Ben Goertzel with Cassio Pennachin & Nil Geisweiller & the OpenCog Team

Engineering General Intelligence, Part 2:

The CogPrime Architecture for Integrative, Embodied AGI

September 19, 2013

EFTA00624128

This book is dedicated by Ben Goertzel to his beloved, departed grandfather, Leo Zwell – an amazingly warm-hearted, giving human being who was also a deep thinker and excellent scientist, who got Ben started on the path of science. As a careful experimentalist, Leo would have been properly skeptical of the big hypotheses made here – but he would have been eager to see them put to the test!

EFTA00624130

Preface

Welcome to the second volume of *Engineering General Intelligence*! This is the second half of a two-part technical treatise aimed at outlining a practical approach to engineering software systems with general intelligence at the human level and ultimately beyond.

Our goal here is an ambitious one and not a modest one: Machines with flexible problem-solving ability, open-ended learning capability, creativity and eventually, their own kind of genius.

Part 1 set the stage, dealing with with a variety of general conceptual issues related to the engineering of advanced AGI, as well as presenting a brief overview of the CogPrime design for Artificial General Intelligence. Now here in Part 2 we plunge deep into the nitty-gritty, and describe the multiple aspects of the CogPrime with a fairly high degree of detail.

First we describe the CogPrime software architecture and knowledge representation in detail; then we review the "cognitive cycle" via which CogPrime perceives and acts in the world and reflects on itself. We then turn to various forms of learning: procedural, declarative (e.g. inference), simulative and integrative. Methods of enabling natural language functionality in CogPrime are then discussed; and the volume concludes with a chapter summarizing the argument that CogPrime can lead to human-level (and eventually perhaps greater) AGI, and a chapter giving a "thought experiment" describing the internal dynamics via which a completed CogPrime system might solve the problem of obeying the request "Build me something with blocks that I haven't seen before."

Reading this book before *Engineering General Intelligence, Part 1* first is not especially recommended, since the prequel not only provides context for this one, but it also defines a number of specific terms and concepts that are used here without explanation (for example, Part One has an extensive Glossary). However, the impatient reader who has not mastered Part 1, or the reader who has finished Part 1 but is tempted to hop through Part 2 nonlinearly, might wish to first skim the final two chapters, and then return to reading in linear order.

While the majority of the text here was written by the lead author Ben Goertzel, the overall work and underlying ideas have been very much a team effort, with major input from the secondary authors Cassio Pennachin and Nil Geisweiller, and large contributions from various other contributors as well. Many chapters have specifically indicated coauthors; but the contributions from various collaborating researchers and engineers go far beyond these. The creation of the AGI approach and design presented here is a process that has occurred over a long period of time among a community of people; and this book is in fact a quite partial view of the existent

EFTA00624132

body of knowledge and intuition regarding CogPrime. For example, beyond the ideas presented here, there is a body of work on the OpenCog wiki site, and then the OpenCog codebase itself.

More extensive introductory remarks may be found in Preface of Part 1, including a brief history of the book and acknowledgements to some of those who helped inspire it.

Also, one brief comment from the Preface of Part 1 bears repeating: At several places in this volume, as in its predecessor, we will refer to the "current" CogPrime implementation (in the OpenCog framework); in all cases this refers to the OpenCog software system as of late 2013.

We fully realize that this book is not "easy reading", and that the level and nature of exposition varies somewhat from chapter to chapter. We have done our best to present these very complex ideas as clearly as we could, given our own time constraints, and the lack of commonly understood vocabularies for discussing many of the concepts and systems involved. Our hope is that the length of the book, and the conceptual difficulty of some portions, will be considered as compensated by the interest of the ideas we present. For, make no mistake – for all their technicality and subtlety, we find the ideas presented here incredibly exciting. We are talking about no less than the creation of machines with intelligence, creativity and genius equaling and ultimately exceeding that of human beings.

This is, in the end, the kind of book that we (the authors) all hoped to find when we first entered the AI field: a reasonably detailed description of how to go about creating thinking machines. The fact that so few treatises of this nature, and so few projects explicitly aimed at the creation of advanced AGI, exist, is something that has perplexed us since we entered the field. Rather than just complain about it, we have taken matters into our own hands, and worked to create a design and a codebase that we believe capable of leading to human-level AGI and beyond.

We feel tremendously fortunate to live in times when this sort of pursuit can be discussed in a serious, scientific way.

Online Appendices

Just one more thing before getting started! This book originally had even more chapters than the ones currently presented in Parts 1 and 2. In order to decrease length and increase focus, however, a number of chapters dealing with peripheral – yet still relevant and interesting – matters were moved to online appendices. These may be downloaded in a single PDF file at http://goertzel.org/engineering_general_Intelligence_appendices_B-H.pdf. The titles of these appendices are:

- Appendix A: Possible Worlds Semantics and Experiential Semantics
- Appendix B: Steps Toward a Formal Theory of Cognitive Structure and Dynamics
- Appendix C: Emergent Reflexive Mental Structures
- Appendix D: GOLEM: Toward an AGI Meta-Architecture Enabling Both Goal Preservation and Radical Self-Improvement
- Appendix E: Lojban++: A Novel Linguistic Mechanism for Teaching AGI Systems
- Appendix F: PLN and the Brain
- Appendix G: Possible Worlds Semantics and Experiential Semantics
- Appendix H: Propositions About Environments in Which CogPrime Components are Useful

EFTA00624133

None of these are critical to understanding the key ideas in the book, which is why they were relegated to online appendices. However, reading them will deepen your understanding of the conceptual and formal perspectives underlying the CogPrime design.

September 2013
Ben Goertzel

EFTA00624134

Contents

Section I Architectural and Representational Mechanisms

19 The OpenCog Framework 3
19.1 Introduction 3
19.1.1 Layers of Abstraction in Describing Artificial Minds 3
19.1.2 The OpenCog Framework 4
19.2 The OpenCog Architecture 5
19.2.1 OpenCog and Hardware Models 5
19.2.2 The Key Components of the OpenCog Framework 6
19.3 The AtomSpace 7
19.3.1 The Knowledge Unit: Atoms 7
19.3.2 AtomSpace Requirements and Properties 8
19.3.3 Accessing the Atomspace 9
19.3.4 Persistence 10
19.3.5 Specialized Knowledge Stores 11
19.4 MindAgents: Cognitive Processes 13
19.4.1 A Conceptual View of CogPrime Cognitive Processes 14
19.4.2 Implementation of MindAgents 15
19.4.3 Tasks 16
19.4.4 Scheduling of MindAgents and Tasks in a Unit 16
19.4.5 The Cognitive Cycle 17
19.5 Distributed AtomSpace and Cognitive Dynamics 18
19.5.1 Distributing the AtomSpace 18
19.5.2 Distributed Processing 23

20 Knowledge Representation Using the Atomspace 27
20.1 Introduction 27
20.2 Denoting Atoms 28
20.2.1 Meta-Language 28
20.2.2 Denoting Atoms 30
20.3 Representing Functions and Predicates 35
20.3.1 Execution Links 36
20.3.2 Denoting Schema and Predicate Variables 39

EFTA00624136

20.3.3 Variable and Combinator Notation . . . 41
20.3.4 Inheritance Between Higher-Order Types . . . 43
20.3.5 Advanced Schema Manipulation . . . 44

21 Representing Procedural Knowledge . . . 49
21.1 Introduction . . . 49
21.2 Representing Programs . . . 50
21.3 Representational Challenges . . . 51
21.4 What Makes a Representation Tractable? . . . 53
21.5 The Combo Language . . . 55
21.6 Normal Forms Postulated to Provide Tractable Representations . . . 55
21.6.1 A Simple Type System . . . 56
21.6.2 Boolean Normal Form . . . 57
21.6.3 Number Normal Form . . . 57
21.6.4 List Normal Form . . . 57
21.6.5 Tuple Normal Form . . . 57
21.6.6 Enum Normal Form . . . 58
21.6.7 Function Normal Form . . . 58
21.6.8 Action Result Normal Form . . . 58
21.7 Program Transformations . . . 59
21.7.1 Reductions . . . 59
21.7.2 Neutral Transformations . . . 60
21.7.3 Non-Neutral Transformations . . . 62
21.8 Interfacing Between Procedural and Declarative Knowledge . . . 63
21.8.1 Programs Manipulating Atoms . . . 63
21.9 Declarative Representation of Procedures . . . 64

Section II The Cognitive Cycle

22 Emotion, Motivation, Attention and Control . . . 67
22.1 Introduction . . . 67
22.2 A Quick Look at Action Selection . . . 68
22.3 Psi in CogPrime . . . 69
22.4 Implementing Emotion Rules atop Psi's Emotional Dynamics . . . 72
22.4.1 Grounding the Logical Structure of Emotions in the Psi Model . . . 73
22.5 Goals and Contexts . . . 73
22.5.1 Goal Atoms . . . 74
22.6 Context Atoms . . . 76
22.7 Ubergoal Dynamics . . . 77
22.7.1 Implicit Ubergoal Pool Modification . . . 77
22.7.2 Explicit Ubergoal Pool Modification . . . 78
22.8 Goal Formation . . . 78
22.9 Goal Fulfillment and Predicate Schematization . . . 79
22.10Context Formation . . . 79
22.11Execution Management . . . 80
22.12Goals and Time . . . 81

EFTA00624137

23 Attention Allocation . . . 83
23.1 Introduction . . . 83
23.2 Semantics of Short and Long Term Importance . . . 85
23.2.1 The Precise Semantics of STI and LTI . . . 86
23.2.2 STI, STIFund, and Juju . . . 89
23.2.3 Formalizing LTI . . . 89
23.2.4 Applications of LTI_{burst} versus LTI_{cont} . . . 90
23.3 Defining Burst LTI in Terms of STI . . . 91
23.4 Valuing LTI and STI in terms of a Single Currency . . . 92
23.5 Economic Attention Networks . . . 94
23.5.1 Semantics of Hebbian Links . . . 94
23.5.2 Explicit and Implicit Hebbian Relations . . . 95
23.6 Dynamics of STI and LTI Propagation . . . 95
23.6.1 ECAN Update Equations . . . 96
23.6.2 ECAN as Associative Memory . . . 101
23.7 Glocal Economic Attention Networks . . . 101
23.7.1 Experimental Explorations . . . 102
23.8 Long-Term Importance and Forgetting . . . 102
23.9 Attention Allocation via Data Mining on the System Activity Table . . . 103
23.10Schema Credit Assignment . . . 104
23.11Interaction between ECANs and other CogPrime Components . . . 106
23.11.1Use of PLN and Procedure Learning to Help ECAN . . . 106
23.11.2Use of ECAN to Help Other Cognitive Processes . . . 106
23.12MindAgent Importance and Scheduling . . . 107
23.13Information Geometry for Attention Allocation . . . 108
23.13.1Brief Review of Information Geometry . . . 108
23.13.2Information-Geometric Learning for Recurrent Networks: Extending the ANGL Algorithm . . . 109
23.13.3Information Geometry for Economic Attention Allocation: A Detailed Example . . . 110

24 Economic Goal and Action Selection . . . 113
24.1 Introduction . . . 113
24.2 Transfer of STI "Requests for Services" Between Goals . . . 114
24.3 Feasibility Structures . . . 116
24.4 Goal Based Schema Selection . . . 116
24.4.1 A Game-Theoretic Approach to Action Selection . . . 117
24.5 SchemaActivation . . . 118
24.6 GoalBasedSchemaLearning . . . 119

25 Integrative Procedure Evaluation . . . 121
25.1 Introduction . . . 121
25.2 Procedure Evaluators . . . 121
25.2.1 Simple Procedure Evaluation . . . 122
25.2.2 Effort Based Procedure Evaluation . . . 122
25.2.3 Procedure Evaluation with Adaptive Evaluation Order . . . 123
25.3 The Procedure Evaluation Process . . . 123

EFTA00624138

25.3.1 Truth Value Evaluation . . . 124
25.3.2 Schema Execution . . . 125

Section III Perception and Action

26 Perceptual and Motor Hierarchies . . . 129
26.1 Introduction . . . 129
26.2 The Generic Perception Process . . . 130
26.2.1 The ExperienceDB . . . 131
26.3 Interfacing CogPrime with a Virtual Agent . . . 131
26.3.1 Perceiving the Virtual World . . . 132
26.3.2 Acting in the Virtual World . . . 133
26.4 Perceptual Pattern Mining . . . 134
26.4.1 Input Data . . . 134
26.4.2 Transaction Graphs . . . 135
26.4.3 Spatiotemporal Conjunctions . . . 135
26.4.4 The Mining Task . . . 136
26.5 The Perceptual-Motor Hierarchy . . . 136
26.6 Object Recognition from Polygonal Meshes . . . 137
26.6.1 Algorithm Overview . . . 138
26.6.2 Recognizing PersistentPolygonNodes (PPNodes) from PolygonNodes . . . 138
26.6.3 Creating Adjacency Graphs from PPNodes . . . 139
26.6.4 Clustering in the Adjacency Graph . . . 140
26.6.5 Discussion . . . 140
26.7 Interfacing the Atomspace with a Deep Learning Based Perception-Action Hierarchy . . . 140
26.7.1 Hierarchical Perception Action Networks . . . 141
26.7.2 Declarative Memory . . . 142
26.7.3 Sensory Memory . . . 142
26.7.4 Procedural Memory . . . 142
26.7.5 Episodic Memory . . . 143
26.7.6 Action Selection and Attention Allocation . . . 144
26.8 Multiple Interaction Channels . . . 144

27 Integrating CogPrime with a Compositional Spatiotemporal Deep Learning Network . . . 147
27.1 Introduction . . . 147
27.2 Integrating CSDLNs with Other AI Frameworks . . . 149
27.3 Semantic CSDLN for Perception Processing . . . 149
27.4 Semantic CSDLN for Motor and Sensorimotor Processing . . . 152
27.5 Connecting the Perceptual and Motoric Hierarchies with a Goal Hierarchy . . . 154

28 Making DeSTIN Representationally Transparent . . . 157
28.1 Introduction . . . 157
28.2 Review of DeSTIN Architecture and Dynamics . . . 158
28.2.1 Beyond Gray-Scale Vision . . . 159
28.3 Uniform DeSTIN . . . 159
28.3.1 Translation-Invariant DeSTIN . . . 160

EFTA00624139

28.3.2 Mapping States of Translation-Invariant DeSTIN into the Atomspace ... 161
28.3.3 Scale-Invariant DeSTIN ... 162
28.3.4 Rotation Invariant DeSTIN ... 163
28.3.5 Temporal Perception ... 164
28.4 Interpretation of DeSTIN's Activity ... 164
28.4.1 DeSTIN's Assumption of Hierarchical Decomposability ... 165
28.4.2 Distance and Utility ... 165
28.5 Benefits and Costs of Uniform DeSTIN ... 166
28.6 Imprecise Probability as a Tool for Linking CogPrime and DeSTIN ... 167
28.6.1 Visual Attention Focusing ... 167
28.6.2 Using Imprecise Probabilities to Guide Visual Attention Focusing ... 168
28.6.3 Sketch of Application to DeSTIN ... 168

29 Bridging the Symbolic/Subsymbolic Gap ... 171
29.1 Introduction ... 171
29.2 Simplified OpenCog Workflow ... 173
29.3 Integrating DeSTIN and OpenCog ... 174
29.3.1 Mining Patterns from DeSTIN States ... 175
29.3.2 Probabilistic Inference on Mined Hypergraphs ... 176
29.3.3 Insertion of OpenCog-Learned Predicates into DeSTIN's Pattern Library 177
29.4 Multisensory Integration, and Perception-Action Integration ... 178
29.4.1 Perception-Action Integration ... 179
29.4.2 Thought-Experiment: Eye-Hand Coordination ... 181
29.5 A Practical Example: Using Subtree Mining to Bridge the Gap Between DeSTIN and PLN ... 182
29.5.1 The Importance of Semantic Feedback ... 184
29.6 Some Simple Experiments with Letters ... 184
29.6.1 Mining Subtrees from DeSTIN States Induced via Observing Letterforms 184
29.6.2 Mining Subtrees from DeSTIN States Induced via Observing Letterforms 185
29.7 Conclusion ... 188

Section IV Procedure Learning

30 Procedure Learning as Program Learning ... 193
30.1 Introduction ... 193
30.1.1 Program Learning ... 193
30.2 Representation-Building ... 195
30.3 Specification Based Procedure Learning ... 196

31 Learning Procedures via Imitation, Reinforcement and Correction ... 197
31.1 Introduction ... 197
31.2 IRC Learning ... 197
31.2.1 A Simple Example of Imitation/Reinforcement Learning ... 198
31.2.2 A Simple Example of Corrective Learning ... 199
31.3 IRC Learning in the PetBrain ... 201
31.3.1 Introducing Corrective Learning ... 203
31.4 Applying A Similar IRC Methodology to Spontaneous Learning ... 203

EFTA00624140

32 Procedure Learning via Adaptively Biased Hillclimbing . . . 205
32.1 Introduction . . . 205
32.2 Hillclimbing . . . 206
32.3 Entity and Perception Filters . . . 207
32.3.1 Entity filter . . . 207
32.3.2 Entropy perception filter . . . 207
32.4 Using Action Sequences as Building Blocks . . . 208
32.5 Automatically Parametrizing the Program Size Penalty . . . 208
32.5.1 Definition of the complexity penalty . . . 208
32.5.2 Parameterizing the complexity penalty . . . 209
32.5.3 Definition of the Optimization Problem . . . 210
32.6 Some Simple Experimental Results . . . 211
32.7 Conclusion . . . 214

33 Probabilistic Evolutionary Procedure Learning . . . 215
33.1 Introduction . . . 215
33.1.1 Explicit versus Implicit Evolution in CogPrime . . . 217
33.2 Estimation of Distribution Algorithms . . . 218
33.3 Competent Program Evolution via MOSES . . . 219
33.3.1 Statics . . . 219
33.3.2 Dynamics . . . 222
33.3.3 Architecture . . . 223
33.3.4 Example: Artificial Ant Problem . . . 224
33.3.5 Discussion . . . 229
33.3.6 Conclusion . . . 229
33.4 Integrating Feature Selection Into the Learning Process . . . 230
33.4.1 Machine Learning, Feature Selection and AGI . . . 231
33.4.2 Data- and Feature- Focusable Learning Problems . . . 232
33.4.3 Integrating Feature Selection Into Learning . . . 233
33.4.4 Integrating Feature Selection into MOSES Learning . . . 234
33.4.5 Application to Genomic Data Classification . . . 234
33.5 Supplying Evolutionary Learning with Long-Term Memory . . . 236
33.6 Hierarchical Program Learning . . . 237
33.6.1 Hierarchical Modeling of Composite Procedures in the AtomSpace . . . 238
33.6.2 Identifying Hierarchical Structure In Combo trees via MetaNodes and Dimensional Embedding . . . 239
33.7 Fitness Function Estimation via Integrative Intelligence . . . 242

Section V Declarative Learning

34 Probabilistic Logic Networks . . . 247
34.1 Introduction . . . 247
34.2 A Simple Overview of PLN . . . 248
34.2.1 Forward and Backward Chaining . . . 249
34.3 First Order Probabilistic Logic Networks . . . 250
34.3.1 Core FOPLN Relationships . . . 250
34.3.2 PLN Truth Values . . . 251

EFTA00624141

34.3.3 Auxiliary FOPLN Relationships . . . 251
34.3.4 PLN Rules and Formulas . . . 252
34.3.5 Inference Trails . . . 253
34.4 Higher-Order PLN . . . 254
34.4.1 Reducing HOPLN to FOPLN . . . 255
34.5 Predictive Implication and Attraction . . . 256
34.6 Confidence Decay . . . 257
34.6.1 An Example . . . 258
34.7 Why is PLN a Good Idea? . . . 260

35 Spatiotemporal Inference . . . 263
35.1 Introduction . . . 263
35.2 Related Work on Spatio-temporal Calculi . . . 264
35.3 Uncertainty with Distributional Fuzzy Values . . . 267
35.4 Spatio-temporal Inference in PLN . . . 270
35.5 Examples . . . 272
35.5.1 Spatiotemporal Rules . . . 272
35.5.2 The Laptop is Safe from the Rain . . . 273
35.5.3 Fetching the Toy Inside the Upper Cupboard . . . 273
35.6 An Integrative Approach to Planning . . . 275

36 Adaptive, Integrative Inference Control . . . 277
36.1 Introduction . . . 277
36.2 High-Level Control Mechanisms . . . 277
36.2.1 The Need for Adaptive Inference Control . . . 278
36.3 Inference Control in PLN . . . 279
36.3.1 Representing PLN Rules as GroundedSchemaNodes . . . 279
36.3.2 Recording Executed PLN Inferences in the Atomspace . . . 279
36.3.3 Anatomy of a Single Inference Step . . . 280
36.3.4 Basic Forward and Backward Inference Steps . . . 281
36.3.5 Interaction of Forward and Backward Inference . . . 282
36.3.6 Coordinating Variable Bindings . . . 282
36.3.7 An Example of Problem Decomposition . . . 284
36.3.8 Example of Casting a Variable Assignment Problem as an Optimization Problem . . . 284
36.3.9 Backward Chaining via Nested Optimization . . . 285
36.4 Combining Backward and Forward Inference Steps with Attention Allocation to Achieve the Same Effect as Backward Chaining (and Even Smarter Inference Dynamics) . . . 288
36.4.1 Breakdown into MindAgents . . . 289
36.5 Hebbian Inference Control . . . 289
36.6 Inference Pattern Mining . . . 293
36.7 Evolution As an Inference Control Scheme . . . 293
36.8 Incorporating Other Cognitive Processes into Inference . . . 294
36.9 PLN and Bayes Nets . . . 295

EFTA00624142

37 Pattern Mining . . . 297
37.1 Introduction . . . 297
37.2 Finding Interesting Patterns via Program Learning . . . 298
37.3 Pattern Mining via Frequent/Surprising Subgraph Mining . . . 299
37.4 Fishgram . . . 300
37.4.1 Example Patterns . . . 300
37.4.2 The Fishgram Algorithm . . . 301
37.4.3 Preprocessing . . . 302
37.4.4 Search Process . . . 303
37.4.5 Comparison to other algorithms . . . 304

38 Speculative Concept Formation . . . 305
38.1 Introduction . . . 305
38.2 Evolutionary Concept Formation . . . 306
38.3 Conceptual Blending . . . 308
38.3.1 Outline of a CogPrime Blending Algorithm . . . 310
38.3.2 Another Example of Blending . . . 311
38.4 Clustering . . . 312
38.5 Concept Formation via Formal Concept Analysis . . . 312
38.5.1 Calculating Membership Degrees of New Concepts . . . 313
38.5.2 Forming New Attributes . . . 313
38.5.3 Iterating the Fuzzy Concept Formation Process . . . 314

Section VI Integrative Learning

39 Dimensional Embedding . . . 319
39.1 Introduction . . . 319
39.2 Link Based Dimensional Embedding . . . 320
39.3 Harel and Koren's Dimensional Embedding Algorithm . . . 322
39.3.1 Step 1: Choosing Pivot Points . . . 322
39.3.2 Step 2: Similarity Estimation . . . 323
39.3.3 Step 3: Embedding . . . 323
39.4 Embedding Based Inference Control . . . 323
39.5 Dimensional Embedding and InheritanceLinks . . . 325

40 Mental Simulation and Episodic Memory . . . 327
40.1 Introduction . . . 327
40.2 Internal Simulations . . . 328
40.3 Episodic Memory . . . 328

41 Integrative Procedure Learning . . . 333
41.1 Introduction . . . 333
41.1.1 The Diverse Technicalities of Procedure Learning in CogPrime . . . 334
41.2 Preliminary Comments on Procedure Map Encapsulation and Expansion . . . 336
41.3 Predicate Schematization . . . 337
41.3.1 A Concrete Example . . . 339
41.4 Concept-Driven Schema and Predicate Creation . . . 340
41.4.1 Concept-Driven Predicate Creation . . . 340

EFTA00624143

41.4.2 Concept-Driven Schema Creation . . . 341
41.5 Inference-Guided Evolution of Pattern-Embodying Predicates . . . 342
41.5.1 Rewarding Surprising Predicates . . . 342
41.5.2 A More Formal Treatment . . . 344
41.6 PredicateNode Mining . . . 345
41.7 Learning Schema Maps . . . 346
41.7.1 Goal-Directed Schema Evolution . . . 347
41.8 Occam's Razor . . . 349

42 Map Formation . . . 351
42.1 Introduction . . . 351
42.2 Map Encapsulation . . . 353
42.3 Atom and Predicate Activity Tables . . . 355
42.4 Mining the AtomSpace for Maps . . . 356
42.4.1 Frequent Itemset Mining for Map Mining . . . 357
42.4.2 Evolutionary Map Detection . . . 359
42.5 Map Dynamics . . . 359
42.6 Procedure Encapsulation and Expansion . . . 360
42.6.1 Procedure Encapsulation in More Detail . . . 361
42.6.2 Procedure Encapsulation in the Human Brain . . . 361
42.7 Maps and Focused Attention . . . 362
42.8 Recognizing and Creating Self-Referential Structures . . . 363
42.8.1 Encouraging the Recognition of Self-Referential Structures in the AtomSpace . . . 364

Section VII Communication Between Human and Artificial Minds

43 Communication Between Artificial Minds . . . 369
43.1 Introduction . . . 369
43.2 A Simple Example Using a PsyneseVocabulary Server . . . 371
43.2.1 The Psynese Match Schema . . . 373
43.3 Psynese as a Language . . . 373
43.4 Psynese Mindplexes . . . 374
43.4.1 AGI Mindplexes . . . 375
43.5 Psynese and Natural Language Processing . . . 376
43.5.1 Collective Language Learning . . . 378

44 Natural Language Comprehension . . . 379
44.1 Introduction . . . 379
44.2 Linguistic Atom Types . . . 381
44.3 The Comprehension and Generation Pipelines . . . 382
44.4 Parsing with Link Grammar . . . 383
44.4.1 Link Grammar vs. Phrase Structure Grammar . . . 385
44.5 The RelEx Framework for Natural Language Comprehension . . . 386
44.5.1 RelEx2Frame: Mapping Syntactico-Semantic Relationships into FrameNet Based Logical Relationships . . . 387
44.5.2 A Priori Probabilities For Rules . . . 389
44.5.3 Exclusions Between Rules . . . 389

EFTA00624144

44.5.4 Handling Multiple Prepositional Relationships 390
44.5.5 Comparatives and Phantom Nodes 391
44.6 Frame2Atom 392
44.6.1 Examples of Frame2Atom 393
44.6.2 Issues Involving Disambiguation 396
44.7 Syn2Sem: A Semi-Supervised Alternative to RelEx and RelEx2Frame 397
44.8 Mapping Link Parses into Atom Structures 398
44.8.1 Example Training Pair 399
44.9 Making a Training Corpus 399
44.9.1 Leveraging RelEx to Create a Training Corpus 399
44.9.2 Making an Experience Based Training Corpus 399
44.9.3 Unsupervised, Experience Based Corpus Creation 400
44.10Limiting the Degree of Disambiguation Attempted 400
44.11Rule Format 401
44.11.1Example Rule 402
44.12Rule Learning 402
44.13Creating a Cyc-Like Database via Text Mining 403
44.14PROWL Grammar 404
44.14.1Brief Review of Word Grammar 405
44.14.2Word Grammar's Logical Network Model 406
44.14.3Link Grammar Parsing vs Word Grammar Parsing 407
44.14.4Contextually Guided Greedy Parsing and Generation Using Word Link Grammar 411
44.15Aspects of Language Learning 413
44.15.1 Word Sense Creation 413
44.15.2Feature Structure Learning 414
44.15.3Transformation and Semantic Mapping Rule Learning 414
44.16Experiential Language Learning 415
44.17Which Path(s) Forward? 416

45 Language Learning via Unsupervised Corpus Analysis 417
45.1 Introduction 417
45.2 Assumed Linguistic Infrastructure 419
45.3 Linguistic Content To Be Learned 421
45.3.1 Deeper Aspects of Comprehension 423
45.4 A Methodology for Unsupervised Language Learning from a Large Corpus 423
45.4.1 A High Level Perspective on Language Learning 424
45.4.2 Learning Syntax 426
45.4.3 Learning Semantics 430
45.5 The Importance of Incremental Learning 434
45.6 Integrating Language Learned via Corpus Analysis into CogPrime's Experiential Learning 435

46 Natural Language Generation 437
46.1 Introduction 437
46.2 SegSim for Sentence Generation 437
46.2.1 NLGen: Example Results 441

EFTA00624145

46.3 Experiential Learning of Language Generation . . . 444
46.4 Sem2Syn . . . 445
46.5 Conclusion . . . 445

47 Embodied Language Processing . . . 447
47.1 Introduction . . . 447
47.2 Semiosis . . . 448
47.3 Teaching Gestural Communication . . . 450
47.4 Simple Experiments with Embodiment and Anaphor Resolution . . . 455
47.5 Simple Experiments with Embodiment and Question Answering . . . 456
47.5.1 Preparing/Matching Frames . . . 456
47.5.2 Frames2RelEx . . . 458
47.5.3 Example of the Question Answering Pipeline . . . 458
47.5.4 Example of the PetBrain Language Generation Pipeline . . . 459
47.6 The Prospect of Massively Multiplayer Language Teaching . . . 460

48 Natural Language Dialogue . . . 463
48.1 Introduction . . . 463
48.1.1 Two Phases of Dialogue System Development . . . 464
48.2 Speech Act Theory and its Elaboration . . . 464
48.3 Speech Act Schemata and Triggers . . . 465
48.3.1 Notes Toward Example SpeechActSchema . . . 467
48.4 Probabilistic Mining of Trigger contexts . . . 471
48.5 Conclusion . . . 473

Section VIII From Here to AGI

49 Summary of Argument for the CogPrime Approach . . . 477
49.1 Introduction . . . 477
49.2 Multi-Memory Systems . . . 477
49.3 Perception, Action and Environment . . . 478
49.4 Developmental Pathways . . . 479
49.5 Knowledge Representation . . . 480
49.6 Cognitive Processes . . . 480
49.6.1 Uncertain Logic for Declarative Knowledge . . . 481
49.6.2 Program Learning for Procedural Knowledge . . . 482
49.6.3 Attention Allocation . . . 483
49.6.4 Internal Simulation and Episodic Knowledge . . . 484
49.6.5 Low-Level Perception and Action . . . 484
49.6.6 Goals . . . 485
49.7 Fulfilling the "Cognitive Equation" . . . 485
49.8 Occam's Razor . . . 486
49.8.1 Mind Geometry . . . 486
49.9 Cognitive Synergy . . . 488
49.9.1 Synergies that Help Inference . . . 488
49.10Synergies that Help MOSES . . . 489
49.10.1Synergies that Help Attention Allocation . . . 489
49.10.2Further Synergies Related to Pattern Mining . . . 489

EFTA00624146

49.10.3Synergies Related to Map Formation 490
49.11Emergent Structures and Dynamics 490
49.12Ethical AGI 491
49.13Toward Superhuman General Intelligence 492
49.13.1Conclusion 492

50 Build Me Something I Haven't Seen: A CogPrime Thought Experiment . . 495
50.1 Introduction 495
50.2 Roles of Selected Cognitive Processes 496
50.3 A Semi-Narrative Treatment 506
50.4 Conclusion 509

A Glossary 511
A.1 List of Specialized Acronyms 511
A.2 Glossary of Specialized Terms 512
References 529

EFTA00624147

Section I
Architectural and Representational Mechanisms

Chapter 19
The OpenCog Framework

19.1 Introduction

The primary burden of this book is to explain the CogPrime architecture for AGI – the broad outline of the design, the main dynamics it's intended to display once complete, and the reasons why we believe it will be capable of leading to general intelligence at the human level and beyond.

The crux of CogPrime lies in its learning algorithms and how they are intended to interact together synergetically, making use of CogPrime's knowledge representation and other tools. Before we can get to this, however, we need to elaborate some of the "plumbing" within which this learning dynamics occurs. We will start out with a brief description of the OpenCog framework in which implementation of CogPrime has been, gradually and incrementally, occurring for the last few years.

19.1.1 Layers of Abstraction in Describing Artificial Minds

There are multiple layers intervening between a conceptual theory of mind and a body of source code. How many layers to explicitly discuss is a somewhat arbitrary decision, but one way to picture it is exemplified in Table 19.1.

In Part 1 of this work we have concerned ourselves mainly with levels 5 and 6 in the table: mathematical/conceptual modeling of cognition and philosophy of mind (with occasional forays into levels 3 and 4). Most of Part 2, on the other hand, deals with level 4 (mathematical/conceptual AI design), verging into level 3 (high-level software design). This chapter however will focus on somewhat lower-level material, mostly level 3 with some dips into level 2. We will describe the basic architecture of CogPrime as a software system, implemented as "OpenCogPrime" within the OpenCog Framework (OCF). The reader may want to glance back at Chapter 6 of Part 1 before proceeding through this one, to get a memory-refresh on basic CogPrime terminology. Also, OpenCog and OpenCogPrime are open-source, so the reader who wishes to dig into the source code (mostly C++, some Python and Scheme) is welcome to; directions to find the code are on the opencog.org website.

EFTA00624150

Level of Abstraction	Description/Example
1 Source code	
2 Detailed software design	
3 Software architecture	Largely programming-language-independent, but not hardware-architecture-independent: much of the material in this chapter, for example, and most of the OpenCog Framework
4 Mathematical and conceptual AI design	e.g., the sort of characterization of CogPrime given in most of this Part of this book
5 Abstract mathematical modeling of cognition	e.g. the SRAM model discussed in chapter 7 of Part 1, which could be used to inspire or describe many different AI systems
6 Philosophy of mind	e.g. Patternism, the Mind-World Correspondence Principle

Table 19.1: Levels of abstractions in CogPrime's implementation and design

19.1.2 The OpenCog Framework

The OpenCog Framework forms a bridge between the mathematical structures and dynamics of CogPrime's *concretely implemented mind,* and the nitty-gritty realities of modern computer technology. While CogPrime could in principle be implemented in a quite different infrastructure, in practice the CogPrime design has been developed closely in conjunction with OpenCog, so that a qualitative understanding of the nature of the OCF is fairly necessary for an understanding of how CogPrime is intended to function, and a detailed understanding of the OCF is necessary for doing concrete implementation work on CogPrime.

Marvin Minsky, in a personal conversation with one of the authors (Goertzel), once expressed the opinion that a human-level general intelligence could probably be implemented on a 486 PC, if we just knew the algorithm. We doubt this is the case – at least not unless the 486 PC were supplied with masses of external memory and allowed to proceed much, much slower than any human being – and it is certainly not the case for CogPrime. By current computing hardware standards, a CogPrime system is a considerable resource hog. And it will remain so for a number of years, even considering technology progress.

It is one of the jobs of the OCF to manage the system's gluttonous behavior. It is the software layer that abstracts the real world efficiency compromises from the rest of the system; this is why we call it a "Mind OS": it provides services, rules, and protection to the Atoms and cognitive processes (see Section 19.4) that live on top of it, which are then allowed to ignore the software architecture they live on.

And so, the nature of the OCF is strongly influenced by the quantitative requirements imposed on the system, as well as the general nature of the structure and dynamics that it must support. The large number and great diversity of Atoms needed to create a significantly intelligent CogPrime, demands that we pay careful attention to such issues as concurrent, distributed processing, and scalability in general. The number of Nodes and Links that we will need in order to create a reasonably complete CogPrime is still largely unknown. But our experiments with learning, natural language processing, and cognition over the past few years have given us an intuition for the question. We currently believe that we are likely to need billions – but probably not trillions, and almost surely not quadrillions – of Atoms in order to achieve a high degree of general intelligence. Hundreds of millions strikes us as possible but overly optimistic.

EFTA00624151

In fact we have already run CogPrime systems utilizing hundreds of millions of Atoms, though in a simplified dynamical regime with only a couple very simple processes acting on most of them.

The operational infrastructure of the OCF is an area where pragmatism must reign over idealism. What we describe here is not the ultimate possible "mind operating system" to underlie a CogPrime system, but rather a workable practical solution given the hardware, networking and software infrastructure readily available today at reasonable prices. Along these lines, it must be emphasized that the ideas presented in this chapter are the result of over a decade of practical experimentation by the authors and their colleagues with implementations of related software systems. The journey began in earnest in 1997 with the design and implementation of the Webmind AI Engine at Intelligenesis Corp., which itself went through a few major design revisions; and then in 2001-2002 the Novamente Cognition Engine was architected and implemented, and evolved progressively until 2008, when a subset of it was adapted for open-sourcing as OpenCog. Innumerable mistakes were made, and lessons learned, along this path. The OCF as described here is significantly different, and better, than these previous architectures, thanks to these lessons, as well as to the changing landscape of concurrent, distributed computing over the past few years.

The design presented here reflects a mix of realism and idealism, and we haven't seen fit here to describe all the alternatives that were pursued on the route to what we present. We don't claim the approach we've chosen is ideal, but it's in use now within the OpenCog system, and it seems both workable in practice and capable of effectively supporting the entire CogPrime design. No doubt it will evolve in some respects as implementation progresses; one of the principles kept in mind during the design and development of OpenCog was modularity, enabling substantial modifications to particular parts of the framework to occur without requiring wholesale changes throughout the codebase.

19.2 The OpenCog Architecture

19.2.1 OpenCog and Hardware Models

The job of the OCF is closely related to the nature of the hardware on which it runs. The ideal hardware platform for CogPrime would be a massively parallel hardware architecture, in which each Atom was given its own processor and memory. The closest thing would have been the Connection Machine [Hil89]: a CM5 was once built with 64000 processors and local RAM for each processor. But even 64000 processors wouldn't be enough for a highly intelligent CogPrime to run in a fully parallelized manner, since we're sure we need more than 64000 Atoms.

Connection Machine style hardware seems to have perished in favor of more standard SMP (Symmetric Multi-Processing) machines. It is true that each year we see SMP machines with more and more processors on the market, and more and more cores per processor. However, the state of the art is still in the hundreds of cores range, many orders of magnitude from what would be necessary for a one Atom per processor CogPrime implementation.

So, at the present time, technological and financial reasons have pushed us to implement the OpenCog system using a relatively mundane and standard hardware architecture. If the CogPrime project is successful in the relatively near term, the first human-level OpenCogPrime system will most likely live on a network of high-end commodity SMP machines. These are

EFTA00624152

machines with dozens of gigabytes of RAM and several processor cores, perhaps dozens but not thousands. A highly intelligent CogPrime would require a cluster of dozens and possibly hundreds or thousands of such machines. We think it's unlikely that tens of thousands will be required, and extremely unlikely that hundreds of thousands will be.

Given this sort of architecture, we need effective ways to swap Atoms back and forth between disk and RAM, and carefully manage the allocation of processor time among the various cognitive processes that demand it. The use of a widely-distributed network of weaker machines for peripheral processing is a serious possibility, and we have some detailed software designs addressing this option; but for the near future we believe that this can best be used as augmentation to core CogPrime processing, which must remain on a dedicated cluster.

Of course, the use of specialized hardware is also a viable possibility, and we have considered a host of possibilities such as

- True supercomputers like those created by IBM or Cray (which these days are distributed systems, but with specialized, particularly efficient interconnection frameworks and overall control mechanisms)
- GPU supercomputers such as the Nvidia Tesla (which are currently being used for vision processing systems considered for hybridization with OCP), such as DeSTIN and Hugo de Garis's Parcone
- custom chips designed to implement the various CogPrime algorithms and data structures in hardware
- More speculatively, it might be possible to use evolutionary quantum computing or adiabatic quantum computing a la Dwave (http://dwave.com) to accelerate CogPrime procedure learning.

All these possibilities and many more are exciting to envision, but the CogPrime architecture does not require any of them in order to be successful.

19.2.2 The Key Components of the OpenCog Framework

Given the realities of implementing CogPrime on clustered commodity servers, as we have seen above, the three key questions that have to be answered in the OCF design are:

1. How do we store CogPrime's knowledge?
2. How do we enable cognitive processes to act on that knowledge, refining and improving it?
3. How do we enable scalable, distributed knowledge storage and cognitive processing of that knowledge?

The remaining sections of this Chapter are dedicated to answering each of these questions in more detail.

While the basic landscape of concurrent, distributed processing is largely the same as it was a decade ago – we're still dealing with distributed networks of multiprocessor von Neumann machines – we can draw on advancements in both computer architecture and software. The former is materialized on the increasing availability of multiple real and virtual cores in commodity processors. The latter reflects the emergence of a number of tools and architectural patterns, largely thanks to the rise of "big data" problems and businesses. Companies and projects dealing

EFTA00624153

with massive datasets face challenges that aren't entirely alike those of building CogPrime, but which share many useful similarities.

These advances are apparent mostly in the architectute of the AtomSpace, a distributed knowledge store for efficient storage of hypergraphs and its use by CogPrime's cognitive dynamics. The AtomSpace, like many *NoSQL* datastores, is heavily distributed, utilizing local caches for read and write operations, and a special purpose design for eventual consistency guarantees.

We also attempt to minimize the complexities of multi-threading in the scheduling of cognitive dynamics, by allowing those to be deployed either as agents sharing a single OS process, or, preferably, as processes of their own. Cognitive dynamics communicate through message queues, which are provided by a sub-system that hides the deployment decision, so the messages exchanged are the same whether delivered within a process, to another process in the same machine, or to a process in another machine in the cluster.

19.3 The AtomSpace

As alluded to above and in Chapter 13, and discussed more fully in Chapter 20 below, the foundation of CogPrime's knowledge representation is the Atom, an object that can be either a Node or a Link. CogPrime's hypergraph is implemented as the AtomSpace, a specialized datastore that comes along with an API designed specifically for CogPrime's requirements.

19.3.1 The Knowledge Unit: Atoms

Atoms are used to represent every kind of knowledge in the system's memory in one way or another. The particulars of Atoms and how they represent knowledge will be discussed in later chapters; here we present only a minimal description in order to motivate the design of the AtomSpace. From that perspective, the most important properties of Atoms are:

- Every Atom has an AtomHandle, which is a universal ID across a CogPrime deployment (possibly involving thousands of networked machines). The AtomHandles are the keys for acessing Atoms in the AtomSpace, and once a handle is assigned to an Atom it can't be changed or reused.
- Atoms have TruthValue and AttentionValue entities associated with them, each of which are small collections of numbers; there are multiple versions of truth values, with varying degrees of detail. TruthValues are context-dependent, and useful Atoms will typically have multiple TruthValues, indexed by context.
- Some Atoms are nodes, and may have names.
- Atoms that are links will have a list of targets, of variable size (as in CogPrime's hypergraph links may connect more than two nodes).

Some Atom attributes are immutable, such as Node names and, most importantly, Link targets, called outgoing sets in AtomSpace lingo. One can remove a Link, but not change its targets. This enables faster implementation of some neighborhood searches, as well as index-

EFTA00624154

ing. Truth and attention values, on the other hand, are mutable, an essential requirement for CogPrime.

For performance reasons, some types of knowledge have alternative representations. These alternative representations are necessary for space or speed reasons, but knowledge stored that way can always be translated back into Atoms in the AtomSpace as needed. So, for instance, procedures are represented as program trees in a ProcedureRepository, which allows for faster execution, but the trees can be expanded into a set of Nodes and Links if one wants to do reasoning on a specific program.

19.3.2 AtomSpace Requirements and Properties

The major high-level requirements for the AtomSpace are the following ones:

- Store Atoms indexed by their immutable AtomHandles as compactly as possible, while still enabling very efficient modification of the mutable properties of an Atom (TruthValues and AttentionValues).
- Perform queries as fast as possible.
- Keep the working set of all Atoms currently being used by CogPrime's cognitive dynamics in RAM.
- Save and restore hypergraphs to disk, a more traditional SQL or non-SQL database, or other structure such as binary files, XML, etc.
- Hold hypergraphs consisting of billions or trillions of Atoms, scaling up to petabytes of data.
- Be transparently distributable across a cluster of machines.

The design trade-offs in the AtomSpace implementation are driven by the needs of CogPrime. The datastore is implemented in a way that maximizes the performance of the cognitive dynamics running on top of it. From this perspective, the AtomSpace differs from most datastores, as the key decisions aren't made in terms of flexibility, consistency, reliability and other common criteria for databases. It is a very specialized database. Among the factors that motivate the AtomSpace's design, we can highlight a few:

1. Atoms tend to be small objects, with very few exceptions (links with many targets or Atoms with many different context-derived TruthValues).
2. Atom creation and deletion are common events, and occur according to complex patterns that may vary a lot over time, even for a particular CogPrime instance.
3. Atoms involved in CogPrime's cognitive dynamics at any given time need to live in RAM. However, the system still needs the ability to save sets of Atoms to disk in order to preserve RAM, and then retrive those later when they get contextually relevant.
4. Some Atoms will remain around for a really long time, others will be ephemeral and get removed shortly after they're created. Removal may be to disk, as outlined above, or plain deletion.

Besides storing Atoms, the AtomSpace also contains a number of indices for fast Atom retrieval according to several criteria. It can quickly search for Atoms given their type, importance, truth value, arity, targets (for Links), name (for Nodes), and any combination of the above. These are built-in indexes. The AtomSpace also allows cognitive processes to create their own

EFTA00624155

indexes, based on the evaluation of a Procedure over the universe of Atoms, or a subset of that universe specified by the process responsible for the index.

The AtomSpace also allows pattern matching queries for a given Atom structure template, which allows for fast search for small subgraphs displaying some desirable properties. In addition to pattern matching, it provides neighborhood searches. Although it doesn't implement any graph-traversal primitives, it's easy for cognitive processes to do so on top of the pattern matching and neighborhood primitives.

Note that, since CogPrime's hypergraph is quite different from a regular graph, using a graph database without modification would probably be inadequate. While it's possible to automatically translate a hypergraph into a regular graph, that process is expensive for large knowledge bases, and leads to higher space requirements, reducing the overall system's scalability.

In terms of database taxonomy, the AtomSpace lies somewhere between a key-value store and a document store, as there is some structure in the contents of each value (an Atom's properties are well defined, and listed above), but no built-in flexibility to add more contents to an existing Atom.

We will now discuss the above requirements in more detail, starting with querying the AtomSpace, followed by persistence to disk, and then handling of specific forms of knowledge that are best handled by specialized stores.

19.3.3 Accessing the Atomspace

The AtomSpace provides an API, which allows the basic operations of creating new Atoms, updating their mutable properties, searching for Atoms and removing Atoms. More specifically, the API supports the following operations:

- Create and store a new Atom. There are special methods for Nodes and Links, in the latter case with multiple convenience versions depending on the number of targets and other properties of the link.
- Remove an Atom. This requires the validation that no Links currently point to that Atom, otherwise they'd be left dangling.
- Look up one or more Atoms. This includes several variants, such as:
  - Look up an Atom by AtomHandle;
  - Look up a Node by name;
  - Find links with an Atom as target;
  - Pattern matching, i.e., find Atoms satisfying some predicate, which is designed as a "search criteria" by some cognitive process, and results in the creation of a specific index for that predicate;
  - Neighborhood search, i.e., find Atoms that are within some radius of a given centroid Atom;
  - Find Atoms by type (this can be combined with the previous queries, resulting in type specific versions);
  - Find Atoms by some AttentionValue criteria, such as the top N most important Atoms, or those with importance above some threshold (can also be combined with previous queries);

EFTA00624156

  - Find Atoms by some TruthValue criteria, similar to the previous one (can also be combined with other queries);
  - Find Atoms based on some temporal or spatial association, a query that relies on the specialized knowledge stores mentioned below;

  Queries can be combined, and the Atom type, AttentionValue and TruthValue criteria are often used as filters for other queries, preventing the result set size from exploding.
- Manipulate an Atom, retrieving or modifying its AttentionValue and TruthValue. In the modification case, this causes the respective indexes to be updated.

19.3.4 Persistence

In many planned CogPrime deployment scenarios, the amount of knowledge that needs to be stored is too vast to fit in RAM, even if one considers a large cluster of machines hosting the AtomSpace and the cognitive processes. The AtomSpace must then be able to persist subsets of that knowledge to disk, and reload them later when necessary.

The decision of whether to keep an Atom in RAM or remove it is made based on its AttentionValue, through the process of economic attention allocation that is the topic of Chapter 23. AttentionValue determines how important an Atom is to the system, and there are multiple levels of importance. For the persistence decisions, the ones that matter are Long Term Importance (LTI) and Very Long Term Importance (VLTI).

LTI is used to estimate the probability that the Atom will be necessary or useful in the not too distant future. If this value is low, below a threshold i_1, then it is safe to remove the Atom from RAM, a process called *forgetting*. When the decision to forget an Atom has been made, VLTI enters the picture. VLTI is used to estimate the probability that the Atom will be useful eventually at some distant point in the future. If VLTI is high enough, the forgotten Atom is persisted to disk so it can be reloaded. Otherwise, the Atom is permanently forgotten.

When an Atom has been forgotten, a proxy is kept in its place. The proxy is more compact than the original Atom, preserving only a crude measure of its LTI. When the proxy's LTI increases above a second threshold i_2, the system understands that the Atom has become relevant again, and loads it from disk.

Eventually, it may happen that the proxy doesn't become important enough over a very long period of time. In this case, the system should remove even the proxy, if its Long Term Importance (LTI) is below a third threshold i_3. Other actions, usually taken by the system administrator, can cause the removal of Atoms and their proxies from RAM. For instance, in a CogPrime system managing information about a number of users of some information system, the deletion of a user from the system would cause all that user's specific Atoms to be removed.

When Atoms are saved to disk and have no proxies in RAM, they can only be reloaded by the system administrator. When reloaded, they will be disconnected from the rest of the AtomSpace, and should be given special attention in order to pursue the creation of new Links with the other Atoms in the system.

It's important that the values of i_1, i_2, and i_3 be set correctly. Otherwise, one or more of the following problems may arise:

- If i_1 and i_2 are too close, the system may spend a lot of resources with saving and loading Atoms.

EFTA00624157

- If i_1 is set too high, important Atoms will be excluded from the system's dynamics, decreasing its intelligence.
- If i_3 is set too high, the system will forget very quickly and will have to spend resources re-creating necessary but no longer available evidence.
- If either i_1 or i_3 is set too low, the system will consume significantly more resources than it needs to with knowledge store, sacrificing cognitive processes.

Generally, we want to enforce a degree of hysteresis for the freezing and defrosting process. What we mean is that:

$$i_2 - i_1 > c_1 > 0$$

$$i_1 - i_3 > c_2 > 0$$

This ensures that when Atoms are reloaded, their importance is still above the threshold for saving, so they will have a chance to be part of cognitive dynamics and become more important, and won't be removed again too quickly. It also ensures that saved Atoms stay in the system for a period of time before their proxies are removed and they're definitely forgotten.

Another important consideration is that forgetting individual Atoms makes little sense, because, as pointed out above, Atoms are relatively small objects. So the forgetting process should prioritize the removal of clusters of highly interconnected Atoms whenever possible. In that case, it's possible that a large subset of those Atoms will only have relations within the cluster, so their proxies aren't needed and the memory savings are maximized.

19.3.5 Specialized Knowledge Stores

Some specific kinds of knowledge are best stored in specialized data structures, which allow big savings in space, query time, or both. The information provided by these specialized stores isn't as flexible as it would be if the knowledge were stored in full fledged Node and Link form, but most of the time CogPrime doesn't need the fully flexible format. Translation between the specialized formats and Nodes and Links is always possible, when necessary.

We note that the ideal set of specialized knowledge stores is application domain specific. The stores we have deemed necessary reflect the pre-school based roadmap towards AGI, and are likely sufficient to get us through most of that roadmap, but not sufficient nor particularly adequate for an architecture where self-modification plays a key role. These specialized stores are a pragmatic compromise between performance and formalism, and their existence and design would need to be revised once CogPrime is mostly functional.

19.3.5.1 Procedure Repository

Procedural knowledge, meaning knowledge that can be used both for the selection and execution of actions, has a specialized requirement – this knowledge needs to be *executable* by the system. While it will be possible, and conceptually straightforward, to execute a procedure that is stored as a set of Atoms in the AtomSpace, it is much simpler, faster, and safer to rely on a specialized repository.

EFTA00624158

Procedural knowledge in CogPrime is stored as *programs* in a special-purpose LISP-like programming language called *Combo*. The motivation and details of this language are the subject of Chapter 21.

Each Combo program is associated with a Node (a GroundedProcedureNode, to be more precise), and the AtomHandle of that Node is used to index the procedure repository, where the executable version of the program is kept, along with specifications of the necessary inputs for its evaluation and what kind of output to expect. Combo programs can also be saved to disk and loaded, like regular Atoms. There is a text representation of Combo for this purpose.

Program execution can be very fast, or, in cognitive dynamics terms, very slow, if it involves interacting with the external world. Therefore, the procedure repository should also facilitate the storage of program states during the execution of procedures. Concurrent execution of many procedures is possible with no significant overhead.

19.3.5.2 3D Space Map

In the AGI Preschool setting, CogPrime is embodied in a three-dimensional world (either a real one, in which it controls a robot, or a virtual one, in which it controls an avatar). This requires the efficient storage and querying of vast amounts of spatial data, including very specialized queries about the spacial interrelationship between entities. This spatial data is a key form of knowledge for CogPrime's world perception, and it also needs to be accessible during learning, action selection, and action execution.

All spatial knowledge is stored in a 3D Space Map, which allows for fast queries about specific regions of the world, and for queries about the proximity and relative placement of objects and entities. It can be used to provide a coarse-grained object level perception for the AtomSpace, or it can be instrumental in supporting a lower level vision layer in which pixels or polygons are used as the units of perception. In both cases, the knowledge stored in the 3D Space Map can be translated into full-fledged Atoms and Links through the AtomHandles.

One characteristic feature of spatial perception is that vast amounts of data are generated constantly, but most of it is very quickly forgotten. The mind abstracts the perceptual data into the relevant concepts, which are linked with other Atoms, and most of the underlying information can then be discarded. The process is repeated at a high frequency as long as something novel is being perceived in the world. 3D Space Map is then optimized for quick inserts and deletes.

19.3.5.3 Time Server

Similarly to spatial information, temporal information poses challenges for a hypergraph-based storage. It can be much more compactly stored in specific data structures, which also allow for very fast querying. The Time Server is the specialized structure for storing and querying temportal data in CogPrime.

Temporal information can be stored by any cognitive process, based on its own criteria for determining that some event should be remembered in a specific temporal context in the future. This can include the perception of specific events, or the agents participation in those, such as the first time it meets a new human teacher. It can also include a collection of concepts describing specific contexts in which a set of actions has been particularly useful. The possibilities are

EFTA00624159

numerous, but from the Time Server perspective, all equivalent. They add up to associating a time point or time interval with a set of Atoms.

The Time Server is a bi-directional storage, as AtomHandles can be used as keys, but also as objects indexed by time points or time intervals. In the former case, the Time Server tells us when an Atom was associated with temporal data. In the latter case, it tells us, for a given time point or interval, which Atoms have been marked as relevant.

Temporal indexing can be based on time points or time intervals. A time point can be at any granularity: from years to sub-seconds could be useful. A time interval is simply a set of two points, the second being necessary after the first one, but their granularities not necessarily the same. The temporal indexing inside the Time Server is hierarchical, so one can query for time points or intervals in granularities other than the ones originally used when the knowledge was first stored.

19.3.5.4 System Activity Table Set

The last relevant specialized store is the System Activity Table Set, which is described in more detail in Chapter 23. This set of tables records, with fine-grained temporal associations, the most important activities that take place inside CogPrime. There are different tables for recording cognitive process activity (at the level of MindAgents, to be described in the next section), for maintaining a history of the level of achievement of each important goal in the system, and for recording other important aspects of the system state, such as the most important Atoms and contexts.

19.4 MindAgents: Cognitive Processes

The AtomSpace holds the system's knowledge, but those Atoms are inert. How is that knowledge used and useful? That is the province of cognitive dynamics. These dynamics, in a CogPrime system, can be considered on two levels.

First, we have the cognitive processes explicitly programmed into CogPrime's source code. These are what we call Concretely-Implemented Mind Dynamics, or *CIM-Dynamics*. Their implementation in software happens through objects called MindAgents. We use the term CIM-Dynamic to discuss a conceptual cognitive process, and the term MindAgents for its actual implementation and execution dynamics.

The second level corresponds to the dynamics that emerge through the system's self-organizing dynamics, based on the cooperative activity of the CIM-Dynamics on the shared AtomSpace.

Most of the material in the following chapters is concerted with particular CIM-Dynamics in the CogPrime system. In this section we will simply give some generalities about the CIM-Dynamics as abstract processes and as software processes, which are largely independent of the actual AI contents of the CIM-Dynamics. In practice the CIM-Dynamics involved in a CogPrime system are fairly stereotyped in form, although diverse in the actual dynamics they induce.

EFTA00624160

19.4.1 A Conceptual View of CogPrime Cognitive Processes

We return now to the conceptual trichotomy of cognitive processes presented in Chapter 3 of Part 1, according to which CogPrime cognitive processes may be divided into:

- Control processes;
- Global cognitive processes;
- Focused cognitive processes.

In practical terms, these may be considered as three categories of CIM-Dynamic.

Control Process CIM-Dynamics are hard to stereotype. Examples are the process of homeostatic parameter adaptation of the parameters associated with the various other CIM-Dynamics, and the CIM-Dynamics concerned with the execution of procedures, especially those whose execution is made lengthy by the interactions with the external world.

Control Processes tend to focus on a limited and specialized subset of Atoms or other entities, and carry out specialized mechanical operations on them (e.g. adjusting parameters, interpreting procedures). To an extent, this may be considered a "grab bag" category containing CIM-Dynamics that are not global or focused cognitive processes according to the definitions of the latter two categories. However, it is a nontrivial observation about the CogPrime system that the CIM-Dynamics that are not global or focused cognitive processes are all explicitly concerned with system control in some way or another, so this grouping makes sense.

Global and Focused Cognitive Process CIM-Dynamics all have a common aspect to their structure. Then, there are aspects in which Global versus Focused CIM-Dynamics diverge from each other in stereotyped ways.

In most cases, the process undertaken by a Global or Focused CIM-Dynamic involves two parts: a selection process and an actuation process. Schematically, such a CIM-Dynamic typically looks something like this:

1. Fetch a set of Atoms that it is judged will be useful to process, according to some selection process.
2. Operate on these Atoms, possibly together with previously selected ones (this is what we sometimes call the *actuation process* of the CIM-Dynamic).
3. Go back to step 1.

The major difference between Global and Focused cognitive processes lies in the selection process. In the case of a Global process, the selection process is very broad, sometimes yielding the whole AtomSpace, or a significant subset of it. This means that the actuation process must be very simple, or the activation of this CIM-Dynamic must be very infrequent.

On the other hand, in the case of a Focused process, the selection process is very narrow, yielding only a small number of Atoms, which can then be processed more intensively and expensively, on a per-Atom basis.

Common selection processes for Focused cognitive processes are fitness-oriented selectors, which pick one or a set of Atoms from the AtomSpace with a probability based on some numerical quantity associated with the atom, such as properties of TruthValue or AttentionValue.

There are also more specific selection processes, which choose for example Atoms obeying some particular combination of relationships in relation to some other Atoms; say choosing only Atoms that inherit from some given Atom already being processed. There is a notion, described in the PLN book, of an *Atom Structure Template*; this is basically just a predicate that applies to Atoms, such as

EFTA00624161

```
P(X).tv
```

equals

```
((InheritanceLink X cat) AND (EvaluationLink eats(X,cheese)).tv
```

which is a template that matches everything that inherits from *cat* and eats cheese. Templates like this allow a much more refined selection than the above fitness-oriented selection process.

Selection processes can be created by composing a fitness-oriented process with further restrictions, such as templates, or simpler type-based restrictions.

19.4.2 Implementation of MindAgents

MindAgents follow a very simple design. They need to provide a single method through which they can be enacted, and they should execute their actions in atomic, incremental steps, where each step should be relatively quick. This design enables collaborative scheduling of MindAgents, at the cost of allowing "opportunistic" agents to have more than their fair share of resources. We rely on CogPrime developers to respect the above guidelines, instead of trying to enforce exact resource allocations on the software level.

Each MindAgent can have a set of system parameters that guide its behavior. For instance, a MindAgent dedicated to inference can provide drastically different conclusions if its parameters tell it to select a small set of Atoms for processing each time, but to spend significant time on each Atom, rather than selecting many Atoms and doing shallow inferences on each one. It's expected that multiple copies of the same MindAgent will exist in the cluster, but delivering different dynamics thanks to those parameters.

In addition to their main action method, MindAgents can also communicate with other MindAgents through message queues. CogPrime has, in its runtime configuration, a list of avaliable MindAgents and their locations in the cluster. Communications between MindAgents typically take the form of specific, one-time requests, which we call Tasks.

The default action of MindAgents and the processing of Tasks constitute the cognitive dynamics of CogPrime. Nearly everything that takes place within a CogPrime deployment is done by either a MindAgent (including the control processes), a Task, or specialized code handling AtomSpace internals or communications with the external world. We now talk about how those dynamics are scheduled.

MindAgents live inside a process called a CogPrime Unit. One machine in a CogPrime cluster can contain one or more Units, and one Unit can contain one or more MindAgents. In practice, given the way the AtomSpace is distributed, which requires a control process in each machine, it typically makes more sense to have a single Unit per machine, as this enables all MindAgents in that machine to make direct function calls to the AtomSpace, instead of using more expensive inter-process communication.

There are exceptions to the above guideline, to accommodate various situations:

1. Very specific MindAgents may not need to communicate with other agents, or only do so very rarely, so it makes sense to give them their own process.

EFTA00624162

2. MindAgents whose implementation is a poor fit for the collaborative processing in small increments design described above also should be given their own process, so they don't interfere with the overall dynamics in that machine.
3. MindAgents whose priority is either much higher or much lower than that of other agents in the same machine should be given their own process, so operating system-level scheduling can be relied upon to reflect those very different priority levels.

19.4.3 Tasks

It is not convenient for CogPrime to do all its work directly via the action of MindAgent objects embodying CIM-Dynamics. This is especially true for MindAgents embodying focused cognitive processes. These have their selection algorithms, which are ideally suited to guarantee that, over the long run, the right Atoms get selected and processed. This, however, doesn't address the issue that, on many occasions, it may be necessary to quickly process a specific set of Atoms in order to execute an action or rapidly respond to some demand. These actions tend to be one-time, rather than the recurring patterns of mind dynamics.

While it would be possible to design MindAgents so that they could both cover their long term processing needs and rapidly respond to urgent demands, we found it much simpler to augment the MindAgent framework with an additional scheduling mechanism that we call the Task framework. In essence, this is a ticketing system, designed to handle cases where MindAgents or Schema spawn one – off tasks to be executed – things that need to be done only once, rather that repeatedly and iteratively as with the things embodied in MindAgents.

For instance, *grab the most important Atoms from the AtomSpace and do shallow PLN reasoning to derive immediate conclusions from them* is a natural job for a MindAgent. But *do search to find entities that satisfy this particular predicate P* is a natural job for a Task.

Tasks have AttentionValues and target MindAgents. When a Task is created it is submitted to the appropriate Unit and then put in a priority queue. The Unit will schedule some resources to processing the more important Tasks, as we'll see next.

19.4.4 Scheduling of MindAgents and Tasks in a Unit

Within each Unit we have one or more MindAgents, a Task queue and, optionally, a subset of the distributed AtomSpace. If that subset isn't held in the unit, it's held in another process running on the same machine. If there is more than one Unit per machine, their relative priorities are handled by the operating system's scheduler.

In addition to the Units, CogPrime has an extra maintenance process per machine, whose job is to handle changes in those priorities as well as reconfigurations caused by MindAgent migration, and machines joining or leaving the CogPrime cluster.

So, at the Unit level, attention allocation in CogPrime has two aspects: how MindAgents and Tasks receive attention from CogPrime, and how Atoms receive attention from different MindAgents and Tasks. The topic of this Section is the former. The latter is dealt with elsewhere, in two ways:

EFTA00624163

- in Chapter 23, which discusses the dynamic updating of the AttentionValue structures associated with Atoms, and how these determine how much attention various focused cognitive processes MindAgents pay to them.
- in the discussion of various specific CIM-Dynamics, each of which may make choices of which Atoms to focus on in its own way (though generally making use of AttentionValue and TruthValue in doing so).

The attention allocation subsystem is also pertinent to MindAgent scheduling, because it discusses dynamics that update ShortTermImportance (STI) values associated with MindAgents, based on the usefulness of MindAgents for achieving system goals. In this chapter, we will not enter into such cognitive matters, but will merely discuss the mechanics by which these STI values are used to control processor allocation to MindAgents.

Each instance of a MindAgent has its own AttentionValue, which is used to schedule processor time within the Unit. That scheduling is done by a Scheduler object which controls a collection of worker threads, whose size is a system parameter. The Scheduler aims to allocate worker threads to the MindAgents in a way that's roughly proportional to their STI, but it needs to account for starvation, as well as the need to process the Tasks in the task queue.

This is an area in which we can safely borrow from reasonably mature computer science research. The requirements of cognitive dynamics scheduling are far from unique, so this is not a topic where new ideas need to be invented for OpenCog; rather, designs need to be crafted meeting CogPrime's specific requirements based on state-of-the-art knowledge and experience.

One example scheduler design has two important inputs: the STI associated with each MindAgent, and a parameter determining how much resources should go to the MindAgents vs the Task queue. In the CogPrime implementation, the Scheduler maps the MindAgent STIs to a set of priority queues, and each queue is run a number of times per cycle. Ideally one wants to keep the number of queues small, and rely on multiple Units and the OS-level scheduler to handle widely different priority levels.

When the importance of a MindAgent changes, one just has to reassign it to a new queue, which is a cheap operation that can be done between cycles. MindAgent insertions and removals are handled similarly.

Finally, Task execution is currently handled via allocating a certain fixed percentage of processor time, each cycle, to executing the top Tasks on the queue. Adaptation of this percentage may be valuable in the long term but was not yet implemented.

Control processes are also implemented as MindAgents, and processed in the same way as the other kinds of CIM-Dynamics, although they tend to have fairly low importance.

19.4.5 The Cognitive Cycle

We have mentioned the concept of a "cycle" in the discussion about scheduling, without explaining what we mean. Let's address that now. All the Units in a CogPrime cluster are kept in sync by a global cognitive cycle, whose purpose is described in Section II.

We mentioned above that each machine in the CogPrime cluster has a housekeeping process. One of its tasks is to keep track of the cognitive cycle, broadcasting when the machine has finished its cycle, and listening to similar broadcasts from its counterparts in the cluster. When all the machines have completed a cycle, a global counter is updated, and each machine is then free to begin the next cycle.

EFTA00624164

One potential annoyance with this global cognitive cycle is that some machines may complete their cycle much faster than others, and then sit idly while the stragglers finish their jobs. CogPrime addresses this issue in two ways:

- Over the long run, a load balancing process will assign MindAgents from overburdened machines to underutilized ones. The MindAgent migration process is described in the next section.
- In a shorter time horizon, during which a machine's configuration is fixed, there are two heuristics to minimize the waste of processor time without breaking the overall cognitive cycle coordination:
  - The Task queue in each of the machine's Units can be processed more extensively than it would by default; in extreme cases, the machine can go through the whole queue.
  - Background process MindAgents can be given extra activations, as their activity is unlikely to throw the system out of sync, unlike with more focused and goal-oriented processes.

  Both heuristics are implemented by the scheduler inside each unit, which has one boolean trigger for each heuristic. The triggers are set by the housekeeping process when it observes that the machine has been frequently idle over the recent past, and then reset if the situation changes.

19.5 Distributed AtomSpace and Cognitive Dynamics

As hinted above, realistic CogPrime deployments will be spread around reasonably large clusters of co-located machines. This section describes how this distributed deployment scenario is planned for in the design of the AtomSpace and the MindAgents, and how the cognitive dynamics take place in such a scenario.

We won't review the standard principles of distributed computing here, but we will focus on specific issues that arise when the CogPrime is spread across a relatively large number of machines. The two key issues that need to be handled are:

- How to distribute knowledge (i.e., the AtomSpace) in a way that doesn't impose a large performance penalty?
- How to allocate resources (i.e., machines) to the different cognitive processes (MindAgents) in a way that's flexible and dynamic?

19.5.1 Distributing the AtomSpace

The design of a distributed AtomSpace was guided by the following high level requirements:

1. Scale up, transparently, to clusters of dozens to hundreds of machines, without requiring a single central master server.
2. The ability to store portions of an Atom repository on a number of machines in a cluster, where each machine also runs some MindAgens. The distribution of Atoms across the ma-

EFTA00624165

chines should benefit from the fact that the cognitive processes on one machine are likely to access local Atoms more often than remote ones.
3. Provide transparent access to all Atoms in RAM to all machines in the cluster, even if at different latency and performance levels.
4. For local access to Atoms in the same machine, performance should be as close as possible to what one would have in a similar, but non-distributed AtomSpace.
5. Allow multiple copies of the same Atom to exist in different machines of the cluster, but only one copy per machine.
6. As Atoms are updated fairly often by cognitive dynamics, provide a mechanism for eventual consistency. This mechanism needs not only to propagate changes to the Atoms, but sometimes to reconcile incompatible changes, such as when two cognitive processes update an Atom's TruthValue in opposite ways. Consistency is less important than efficiency, but should be guaranteed eventually.
7. Resolution of inconsistencies should be guided by the importance of the Atoms involved, so the more important ones are more quickly resolved.
8. System configuration can explicitly order the placement of some Atoms to specific machines, and mark a subset of those Atoms as immovable, which should ensure that local copies are always kept.
9. Atom placement across machines, aside from the immovable Atoms, should be dynamic, rebalancing based on frequency of access to the Atom by the different machines.

The first requirement follows obviously from our estimates of how many machines CogPrime will require to display advanced intelligence.

The second requirement above means that we don't have two kinds of machines in the cluster, where some are processing servers and some are database servers. Rather, we prefer each machine to store some knowledge and host some processes acting on that knowledge. This design assumes that there are simple heuristic ways to partition the knowledge across the machines, resulting in allocations that, most of the time, give the MindAgents local access to the Atoms they need most often.

Alas, there will always be some cases in which a MindAgent needs an Atom that isn't available locally. In order to keep the design on the MindAgents simple, this leads to the third requirement, transparency, and to the fourth one, performance.

This partition design, on the other hand, means that there must be some replication of knowledge, as there will always be some Atoms that are needed often by MindAgents on different machines. This leads to requirement five (allow redundant copies of an Atom). However, as MindAgents frequently update the mutable components of Atoms, requirements six and seven are needed, to minimize the impact of conflicts on system performance while striving to guarantee that conflicts are eventually solved, and with priority proportional to the importance of the impacted Atoms.

19.5.1.1 Mechanisms of Managing Distributed Atomspaces

When one digs into the details of distributed AtomSpaces, a number of subtleties emerge. Going into these in full detail here would not be appropriate, but we will make a few comments, just to give a flavor of the sorts of issues involved.

To discuss these issues clearly, some special terminology is useful. In this context, it is useful to reserve the word "Atom" for its pure, theoretical definition, viz: "a Node is uniquely determined

EFTA00624166

by its name. A Link is uniquely determined by its outgoing set". Atoms sitting in RAM may then be called "Realized Atoms". Thus, given a single, pure "abstract/theoretical" Atom, there might be two different Realized Atoms, on two different servers, having the same name/outgoing-set. It's OK to think of a RealizedAtom as a clone of the pure, abstract Atom, and to talk about it that way. Analogously, we might call atoms living on disk, or flying on a wire, "Serialized Atoms"; and, when need be, use specialized terms like "ZMQ-serialized atoms", or "BerkeleyDB-serialized Atoms", etc.

An important and obvious coherency requirement is: "If a MindAgent asks for the Handle of an Atom at time A, and then asks, later on, for the Handle of the same Atom, it should receive the same Handle."

By the "AtomSpace", in general, we mean the container(s) that are used to store the set of Atoms used in an OpenCog system, both in RAM and on disk. In the case of an Atom space that is distributed across multiple machines or other data stores, we will call each of these an "Atom space portion"

Atoms and Handles

Each OpenCog Atom is associated with a Handle object, which is used to identify the Atom uniquely. The Handle is a sort of "key" used, at the infrastructure level, to compactly identify the Atom. In a single-machine, non-distributed Atomspace, one can effectively just use long ints as Handles, and assign successive ints as Handles to successively created new Atoms. In a distributed Atomspace, it's a little subtler. Perhaps the cleanest approach in this case is to use a hash of the serialized Atom data as the handle for an Atom. That way, if an Atom is created in any portion, it will inherently have the same handle as any of its clones.

The issue of Handle collisions then occurs – it is possible, though it will be rare, that two different Atoms will be assigned the same Handle via the hashing function. This situation can be identified via checking, when an Atom is imported into a portion, whether there is already some Atom in that portion with the same Handle but different fundamental aspects. In the rare occasion where this situation does occur, one of the Atoms must then have its Handle changed. Changing an Atom's handle everywhere it's referenced in RAM is not a big deal, so long as it only happens occasionally. However, some sort of global record of Handle changes should be kept, to avoid confusion in the process of loading saved Atoms from disk. If a loaded Atomspace contains Atoms that have changed Handle since the file was saved, the Atom loading process needs to know about this.

The standard mathematics of hash functions collisions, shows that if one has a space of H possible Handles, one will get two Atoms with the same Handle after $1.25 \times \sqrt{(H)}$ tries, on average.... Rearranging this, it means we'd need a space of around N^2 Handles to have a space of Handles for N possible Atoms, in which one collision would occur on average.... So to have a probability of one collision, for N possible Atoms, one would have to use a handle range up to N^2. The number of bits needed to encode N^2 is twice as many as the number needed to encode N. So, if one wants to minimize collisions, one may need to make Handles twice as long, thus taking up more memory.

However, this memory cost can be palliated via introducing "local Handles" separate from the global, system-wide Handles. The local Handles are used internally within each local Atomspace, and then each local Atomspace contains a translation table going back and forth between local

EFTA00624167

and global Handles. Local handles may be long ints, allocated sequentially to each new Atom entered into a portion. Persistence to disk would always use the global Handles.

To understand the memory tradeoffs involved in these solutions, assume that the global Handles were k times as long as the local handles... and suppose that the average Handle occurred r times in the local Atomspace. Then the memory inflation ratio of the local/global solution as opposed to a solution using only the shorter local handles, would be

$$(1 + k + r)/r = 1 + (k + 1)/r$$

if k=2 and r=10 (each handle is used 10 times on average, which is realistic based on current real-world OpenCog Atomspaces), then the ratio is just $1.3x$ – suggesting that using hash codes for global Handles, and local Handles to save memory in each local AtomSpace, is acceptable memory-wise.

19.5.1.2 Distribution of Atoms

Given the goal of maximizing the probability that an Atom will be local to the machines of the MindAgents that need it, the two big decisions are how to allocate Atoms to machines, and then how to reconcile the results of MindAgents actuating on those Atoms.

The initial allocation of Atoms to machines may be done via explicit system configuration, for Atoms known to have different levels of importance to specific MindAgents. That is, after all, how MindAgents are initially allocated to machines as well.

One may, for instance, create a CogPrime cluster where one machine (or group) focuses on visual perception, one focuses on language processing, one focuses on abstract reasoning, etc. In that case one can hard-wire the location of Atoms.

What if one wants to have three abstract-reasoning machines in one's cluster? Then one can define an abstract-reasoning zone consisting of three Atom repository portions. One can hard-wire that Atoms created by MindAgents in the zone must always remain in that zone – but can potentially be moved among different portions within that zone, as well as replicated across two or all of the machines, if need be. By default they would still initially be placed in the same portion as the MindAgent that created them.

However Atoms are initially placed in portions, sometimes it will be appropriate to move them. And sometimes it will be appropriate to clone an Atom, so there's a copy of it in a different portion from where it exists. Various algorithms could work for this, but the following is one simple mechanism:

- When an Atom A in machine M_1 is requested by a MindAgent in machine M_2, then a clone of A is temporarily created in M_2.
- When an Atom is forgotten (due to low LTI), then a check is made if it has any clones, and any links to it are changed into links to its clones.
- The LTI of an Atom may get a boost if that Atom has no clones (the amount of this boost is a parameter that may be adjusted).

EFTA00624168

19.5.1.3 MindAgents and the Distributed AtomSpace

In the context of a distributed AtomSpace, the interactions between MindAgents and the knowledge store become subtler, as we'll now discuss.

When a MindAgent wants to create an Atom, it will make this request of the local AtomSpace process, which hosts a subset of the whole AtomSpace. It can, on Atom creation, specify whether the Atom is immovable or not. In the former case, it will initially only be accessible by the MindAgents in the local machine.

The process of assigning the new Atom an AtomHandle needs to be taken care of, in a way that doesn't introduce a central master. One way to achieve that is to make handles hierarchical, so the higher order bits indicate the machine. This, however, means that AtomHandles are no longer immutable. A better idea is to automatically allocate a subset of the AtomHandle universe to each machine. The initial use of those AtomHandles is the privilege of that machine but, as Atoms migrate or are cloned, the handles can move through the cluster.

When a MindAgent wants to retrieve one or more Atoms, it will perform a query on the local AtomSpace subset, just as it would with a single machine repository. Along with the regular query parameters, it may specify whether the request should be processed locally only, or globally. Local queries will be fast, but may fail to retrieve the desired Atoms, while global queries may take a while to return. In the approach outlined above for MindAgent dynamics and scheduling, this would just cause the MindAgent to wait until results are available.

Queries designed to always return a set of Atoms can have a third mode, which is "prioritize local Atoms". In this case, the AtomSpace, when processing a query that looks for Atoms that match a certain pattern would try to find all local responses before asking other machines.

19.5.1.4 Conflict Resolution

A key design decision when implementing a distributed AtomSpace is the trade-off between consistency and efficiency. There is no universal answer to this conflict, but the usage scenarios for CogPrime, current and planned, tend to fall on the same broad category as far consistency goes. CogPrime's cognitive processes are relatively indifferent to conflicts and capable of working well with outdated data, especially if the conflicts are temporary. For applications such as the AGI Preschool, it is unlikely that outdated properties of single Atoms will have a large, noticeable impact on the system's behavior; even if that were to happen on rare occasions, this kind of inconsistency is often present in human behavior as well.

On the other hand, CogPrime assumes fairly fast access to Atoms by the cognitive processes, so efficiency shouldn't be too heavily penalized. The robustness against mistakes and the need for performance mean that a distributed AtomSpace should follow the principle of "eventual consistency". This means that conflicts are allowed to arise, and even to persist for a while, but a mechanism is needed to reconcile them.

Before describing conflict resolution, which in CogPrime is a bit more complicated than in most applications, we note that there are two kinds of conflicts. The simple one happens when an Atom that exists in multiple machines is modified in one machine, and that change isn't immediately propagated. The less obvious one happens when some process creates a new Atom in its local AtomSpace repository, but that Atom conceptually "already exists" elsewhere in the system. Both scenarios are handled in the same way, and can become complicated when, instead of a single change or creation, one needs to reconcile multiple operations.

EFTA00624169

The way to handle conflicts is to have a special purpose control process, a reconciliation MindAgent, with one copy running on each machine in the cluster. This MindAgent keeps track of all recent write operations in that machine (Atom creations or changes).

Each time the reconciliation MindAgent is called, it processes a certain number of Atoms in the recent writes list. It chooses the Atoms to process based on a combination of their STI, LTI and recency of creation/change. Highest priority is given to Atoms with higher STI and LTI that have been around longer. Lowest priority is given to Atoms with low STI or LTI that have been very recently changed – both because they may change again in the very near future, and because they may be forgotten before it's worth solving any conflicts. This will be the case with most perceptual Atoms, for instance.

By tuning how many Atoms this reconciliation MindAgent processes each time it's activated we can tweak the consistency vs efficiency trade-off.

When the AtomReconciliation agent processes an Atom, what it does is:

- Searches all the machines in the cluster to see if there are other equivalent Atoms (for Nodes, these are Atoms with the same name and type; for Links, these are Atoms with the same type and targets).
- If it finds equivalent Atoms, and there are conflicts to be reconciled, such as different TruthValues or AttentionValues, the decision of how to handle the conflicts is made by a special probabilistic reasoning rule, called the Rule of Choice (see Chapter 34). Basically, this means:
  - It decided whether to merge the conflicting Atoms. We always merge Links, but some Nodes may have different semantics, such as Nodes representing different procedures that have been given the same name.
  - In the case that the two Atoms A and B should be merged, it creates a new Atom C that has all the same immutable properties as A and B. It merges their TruthValues according to the probabilistic revision rule (see Chapter 34). The AttentionValues are merged by prioritizing the higher importances.
  - In the case that two Nodes should be allowed to remain separate, it allocates one of them (say, B) a new name. Optionally, it also evaluates whether a SimilarityLink should be created between the two different Nodes.

Another use for the reconcilitation MindAgent is maintaining approximate consistency between clones, which can be created by the AtomSpace itself, as described above in Subsection 19.5.1.2. When the system knows about the multiple clones of an Atom, it keeps note of these versions in a list, which is processed periodically by a conflict resolution MindAgent, in order to prevent the clones from drifting too far apart by the actions of local cognitive processes in each machine.

19.5.2 Distributed Processing

The OCF infrastructure as described above already contains a lot of distributed processing implicit in it. However, it doesn't tell you how to make the complex cognitive processes that are part of the CogPrime design distributed unto themselves – say, how to make PLN or MOSES themselves distributed. This turns out to be quite possible, but becomes quite intricate and

EFTA00624170

specific depending on the particular algorithms involved. For instance, the current MOSES implementation is now highly amenable to distributed and multiprocessor implementation, but in a way that depends subtly on the specifics of MOSES and has little to do with the role of MOSES in CogPrime as a whole. So we will not delve into these topics here.

Another possibility worth mentioning is broadly distributed processing, in which CogPrime intelligence is spread across thousands or millions of relatively weak machines networked via the Internet. Even if none of these machines is exclusively devoted to CogPrime, the total processing power may be massive, and massively valuable. The use of this kind of broadly distributed computing resource to help CogPrime is quite possible, but involves numerous additional control problems which we will not address here.

A simple case is massive global distribution of MOSES fitness evaluation. In the case where fitness evaluation is isolated and depends only on local data, this is extremely straightforward. In the more general case where fitness evaluation depends on knowledge stored in a large AtomSpace, it requires a subtler design, wherein each globally distributed MOSES subpopulation contains a pool of largely similar genotypes and a cache of relevant parts of the AtomSpace, which is continually refreshed during the fitness evaluation process. This can work so long as each globally distributed lobe has a reasonably reliable high bandwidth, low latency connection to a machine containing a large AtomSpace.

On the more mundane topic of distributed processing within the main CogPrime cluster, three points are worth discussing:

- Distributed communication and coordination between MindAgents.
- Allocation of machines to functional groups, and MindAgent migration.
- Machines entering and leaving the cluster.

19.5.2.1 Distributed Communication and Coordination

Communications between MindAgents, Units and other CogPrime components are handled by a message queue subsystem. This subsystem provides a unified API, so the agents involved are unaware of the location of their partners: distributed messages, inter-process messages in the same machine, and intra-process messages in the same Unit are sent through the same API, and delivered to the same target queues. This design enables transparent distribution of MindAgents and other components.

In the simplest case, of MindAgents within the same Unit, messages are delivered almost immediately, and will be available for processing by the target agent the next time it's enacted by the scheduler. In the case of messages sent to other Units or other machines, they're delivered to the messaging subsystem component of that unit, which has a dedicated thread for message delivery. That subsystem is scheduled for processing just like any other control process, although it tends to have a reasonably high importance, to ensure speedy delivery.

The same messaging API and subsystem is used for control-level communications, such as the coordination of the global cognitive cycle. The cognitive cycle completion message can be used for other housekeeping contents as well.

EFTA00624171

19.5.2.2 Functional Groups and MindAgent Migration

A CogPrime cluster is composed of groups of machines dedicated to various high-level cognitive tasks: perception processing, language processing, background reasoning, procedure learning, action selection and execution, goal achievement planning, etc. Each of these high-level tasks will probably require a number of machines, which we call functional groups.

Most of the support needed for functional groups is provided transparently by the mechanisms for distributing the AtomSpace and by the communications layer. The main issue is how much resources (i.e., how many machines) to allocate to each functional group. The initial allocation is determined by human administrators via the system configuration – each machine in the cluster has a local configuration file which tells it exactly which processes to start, along with the collection of MindAgents to be loaded onto each process and their initial AttentionValues.

Over time, however, it may be necessary to modify this allocation, adding machines to overworked or highly important functional groups. For instance, one may add more machines to the natural language and perception processing groups during periods of heavy interaction with humans in the preschool environment, while repurposing those machines to procedure learning and background inference during periods in which the agent controlled by CogPrime is resting or "sleeping".

This allocation of machines is driven by attention allocation in much the same way that processor time is allocated to MindAgents. Functional groups can be represented by Atoms, and their importance levels are updated according to the importance of the system's top level goals, and the usefulness of each functional group to their achievement. Thus, once the agent is engaged by humans, the goals of pleasing them and better understanding them would become highly important, and would thus drive the STI of the language understanding and language generation functional groups.

Once there is an imbalance between a functional group's STI and its share of the machines in the cluster, a control process CIM-Dynamic is triggered to decide how to reconfigure the cluster. This CIM-Dynamic works approximately as follows:

- First, it decides how many extra machines to allocate to each sub-represented functional group.
- Then, it ranks the machines not already allocated to those groups based on a combination of their workload and the aggregate STI of their MindAgents and Units. The goal is to identify machines that are both relatively unimportant and working under capacity.
- It will then migrate the MindAgents of those machines to other machines in the same functional group (or just remove them if clones exist), freeing them up.
- Finally, it will decide how best to allocate the new machines to each functional group. This decision is heavily dependent on the nature of the work done by the MindAgents in that group, so in CogPrime these decisions will be somewhat hardcoded, as is the set of functional groups. For instance, background reasoning can be scaled just by adding extra inference MindAgents to the new machines without too much trouble, but communicating with humans requires MindAgents responsible for dialog management, and it doesn't make sense to clone those, so it's better to just give more resources to each MindAgent without increasing their numbers.

The migration of MindAgents becomes, indirectly, a key driver of Atom migration. As MindAgents move or are cloned to new machines, the AtomSpace repository in the source machine

EFTA00624172

should send clones of the Atoms most recently used by these MindAgents to the target machine(s), anticipating a very likely distributed request that would create those clones in the near future anyway. If the MindAgents are moved but not cloned, the local copies of those Atoms in the source machine can then be (locally) forgotten.

19.5.2.3 Adding and Removing Machines

Given the support for MindAgent migration and cloning outlined above, the issue of adding new machines to the cluster becomes a specific application of the heuristics just described. When a new machine is added to the cluster, CogPrime initially decides on a functional group for it, based both on the importance of each functional group and on their current performance – if a functional group consistently delays the completion of the cognitive cycle, it should get more machines, for instance. When the machine is added to a functional group, it is then populated with the most important or resource starved MindAgents in that group, a decision that is taken by economic attention allocation.

Removal of a machine follows a similar process. First the system checks if the machine can be safely removed from its current functional group, without greatly impacting its performance. If that's the case, the non-cloned MindAgents in that machine are distributed among the remaining machines in the group, following the heuristic described above for migration. Any local-only Atoms in that machine's AtomSpace container are migrated as well, provided their LTI is high enough.

In the situation in which removing a machine M_1 would have an intolerable impact on the functional group's performance, a control process selects another functional group to lose a machine M_2. Then, the MindAgents and Atoms in M_1 are migrated to M_2, which goes through the regular removal process first.

In principle, one might use the insertion or removal of machines to perform a global optimization of resource allocation within the system, but that process tends to be much more expensive than the simpler heuristics we just described. We believe these heuristics can give us most of the benefits of global re-allocation at a fraction of the disturbance for the system's overall dynamics during their execution.

EFTA00624173

Chapter 20
Knowledge Representation Using the Atomspace

20.1 Introduction

CogPrime's knowledge representation must be considered on two levels: implicit and explicit. This chapter considers mainly explicit knowledge representation, with a focus on representation of declarative knowledge. We will describe the Atom knowledge representation, a generalized hypergraph formalism which comprises a specific vocabulary of Node and Link types, used to represent declarative knowledge but also, to a lesser extent, other types of knowledge as well. Other mechanisms of representing procedural, episodic, attentional, and intentional knowledge will be handled in later chapters, as will the subtleties of implicit knowledge representation.

The AtomSpace Node and Link formalism is the most obviously distinctive aspect of the OpenCog architecture, from the point of view of a software developer building AI processes in the OpenCog framework. But yet, the features of CogPrime that are most important, in terms of our theoretical reasons for estimating it likely to succeed as an advanced AGI system, are not really dependent on the particulars of the AtomSpace representation.

What's important about the AtomSpace knowledge representation is mainly that it provides a flexible means for compactly representing multiple forms of knowledge, in a way that allows them to interoperate – where by "interoperate" we mean that e.g. a fragment of a chunk of declarative knowledge can link to a fragment of a chunk of attentional or procedural knowledge; or a chunk of knowledge in one category can overlap with a chunk of knowledge in another category (as when the same link has both a (declarative) truth value and an (attentional) importance value). In short, any representational infrastructure sufficiently flexible to support

- compact representation of all the key categories of knowledge playing dominant roles in human memory
- the flexible creation of specialized sub-representations for various particular subtypes of knowledge in all these categories, enabling compact and rapidly manipulable expression of knowledge of these subtypes
- the overlap and interlinkage of knowledge of various types, including that represented using specialized sub-representations

will probably be acceptable for CogPrime's purposes. However, precisely formulating these general requirements is tricky, and is significantly more difficult than simply articulating a single acceptable representational scheme, like the current OpenCog Atom formalism. The Atom

EFTA00624174

formalism satisfies the relevant general requirements and has proved workable from a practical software perspective.

In terms of the Mind-World Correspondence Principle introduced in Chapter 10, the important point regarding the Atom representation is that it must be flexible enough to allow the compact and rapidly manipulable representation of knowledge that has aspects spanning the multiple common human knowledge categories, in a manner that allows easy implementation of cognitive processes that will manifest the Mind-World Correspondence Principle in everyday human-like situations. The actual manifestation of mind-world correspondence is the job of the cognitive processes acting on the AtomSpace – the job of the AtomSpace is to be an efficient and flexible enough representation that these cognitive processes can manifest mind-world correspondence in everyday human contexts given highly limited computational resources.

20.2 Denoting Atoms

First we describe the textual notation we'll use to denote various sorts of Atoms throughout the following chapters. The discussion will also serve to give some particular examples of cognitively meaningful Atom constructs.

20.2.1 Meta-Language

As always occurs when discussing (even partially) logic-based systems, when discussing CogPrime there is some potential for confusion between logical relationships inside the system, and logical relationships being used to describe parts of the system. For instance, we can state as observers that two Atoms inside CogPrime are equivalent, and this is different from stating that CogPrime itself contains an Equivalence relation between these two Atoms. Our formal notation needs to reflect this difference.

Since we will not be doing any fancy mathematical analyses of CogPrime structures or dynamics here, there is no need to formally specify the logic being used for the metalanguage. Standard predicate logic may be assumed.

So, for example, we will say things like

```
(IntensionalInheritanceLink Ben monster).TruthValue.strength = .5
```

This is a metalanguage statement, which means that the strength field of the TruthValue object associated with the link (IntensionalInheritance Ben monster) is equal to .5. This is different than saying

```
EquivalenceLink
    ExOutLink
        GetStrength
        ExOutLink
            GetTruthValue
            IntensionalInheritanceLink Ben monster
    NumberNode 0.5
```

EFTA00624175

which refers to an equivalence relation represented inside CogPrime. The former refers to an equals relationship observed by the authors of the book, but perhaps never represented explicitly inside CogPrime.

In the first example above we have used the C++ convention

```
structure_variable_name.field_name
```

for denoting elements of composite structures; this convention will be stated formally below.

In the second example we have used schema corresponding to TruthValue and Strength; these schema extract the appropriate fields from the Atoms they're applied to, so that e.g.

```
ExOutLink
    GetTruthValue
    A
```

returns the number

```
A.TruthValue
```

Following a convention from mathematical logic, we will also sometimes use the special symbol

```
|-
```

to mean "implies in the metalanguage." For example, the first-order PLN deductive inference strength rule may be written

```
InheritanceLink A B <sAB>
InheritanceLink B C <sBC>
|-
InheritanceLink A C <sAC>
```

where

```
sAC = sAB sBC + (1-sAB) ( sC - sB sBC ) / (1- sB )
```

This is different from saying

```
ForAll $A, $B, $C, $sAB, $sBC, $sAC

ExtensionalImplicationLink_HOJ
    AND
        InheritanceLink $A $B <$sAB>
        InheritanceLink $B $C <$sBC>
    AND
        InheritanceLink $A $C <$sAC>
        $sAC = $sAB $sBC + (1-$sAB) ($sC - $sB $sBC) / (1- $sB)
```

which is the most natural representation of the independence-based PLN deduction rule (for strength-only truth values) as a logical statement within CogPrime. In the latter expression the variables $A, $sAB, and so forth represent actual Variable Atoms within CogPrime. In the former expression the variables represent concrete, non-Variable Atoms within CogPrime, which however are being considered as variables within the metalanguage.

(As explained in the PLN book, a link labeled with "HOJ" refers to a "higher order judgment", meaning a relationship that interprets its relations as entities with particular truth values. For instance,

```
ImplicationLink_HOJ
   Inh $X stupid <.9>
   Inh $X rich <.9>
```

EFTA00624176

means that if (Inh $X stupid) has a strength of .9, then (Inh $X rich) has a strength of .9). WIKISOURCE:AtomNotation

20.2.2 Denoting Atoms

Atoms are the basic objects making up CogPrime knowledge. They come in various types, and are associated with various dynamics, which are embodied in MindAgents. Generally speaking Atoms are endowed with TruthValue and AttentionValue objects. They also sometimes have names, and other associated Values as previously discussed. In the following subsections we will explain how these are notated, and then discuss specific notations for Links and Nodes, the two types of Atoms in the system.

20.2.2.1 Names

In order to denote an Atom in discussion, we have to call it something. Relatedly but separately, Atoms may also have names within the CogPrime system. (As a matter of implementation, in the current OpenCog version, no Links have names; whereas, all Nodes have names, but some Nodes have a null name, which is conceptually the same as not having a name.)

(name,type) pairs must be considered as unique within each Unit within a OpenCog system, otherwise they can't be used effectively to reference Atoms. It's OK if two different OpenCog Units both have SchemaNodes named "+", but not if one OpenCog Unit has two SchemaNodes both named "+" - this latter situation is disallowed on the software level, and is assumed in discussions not to occur.

Some Atoms have natural names. For instance, the SchemaNode corresponding to the elementary schema function + may quite naturally be named "+". The NumberNode corresponding to the number .5 may naturally be named ".5", and the CharacterNode corresponding to the character *c* may naturally be named "c". These cases are the minority, however. For instance, a SpecificEntityNode representing a particular instance of + has no natural name, nor does a SpecificEntityNode representing a particular instance of c.

Names should not be confused with Handles. Atoms have Handles, which are unique identifiers (in practice, numbers) assigned to them by the OpenCog core system; and these Handles are how Atoms are referenced internally, within OpenCog, nearly all the time. Accessing of Atoms by name is a special case - not all Atoms have names, but all Atoms have Handles. An example of accessing an Atom by name is looking up the CharacterNode representing the letter "c" by its name "c". There would then be two possible representations for the word "cat":

1. this word might be associated with a ListLink - and the ListLink corresponding to "cat" would be a list of the Handles of the Atoms of the nodes named "c", "a", and "t".
2. for expedience, the word might be associated with a WordNode named "cat."

In the case where an Atom has multiple versions, this may happen for instance if the Atom is considered in a different context (via a ContextLink), each version has a VersionHandle, so that accessing an AtomVersion requires specifying an AtomHandle plus a VersionHandle. See Chapter 19 for more information on Handles.

EFTA00624177

OpenCog never assigns Atoms names *on its own*; in fact, Atom names are assigned only in the two sorts of cases just mentioned:

1. Via preprocessing of perceptual inputs (e.g. the names of NumberNode, CharacterNodes)
2. Via hard-wiring of names for SchemaNodes and PredicateNodes corresponding to built-in elementary schema (e.g. +, AND, Say)

If an Atom A has a name n in the system, we may write

```
A.name = n
```

On the other hand, if we want to assign an Atom an *external* name, we may make a meta-language assertion such as

```
L1 := (InheritanceLink Ben animal)
```

indicating that we decided to name that link L1 for our discussions, even though inside OpenCog it has no name.

In denoting (nameless) Atoms we may use arbitrary names like L1. This is more convenient than using a Handle based notation which Atoms would be referred to as 1, 3433322, etc.; but sometimes we will use the Handle notation as well.

Some ConceptNodes and conceptual PredicateNode or SchemaNodes may correspond with human-language words or phrases like *cat*, *bite*, and so forth. This will be the minority case; more such nodes will correspond to parts of human-language concepts or fuzzy collections of human-language concepts. In discussions in this book, however, we will often invoke the unusual case in which Atoms correspond to individual human-language concepts. This is because such examples are the easiest ones to write about and discuss intuitively. The preponderance of named Atoms in the examples in the book implies no similar preponderance of named Atoms in the real OpenCog system. It is merely easier to talk about a hypothetical Atom named "cat" than it is about a hypothetical Atom with Handle 434. It is not impossible that a OpenCog system represents "cat" as a single ConceptNode, but it is just as likely that it will represent "cat" as a map composed of many different nodes without any of these having natural names. Each OpenCog works out for itself, implicitly, which concepts to represent as single Atoms and which in distributed fashion.

For another example,

```
ListLink
   CharacterNode "c"
   CharacterNode "a"
   CharacterNode "t"
```

corresponds to the character string

```
("c", "a", "t")
```

and would naturally be named using the string *cat*. In the system itself, however, this ListLink need not have any name.

20.2.2.2 Types

Atoms also have types. When it is necessary to explicitly indicate the type of an atom, we will use the keyword Type, as in

EFTA00624178

```
A.Type = InheritanceLink

N_345.Type = ConceptNode
```

On the other hand, there is also a built-in schema HasType which lets us say

```
EvaluationLink HasType A InheritanceLink

EvaluationLink HasType N_345 ConceptNode
```

This covers the case in which type evaluation occurs explicitly in the system, which is useful if the system is analyzing its own emergent structures and dynamics.

Another option currently implemented in OpenCog is to explicitly restrict the type of a variable using `TypedVariableLink` such as follows

```
TypedVariableLink
    VariableNode $X
    VariableTypeNode "ConceptNode"
```

Note also that we will frequently remove the suffix `Link` or `Node` from their type name, such as

```
Inheritance
    Concept A
    Concept B
```

instead of

```
InheritanceLink
    ConceptNode A
    ConceptNode B
```

20.2.2.3 Truth Values

The truth value of an atom is a bundle of information describing how *true* the Atom is, in one of several different senses depending on the Atom type. It is encased in a TruthValue object associated with the Atom. Most of the time, we will denote the truth value of an atom in <>'s following the expression denoting the atom. This very handy notation may be used in several different ways.

A complication is that some Atoms may have CompositeTruthValues, which consist of different estimates of their truth value made by different sources, which for whatever reason have not been reconciled (maybe no process has gotten around to reconciling them, maybe they correspond to different truth values in different contexts and thus logically need to remain separate, maybe their reconciliation is being delayed pending accumulation of more evidence, etc.). In this case we can still assume that an Atom has a default truth value, which corresponds to the highest-confidence truth value that it has, in the Universal Context.

Most frequently, the notation is used with a single number in the brackets, e.g.

```
A <.4>
```

to indicate that the atom A has truth value .4; or

```
IntensionalInheritanceLink Ben monster <.5>
```

EFTA00624179

to indicate that the IntensionalInheritance relation between Ben and monster has truth value strength .5. In this case, <tv> indicates (roughly speaking) that the truth value of the atom in question involves a probability distribution with a mean of tv. The precise semantics of the strength values associated with OpenCog Atoms is described in Probabilistic Logic Networks (see Chapter 34). Please note, though: This notation does not imply that the only data retained in the system about the distribution is the single number .5.

If we want to refer to the truth value of an Atom A in the context C, we can use the construct

```
ContextLink <truth value>
  C
  A
```

Sometimes, Atoms in OpenCog are labeled with two truth value components as defined by PLN: strength and weight-of-evidence. To denote these two components, we might write

```
IntensionalInheritanceLink Ben scary <.9,.1>
```

indicating that there is a relatively small amount of evidence in favor of the proposition that Ben is very scary.

We may also put the TruthValue indicator in a different place, e.g. using indent notation,

```
IntensionalInheritanceLink <.9,.1>
    Ben
    scary
```

This is mostly useful when dealing with long and complicated constructions.

If we want to denote a composite truth value (whose components correspond to different "versions" of the Atom), we can use a list notation, e.g.

```
IntensionalInheritance (<.9,.1>, <.5,.9> [h,123],<.6,.7> [c,655])
    Ben
    scary
```

where e.g.

```
<.5,.9> [h,123]
```

denotes the TruthValue version of the Atom indexed by Handle 123. The `h` denotes that the AtomVersion indicated by the VersionHandle h,123 is a Hypothetical Atom, in the sense described in the PLN book. Some versions may not have any index Handles.

The semantics of composite TruthValues are described in the PLN book, but roughly they are as follows. Any version not indexed by a VersionHandle is a "primary TruthValue" that gives the truth value of the Atom based on some body of evidence. A version indexed by a VersionHandle is either contextual or hypothetical, as indicated notationally by the c or h in its VersionHandle. So, for instance, if a TruthValue version for Atom A has VersionHandle h,123 that means it denotes the truth value of Atom A under the hypothetical context represented by the Atom with handle 123. If a TruthValue version for Atom A has VersionHandle c,655 this means it denotes the truth value of Atom A in the context represented by the Atom with Handle 655.

Alternately, truth values may be expressed sometimes in <L,U,b> or <L,U,b,N> format, defined in terms of indefinite probability theory as defined in the PLN book and recalled in Chapter 34. For instance,

```
IntensionalInheritanceLink Ben scary <.7,.9,.8,20>
```

EFTA00624180

has the semantics that *There is an estimated 80% chance that after 20 more observations have been made, the estimated strength of the link will be in the interval (.7,.9).*

The notation may also be used to specify a TruthValue probability distribution, e.g.

```
A <g(5,7,12)>
```

would indicate that the truth value of A is given by distribution g with parameters (5,7,12), or

```
A <M>
```

where M is a table of numbers, would indicate that the truth value of A is approximated by the table M.

The <> notation for truth value is an unabashedly incomplete and ambiguous notation, but it is very convenient. If we want to specify, say, that the truth value strength of IntensionalInheritanceLink Ben monster is in fact the number .5, and no other truth value information is retained in the system, then we need to say

```
(IntensionalInheritance Ben monster).TruthValue = [(strength, .5)]
```

(where a hashtable form is assumed for TruthValue objects, i.e. a list of name-value pairs). But this kind of issue will rarely arise here and the <> notation will serve us well.

20.2.2.4 Attention Values

The AttentionValue object associated with an Atom does not need to be notated nearly as often as truth value. When it does however we can use similar notational methods.

AttentionValues may have several components, but the two critical ones are called short-term importance (STI) and long-term importance (LTI). Furthermore, multiple STI values are retained: for each (Atom, MindAgent) pair there may be a Mind-Agent-specific STI value for that Atom. The pragmatic import of these values will become clear in a later chapter when we discuss attention allocation.

Roughly speaking, the long-term importance is used to control memory usage: when memory gets scarce, the atoms with the lowest LTI value are removed. On the other hand, the short-term importance is used to control processor time allocation: MindAgents, when they decide which Atoms to act on, will generally, but not only, choose the ones that have proved most useful to them in the recent past, and additionally those that have been useful for other MindAgents in the recent past.

We will use the double bracket <<>> to denote attention value (in the rare cases where such denotation is necessary). So, for instance,

```
Cow_7 <<.5>>
```

will mean the node Cow_7 has an importance of .5; whereas,

```
Cow_7 <<STI=.1, LTI = .8>>
```

or simply

```
Cow_7 <<.1, .8>>
```

will mean the node Cow_7 has short-term importance = .1 and long-term importance = .8 .

Of course, we can also use the style

EFTA00624181

```
(IntensionalInheritanceLink Ben monster).AttentionValue
                                    = [(STI,.1), (LTI, .8)]
```

where appropriate.

20.2.2.5 Links

Links are represented using a simple notation that has already occurred many times in this book. For instance,

```
Inheritance A B
```

```
Similarity A B
```

Note that here the symmetry or otherwise of the link is not implicit in the notation. SimilarityLinks are symmetrical, InheritanceLinks are not. When this distinction is necessary, it will be explicitly made. WIKISOURCE:FunctionNotation

20.3 Representing Functions and Predicates

SchemaNodes and PredicateNodes contain functions internally; and Links may also usefully be considered as functions. We now briefly discuss the representations and notations we will use to indicate functions in various contexts.

Firstly, we will make some use of the currying notation drawn from combinatory logic, in which adjacency indicates function application. So, for instance, using currying,

```
f x
```

means the function f evaluated at the argument x; and (f x y) means (f(x))(y). If we want to specify explicitly that a block of terminology is being specified using currying we will use the notation @[expression], for instance

```
@[f x y z]
```

means

```
((f(x))(y))(z)
```

We will also frequently use conventional notation to refer to functions, such as f(x,y). Of course, this is consistent with the currying convention if (x,y) is interpreted as a list and f is then a function that acts on 2-element lists. We will have many other occasions than this to use list notation.

Also, we will sometimes use a non-curried notation, most commonly with Links, so that e.g.

```
InheritanceLink x y
```

does not mean a curried evaluation but rather means InheritanceLink(x,y).

EFTA00624182

20.3.0.6 Execution Output Links

In the case where f refers to a schema, the occurrence of the combination f x in the system is represented by

```
ExOutLink f x
```

or graphically

Note that, just as when we write

```
f (g x)
```

we mean to apply f to the result of applying g to x, similarly when we write

```
ExOutLink f (ExOutLink g x)
```

we mean the same thing. So for instance

```
EvaluationLink (ExOutLink g x) y <.8>
```

means that the result of applying g to x is a predicate r, so that r(y) evaluates to True with strength .8.

This approach, in its purest incarnation, does not allow multi-argument schemata. Now, multi-argument schemata are never actually necessary, because one can use argument currying to simulate multiple arguments. However, this is often awkward, and things become simpler if one introduces an explicit tupling operator, which we call ListLink. Simply enough,

```
ListLink A1 ... An
```

denotes an ordered list (A1, . . . , An)

20.3.1 Execution Links

ExecutionLinks give the system an easy way to record acts of schema execution. These are ternary links of the form:

```
SchemaNode: S

Atom: A, B

ExecutionLink S A B
```

In words, this says the procedure represented by SchemaNode S has taken input A and produced output B.

There may also be schemata that do not take output, or do not take input. But these are treated as PredicateNodes, to be discussed below; their activity is recorded by EvaluationLinks, not ExecutionLinks.

The TruthValue of an ExecutionLink records how frequently the result encoded in the ExecutionLink occurs. Specifically,

EFTA00624183

- the TruthValue of (ExecutionLink S A B) tells you the probability of getting B as output, given that you have run schema S on input A
- the TruthValue of (ExecutionLink S A) tells you the probability that if S is run, it is run on input A

Often it is useful to record the time at which a given act of schema execution was carried out; in that case one uses the atTime link, writing e.g.

```
atTimeLink
    T
    ExecutionLink S A B
```

where T is a TimeNode, or else one uses an implicit method such as storing the time-stamp of the ExecutionLink in a core-level data-structure called the TimeServer. The implicit method is logically equivalent to explicitly using atTime, and is treated the same way by PLN inference, but provides significant advantages in terms of memory usage and lookup speed.

For purposes of logically reasoning about schema, it is useful to create binary links representing ExecutionLinks with some of their arguments fixed. We name these as follows:

```
ExecutionLink1 A B means: X so that ExecutionLink X A B

ExecutionLink2 A B means: X so that ExecutionLink A X B

ExecutionLink3 A B means: X so that ExecutionLink A B X
```

Finally, a SchemaNode may be associated with a structure called a Graph.

Where S is a SchemaNode,

```
Graph(S) = { (x,y): ExecutionLink S x y }
```

Sometimes, the graph of a SchemaNode may be explicitly embodied as a ConceptNode; other times, it may be constructed implicitly by a MindAgent in analyzing the SchemaNode (e.g. the inference MindAgent).

Note that the set of ExecutionLinks describing a SchemaNode may not define that SchemaNode exactly, because some of them may be derived by inference. This means that the model of a SchemaNode contained in its ExecutionLinks may not actually be a mathematical function, in the sense of assigning only one output to each input. One may have

```
ExecutionLink S X A <.5>

ExecutionLink S X B <.5>
```

meaning that the system does not know whether S(X) evaluates to A or to B. So the set of ExecutionLinks modeling a SchemaNode may constitute a non-function relation, even if the schema inside the SchemaNode is a function.

Finally, what of the case where f x represents the action of a built-in system function f on an argument x? This is an awkward case that would not be necessary if the CogPrime system were revised so that all cognitive functions were carried out using SchemaNodes. However, in the current CogPrime version, where most cognitive functions are carried out using C++ MindAgent objects, if we want CogPrime to study its own cognitive behavior in a statistical way, we need BuiltInSchemaNodes that refer to MindAgents rather than to ComboTrees (or else, we need to represent MindAgents using ComboTrees, which will become practicable once we have a sufficiently efficient Combo interpreter). The semantics here is thus basically the same as where f refers to a schema. For instance we might have

EFTA00624184

```
ExecutionLink FirstOrderInferenceMindAgent (L1, L2) L3
```

where L1, L2 and L3 are links related by

```
L1
L2
|-
L3
```

according to the first-order PLN deduction rules.

20.3.1.1 Predicates

Predicates are related but not identical to schema, both conceptually and notationally. PredicateNodes involve *predicate schema* which output TruthValue objects. But there is a difference between a SchemaNode embodying a predicate schema and a PredicateNode, which is that a PredicateNode doesn't output a TruthValue, it adjusts its own TruthValue as a result of the output of its own internal predicate schema.

The record of the activity of a PredicateNode is given not by an ExecutionLink but rather by an:

```
EvaluationLink P A <tv>
```

where P is a PredicateNode, A is its input, and <tv> is the truth value assumed by the EvaluationLink corresponding to the PredicateNode being fed the input A. There is also the variant

```
EvaluationLink P <tv>
```

for the case where the PredicateNode P embodies a schema that takes no inputs[1].

A simple example of a PredicateNode is the predicate GreaterThan. In this case we have, for instance

```
EvaluationLink GreaterThan 5 6 <0>

EvaluationLink GreaterThan 5 3 <1>
```

and we also have:

```
EquivalenceLink
    GreaterThan
    ExOutLink
        And
        ListLink
            ExOutLink
                Not
                LessThan
            ExOutLink
                Not
                EqualTo
```

Note how the variables have been stripped out of the expression, see the PLN book for more explanation about that. We will also encounter many commonsense-semantics predicates such as isMale, with e.g.

[1] actually, if P does take some inputs, `EvaluationLink P <tv>` is defined too and tv corresponds to the average of P(X) over all inputs X, this is explained in more depth in the PLN book.

EFTA00624185

```
EvaluationLink isMale Ben_Goertzel <1>
```

Schemata that return no outputs are treated as predicates, and handled using EvaluationLinks. The truth value of such a predicate, as a default, is considered as True if execution is successful, and False otherwise.

And, analogously to the Graph operator for SchemaNodes, we have for PredicateNodes the SatisfyingSet operator, defined so that the SatisfyingSet of a predicate is the set whose members are the elements that satisfy the predicate. Formally, that is:

```
S = SatisfyingSet P
```

means

```
TruthValue(MemberLink X S)
```

equals

```
TruthValue(EvaluationLink P X)
```

This operator allows the system to carry out advanced logical operations like higher-order inference and unification.

20.3.2 Denoting Schema and Predicate Variables

CogPrime sometimes uses variables to represent the expressions inside schemata and predicates, and sometimes uses variable-free, combinatory-logic-based representations. There are two sorts of variables in the system, either of which may exist either inside compound schema or predicates, or else in the AtomSpace as VariableNodes:

It is important to distinguish between two sorts of variables that may exist in CogPrime:

- Variable Atoms, which may be quantified (bound to existential or universal quantifiers) or unquantified
- Variables that are used solely as function-arguments or local variables inside the "Combo tree" structures used inside some ProcedureNodes (PredicateNodes or SchemaNodes) (to be described below), but are not related to Variable Atoms

Examples of quantified variables represented by Variable Atoms are $X and $Y in:

```
ForAll $X <.0001>
    ExtensionalImplicationLink
        ExtensionalInheritanceLink $X human
        ThereExists $Y
            AND
                ExtensionalInheritanceLink $Y human
                EvaluationLink parent_of ($X, $Y)
```

An example of an unquantified Variable Atom is $X in

```
ExtensionalImplicationLink <.3>
    ExtensionalInheritanceLink $X human
    ThereExists $Y
        AND
            ExtensionalInheritanceLink $Y human
            EvaluationLink parent_of ($X, $Y)
```

EFTA00624186

This ImplicationLink says that 30% of humans are parents: a more useful statement than the ForAll Link given above, which says that it is very very unlikely to be true that all humans are parents.

We may also say, for instance,

```
SatisfyingSet( EvaluationLink eats (cat, $X) )
```

to refer to the set of X so that eats(cat, X).

On the other hand, suppose we have the implication

```
Implication
    Evaluation f $X
    Evaluation
        f
        ExOut reverse $X
```

where f is a PredicateNode embodying a mathematical operator acting on pairs of NumberNodes, and reverse is an operator that reverses a list. So, this implication says that the f predicate is commutative. Now, suppose that f is grounded by the formula

```
f(a,b) = (a > b - 1)
```

embodied in a Combo Tree object (which is not commutative but that is not the point), stored in the ProcedureRepository and linked to the PredicateNode for f. These f-internal variables, which are expressed here using the letters a and b, are not VariableNodes in the CogPrime AtomTable. The notation we use for these within the textual Combo language, that goes with the Combo Tree formalism, is to replace a and b in this example with #1 and #2, so the above grounding would be denoted

```
f -> (#1 > #2 - 1)
```

version, it is assumed that type restrictions are always crisp, not probabilistically truth-valued. This assumption may be revisited in a later version of the system.

20.3.2.1 Links as Predicates

It is conceptually important to recognize that CogPrime link types may be interpreted as predicates. For instance, when one says

```
InheritanceLink cat animal <.8>
```

indicating an Inheritance relation between cat and animal with a strength .8, effectively one is declaring that one has a predicate giving an output of .8. Depending on the interpretation of InheritanceLink as a predicate, one has either the predicate

```
InheritanceLink cat $X
```

acting on the input

```
animal
```

or the predicate

```
InheritanceLink $X animal
```

acting on the input

EFTA00624187

```
cat
```

or the predicate

```
InheritanceLink $X $Y
```

acting on the list input

```
(cat, animal)
```

This means that, if we wanted to, we could do away with all Link types except OrderedLink and UnorderedLink, and represent all other Link types as PredicateNodes embodying appropriate predicate schema.

This is not the approach taken in the current codebase. However, the situation is somewhat similar to that with CIM-Dynamics:

- In future we will likely create a revision of CogPrime that regularly revises its own vocabulary of Link types, in which case an explicit representation of link types as predicate schema will be appropriate.
- In the shorter term, it can be useful to treat link types as *virtual predicates,* meaning that one lets the system create SchemaNodes corresponding to them, and hence do some *meta level reasoning* about its own link types.

20.3.3 Variable and Combinator Notation

One of the most important aspects of combinatory logic, from a CogPrime perspective, is that it allows one to represent arbitrarily complex procedures and patterns without using variables in any direct sense. In CogPrime, variables are optional, and the choice of whether or how to use them may be made (by CogPrime itself) on a contextual basis.

This section deals with the representation of *variable expressions* in a variable-free way, in a CogPrime context. The general theory underlying this is well-known, and is usually expressed in terms of the elimination of variables from lambda calculus expressions (*lambda lifting*). Here we will not present this theory but will restrict ourselves to presenting a simple, hopefully illustrative example, and then discussing some conceptual implications.

20.3.3.1 Why Eliminating Variables is So Useful

Before launching into the specifics, a few words about the general utility of variable-free expression may be worthwhile.

Some expressions look simpler to the trained human eye with variables, and some look simpler without them. However, the main reason why eliminating all variables from an expression is sometimes very useful, is that there are automated program-manipulation techniques that work much more nicely on programs (schemata, in CogPrime lingo) without any variables in them.

As will be discussed later (e.g. Chapter 33 on evolutionary learning, although the same process is also useful for supporting probabilistic reasoning on procedures), in order to mine patterns among multiple schema that all try to do the same (or related) things, we want to put schema into a kind of "hierarchical normal form." The normal form we wish to use generalizes

EFTA00624188

Holman's Elegant Normal Form (which is discussed in Moshe Looks' PhD thesis) to program trees rather than just Boolean trees.

But, putting computer programs into a useful, nicely-hierarchically-structured normal form is a hard problem - it requires one to have a pretty nice and comprehensive set of *program transformations*.

But the only general, robust, systematic program transformation methods that exist in the computer science literature require one to remove the variables from one's programs, so that one can use the theory of functional programming (which ties in with the theory of monads in category theory, and a lot of beautiful related math).

In large part, we want to remove variables so we can use functional programming tools to normalize programs into a standard and pretty hierarchical form, in order to mine patterns among them effectively.

However, we don't *always* want to be rid of variables, because sometimes, from a logical reasoning perspective, theorem-proving is easier with the variables in there. (Sometimes not.)

So, we want to have the option to use variables, or not.

20.3.3.2 An Example of Variable Elimination

Consider the PredicateNode

```
AND
  InheritanceLink X cat
  eats X mice
```

Here we have used a *syntactically sugared* representation involving the variable X. How can we get rid of the X?

Recall the C combinator (from combinatory logic), defined by

```
C f x y = f y x
```

Using this tool,

```
InheritanceLink X cat
```

becomes

```
C InheritanceLink cat X
```

and

```
eats X mice
```

becomes

```
C eats mice X
```

so that overall we have

```
AND
  C InheritanceLink cat
  C eats mice
```

where the C combinators essentially give instructions as to where the *virtual argument* X should go.

In this case the variable-free representation is basically just as simple as the variable-based representation, so there is nothing to lose and a lot to gain by getting rid of the variables. This

EFTA00624189

won't always be the case - sometimes execution efficiency will be significantly enhanced by use of variables.

WIKISOURCE:TypeInheritance

20.3.4 Inheritance Between Higher-Order Types

Next, this section deals with the somewhat subtle matter of Inheritance between higher-order types. This is needed, for example, when one wants to cross over or mutate two complex schemata, in an evolutionary learning context. One encounters questions like: When mutation replaces a schema that takes integer input, can it replace it with one that takes general numerical input? How about vice versa? These questions get more complex when the inputs and outputs of schema may themselves be schema with complex higher-order types. However, they can be dealt with elegantly using some basic mathematical rules.

Denote the type of a mapping from type T to type S, as `T -> S`. Use the shorthand *inh* to mean *inherits from*. Then the basic rule we use is that

```
T1 -> S1 inh T2 -> S2
```

iff

```
T2 inh T1
S1 inh S2
```

In other words, we assume higher-order type inheritance is countervariant. The reason is that, if R1 = T1 -> S1 is to be a special case of R2 = T2 -> S2, then one has to be able to use the latter everywhere one uses the former. This means that any input R2 takes, has to also be taken by R1 (hence T2 inherits from T1). And it means that the outputs R2 gives must be able to be accepted by any function that accepts outputs of R1 (hence S1 inherits from S2).

This type of issue comes up in programming language design fairly frequently, and there are a number of research papers debating the pros and cons of countervariance versus covariance for complex type inheritance. However, for the purpose of schema type inheritance in CogPrime, the greater logical consistency of the countervariance approach holds sway.

For instance, in this approach, `INT -> INT` is not a subtype of `NO -> INT` (where `NO` denotes `FLOAT`), because `NO -> INT` is the type that includes all functions which take a real and return an int, and an `INT -> INT` does not take a real. Rather, the containment is the other way around: every `NO -> INT` function is an example of an `INT -> INT` function. For example, consider the `NO -> INT` that takes every real number and rounds it up to the nearest integer. Considered as an `INT -> INT` function, this is simply the identity function: it is the function that takes an integer and rounds it up to the nearest integer.

Of course, tupling of types is different, it's covariant. If one has an ordered pair whose elements are of different types, say `(T1, T2)`, then we have

```
(T1 , S1) inh (T2, S2)
```

iff

```
T1 inh T2
S1 inh S2
```

As a mnemonic formula, we may say

EFTA00624190

```
(general -> specific) inherits from (specific -> general)
```

```
(specific, specific) inherits from (general, general)
```

In schema learning, we will also have use for abstract type constructions, such as

```
(T1, T2) where T1 inherits from T2
```

Notationally, we will refer to variable types as Xv1, Xv2, etc., and then denote the inheritance relationships by using numerical indices, e.g. using

```
[1 inh 2]
```

to denote that

```
Xv1 inh Xv2
```

So for example,

```
(INT, VOID) inh (Xv1, Xv2)
```

is true, because there are no restrictions on the variable types, and we can just assign Xv1 = INT, Xv2 = VOID.

On the other hand,

```
( INT, VOID ) inh ( Xv1, Xv2 ), [ 1 inh 2 ]
```

is false because the restriction Xv1 inh Xv2 is imposed, but it's not true that INT inh VOID.

The following list gives some examples of type inheritance, using the elementary types INT, FLOAT (FL), NUMBER (NO), CHAR and STRING (STR), with the elementary type inheritance relationships

- INT inh NUMBER
- FLOAT inh NUMBER
- CHAR inh STRING
- (NO -> FL) inh (INT -> FL)
- (FL -> INT) inh (FL -> NO)
- ((INT -> FL) -> (FL -> INT)) inh ((NO -> FL) -> (FL -> NO)) WIKISOURCE:AbstractSchemaManipulation

20.3.5 Advanced Schema Manipulation

Now we describe some special schema for manipulating schema, which seem to be very useful in certain contexts.

20.3.5.1 Listification

First, there are two ways to represent n-ary relations in CogPrime's Atom level knowledge representation language: using lists as in

```
f_list (x1, ..., xn)
```

or using currying as in

EFTA00624191

```
f_curry x1 ... xn
```

To make conversion between list and curried forms easier, we have chosen to introduce special schema (combinators) just for this purpose:

```
listify f = f_list so that f_list (x1, ..., xn ) = f x1 ... xn

unlistify listify f = f
```

For instance

```
kick_curry Ben Ken
```

denotes

```
(kick_curry Ben) Ken
```

which means that kick is applied to the argument Ben to yield a predicate schema applied to Ken. This is the curried style. The list style is

```
kick_List (Ben, Ken)
```

where kick is viewed as taking as an argument the List (Ben, Ken). The conversion between the two is done by

```
listify kick_curry = kick_list

unlistify kick_list = kick_curry
```

As a more detailed example of unlistification, let us utilize a simple mathematical example, the function $(X-1)^2$. If we use the notations - and *pow* to denote SchemaNodes embodying the corresponding operations, then this formula may be written in variable-free node-and-link form as

```
ExOutLink
    pow
    ListLink
        ExOutLink
            -
            ListLink
                X
                1
        2
```

But to get rid of the nasty variable X, we need to first unlistify the functions *pow* and -, and then apply the C and B combinators a couple times to move the variable X to the front. The B combinator (see Combinatory Logic REF) is recalled below:

```
B f g h = f (g h)
```

This is accomplished as follows (using the standard convention of left-associativity for the application operator, denoted @ in the tree representation given in Section 20.3.0.6)

```
pow(-(x, 1), 2)
unlistify pow (-(x, 1) 2)
C (unlistify pow) 2 (-(x,1))
C (unlistify pow) 2 ((unlistify -) x 1)
C (unlistify pow) 2 (C (unlistify -) 1 x)
B (C (unlistify pow) 2) (C (unlistify -) 1) x
```

EFTA00624192

yielding the final schema

```
B (C (unlistify pow) 2) (C (unlistify -) 1)
```

By the way, a variable-free representation of this schema in CogPrime would look like

```
ExOutLink
    ExOutLink
        B
        ExOutLink
            ExOutLink
                C
                ExOutLink
                    unlistify
                    pow
            2
    ExOutLink
        ExOutLink
            C
            ExOutLink
                unlistify
                -
        1
```

The main thing to be observed is that the introduction of these extra schema lets us remove the variable X. The size of the schema is increased slightly in this case, but only slightly - an increase that is well - justified by the elimination of the many difficulties that explicit variables would bring to the system. Furthermore, there is a shorter rendition which looks like

```
ExOutLink
    ExOutLink
        B
        ExOutLink
            ExOutLink
                C
                pow_curried
            2
    ExOutLink
        ExOutLink
            C
            -_curried
        1
```

This rendition uses alternate variants of - and pow schema, labeled `-_curried` and `pow_curried`, which do not act on lists but are *curried* in the manner of combinatory logic and Haskell. It is 13 lines whereas the variable-bearing version is 9 lines, a minor increase in length that brings a lot of operational simplification.

20.3.5.2 Argument Permutation

In dealing with List relationships, there will sometimes be use for an argument-permutation operator, let us call it P, defined as follows

```
(P p f) (v1, ..., vn) = f (p (v1, ..., vn ))
```

where p is a permutation on n letters. This deals with the case where we want to say, for instance that

EFTA00624193

```
Equivalence parent(x,y) child(y,x)
```

Instead of positing variable names x and y that span the two relations parent(x,y) and child(y,x), what we can instead say in this example is

```
Equivalence parent (P {2,1} child)
```

For the case of two-argument functions, argument permutation is basically doing on the list level what the C combinator does in the curried function domain. On the other hand, in the case of n-argument functions with n>2, argument permutation doesn't correspond to any of the standard combinators.

Finally, let's conclude with a similar example in a more standard predicate logic notation, involving both combinators and the permutation argument operator introduced above. We will translate the variable-laden predicate

```
likes(y,x) AND likes(x,y)
```

into the equivalent combinatory logic tree. Let us first recall the combinator S whose function is to distribute an argument over two terms.

```
S f g x = (f x) (g x)
```

Assume that the two inputs are going to be given to us as a list. Now, the combinatory logic representation of this is

```
S (B AND (B (P {2,1} likes))) likes
```

We now show how this would be evaluated to produce the correct expression:

```
S (B AND (B (P {2,1} likes))) likes (x,y)
```

S gets evaluated first, to produce

```
(B AND (B (P {2,1} likes)) (x,y)) (likes (x,y))
```

now the first B

```
AND ((B (P {2,1} likes)) (x,y)) (likes (x,y))
```

now the second one

```
AND ((P {2,1} likes) (x,y)) (likes (x,y))
```

now P

```
AND (likes (y,x)) (likes (x,y))
```

which is what we wanted.

EFTA00624194

Chapter 21
Representing Procedural Knowledge

21.1 Introduction

We now turn to CogPrime's representation and manipulation of *procedural knowledge*. In a sense this is the most fundamental kind of knowledge – since intelligence is most directly about action selection, and it is procedures which generate actions.

CogPrime involves multiple representations for procedures, including procedure maps and (for sensorimotor procedures) neural nets or similar structures. Its most basic procedural knowledge representation, however, is the *program*. The choice to use programs to represent procedures was made after considerable reflection – they are not of course the only choice, as other representations such as recurrent neural networks possess identical representational power, and are preferable in some regards (e.g. resilience with respect to damage). Ultimately, however, we chose programs due to their consilience with the software and hardware underlying CogPrime (and every other current AI program). CogPrime is a program, current computers and operating systems are optimized for executing and manipulating programs; and we humans now have many tools for formally and informally analyzing and reasoning about programs. The human brain probably doesn't represent most procedures as programs in any simple sense, but CogPrime is not intended to be an emulation of the human brain. So, the representation of programs as procedures is one major case where CogPrime deviates from the human cognitive architecture in the interest of more effectively exploiting its own hardware and software infrastructure.

CogPrime represents procedures as programs in an internal programming language called "Combo." While Combo has a textual representation, described online at the OpenCog wiki, this isn't one of its more important aspects (and may be redesigned slightly or wholly without affecting system intelligence or architecture); the essence of Combo programs lies in their tree representation not their text representation. One could fairly consider Combo as a dialect of LISP, although it's not equivalent to any standard dialect, and it hasn't particularly been developed with this in mind. In this chapter we discuss the key concepts underlying the Combo approach to program representation, seeking to make clear at each step the motivations for doing things in the manner proposed.

In terms of the overall CogPrime architecture diagram given in Chapter 6 of Part 1, this chapter is about the box labeled "Procedure Repository." The latter, in OpenCog, is a specialized component connected to the AtomSpace, storing Combo tree representations of programs;

EFTA00624196

each program in the repository is linked to a SchemaNode in the AtomSpace, ensuring full connectivity between procedural and declarative knowledge.

21.2 Representing Programs

What is a "program" anyway? What distinguishes a program from an arbitrary representation of a procedure?

The essence of programmatic representations is that they are well-specified, compact, combinatorial, and hierarchical:

- *Well-specified:* unlike sentences in natural language, programs are unambiguous; two distinct programs can be precisely equivalent.
- *Compact:* programs allow us to compress data on the basis of their regularities. Accordingly, for the purposes of this chapter, we do not consider overly constrained representations such as the well-known conjunctive and disjunctive normal forms for Boolean formulae to be programmatic. Although they can express any Boolean function (data), they dramatically limit the range of data that can be expressed compactly, compared to unrestricted Boolean formulae.
- *Combinatorial:* programs access the results of running other programs (e.g. via function application), as well as delete, duplicate, and rearrange these results (e.g., via variables or combinators).
- *Hierarchical:* programs have intrinsic hierarchical organization, and may be decomposed into subprograms.

Eric Baum has advanced a theory "under which one understands a problem when one has mental programs that can solve it and many naturally occurring variations" [Bau06]. In this perspective – which we find an agreeable way to think about procedural knowledge, though perhaps an overly limited perspective on mind as a whole – one of the primary goals of artificial general intelligence is systems that can represent, learn, and reason about such programs [Bau06, Bau04]. Furthermore, integrative AGI systems such as CogPrime may contain subsystems operating on programmatic representations. Would-be AGI systems with no direct support for programmatic representation will clearly need to represent procedures and procedural abstractions *somehow*. Alternatives such as recurrent neural networks have serious downsides, including opacity and inefficiency, but also have their advantages (e.g. recurrent neural nets can be robust with regard to damage, and learnable via biologically plausible algorithms).

Note that the problem of how to represent programs for an AGI system dissolves in the unrealistic case of unbounded computational resources. The solution is algorithmic information theory [Cha08], extended recently to the case of sequential decision theory [Hut05a]. The latter work defines the universal algorithmic agent AIXI, which in effect simulates all possible programs that are in agreement with the agent's set of observations. While AIXI is uncomputable, the related agent AIXI^{tl} may be computed, and is superior to any other agent bounded by time t and space l [Hut05b]. The choice of a representational language for programs[1] is of no consequence, as it will merely introduce a bias that will disappear within a constant number of time steps.[2]

[1] As well as a language for proofs in the case of AIXI^{tl}.

[2] The universal distribution converges quickly.

EFTA00624197

Our goal in this chapter is to provide practical techniques for approximating the ideal provided by algorithmic probability, based on what Pei Wang has termed the *assumption of insufficient knowledge and resources* [Wan06], and assuming an AGI architecture that's at least vaguely humanlike in nature, and operates largely in everyday human environments, but uses programs to represent many procedures. Given these assumptions, how programs are represented is of paramount importance, as we shall see in the next two sections, where we give a conceptual formulation of what we mean by *tractable program representations*, and introduce tools for formalizing such representations. The fourth section delves into effective techniques for representing programs. A key concept throughout is *syntactic-semantic correlation*, meaning that programs which are similar on the syntactic level, within certain constraints will tend to also be similar in terms of their behavior (i.e. on the semantic level). Lastly, the fifth section changes direction a bit and discusses the translation of programmatic structure into declarative form for the purposes of logical inference.

In the future, we will experimentally validate that these normal forms and heuristic transformations *do* in fact increase the syntactic-semantic correlation in program spaces, as has been shown so far only in the Boolean case. We would also like to explore the extent to which even stronger correlation, and additional tractability properties, can be observed when realistic probabilistic constraints on "natural" environment and task spaces are imposed.

The importance of a good programmatic representation of procedural knowledge becomes quite clear when one thinks about it in terms of the Mind-World Correspondence Principle introduced in Chapter 10. That principle states, roughly, that transition paths between world-states should map naturally onto transition paths between mind-states. This suggests that there should be a natural, smooth mapping between *real-world action series* and the corresponding *series of internal states*. Where internal states are driven by explicitly given programs, this means that the transitions between internal program states should nicely mirror transitions between the states of the real world as it interacts with the system controlled by the program. The extent to which this is true will depend on the specifics of the programming language – and it will be true for a much greater extent, on the whole, if the programming language displays high syntactic-semantic correlation for behaviors that commonly occur when the program is used to control the system in the real world. So, the various technical issues mentioned above and considered below, regarding the qualities desired in a programmatic representation, are merely the manifestation of the general Mind-World Correspondence Principle in the context of procedural knowledge, under the assumption that procedures are represented as programs. The material in this chapter may be viewed as an approach to ensuring the validity of the Mind-World Correspondence principle for programmatically-represented procedural knowledge, for CogPrime systems concerned with achieving humanly meaningful goals in everyday human environments.

21.3 Representational Challenges

Despite the advantages outlined in the previous section, there are a number of challenges in working with programmatic representations:

- **Open-endedness** – in contrast to some other knowledge representations current in machine learning, programs vary in size and "shape", and there is no obvious problem-independent

EFTA00624198

upper bound on program size. This makes it difficult to represent programs as points in a fixed-dimensional space, or to learn programs with algorithms that assume such a space.

- **Over-representation** – often, syntactically distinct programs will be semantically identical (i.e. represent the same underlying behavior or functional mapping). Lacking prior knowledge, many algorithms will inefficiently sample semantically identical programs repeatedly [Loo07a, GBK04].
- **Chaotic Execution** – programs that are very similar, syntactically, may be very different, semantically. This presents difficulties for many heuristic search algorithms, which require syntactic and semantic distance to be correlated [Loo07b, TVCC05].
- **High resource-variance** – programs in the same space vary greatly in the space and time they require to execute.

It's easy to see how the latter two issues may present a challenge for mind-world correspondence! Chaotic execution makes it hard to predict whether a program will indeed manifest state-sequences mapping nicely to a corresponding world-sequences; and high resource-variance makes it hard to predict whether, for a given program, this sort of mapping can be achieved for relevant goals given available resources.

Based on these concerns, it is no surprise that search over program spaces quickly succumbs to combinatorial explosion, and that heuristic search methods are sometimes no better than random sampling [LP02]. However, alternative representations of procedures also have their difficulties, and so far we feel the thornier aspects of programmatic representation are generally an acceptable price to pay in light of the advantages.

For some special cases in CogPrime we have made a different choice – e.g. when we use DeSTIN for sensory perception (see Chapter 28 we utilize a more specialized representation comprising a hierarchical network of more specialized elements. DeSTIN doesn't have problems with resource variance or chaotic execution, though it does suffer from over-representation. It is not very open-ended, which helps increase its efficiency in the perceptual processing domain, but may limit its applicability to more abstract cognition. In short we feel that, for general representation of cognitive procedures, the benefits of programmatic representation outweigh the costs; but for some special cases such as low-level perception and motor procedures, this may not be true and one may do better to opt for a more specialized, more rigid but less problematic representation.

It would be possible to modify CogPrime to use, say, recurrent neural nets for procedure representation, rather than programs in an explicit language. However, this would rate as a rather major change in the architecture, and would cause multiple problems in other aspects of the system. For example, programs are reasonably straightforward to reason about using PLN inference, whereas reasoning about the internals of recurrent neural nets is drastically more problematic, though not impossible. The choice of a procedure representation approach for CogPrime has been made considering not only procedural knowledge in itself, but the interaction of procedural knowledge with other sorts of knowledge. This reflects the general synergetic nature of the CogPrime design.

There are also various computation-theoretic issues regarding programs; however, we suspect these are not particularly relevant to the task of creating human-level AGI, though they may rear their heads when one gets into the domain of super-human, profoundly self-modifying AGI systems. For instance, in the context of the the difficulties caused by over-representation and high resource-variance, one might observe that determinations of e.g. programmatic equivalence for the former, and e.g. halting behavior for the latter, are uncomputable. But we feel that, given the assumption of insufficient knowledge and resources, these concerns dissolve into the

EFTA00624199

larger issue of computational intractability and the need for efficient heuristics. Determining the equivalence of two Boolean formulae over 500 variables by computing and comparing their truth tables is trivial from a computability standpoint, but, in the words of Leonid Levin, "only math nerds would call 2^{500} finite" [Lev94]. Similarly, a program that never terminates is a special case of a program that runs too slowly to be of interest to us.

One of the key ideas underlying our treatment of programmatic knowledge is that, in order to tractably learn and reason about programs, an AI system must have prior knowledge of programming language semantics. That is, in the approach we advocate, the mechanism whereby programs are executed is assumed known *a priori*, and assumed to remain constant across many problems. One may then craft AI methods that make specific use of the programming language semantics, in various ways. Of course in the long run a sufficiently powerful AGI system could modify these aspects of its procedural knowledge representation; but in that case, according to our approach, it would also need to modify various aspects of its procedure learning and reasoning code accordingly.

Specifically, we propose to exploit prior knowledge about program structure via enforcing programs to be represented in normal forms that preserve their hierarchical structure, and to be heuristically simplified based on reduction rules. Accordingly, one formally equivalent programming language may be preferred over another by virtue of making these reductions and transformations more explicit and concise to describe and to implement. The current OpenCog-Prime system uses a simple LISP-like language called Combo (which takes both tree form and textual form) to represent procedures, but this is not critical; the main point is using some language or language variant that is "tractable" in the sense of providing a context in which the semantically useful reductions and transformations we've identified are naturally expressible and easily usable.

21.4 What Makes a Representation Tractable?

Creating a comprehensive formalization of the notion of a *tractable program representation* would constitute a significant achievement; and we will not answer that summons here. We will, however, take a step in that direction by enunciating a set of positive principles for tractable program representations, corresponding closely to the list of representational challenges above. While the discussion in this section is essentially conceptual rather than formal, we will use a bit of notation to ensure clarity of expression; S to denote a space of programmatic functions of the same type (e.g. all pure *Lisp* λ-expressions mapping from lists to numbers), and B to denote a metric space of *behaviors*.

In the case of a deterministic, side-effect-free program, execution maps from programs in S to points in B, which will have separate dimensions for the function's output across various inputs of interest, as well as dimensions corresponding to the time and space costs of executing the program. In the case of a program that interacts with an external environment, or is intrinsically nondeterministic, execution will map from S to probability distributions over points in B, which will contain additional dimensions for any side-effects of interest that programs in S might have. Note the distinction between *syntactic distance*, measured as e.g. tree-edit distance between programs in S, and *semantic distance*, measured between program's corresponding points in or probability distributions over B. We assume that semantic distance accurately quantifies our

EFTA00624200

preferences in terms of a weighting on the dimensions of B; i.e., if variation along some axis is of great interest, our metric for semantic distance should reflect this.

Let $\mathcal{P}$ be a probability distribution over B that describes our knowledge of what sorts of problems we expect to encounter, let $R(n) \subseteq S$ be all the programs in our representation with (syntactic) size no greater than n. We will say that $R(n)$ *d-covers* the pair $(B, \mathcal{P})$ to extent p if the probability that, for a random behavior $b \in B$ chosen according to $\mathcal{P}$, there is some program in R whose behavior is within semantic distance d of b, is greater than or equal to p. Then, some among the various properties of tractability that seem important based on the above discussion are as follows:

- for fixed d, p quickly goes to 1 as n increases,
- for fixed p, d quickly goes to 0 as n increases,
- for fixed d and p, the minimal n needed for $R(n)$ to *d-cover* $(B, \mathcal{P})$ to extent p should be as small as possible,
- ceteris paribus, syntactic and semantic distance (measured according to $\mathcal{P}$) are highly correlated.

This is closely related to the Mind-Brain Correspondence Principle articulated in Chapter 10, and to the geometric formulation of cognitive synergy posited in Appendix **??**. Syntactic distance has to do with distance along paths in mind-space related to formal program structures, and semantic distance has to do with distance along paths in mind-space and world-space corresponding to the record of the program's actual behavior. If syntax-semantics correlation failed, then there would be paths through mind-space (related to formal program structures) that were poorly matched to their closest corresponding paths through the rest of mind-space and world-space, hence causing a failure (or significant diminution) of cognitive synergy and mind-world correspondence.

Since execution time and memory usage considerations may be incorporated into the definition of program behavior, minimizing chaotic execution and managing resource variance emerges conceptually here as subcases of maximizing correlation between syntactic and semantic distance. Minimizing over-representation follows from the desire for small program size: roughly speaking the less over-representation there is, the smaller average program size can be achieved.

In some cases one can achieve fairly strong results about tractability of representations without any special assumptions about $\mathcal{P}$: for example in prior work we have shown that adoption of an appropriate hierarchical normal form can generically increase correlation between syntactic and semantic distance in the space of Boolean functions [**?**, Loo07b]. In this case we may say that we have a *generically tractable* representation. However, to achieve tractable representation of more complex programs, some fairly strong assumptions about $\mathcal{P}$ will be necessary. This should not be philosophically disturbing, since it's clear that human intelligence has evolved in a manner strongly conditioned by certain classes of environments; and similarly, what we need to do to create a viable program representation system for pragmatic AGI usage, is to achieve tractability relative to the distribution $\mathcal{P}$ corresponding to the actual problems the AGI is going to need to solve. Formalizing the distributions $\mathcal{P}$ of real-world interest is a difficult problem, and one we will not address here (recall the related, informal discussions of Chapter 9, where we considered the various important peculiarities of the human everyday world). However, we hypothesize that the representations presented in the following section may be tractable to a significant extent irrespective of $\mathcal{P}$,[3] and even more powerfully tractable with

[3] Specifically, with only weak biases that prefer smaller and faster programs with hierarchical decompositions.

EFTA00624201

respect to this as-yet unformalized distribution. As weak evidence in favor of this hypothesis, we note that many of the representations presented have proved useful so far in various narrow problem-solving situations.

21.5 The Combo Language

The current version of OpenCogPrime uses a simple language called Combo, which is an example of a language in which the transformations we consider important for AGI-focused program representation are relatively simple and natural. Here we illustrate the Combo language by example, referring the reader to the OpenCog wiki site for a formal presentation.

The main use of the Combo language in OpenCog is behind-the-scenes, i.e. using tree representations of Combo programs; but there is also a human-readable syntax, and an interpreter that allows humans to write Combo programs when needed. The main use of Combo, however, is not for human-coded programs, but rather for programs that are learned via various AI methods.

In Combo all expressions are in prefix form like LISP, but the left parenthesis is placed after the operator instead of before, for example:

- `+(4 5)`

  is a 0-ari expression that returns $4 + 5$
- `and(#1 0<(#2))`

  is a binary expression of type $bool \times float \mapsto bool$ that returns true if and only if the first input is true and the second input positive. $\#n$ designates the $n\text{-}th$ input.
- `fact(1) := if(0<(#1) *(#1 fact(+(#1 -1))) 1)`

  is a recursive definition of factorial.
- `and_seq(goto(stick) grab(stick) goto(owner) drop)`

  is a 0-ari expression with side effects, it evaluates a sequence of actions until completion or failure of one of them. Each action is executed in the environment the agent is connected to and returns $action_success$ upon success or $action_failure$ otherwise. The action sequence returns $action_success$ if it completes or $action_failure$ if it does not.
- ```
  if(near(owner self)
      lick(owner)
      and_seq(goto(owner) wag)
  ```

  is a 0-ary expression with side effects; it means that if at the time of its evaluation the agent referred as self (here a virtual pet) is near its owner then lick him/her, otherwise go to the owner and wag the tail.

21.6 Normal Forms Postulated to Provide Tractable Representations

We now present a series of normal forms for programs, postulated to provide tractable representations in the contexts relevant to human-level, roughly human-like general intelligence.

EFTA00624202
  ```

21.6.1 A Simple Type System

We use a simple type system to distinguish between the various normal forms introduced below. This is necessary to convey the minimal information needed to correctly apply the basic functions in our canonical forms. Various systems and applications may of course augment these with additional type information, up to and including the satisfaction of arbitrary predicates (e.g. a type for prime numbers). This can be overlaid on top of our minimalist system to convey additional bias in selecting which transformations to apply, and introducing constraints as necessary. For instance, a call to a function expecting a prime number, called with a potentially composite argument, may be wrapped in a conditional testing the argument's primality. A similar technique is used in the normal form for functions to deal with list arguments that may be empty.

Normal forms are provided for *Boolean* and *number* primitive types, and the following parametrized types:

- list types, $list_T$, where T is any type,
- tuple types, $tuple_{T_1,T_2,\ldots T_N}$, where all T_i are types, and N is a positive natural number,
- enum types, $\{s_1, s_2, \ldots s_N\}$, where N is a positive number and all s_i are unique identifiers,
- function types $T_1, T_2, \ldots T_N \to O$, where O and all T_i are types,
- action result types.

A list of type $list_T$ is an ordered sequence of any number of elements, all of which must have type T. A tuple of type $tuple_{T_1,T_2,\ldots T_N}$ is an ordered sequence of exactly N elements, where every ith element is of type T_i. An enum of type $\{s_1, s_2, \ldots s_N\}$ is some element s_i from the set. Action result types concern side-effectful interaction with some world external to the system (but perhaps simulated, of course), and will be described in detail in their subsection below. Other types may certainly be added at a later date, but we believe that those listed above provide sufficient expressive power to conveniently encompass a wide range of programs, and serve as a compelling proof of concept.

The normal form for a type T is a set of elementary functions with codomain T, a set of constants of type T, and a tree grammar. Internal nodes for expressions described by the grammar are elementary functions, and leaves are either U_{var} or $U_{constant}$, where U is some type (often $U = T$).

Sentences in a normal form grammar may be transformed into normal form expressions. The set of expressions that may be generated is a function of a set of bound variables and a set of external functions that must be provided (both bound variables and external functions are typed). The transformation is as follows:

- $T_{constant}$ leaves are replaced with constants of type T,
- T_{var} leaves are replaced with either bound variables matching type T, or expressions of the form $f(expr_1, expr_2, \ldots expr_M)$, where f is an external function of type $T_1, T_2, \ldots T_M \to T$, and each $expr_i$ is a normal form expression of type T_i (given the available bound variables and external functions).

EFTA00624203

21.6.2 Boolean Normal Form

The elementary functions are *and*, *or*, and *not*. The constants are $\{true, false\}$. The grammar is:

```
bool_root = or_form  | and_form | literal  | bool_constant
literal   = bool_var | not( bool_var )
or_form   = or( {and_form | literal}{2,} )
and_form  = and( {or_form | literal}{2,} ) .
```

The construct `foo{x,}` refers to `x` or more matches of `foo` (e.g. `{x | y}{2,}` is two or more items in sequences where each item is either an `x` or a `y`).

21.6.3 Number Normal Form

The elementary functions are $*$ (times) and $+$ (plus). The constants are some subset of the rationals (e.g. those with IEEE single-precision floating-point representations). The grammar is:

```
num_root   = times_form | plus_form | num_constant | num_var
times_form = *( (num_constant | plus_form) plus_form{1,} )
           | num_var
plus_form  = +( (num_constant | times_form) times_form{1,} )
           | num_var
```

21.6.4 List Normal Form

For list types $list_T$, the elementary functions are *list* (an n-ary list constructor) and *append*. The only constant is the empty list (*nil*). The grammar is:

```
list_T_root = append_form | list_form | list_T_var
            | list_T_constant
append_form = append( {list_form | list_T_var}{2,} )
list_form   = list( T_root{1,} )
```

21.6.5 Tuple Normal Form

For tuple types $tuple_{T_1,T_2,\ldots T_N}$, the only elementary function is the tuple constructor (*tuple*). The constants are

$$T_1\texttt{_constant} \times T_2\texttt{_constant} \times \cdots \times T_N\texttt{_constant}$$

The normal form is either a constant, a var, or

$$\texttt{tuple(}\ T_1\texttt{_root}\ \ T_2\texttt{_root}\ \ldots\ T_N\texttt{_root}\ \texttt{)}$$

EFTA00624204

21.6.6 Enum Normal Form

Enums are atomic tokens with no internal structure - accordingly, there are no elementary functions. The constants for the enum $\{s_1, s_2, \ldots s_N\}$ are the s_is. The normal form is either a constant or a variable.

21.6.7 Function Normal Form

For $T_1, T_2, \ldots T_N \rightarrow O$, the normal form is a lambda-expression of arity N whose body is of type O. The list of variable names for the lambda-expression is not a "proper" argument - it does not have a normal form of its own. Assuming that none of the T_is is a list type, the body of the lambda-expression is simply in the normal form for type O (with the possibility of the lambda-expressions arguments appearing with their appropriate types). If one or more T_is are list types, then the body is a call to the *split* function with all arguments in normal form.

Split is a family of functions with type signatures

$$(T_1, list_{T_1}, T_2, list_{T_2}, \ldots T_k, list_{T_k} \rightarrow O),$$
$$tuple_{list_{T_1},O}, tuple_{list_{T_2},O}, \ldots tuple_{list_{T_k},O} \rightarrow O.$$

To evaluate $split(f, tuple(l_1, o_1), tuple(l_2, o_2), \ \ldots \ tuple(l_k, o_k))$, the list arguments $l_1, l_2, \ldots l_k$ are examined sequentially. If some l_i is found that is empty, then the result is the corresponding value o_i. If all l_i are nonempty, we deconstruct each of them into $x_i : xs_i$, where x_i is the first element of the list and xs_i is the rest. The result is then $f(x_1, xs_1, x_2, xs_2, \ldots x_k, xs_k)$. The *split* function thus acts as an implicit case statement to deconstruct lists only if they are nonempty.

21.6.8 Action Result Normal Form

An action result type *act* corresponds to the result of taking an action in some world. Every action result type has a corresponding world type, *world*. Associated with action results and worlds are two special sorts of functions.

- *Perceptions* - functions that take a *world* as their first argument and regular (non-world and non-action-result) types as their remaining arguments, and return regular types. Unlike other function types, the result of evaluating a perception call may be different at different times, because the world will have different configurations at different times.
- *Actions* - functions that take a *world* as their first argument and regular types as their remaining arguments, and return action results (of the type associated with the type of their world argument). As with perceptions, the result of evaluating an action call may be different at different times. Furthermore, actions may have *side effects* in the associated world that they are called in. Thus, unlike any other sort of function, actions *must* be evaluated, even if their return values are ignored.

Other sorts of functions acting on worlds (e.g. ones that take multiple worlds as arguments) are disallowed.

EFTA00624205

Note that an action result expression cannot appear nested inside an expression of any other type. Consequently, there is no way to convert e.g. an action result to a Boolean, although conversion in the opposite direction is permitted. This is required because mathematical operations in our language have classical mathematical semantics; x *and* y must equal y *and* x, which will not generally be the case if x or y can have side-effects. Instead, there are special sequential versions of logical functions which may be used instead.

The elementary functions for action result types are and_{seq} (sequential and, equivalent to C's short-circuiting `&&`), or_{seq} (sequential or, equivalent to C's short-circuiting `||`), and $fails$ (negates success to failure and vice versa). The constants may vary from type to type but must at least contain *success* and $failure$, indicating absolute success/failure in execution.[4] The normal form is as follows:

```
act_root     = orseq_form | andseq_form | seqlit
seqlit       = act | fails( act )
act          = act_constant | act_var
orseq_form   = orseq( {andseq_form | seqlit}{2,} )
andseq_form  = andseq( {orseq_form | seqlit}{2,} )
```

21.7 Program Transformations

A program transformation is any type-preserving mapping from expressions to expressions. Transformations may be guaranteed to preserve semantics. When doing program evolution there is an intermediate category of fitness preserving transformations that may alter semantics, but not fitness. In general, the only way that fitness preserving transformations will be uncovered is by scoring programs that have had their semantics potentially transformed to determine their fitness, which is what most fitness function does. On the other hand if the fitness function is encompassed in the program itself, so a candidate directly outputs the fitness itself, then only preserving semantics transformations are needed.

21.7.1 Reductions

These are semantics preserving transformations that do not increase some size measure (typically number of symbols), and are idempotent. For example, $and(x, x, y) \rightarrow and(x, y)$ is a reduction for Boolean expressions. A set of *canonical reductions* is defined for every type that has a normal form. For numerical functions, the simplifier in a computer algebra system may be used. The full list of reductions is omitted in for brevity. An expression is *reduced* if it maps to itself under all canonical reductions for its type, and all of its children are reduced.

Another important set of reductions are the *compressive abstractions*, which reduce or keep constant the size of expressions by introducing new functions. Consider

```
list(*(+(a p q) r)
     *(+(b p q) r)
     *(+(c p q) r))
```

[4] A $do(arg_1, arg_2, \ldots arg_N)$ statement (known as *progn* in Lisp), which evaluates its arguments sequentially regardless of success or failure, is equivalent to $and_{seq}(or_{seq}(arg_1, success), or_{seq}(arg_2, success), \ldots or_{seq}(arg_N, success))$.

EFTA00624206

which contains 19 symbols. Transforming this to

```
f(x) = *(+(x p q) r)
list(f(a) f(b) f(c))
```

reduces the total number of symbols to 15. One can generalize this notion to consider compressive abstractions across a set of programs. Compressive abstractions appear to be rather expensive to uncover, although not prohibitively so, the computation may easily be parallelized and may rely heavily on subtree mining [TODO REF].

21.7.1.1 A Simple Example of Reduction

We now give a simple example of how CogPrime's reduction engine can transform a program into a semantically equivalent but shorter one.

Consider the following program and the chain of reduction:

1. We start with the expression

   ```
   if(P and_seq(if(P A B) B) and_seq(A B))
   ```

2. A reduction rule permits to reduce the conditional `if(P A B)` to `if(true A B)`. Indeed if `P` is true, then the first branch is evaluated and `P` must still be true.

   ```
   if(P and_seq(if(true A B) B) and_seq(A B))
   ```

3. Then a rule can reduce `if(true A B)` to `A`.

   ```
   if(P and_seq(A B) and_seq(A B))
   ```

4. And finally another rule replaces the conditional by one of its branches since they are identical

   ```
   and_seq(A B)
   ```

Note that the reduced program is not only smaller (3 symbols instead of 11) but a bit faster too. Of course it is not generally true that smaller programs are faster but in the restricted context of our experiments it has often been the case.

21.7.2 Neutral Transformations

Semantics preserving transformations that are not reductions are not useful on their own - they can only have value when followed by transformations from some other class. They are thus more speculative than reductions, and more costly to consider. I will refer to these as *neutral transformations* [Ols95].

- **Abstraction** - given an expression E containing non-overlapping subexpressions E_1, E_2, $\ldots E_N$, let E' be E with all E_i replaced by the unbound variables v_i. Define the function $f(v_1, v_2, \ldots v_3) = E'$, and replace E with $f(E_1, E_2, \ldots E_N)$. Abstraction is distinct from compressive abstraction because only a single call to the new function f is introduced.[5]
- **Inverse abstraction** - replace a call to a user-defined function with the body of the function, with arguments instantiated (note that this can also be used to partially invert a compressive abstraction).

[5] In compressive abstraction there must be at least two calls in order to avoid increasing the number of symbols.

EFTA00624207

- **Distribution** - let E be a call to some function f, and let E' be an expression of E's ith argument that is a call to some function g, such that f is distributive over g's arguments, or a subset thereof. We shall refer to the actual arguments to g in these positions in E' as $x_1, x_2, \ldots x_n$. Now, let $D(F)$ be the function that is obtained by evaluating E with its ith argument (the one containing E') replaced with the expression F. Distribution is replacing E with E', and then replacing each x_j $(1 \leq j \leq n)$ with $D(x_j)$. For example, consider

  ```
  +(x *(y if(cond a b)))
  ```

  Since both + and * are distributive over the result branches of *if*, there are two possible distribution transformations, giving the expressions

  ```
  if(cond +(x *(y a)) +(x *(y b)))
  +(x(if(cond *(y a) *(y b))))
  ```

- **Inverse distribution (factorization)** - the opposite of distribution. This is nearly a reduction; the exceptions are expressions such as $f(g(x))$, where f and g are mutually distributive.
- **Arity broadening** - given a function f, modify it to take an additional argument of some type. All calls to f must be correspondingly broadened to pass it an additional argument of the appropriate type.
- **List broadening**[6] - given a function f with some ith argument x of type T, modify f to instead take an argument y of type $list_T$, which gets split into $x : xs$. All calls to f with ith argument x' must be replaced by corresponding calls with ith argument $list(x')$.
- **Conditional insertion** - an expression x is replaced by $if(true, x, y)$, where y is some expression of the same type of x.

As a technical note, action result expressions (which may cause side-effects) complicate neutral transformations. Specifically, abstractions and compressive abstractions must take their arguments lazily (i.e. not evaluate them before the function call itself is evaluated), in order to be neutral. Furthermore, distribution and inverse distribution may only be applied when f has no side-effects that will vary (e.g. be duplicated or halved) in the new expression, or affect the nested computation (e.g. change the result of a condition within a conditional). Another way to think about this issue is to consider the action result type as a lazy domain-specific language embedded within a pure functional language (where evaluation order is unspecified). Spector has performed an empirical study of the tradeoffs in lazy vs. eager function abstraction for program evolution [Spe96].

The number of neutral transformations applicable to any given program grows quickly with program size.[7] Furthermore, synthesis of complex programs and abstractions does not seem to be possible without them. Thus, a key hypothesis of any approach to AGI requiring significant program synthesis, without assuming the currently infeasible computational capacities required to brute-force the problem, is that the inductive bias to select promising neutral transformations can be learned and/or programmed. Referring back to the initial discussion of what constitutes a tractable representation, we speculate that perhaps, whereas well-chosen reductions are valuable for generically increasing program representation tractability, well-chosen neutral transformations will be valuable for increasing program representation tractability relative to distributions $\mathcal{P}$ to which the transformations have some (possibly subtle) relationship.

[6] Analogous tuple-broadening transformations may be defined as well, but are omitted for brevity.

[7] Exact calculations are given by Olsson [Ols95].

EFTA00624208

21.7.3 Non-Neutral Transformations

Non-neutral transformations are the general class defined by removal, replacement, and insertion of subexpressions, acting on expressions in normal form, and preserving the normal form property. Clearly these transformations are sufficient to convert any normal form expression into any other. What is desired is a subclass of the non-neutral transformations that is combinatorially complete, where each individual transformation is nonetheless a semantically small step.

The full set of transformations for Boolean expressions is given in [Loo06]. For numerical expressions, the transcendental functions sin, log, and e^x are used to construct transformations. These obviate the need for division ($a/b = e^{log(a)-log(b)}$), and subtraction ($a - b = a + -1 * b$). For lists, transformations are based on insertion of new leaves (e.g. to append function calls), and "deepening" of the normal form by insertion of subclauses (see [Loo06] for details). For tuples, we take the union of the transformations of all the subtypes. For other mixed-type expressions the union of the non-neutral transformations for all types must be considered as well. For enum types the only transformation is replacing one symbol with another. For function types, the transformations are based on function composition. For action result types, actions are inserted/removed/altered, akin to the treatment of Boolean literals for the Boolean type.

We propose an additional class of non-neutral transformations based on the marvelous *fold* function:

$$fold(f, v, l) = if(empty(l), v, f(first(l), fold(f, v, rest(l))))$$

With *fold* we can express a wide variety of iterative constructs, with guaranteed termination and a bias towards low computational complexity. In fact, fold allows us to represent exactly the primitive recursive functions [Hut99].

Even considering only this reduced space of possible transformations, in many cases there are still too many possible programs "nearby" some target to effectively consider all of them. For example many probabilistic model-building algorithms, such as learning the structure of a Bayesian network from data, can require time cubic in the number of variables (in this context each independent non-neutral transformation can correspond to a variable). Especially as the size of the programs we wish to learn grows, and as the number of typologically matching functions increases, there will be simply too many variables to consider each one intensively, let alone apply a quadratic-time algorithm.

To alleviate this scaling difficulty, we propose three techniques.

The first is to consider each potential variable (i.e. independent non-neutral transformation) to heuristically determine its usefulness in expressing constructive semantic variation. For example, a Boolean transformation that collapses the overall expression into a tautology is assumed to be useless.[8]

The second is heuristic coupling rules that allow us to calculate, for a pair of transformations, the expected utility of applying them in conjunction.

Finally, while *fold* is powerful, it may need to be augmented by other methods in order to provide tractable representation of complex programs that would normally be written using numerous variables with diverse scopes. One approach that we have explored involves application of [SMI97]'s ideas about *director strings as combinators*. In Sinot's approach, special program

[8] This is heuristic because such a transformation might be useful together with other transformations.

EFTA00624209

tree nodes are labeled with director strings, and special algebraic operators interrelate these strings. One then achieves the representational efficiency of local variables with diverse scopes, without needing to do any actual variable management. Reductions and other (non-)neutral transformation rules related to broadening and reducing variable scope may then be defined using the director string algebra.

21.8 Interfacing Between Procedural and Declarative Knowledge

Finally, another critical aspect of procedural knowledge is its interfacing with declarative knowledge. We now discuss the referencing of declarative knowledge within procedures, and the referencing of the details of procedural knowledge within CogPrime's declarative knowledge store.

21.8.1 Programs Manipulating Atoms

Above we have used Combo syntax implicitly, referring to Appendix **??** for the formal definitions. Now we introduce one additional, critical element of Combo syntax: the capability to explicitly reference declarative knowledge within procedures.

For this purpose Combo must contain the following types:

$$Atom, Node, Link, TruthValue, AtomType, AtomTable$$

Atom is the union of Node and Link.

So a type Node within a Combo program refers to a Node in CogPrime's AtomTable. The mechanisms used to evaluate these entities during program evaluation are discussed in Chapter 25.

For example, suppose one wishes to write a Combo program that creates Atoms embodying the predicate-argument relationship $eats(cat, fish)$, represented

```
Evaluation eats (cat, fish)
```

aka

```
Evaluation
   eats
      List
         cat
         fish
```

To do this, one could say for instance,

```
new-link(EvaluationLink
         new-node(PredicateNode ``eats'')
         new-link(ListLink
                  new-node(ConceptNode ``cat'')
                  new-node(ConceptNode ``fish''))
         (new-stv .99 .99))
```

EFTA00624210

21.9 Declarative Representation of Procedures

Next, we consider the representation of program tree internals using declarative data structures. This is important if we want OCP to inferentially *understand* what goes on inside programs. In itself, it is more of a "bookkeeping" issue than a deep conceptual issue, however.

First, note that each of the entities that can live at an internal node of a program, can also live in its own Atom. For example, a number in a program tree corresponds to a NumberNode; an argument in a Combo program already corresponds to some Atom; and an operator in a program can be wrapped up in a SchemaNode all its own, and considered as a one-leaf program tree.

Thus, one can build a kind of virtual, distributed program tree by linking a number of ProcedureNodes (i.e. PredicateNodes or SchemaNodes) together. All one needs in order to achieve this is an analogue of the @ symbol (as defined in Section 20.3 of Chapter 20) for relating ProcedureNodes. This is provided by the ExecutionLink type, where

```
(ExecutionLink f g)
```

essentially means the same as

```
f g
```

in curried notation or

```
   @
  / \
 f   g
```

The same generalized evaluation rules used inside program trees may be thought of in terms of ExecutionLinks; formally, they are crisp ExtensionalImplicationLinks among ExecutionLinks.

Note that we are here using ExecutionLink as a curried function; that is, we are looking at (ExecutionLink f g) as a function that takes an argument x, where the truth value of

```
(ExecutionLink f g) x
```

represents the probability that executing f, on input g, will give output x.

One may then construct combinator expressions linking multiple ExecutionLinks together; these are the analogues of program trees.

For example, using ExecutionLinks, one equivalent of y = x + x^2 is:

```
Hypothetical
    SequentialAND
        ExecutionLink
            pow
            List v1 2
            v2
        ExecutionLink
            +
            List v1 v2
            v3
```

Here the $v1, v2, v3$ are variables which may be internally represented via combinators. This AND is sequential in case the evaluation order inside the program interpreter makes a difference.

As a practical matter, it seems there is no purpose to explicitly storing program trees in conjunction-of-ExecutionLinks form. The information in the ExecutionLink conjunct is already there in the program tree. However, the PLN reasoning system, when reasoning on program trees, may carry out this kind of expansion internally as part of its analytical process.

EFTA00624211

Section II
The Cognitive Cycle

Chapter 22
Emotion, Motivation, Attention and Control

Co-authored with Zhenhua Cai

22.1 Introduction

This chapter begins the heart of the book: the part that explains *how the CogPrime design aims to implement roughly human-like general intelligence, at the human level and ultimately beyond*. First, here in Section II we explain how CogPrime can be used to implement a simplistic animal-like agent without much learning: an agent that perceives, acts and remembers, and chooses actions that it thinks will achieve its goals; but doesn't do any sophisticated learning or reasoning or pattern recognition to help it better perceive, act, remember or figure out how to achieve its goals. We're not claiming CogPrime is the best way to implement such an animal-like agent, though we suggest it's not a bad way and depending on the complexity and nature of the desired behaviors, it *could* be the best way. We have simply chosen to split off the parts of CogPrime needed for animal-like behavior and present them first, prior to presenting the various "knowledge creation" (learning, reasoning and pattern recognition) methods that constitute the more innovative and interesting part of the design.

In Stan Franklin's terms, what we explain here in Section II is how a basic *cognitive cycle* may be achieved within CogPrime. In that sense, the portion of CogPrime explained in this Sectionis somewhat similar to the parts of Stan's LIDA architecture that have currently been worked out in detail, and that . However, while LIDA has not yet been extended in detail (in theory or implementation) to handle advanced learning, cognition and language, those aspects of CogPrime *have* been developed and in fact constitute the largest portion of this book.

Looking back to the integrative diagram from Chapter 5, the cognitive cycle is mainly about integrating vaguely LIDA-like structures and mechanisms with heavily Psi-like structures and mechanisms – but doing so in a way that naturally links in with perception and action mechanisms "below," and more abstract and advanced learning mechanisms "above."

In terms of the general theory of general intelligence, the basic CogPrime cognitive cycle can be seen to have a foundational importance in biasing the CogPrime system toward the problem of controlling an agent in an environment requiring a variety of real-time and near-real-time responses based on a variety of kinds of knowledge. Due to its basis in human and animal cognition, the CogPrime cognitive cycle likely incorporates many useful biases in ways that are not immediately obvious, but that would become apparent if comparing intelligent agents controlled by such a cycle versus intelligent agents controlled via other means.

The cognitive cycle also provides a framework in which other cognitive processes, relating to various aspects of the goals and environments relevant to human-level general intelligence,

EFTA00624214

may conveniently dynamically interoperate. The "Mind OS" aspect of the CogPrime architecture provides general mechanisms in which various cognitive processes may interoperate on a common knowledge store; the cognitive cycle goes further and provides a specific dynamical pattern in which multiple cognitive processes may intersect. Its effective operation places strong demands on the cognitive synergy between the various cognitive processes involved, but also provides a framework that encourages this cognitive synergy to develop and persist.

Finally, it should be stressed that the cognitive cycle is not all-powerful nor wholly pervasive in CogPrime's dynamics. It's critical for the real-time interaction of a CogPrime-controlled agent with a virtual or physical world; but there may be many processes within CogPrime that most naturally operate outside such a cycle. For instance, humans will habitually do deep intellectual thinking (even something so abstract as mathematical theorem proving) within a cognitive cycle somewhat similar to the one they use for practical interaction with the external world. But, there's no reason that CogPrime systems need to be constrained in this way. Deviating from a cognitive cycle based dynamic may cause a CogPrime system to deviate further from human-likeness in its intelligence, but may also help it to perform better than humans on some tasks, e.g. tasks like scientific data analysis or mathematical theorem proving that benefit from styles of information processing that humans aren't particularly good at.

22.2 A Quick Look at Action Selection

We will begin our exposition of CogPrime's cognitive cycle with a quick look at *action selection*. As Stan Franklin likes to point out, the essence of an intelligent *agent* is that it does things; it takes *actions*. The particular mechanisms of action selection in CogPrime are a bit involved and will be given in Chapter 24; in this chapter we will give the basic idea of the action selection mechanism and then explain how a variant of the Psi model (described in Chapter 4 of Part 1 above) is used to handle *motivation* (emotions, drives, goals, etc.) in CogPrime, including the guidance of action selection.

The crux of CogPrime's action selection mechanism is as follows

- the action selector chooses procedures that seem likely to help achieve important goals in the current context
  - *Example:* If the goal is to create a block structure that will surprise Bob, and there is plenty of time, one procedure worth choosing might be a memory search procedure for remembering situations involving Bob and physical structures. Alternately, if there isn't much time, one procedure worth choosing might be a procedure for building the base of a large structure – as this will give something to use as part of whatever structure is eventually created. Another procedure worth choosing might be one that greedily assembles structures from blocks without any particular design in mind.
- to support the action selector, the system builds implications of the form $Context \& Procedure \rightarrow Goal$, where Context is a predicate evaluated based on the agent's situation
  - *Example:* If Bob has asked the agent to do something, and it knows that Bob is very insistent on being obeyed, then implications such as
    - "Bob instructed to do X" and "do X" $\rightarrow$ "please Bob" $< .9, .9 >$

    will be utilized

EFTA00624215

  - *Example:* If the agent wants to make a tower taller, then implications such as
    - "T is a blocks structure" and "place block atop T" → "make T taller" $< .9, .9 >$
    will be utilized

- the truth values of these implications are evaluated based on experience and inference
  - *Example:* The above implication involving Bob could be evaluated based on experience, by assessing it against remembered episodes involving Bob giving instructions
  - *Example:* The same implication could be evaluated based on inference, using analogy to experiences with instructions from other individuals similar to Bob; or using things Bob has explicitly said, combined with knowledge that Bob's self-descriptions tend to be reasonably accurate

- Importance values are propagated between goals using economic attention allocation (and, inference is used to learn subgoals from existing goals)
  - *Example:* If Bob has told the agent to do X, and the agent has then derived (from the goal of pleasing Bob) the goal of doing X, then the "please Bob" goal will direct some of its currency to the "do X" goal (which the latter goal can then pass to its subgoals, or spend on executing procedures)

These various processes are carried out in a manner orchestrated by Dorner's Psi model as refined by Joscha Bach (as reviewed in Chapter 4 above), which supplies (among other features)

- a specific theory regarding what "demands" should be used to spawn the top-level goals
- a set of (four) interrelated system parameters governing overall system state in a useful manner reminiscent of human and animal psychology
- a systematic theory of how various emotions (wholly or partially) emerge from more fundamental underlying phenomena

22.3 Psi in CogPrime

The basic concepts of the Psi approach to motivation, as reviewed in Chapter 4 of Part 1 above, are incorporated in CogPrime as follows (note that the following list includes many concepts that will be elaborated in more detail in later chapters):

- Demands are GroundedPredicateNodes (GPNs), i.e. Nodes that have their truth value computed at each time by some internal C++ code or some Combo procedure in the ProcedureRepository
  - *Examples:* Alertness, perceived novelty, internal novelty, reward from teachers, social stimulus
  - Humans and other animals have familiar demands such as hunger, thirst and excretion; to create an AGI closely emulating a human or (say) a dog one may wish to simulate these in one's AGI system as well

- Urges are also GPNs, with their truth values defined in terms of the truth values of the Nodes for corresponding Demands. However in CogPrime we have chosen the term "Ubergoal" instead of Urge, as this is more evocative of the role that these entities play in the system's dynamics (they are the top-level goals).

EFTA00624216

- Each system comes with a fixed set of Ubergoals (and only very advanced CogPrime systems will be able to modify their Ubergoals)
  - *Example:* Stay alert and alive now and in the future; experience and learn new things now and in the future; get reward from the teachers now and in the future; enjoy rich social interactions with other minds now and in the future
  - A more advanced CogPrime system could have abstract (but experientially grounded) ethical principles among its Ubergoals, e.g. an Ubergoal to promote joy, an Ubergoal to promote growth and an Ubergoal to promote choice, in accordance with the ethics described in [Goe06]
- The ShortTermImportance of an Ubergoal indicates the urgency of the goal, so if the Demand corresponding to an Ubergoal is within its target range, then the Ubergoal will have zero STI. But all Ubergoals can be given maximal LTI to guarantee they don't get deleted.
  - *Example:* If the system is in an environment continually providing an adequate level of novelty (according to its Ubergoal), then the Ubergoal corresponding to external novelty will have low STI but high LTI. The system won't expend resources seeking novelty. But then, if the environment becomes more monotonous, the urgency of the external novelty goal will increase, and its STI will increase correspondingly, and resources will begin getting allocated toward improving the novelty of the stimuli received by the agent.
- Pleasure is a GPN, and its internal truth value computing program compares the satisfaction of Ubergoals to their expected satisfaction
  - Of course, there are various mathematical functions (e.g. $p'th$ power averages[1] for different p) that one can use to average the satisfaction of multiple Ubergoals; and choices here, i.e. different specific ways of calculating Pleasure, could lead to systems with different "personalities"
- Goals are Nodes or Links that are on the system's list of goals (the GoalPool). Ubergoals are automatically Goals, but there will also be many other Goals also
  - *Example:* The Ubergoal of getting reward from teachers might spawn subgoals like "getting reward from Bob" (if Bob is a teacher), or "making teachers smile" or "create surprising new structures" (if the latter often garners teacher reward). The subgoal of "create surprising new structures" might, in the context of a new person entering the agent's environment with a bag of toys, lead to the creation of a subgoal of asking for a new toy of the sort that could be used to help create new structures. Etc.
- Psi's memory is CogPrime's AtomTable, with associated structures like the ProcedureRepository (explained in Chapter 19), the SpaceServer and TimeServer (explained in Chapter 26), etc.
  - *Examples:* The knowledge of what blocks look like and the knowledge that tall structures often fall down, go in the AtomTable; specific procedures for picking up blocks of different shapes go in the ProcedureRepository; the layout of a room or a pile of blocks at a specific point in time go in the SpaceServer; the series of events involved in the building-up of a tower are temporally indexed in the TimeServer.

[1] the $p'th$ power average is defined as $\sqrt[p]{\sum X^p}$

EFTA00624217

  - In Psi and MicroPsi, these same phenomena are stored in memory in a rather different way, yet the basic Psi motivational dynamics are independent of these representational choices

- Psi's "motive selection" process is carried out in CogPrime by economic attention allocation, which allocates ShortTermImportance to Goal nodes
  - *Example:* The flow of importance from "Get reward from teachers" to "get reward from Bob" to "make an interesting structure with blocks" is an instance of what Psi calls "motive selection". No action is being taken yet, but choices are being made regarding what specific goals are going to be used to guide action selection.
- Psi's action selection plays the same role as CogPrime's action selection, with the clarification that in CogPrime this is a matter of selecting which *procedures* (i.e. schema) to run, rather than which individual actions to execute. However, this notion exists in Psi as well, which accounts for "automatized behaviors" that are similar to CogPrime schemata; the only (minor) difference here is that in CogPrime automatized behaviors are the default case.
  - *Example:* If the goal "make an interesting structure with blocks" has a high STI, then it may be used to motivate choice of a procedure to execute, e.g. a procedure that finds an interesting picture or object seen before and approximates it with blocks, or a procedure that randomly constructs something and then filters it based on interestingness. Once a blocks-structure-building procedure is chosen, this procedure may invoke the execution of sub-procedures such as those involved with picking up and positioning particular blocks.
- Psi's planning is carried out via various learning processes in CogPrime, including PLN plus procedure learning methods like MOSES or hillclimbing
  - *Example:* If the agent has decided to build a blocks structure emulating a pyramid (which it saw in a picture), and it knows how to manipulate and position individual blocks, then it must figure out a procedure for carrying out individual-block actions that will result in production of the pyramid. In this case, a very inexperienced agent might use MOSES or hillclimbing and "guidedly-randomly" fiddle with different construction procedures until it hit on something workable. A slightly more experienced agent would use reasoning based on prior structures it had built, to figure out a rational plan (like: "start with the base, then iteratively pile on layers, each one slightly smaller than the previous.")
- The modulators are system parameters which may be represented by PredicateNodes, and which must be incorporated appropriately in the dynamics of various MindAgents, e.g.
  - *activation* affects action selection. For instance this may be effected by a process that, each cycle, causes a certain amount of STICurrency to pass to schema satisfying certain properties (those involving physical action, or terminating rapidly). The amount of currency passed in this way would be proportional to the *activation*
  - *resolution level* affects perception schema and MindAgents, causing them to expend less effort in processing perceptual data
  - *certainty* affects inference and pattern mining and concept creation processes, causing them to place less emphasis on certainty in guiding their activities, i.e. to be more

EFTA00624218

accepting of uncertain conclusions. To give a single illustrative example: When backward chaining inference is being used to find values for variables, a "fitness target" of the form *strength* × *confidence* is sometimes used; this may be replaced with $strength^p \times confidence^{2-p}$, where *activation* parameter affects the exponent p, so when p tends to 0 confidence is more important, when p tends to 2 strength is more important and when p tends to 1 strength and confidence are equally important.

– *selection threshold* may be used to effect a process that, each cycle, causes a certain amount of STICurrency (proportional to the selection threshold) to pass to the Goal Atoms that were wealthiest at the previous cycle.

Based on this run-down, Psi and CogPrime may seem very similar, but that's because we have focused here only on the motivation and emotion aspect. Psi uses a very different knowledge representation than CogPrime; and in the Psi architecture diagram, nearly all of CogPrime is pushed into the role of "background processes that operate in the memory box." According to the theoretical framework underlying CogPrime, the multiple synergetic processes operating in the memory box are actually the crux of general intelligence. But getting the motivation/emotion framework right is also very important, and Psi seems to do an admirable job of that.

22.4 Implementing Emotion Rules atop Psi's Emotional Dynamics

Human motivations are largely determined by human emotions, which are the result of humanity's evolutionary heritage and embodiment, which are quite different than the heritage and embodiment of current AI systems. So, if we want to create AGI systems that lack humanlike bodies, and didn't evolve to adapt to the same environments as humans did, yet still have vaguely human-like emotional and motivational structures, the latter will need to be explicitly engineered or taught in some way.

For instance, if one wants to make a CogPrime agent display anger, something beyond Psi's model of emotion needs to be coded into the agent to enable this. After all, the rule that when angry the agent has some propensity to harm other beings, is not implicit in Psi and needs to be programmed in. However, making use of Psi's emotion model, anger could be characterized as an emotion consisting of high arousal, low resolution, strong motive dominance, few background checks, strong goal-orientedness (as the Psi model suggests) *and* a propensity to cause harm to agents or objects. This is much simpler than specifying a large set of detailed rules characterizing angry behavior.

The "anger" example brings up the point that desirability of giving AGI systems closely humanlike emotional and motivational systems is questionable. After all we humans cause ourselves a lot of problems with these aspects of our mind/brains, and we sometimes put our more ethical and intellectual sides at war with our emotional and motivational systems. Looking into the future, an AGI with greater power than humans yet a humanlike motivational and emotional system, could be a very dangerous thing.

On the other hand, if an AGI's motivational and emotional system is *too different* from human nature, we might have trouble understanding it, and it understanding us. This problem shouldn't be overblown – it seems possible that an AGI with a more orderly and rational motivational system than the human one might be able to understand us *intellectually* very well, and that we might be able to understand it well using our analytical tools. However, if we want to have mutual *empathy* with an AGI system, then its motivational and emotional

EFTA00624219

framework had better have at least *some* reasonable overlap with our own. The value of empathy for ethical behavior was stressed extensively in Chapter 12 of Part 1.

This is an area where experimentation is going to be key. Our initial plan is to supply CogPrime with rough emulations of some but not all human emotions. We see no need to take explicit pains to simulate emotions like anger, jealousy and hatred. On the other hand, joy, curiosity, sadness, wonder, fear and a variety of other human emotions seem both natural in the context of a robotically or virtually embodied CogPrime system, and valuable in terms of allowing mutual human/CogPrime empathy.

22.4.1 Grounding the Logical Structure of Emotions in the Psi Model

To make this point in a systematic way, we point out that Ortony et al's [OCC90] "cognitive theory of emotions" can be grounded in CogPrime's version of Psi in a natural way. This theory captures a wide variety of human and animal emotions in a systematic logical framework, so that grounding their framework in CogPrime Psi goes a long way toward explaining how CogPrime Psi accounts for a broad spectrum of human emotions.

The essential idea of the cognitive theory of emotions can be seen in Figure 22.1. What we see there is that common emotions can be defined in terms of a series of choices:

- Is it positive or negative?
- Is it a response to an agent, an event or an object?
- Is it focused on consequences for oneself, or for another?
  - If on another, is it good or bad for the other?
  - If on oneself, is it related to some event whose outcome is uncertain?
    - if it's related to an uncertain outcome, did the expectation regarding the outcome get fulfilled or not?

Figure 22.1 shows how each set of answers to these questions leads to a different emotion. For instance: what is a negative emotion, responding to events, focusing on another, and undesirable to the other? Pity.

In the list of questions, we see that two of them – positive vs. negative, and expectation fulfillment vs. otherwise – are foundational in the Psi model. The other questions are evaluations that an intelligent agent would naturally make, but aren't bound up with Psi's emotion/motivation infrastructure in such a deep way. Thus, the cognitive theory of emotion emerges as a combination of some basic Psi factors with some more abstract cognitive properties (good vs. bad for another; agents vs. events vs. objects).

22.5 Goals and Contexts

Now we dig deeper into the details of motivation in CogPrime. Just as we have both explicit (local) and implicit (global) memory in CogPrime, we also have both explicit and implicit goals. An explicit goal is formulated as a Goal Atom, and then MindAgents specifically orient the system's activity toward achievement of that goal. An implicit goal is something that the

EFTA00624220

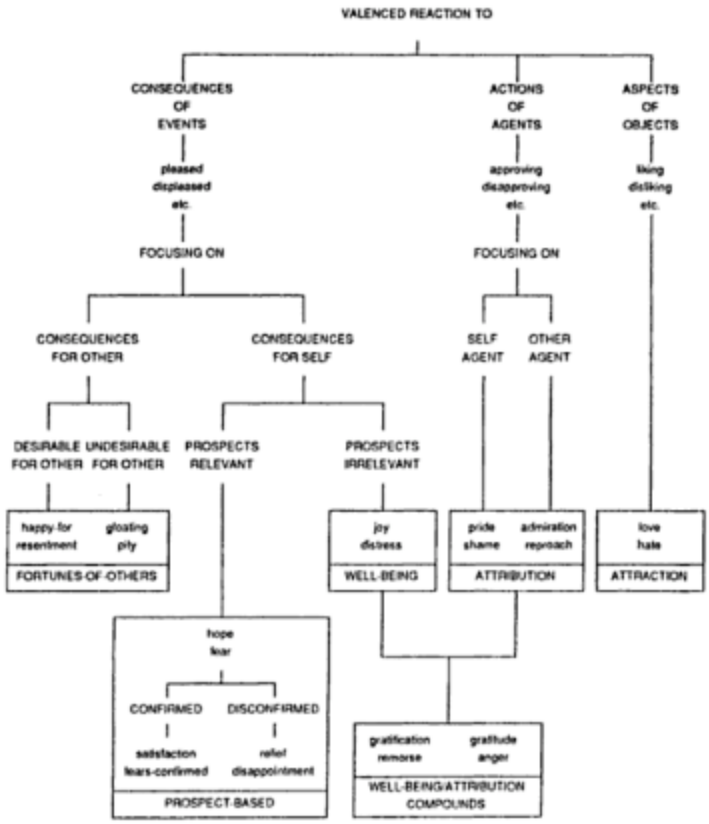


Fig. 22.1: Ontology of Emotions from [OCC90]

system works toward, but in a more loosely organized way, and without necessarily explicitly representing the knowledge that it is working toward that goal.

Here we will focus mainly on explicit motivation, beginning with a description of Goal Atoms, and the Contexts in which Goals are worked toward via executing Procedures. Figure 22.2 gives a rough depiction of the relationship between goals, procedures and context, in a simple example relevant to an OpenCogPrime-controlled virtual agent in a game world.

22.5.1 Goal Atoms

A Goal Atom represents a *target system state* and is true to the extent that the system satisfied the conditions it represents. A Context Atom represents an *observed state of the world/mind,* and is true to the extent that the state it defines is observed. Taken together, these two Atom types provide the infrastructure CogPrime needs to orient its actions in specific contexts toward specific goals. Not all of CogPrime's activity is guided by these Atoms; much of it is non-goal-directed and spontaneous, or *ambient* as we sometimes call it. But it is important that some

EFTA00624221

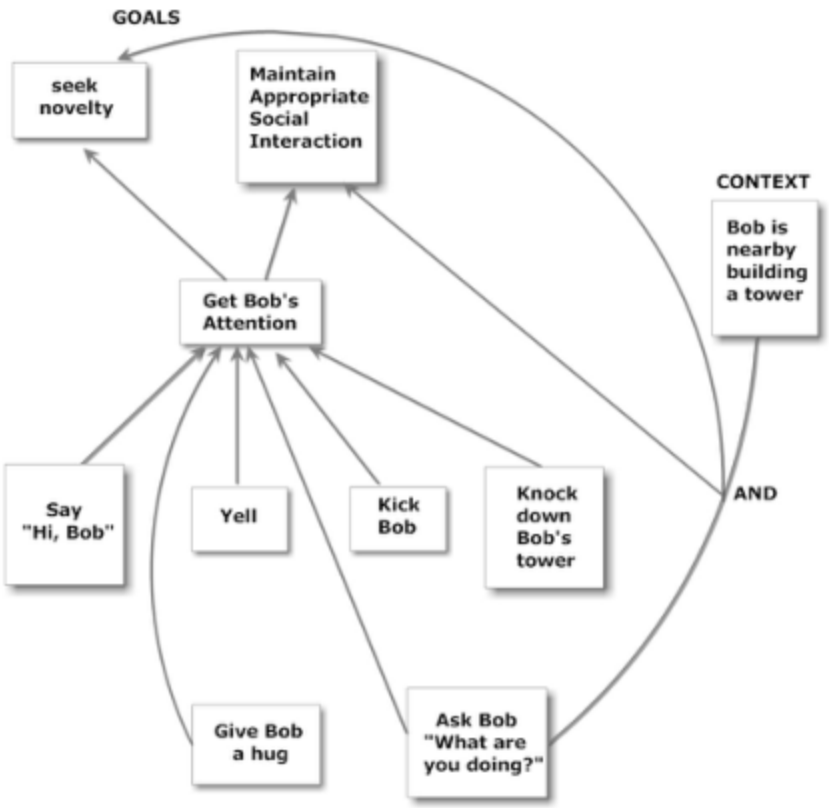


Fig. 22.2: **Context, Procedures and Goals**. Examples of the basic "goal/context/procedure" triad in a simple game-agent situation.

of the system's activity - and in some cases, a substantial portion - is controlled explicitly via goals.

Specifically, a Goal Atom is simply an Atom (usually a PredicateNode, sometimes a Link, and potentially another type of Atom) that has been selected by the GoalRefinement MindAgent as one that represents a state of the atom space which the system finds important to achieve. The extent to which an Atom is considered a Goal Atom at a particular point in time is determined by how much of a certain kind of financial instrument called an RFS (Request For Service) it possesses (as will be explained in Chapter 24).

A CogPrime instance must begin with some initial *Ubergoals* (aka *top level supergoals*), but may then refine these goals in various ways using inference. Immature, "childlike" CogPrime systems cannot modify their Ubergoals nor add nor delete Ubergoals. Advanced CogPrime systems may be allowed to modify, add or delete Ubergoals, but this is a critical and subtle aspect of system dynamics that must be treated with great care. WIKISOURCE:ContextAtom

EFTA00624222

22.6 Context Atoms

Next, a Context is simply an Atom that is used as the source of a ContextLink, for instance

```
Context
    quantum_computing
    Inheritance Ben amateur
```

or

```
Context
    game_of_fetch
    PredictiveAttraction
        Evaluation give (ball, teacher)
        Satisfaction
```

The former simply says that Ben is an amateur in the context of quantum computing. The latter says that in the context of the game of fetch, giving the ball to the teacher implies satisfaction. A more complex instance pertinent to our running example would be

```
Context
   Evaluation
      Recently
      List
         Minute
         Evaluation
            Ask
            List
               Bob
               ThereExists $X
                  And
                     Evaluation
                        Build
                        List
                           self
                           $X
                     Evaluation
                        surprise
                        List
                           $X
                           Bob
   AverageQuantifier $Y
      PredictiveAttraction
         And
            Evaluation
               Build
               List
                  self
                  $Y
            Evaluation
               surprise
               List
                  $Y
                  Jim
         Satisfaction
```

EFTA00624223

which says that, if the context is that Bob has recently asked for something surprising to be built, then one strategy for getting satisfaction is to build something that seems likely to satisfy Jim.

An implementation-level note: in the current OpenCogPrime implementation of CogPrime, ContextLinks are implicit rather than explicit entities. An Atom can contain a ComplexTruthValue which in turn contains a number of VersionHandles. Each VersionHandle associates a Context or a Hypothetical with a TruthValue. This accomplishes the same thing as a formal ContextLink, but without the creation of a ContextLink object. However, we continue to use ContextLinks in this book and other documents about CogPrime; and it's quite possible that future CogPrime implementations might handle them differently.

22.7 Ubergoal Dynamics

In the early phases of a CogPrime system's cognitive development, the goal system dynamics will be quite simple. The Ubergoals are supplied by human programmers, and the system's adaptive cognition is used to derive subgoals. Attentional currency allocated to the Ubergoals is then passed along to the subgoals, as judged appropriate.

As the system becomes more advanced, however, more interesting phenomena may arise regarding Ubergoals: implicit and explicit Ubergoal creation.

22.7.1 Implicit Ubergoal Pool Modification

First of all, *implicit Ubergoal creation or destruction* may occur. Implicit Ubergoal destruction may occur when there are multiple Ubergoals in the system, and some prove easier to achieve than others. The system may then decide not to bother achieving the more difficult Ubergoals. Appropriate parameter settings may mitigate against this phenomenon, of course.

Implicit Ubergoal creation may occur if some Goal Node G arises that inherits as a subgoal from multiple Ubergoals. This Goal G may then come to act implicitly as an Ubergoal, in that it may get more attentional currency than any of the Ubergoals.

Also, implicit Ubergoal creation may occur via forgetting. Suppose that G becomes a goal via inferred inheritance from one or more Ubergoals. Then, suppose G forgets *why* this inheritance exists, and that in fact the reason becomes obsolete, but the system doesn't realize that and keeps the inheritance there. Then, G is an implicit Ubergoal in a strong sense: it gobbles up a lot of attentional currency, potentially more than any of the actual Ubergoals, but actually doesn't help achieve the Ubergoals, even though the system thinks it does. This kind of dynamic is obviously very bad and should be avoided – and *can* be avoided with appropriate tuning of system parameters (so that the system pays a lot of attention to making sure that its subgoaling-related inferences are correct and are updated in a timely way).

EFTA00624224

22.7.2 Explicit Ubergoal Pool Modification

An advanced CogPrime system may be given the ability to explicitly modify its Ubergoal pool. This is a very interesting but very subtle type of dynamic, which is not currently well understood and which potentially could lead to dramatically unpredictable behaviors.

However, modification, creation and deletion of goals is a key aspect of human psychology, and the granting of this capability to mature CogPrime systems must be seriously considered.

In the case that Ubergoal pool modification is allowed, one useful heuristic may be to make *implicit Ubergoals* into explicit Ubergoals. For instance: if an Atom is found to consistently receive a lot of RFSs, and has a long time-scale associated with it, then the system should consider making it an Ubergoal. But this heuristic is certainly not sufficient, and any advanced CogPrime system that is going to modify its own Ubergoals should definitely be tuned to put a lot of thought into the process!

The science of Ubergoal pool dynamics basically does not exist at the moment, and one would like to have some nice mathematical models of the process prior to experimenting with it in any intelligent capable CogPrime system. Although Schmiddhuber's Gödel machine [Sch06] has the theoretical capability to modify its ubergoal (note that CogPrime is, in some way, a Gödel machine), there is currently no mathematics allowing us to assess the time and space complexity of such process in a realistic context, given a certain safety confidence target.

22.8 Goal Formation

Goal formation in CogPrime is done via PLN inference. In general, what PLN does for goal formation is to look for predicates that can be proved to probabilistically imply the existing goals. These new predicates will then tend to receive RFS currency, according to the logic of RFS's to be outlined in Chapter 24, which (according to goal-driven attention allocation dynamics) will make the system more likely to enact procedures that lead to their satisfaction.

As an example of the goal formation process, consider the case where ExternalNovelty is an Ubergoal. The agent may then learn that whenever Bob gives it a picture to look at, its quest for external novelty is satisfied to a singificant degree. That is, it learns

```
Attraction
   Evaluation give (Bob, me, picture)
   ExternalNovelty
```

where *Attraction A B* measures how much A versus $\neg A$ implies B (as explained in Chapter 34). This information allows the agent (the Goal Formation MindAgent) to nominate the atom:

```
EvaluationLink give (Bob, me, picture)
```

as a goal (a subgoal of the original Ubergoal). This is an example of goal refinement, which is one among many ways that PLN can create new goals from existing ones.

EFTA00624225

22.9 Goal Fulfillment and Predicate Schematization

When there is a Goal Atom G important in the system (with a lot of RFS), the GoalFulfillment MindAgent seeks SchemaNodes S that it has reason to believe, if enacted, will cause G to become true (satisfied). It then adds these to the ActiveSchemaPool, an object to be discussed below. The dynamics by which the GoalFulfillment process works will be discussed in Chapter 24 below.

For example, if a Context Node Chas a high truth value at that time (because it is currently satisfied), and is involved in a relation:

```
Attraction
   C
   PredictiveAttraction S G
```

(for some SchemaNode S and Goal Node G) then this SchemaNode S is likely to be selected by the GoalFulfillment process for execution. This is the fully formalized version of the $Context\&Schema \rightarrow Goal$ notion discussed frequently above. The process may also allow the importance of various schema S to bias its choices of which schemata to execute.

For instance, following up previous examples, we might have

```
Attraction
   Evaluation
      near
         List
            self
            Bob
   PredictiveAttraction
      Evaluation
         ask
         List
            Bob
            ``Show me a picture''
      ExternalNovelty
```

Of course this is a very simplistic relationship but it's similar to a behavior a young child might display. A more advanced agent would utilize a more abstract relationship that distinguishes various situations in which Bob is nearby, and also involves expressing a concept rather than a particular sentence.

The formation of these schema-context-goal triads may occur according to generic inference mechanisms. However, a specially-focused PredicateSchematization MindAgent is very useful here as a mechanism of inference control, increasing the number of such relations that will exist in the system.

22.10 Context Formation

New contexts are formed by a combination of processes:

- The MapEncapsulation MindAgent, which creates Context Nodes embodying repeated patterns in the perceived world. This process encompasses
  - Maps creating Context Nodes involving Atoms that have high STI at the same time

EFTA00624226

    - *Example:* A large number of Atoms related to towers could be joined into a single map, which would then be a ConceptNode pointing to "tower-related ideas, procedures and experiences"
  - Maps creating Context Nodes that are involved in a temporal activation pattern that recurs at multiple points in the system's experience.
    - *Example:* There may be a common set of processes involving creating a building out of blocks: first build the base, then the walls, then the roof. This could be encapsulated as a temporal map embodying the overall nature of the process. In this case, the map contains information of the nature: *first do things related to this, then do things related to this, then do things related to this...*

- A set of concept creation MindAgents (see Chapter 38, which fuse and split Context Nodes to create new ones.
  - The concept of a building and the concept of a person can be merged to create the concept of a BuildingMan
  - The concept of a truck built with Legos can be subdivided into trucks you can actually carry Lego blocks with, versus trucks that are "just for show" and can't really be loaded with objects and then carry them around

22.11 Execution Management

The GoalFulfillment MindAgent chooses schemata that are found likely to achieve current goals, but it doesn't actually execute these schemata. What it does is to take these schemata and place them in a container called the ActiveSchemaPool.

The ActiveSchemaPool contains a set of schemata that have been determined to be reasonably likely, if enacted, to significantly help with achieving the current goal-set. I.e., everything in the active schema pool should be a schema S so that it has been concluded that

```
Attraction
   C
   PredictiveAttraction S G
```

– where C is a currently applicable context and G is one of the goals in the current goal pool – has a high truth value compared to what could be obtained from other known schemata S or other schemata S that could be reasonably expected to be found via reasoning.

The decision of which schemata in the ActiveSchemaPool to enact is made by an object called the ExecutionManager, which is invoked each time the SchemaActivation MindAgent is executed. The ExecutionManager is used to select which schemata to execute, based on doing reasoning and consulting memory regarding which active schemata can usefully be executed simultaneously without causing *destructive interference* (and hopefully causing constructive interference). This process will also sometimes (indirectly) cause new schemata to be created and/or other schemata from the AtomTable to be made active. This process is described more fully in Chapter 24 on action selection. WIKISOURCE:GoalsAndTime

For instance, if the agent is involved in building a blocks structure intended to surprise or please Bob, then it might simultaneously carry out some blocks-manipulation schema, and also a schema involving looking at Bob to garner his approval. If it can do the blocks manipulation without constantly looking at the blocks, this should be unproblematic for the agent.

EFTA00624227

22.12 Goals and Time

The CogPrime system maintains an explicit list of "Ubergoals", which as will be explained in Chapter 24, receive attentional currency which they may then allocate to their subgoals according to a particular mechanism.

However, there is one subtle factor involved in the definition of the Ubergoals: time. The truth value of a Ubergoal is typically defined as the average level of satisfaction of some Demand over some period of time – but the time scale of this averaging can be very important. In many cases, it may be worthwhile to have separate Ubergoals corresponding to the same Demand but doing their truth-value time-averaging over different time scales. For instance, corresponding to Demands such as Novelty or Health, we may posit both long-term and short-term versions, leading to Ubergoals such as CurrentNovelty, LongTermNovelty, CurrentHealth, LongTermHealth, etc. Of course, one could also wrap multiple Ubergoals corresponding to a single Demand into a single Ubergoal combining estimates over multiple time scales; this is not a critical issue and the only point of splitting Demands into multiple Ubergoals is that it can make things slightly simpler for other cognitive processes.

For instance, if the agent has a goal of pleasing Bob, and it knows Bob likes to be presented with surprising structures and ideas, then the agent has some tricky choices to make. Among other choices it must balance between focusing on

- creating things and then showing them to Bob
- studying basic knowledge and improving its skills.

Perhaps studying basic knowledge and skills will give it a foundation to surprise Bob much more dramatically in the mid-term future ... but in the short run will not allow it to surprise Bob much at all, because Bob already knows all the basic material. This is essentially a variant of the general "exploration versus exploitation" dichotomy, which lacks any easy solution. Young children are typically poor at carrying out this kind of balancing act, and tend to focus overly much on near-term satisfaction. There are also significant cultural differences in the heuristics with which adult humans face these issues; e.g. in some contexts Oriental cultures tend to focus more on mid to long term satisfaction whereas Western cultures are more short term oriented.

EFTA00624228

Chapter 23
Attention Allocation

Co-authored with Joel Pitt and Matt Ikle' and Rui Liu

23.1 Introduction

The critical factor shaping real-world general intelligence is resource constraint. Without this issue, we could just have simplistic program-space-search algorithms like AIXI^{tl} instead of complicated systems like the human brain or CogPrime. Resource constraint is managed implicitly within various components of CogPrime, for instance in the population size used in evolutionary learning algorithms, and the depth of forward or backward chaining inference trees in PLN. But there is also a component of CogPrime that manages resources on a global and cognitive-process-independent manner: the *attention allocation* component.

The general principles the attention allocation process should follow are easy enough to see: History should be used as a guide, and an intelligence should make probabilistic judgments based on its experience, guessing which resource-allocation decisions are likely to maximize its goal-achievement. The problem is that this is a difficult learning and inference problem, and to carry it out with excellent accuracy would require a limited-resources intelligent system to spend nearly all its resources deciding what to pay attention to and nearly none of them actually paying attention to anything else. Clearly this would be a very poor allocation of an AI system's attention! So simple heuristics are called for, to be supplemented by more advanced and expensive procedures on those occasions where time is available and correct decisions are particularly crucial.

Attention allocation plays, to a large extent, a "meta" role in enabling mind-world correspondence. Without effective attention allocation, the other cognitive processes can't do their jobs of helping an intelligent agent to achieve its goals in an environment, because they won't be able to pay attention to the most important parts of the environment, and won't get computational resources at the times when they need it. Of course this need could be addressed in multiple different ways. For example, in a system with multiple complex cognitive processes, one could have attention allocation handled separately within each cognitive process, and then a simple "top layer" of attention allocation managing the resources allocated to each cognitive process. On the other hand, one could also do attention allocation via a single dynamic, pervasive both within and between individual cognitive processes. The CogPrime design gravitates more toward the latter approach, though also with some specific mechanisms within various MindAgents; and efforts have been made to have these specific mechanisms modulated by the generic attention allocation structures and dynamics wherever possible.

EFTA00624230

In this chapter we will dig into the specifics of how these *attention allocation* issues are addressed in the CogPrime design. In short, they are addressed via a set of mechanisms and equations for dynamically adjusting *importance values* attached to Atoms and MindAgents. Different importance values exist pertinent to different time scales, most critically the short-term (STI) and long-term (LTI) importances. The use of two separate time-scales here reflects fundamental aspects of human-like general intelligence and real-world computational constraints.

The dynamics of STI is oriented partly toward the need for real-time responsiveness, and the more thoroughgoing need for cognitive processes at speeds vaguely resembling the speed of "real time" social interaction. The dynamics of LTI is based on the fact that some data tends to be useful over long periods of time, years or decades in the case of human life, but the practical capability to store large amounts of data in a rapidly accessible way is limited. One could imagine environments in which very-long-term multiple-type memory was less critical than it is in typical human-friendly environments; and one could envision AGI systems carrying out tasks in which real-time responsiveness was unnecessary (though even then some attention focusing would certainly be necessary). For AGI systems like these, an attention allocation system based on STI and LTI with CogPrime-like equations would likely be inappropriate. But for an AGI system intended to control a vaguely human-like agent in an environment vaguely resembling everyday human environments, the focus on STI and LTI values, and the dynamics proposed for these values in CogPrime, appear to make sense.

Two basic innovations are involved in the mechanisms attached to these STI and LTI importance values:

- treating attention allocation as a data mining problem: the system records information about what it's done in the past and what goals it's achieved in the past, and then recognizes patterns in this history and uses them to guide its future actions via probabilistically adjusting the (often context-specific) *importance values* associated with internal terms, actors and relationships, and adjusting the "effort estimates" associated with Tasks
- using an artificial-economics approach to update the importance values (attached to Atoms, MindAgents, and other actors in the CogPrime system) that regulate system attention. (And, more speculatively, using an information geometry based approach to execute the optimization involved in the artificial economics approach efficiently and accurately.)

The integration of these two aspects is crucial. The former aspect provides fundamental data about what's of value to the system, and the latter aspect allows this fundamental data to be leveraged to make sophisticated and integrative judgments rapidly. The need for the latter, rapid-updating aspect exists partly because of the need for real-time responsiveness, imposed by the need to control a body in a rapidly dynamic world, and the prominence in the architecture of an animal-like cognitive cycle. The need for the former, data-mining aspect (or something functionally equivalent) exists because, in the context of the tasks involved in human-level general intelligence, the assignment of credit problem is hard – the relations between various entities in the mind and the mind's goals are complex, and identifying and deploying these relationships is a difficult learning problem requiring application of sophisticated intelligence.

Both of these aspects of attention allocation dynamics may be used in computationally lightweight or computationally sophisticated manners:

- For routine use in real-time activity

EFTA00624231

  - "data mining" consists of forming HebbianLinks (involved in the associative memory and inference control, see Section 23.5), where the weight of the link from Atom A to Atom B is based on the probability of shared utility of A and B
  - economic attention allocation consists of spreading ShortTermImportance and LongTermImportance "artificial currency" values (both grounded in the universal underlying "juju" currency value defined further below) between Atoms according to specific equations that somewhat resemble neural net activation equations but respect the conservation of currency

- For use in cases where large amounts of computational resources are at stake based on localized decisions, hence allocation of substantial resources to specific instances of attention-allocation is warranted
  - "data mining" may be more sophisticated, including use of PLN, MOSES and pattern mining to recognize patterns regarding what probably deserves more attention in what contexts
  - economic attention allocation may involve more sophisticated economic calculations involving the expected future values of various "expenditures" of resources

The particular sort of "data mining" going on here is definitely not exactly what the human brain does, but we believe this is a case where slavish adherence to neuroscience would be badly suboptimal (even if the relevant neuroscience were well known, which is not the case). Doing attention allocation entirely in a distributed, formal-neural-net-like way is, we believe, extremely and unnecessarily inefficient, and given realistic resource constraints it leads to the rather poor attention allocation that we experience every day in our ordinary waking state of consciousness. Several aspects of attention allocation can be fruitfully done in a distributed, neural-net-like way, but not having a logically centralized repository of system-history information (regardless of whether it's physically distributed or not) seems intrinsically problematic in terms of effective attention allocation. And we argue that, even for those aspects of attention allocation that are best addressed in terms of distributed, vaguely neural-net-like dynamics, an artificial-economics approach has significant advantages over a more strictly neural-net-like approach, due to the greater ease of integration with other cognitive mechanisms such as forgetting and data mining.

23.2 Semantics of Short and Long Term Importance

We now specify the two types of importance value (short and long term) that play a key role in CogPrime dynamics. Conceptually, ShortTermImportance (STI) is defined as

$$STI(A) = P(A \text{ will be useful in the near future})$$

whereas LongTermImportance (LTI) is defined as

$$LTI(A) = P(A \text{ will be useful eventually, in the foreseeable future})$$

Given a time-scale T, in general we can define an importance value relative to T as

$$I_T(A) = P(A \text{ will be useful during the next } T \text{ seconds})$$

EFTA00624232

In the ECAN module in CogPrime, we deal only with STI and LTI rather than any other importance values, and the dynamics of STI and LTI are dealt with by treating them as two separate "artificial currency" values, which however are interconvertible via being mutually grounded in a common currency called "juju."

For instance, if the agent is intensively concerned with trying to build interesting blocks structures, then knowledge about interpreting biology research paper abstracts is likely to be of very little current importance. So its biological knowledge will get low STI, but – assuming the agent expects to use biology again – it should maintain reasonably high LTI so it can remain in memory for future use. And if in its brainstorming about what blocks structures to build, the system decides to use some biological diagrams as inspiration, STI can always spread to some of the biology-related Atoms, increasing their relevance and getting them more attention. While the attention allocation system contains mechanisms to convert STI to LTI, it also has parameter settings biasing it to spend its juju on both kinds of importance – i.e. it contains an innate bias to both focus its attention judiciously, and manage its long-term memory conscientiously.

Because in CogPrime most computations involving STI and LTI are required to be very rapid (as they're done for many Atoms in the memory very frequently), in most cases when dealing with these quantities, it will be appropriate to sacrifice accuracy for efficiency. On the other hand, it's useful to *occasionally* be able to carry out expensive, highly accurate computations involving importance.

An example where doing expensive computations about attention allocation might pay off, would be the decision whether to use biology-related or engineering-related metaphors in creating blocks structures to please a certain person. In this case it could be worth doing a few steps of inference to figure out whether there's a greater intensional similarity between that person's interests and biology or engineering; and then using the results to adjust the STI levels of whichever of the two comes out most similar. This would not be a particularly expensive inference to carry out, but it's still much more effort than what can be expended on Atoms in the memory most of the time. Most attention allocation in CogPrime involves simple neural-net type spreading dynamics rather than explicit reasoning.

Figure 23.1 illustrates the key role of LTI in the forgetting process. Figure 23.2 illustrates the key role of STI in maintaining a "moving bubble of attention", which we call the system's AttentionalFocus.

23.2.1 The Precise Semantics of STI and LTI

Now we precisiate the above definitions of STI and LTI.

First, we introduce the notion of *reward*. Reward is something that Goals give to Atoms. In principle a Goal might give an Atom reward in various different forms, though in the design given here, reward will be given in units of a currency called *juju*. The process by which Goals assign reward to Atoms is part of the "assignment of credit" process (and we will later discuss the various time-scales on which assignment of credit may occur and their relationship to the time-scale parameter within LTI).

Next, we define

$J(A, t_1, t_2, r)$ = expected amount of reward A will receive between t_1 and t_2 time-steps in the future, if its STI has percentile rank r among all Atoms in the AtomTable

EFTA00624233

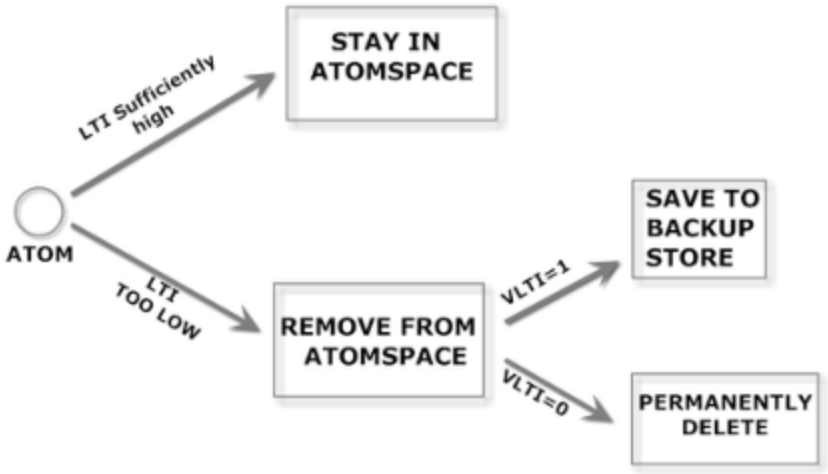


Fig. 23.1: **LongTermImportance and Forgetting**.

The percentile rank r of an Atom is the rank of that Atom in a list of Atoms ordered by decreasing STI, divided by the total number of Atoms. The reason for using a percentile rank instead of the STI itself is because at any given time only a limited number of atoms can be given attention, so all atoms below a certain perceptible rank, depending on the amount of available resource, will simply be ignored.

This is a fine-grained measure of how worthwhile it is expected to be to increase A's STI, in terms of getting A rewarded by Goals.

For practical purposes it is useful to collapse $J(A, t_1, t_2, r)$ to a single number:

$$J(A, t_1, t_2) = \frac{\sum_r J(A, t_1, t_2, r) w_r}{\sum_r w_r}$$

where w_r weights the different percentile ranks (and should be chosen to be monotone increasing in r). This is a single-number measure of the responsiveness of an Atom's utility to its STI level. So for instance if A has a lot of STI and it turns out to be rewarded then J(A, t1, t2) will be high. On the other hand if A has little STI then whether it gets rewarded or not will not influence J(A, t1, t2) much.

To simplify notation, it's also useful to define a single-time-point version

$$J(A, t) = J(A, t, t)$$

23.2.1.1 Formalizing STI

Using these definitions, one simple way to make the STI definition precise is:

$$STI_{thresh}(A, t) = P(J(A, t, t + t_{short}) \geq s_{threshold})$$

where $s_{threshold}$ demarcates the "attentional focus boundary." Which is a way of saying that we don't want to give STI to atoms that would not get rewarded if they were given attention.

EFTA00624234

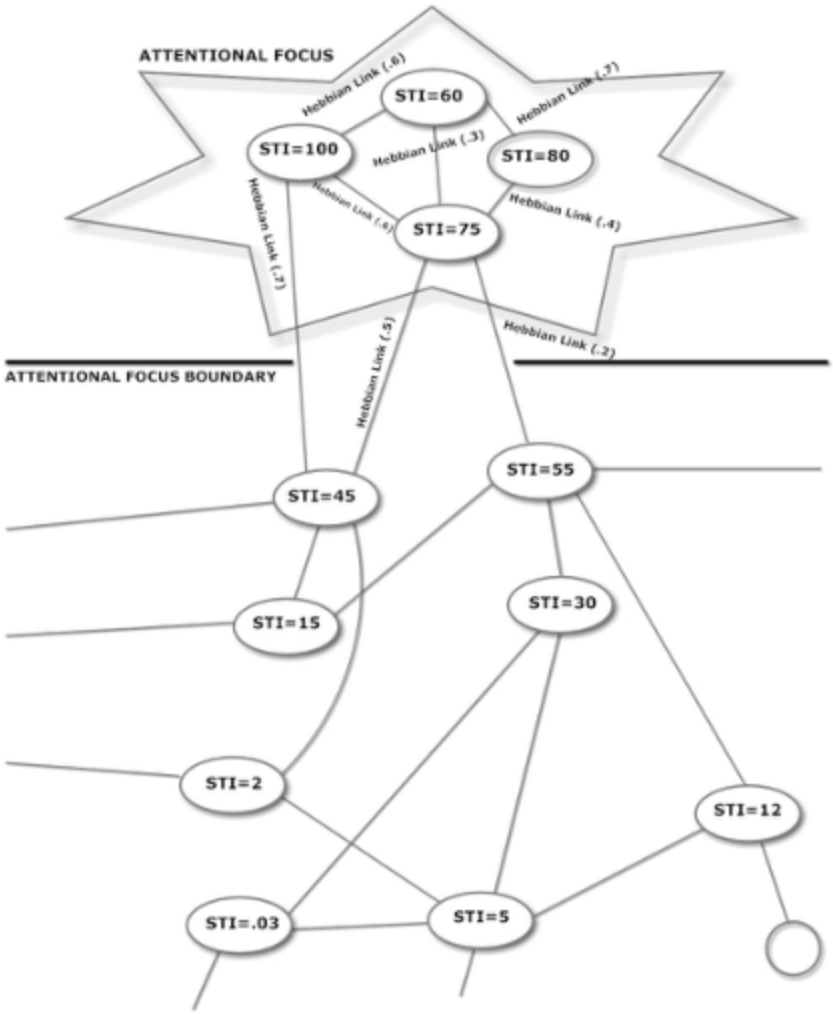


Fig. 23.2: **Formation of the AttentionalFocus**. The dynamics of STI is configured to encourage the emergence of richly cross-connected networks of Atoms with high STI (above a threshold called the AttentionalFocusBoundary), passing STI among each other as long as this is useful and forming new HebbianLinks among each other. The collection of these Atoms is called the AttentionalFocus.

Or one could make the STI definition precise in a fuzzier way, and define

$$STI_{fuzzy}(A,t) = \sum_{s=0}^{\infty} J(A,t+s)c^{-s}$$

for some appropriate parameter c (or something similar with a decay function less severe than exponential)

EFTA00624235

In either case, the goal of the ECAN subsystem, regarding STI, is to assign each Atom A an STI value that corresponds as closely as possible to the theoretical STI values defined by whichever one of the above equations is selected (or some other similar equation).

23.2.2 STI, STIFund, and Juju

But how can one estimate these probabilities in practice? In some cases they may be estimated via explicit inference. But often they must be estimated by heuristics.

The estimative approach taken in current CogPrime design is an artificial economy, in which each Atom maintains a certain fund of artificial currency. In the current proposal this currency is called juju and is the same currency used to value LTI. Let us call the amount of juju owned by Atom A the STIFund of A. Then, one way to formalize the goal of the artificial economy is to state that: if one ranks all Atoms by the wealth of their STIFund, and separately ranks all Atoms by their theoretical STI value, the rankings should be as close as possible to the same. One may also formalize the goal in terms of value correlation instead of rank correlation, of course.

Proving conditions under which the STIFund values will actually correlate well with the theoretical STI values, is an open math problem. Heuristically, one may map STIFund values into theoretical STI values by a mapping such as

$$A.STI = \alpha + \beta \frac{A.STIFund - STIFund.\min}{STIFund.\max - STIFund.\min}$$

where $STIFund.\min = \min_X X.STIFund$. However, we don't currently have rigorous grounding for any particular functional form for such a mapping; the above is just a heuristic approximation.

The artificial economy approach leads to a variety of supporting heuristics. For instance, one such heuristic is: if A has been used at time t, then it will probably be useful at time $t + s$ for small s. Based on this heuristic, whenever a MindAgent uses an Atom A, it may wish to increase A's STIFund (so as to hopefully increase correlation of A's STIFund with its theoretical STI). It does so by transferring some of its juju to A's STIFund.

23.2.3 Formalizing LTI

Similarly to STI, with LTI we will define theoretical LTI values, and posit an LTIFund associated with each Atom, which seeks to create values correlated with the theoretical LTI values.

For LTI, the theoretical issues are subtler. There is a variety of different ways to precisiate the above loose conceptual definition of LTI. For instance, one can (and we will below) create formalizations of both:

1. $LTI_{cont}(A)$ = (some time-weighting or normalization of) the expected value of A's total usefulness over the long-term future
2. $LTI_{burst}(A)$ = the probability that A ever becomes *highly useful* at some point in the long-term future

EFTA00624236

(here "cont" stands for "continuous"). Each of these may be formalized, in similar but nonidentical ways.

These two forms of LTI may be viewed as extremes along a continuum; one could posit a host of intermediary LTI values between them. For instance, one could define

$LTI_p(A)$ = the $p'th$ power average[1] of expectation of the utility of A over brief time intervals, measured over the long-term future

Then we would have

$$LTI_{burst} = LTI_\infty$$

$$LTI_{cont} = LTI_1$$

and could vary p to vary the sharpness of the LTI computation. This might be useful in some contexts, but our guess is that it's overkill in practice and that looking at LTI_{burst} and LTI_{cont} is enough (or more than enough; the current OCP code uses only one LTI value and that has not been problematic so far).

23.2.4 Applications of LTI_{burst} versus LTI_{cont}

It seems that the two forms of LTI discussed above might be of interest in different contexts, depending on the different ways that Atoms may be used so as to achieve reward.

If an Atom is expected to get rewarded for the results of its being selected by MindAgents that carry out diffuse, background thinking (and hence often select low-STI Atoms from the AtomTable), then it may be best associated with LTI_{cont}.

On the other hand, if an Atom is expected to get rewarded for the results of its being selected by MindAgents that are focused on intensive foreground thinking (and hence generally only select Atoms with very high STI), it may be best associated with LTI_{burst}.

In principle, Atoms could be associated with particular LTI_p based on the particulars of the selection mechanisms of the MindAgents expected to lead to their reward. But the issue with this is, it would result in Atoms carrying around an excessive abundance of different LTI_p values for various p, resulting in memory bloat; and it would also require complicated analyses of MindAgent dynamics. If we do need more than one LTI value, one would hope that two will be enough, for memory conservation reasons.

And of course, if an Atom has only one LTI value associated with it, this can reasonably be taken to stand in for the other one: either of LTI_{burst} or LTI_{cont} may, in the absence of information to the contrary, be taken as an estimate of the other.

23.2.4.1 LTI with Various Time Lags

The issue of the p value in the average in the definition of LTI is somewhat similar to (though orthogonal to) the point that there are many different interpretations of LTI, achieved via

[1] the $p'th$ power average is defined as $\sqrt[p]{\sum X^p}$

EFTA00624237

considering various time-lags. Our guess is that a small set of time-lags will be sufficient. Perhaps one wants an exponentially increasing series of time-lags: i.e. to calculate LTI over k cycles where k is drawn from $\{r, 2r, 4r, 8r, ...2^N r\}$.

The time-lag in LTI seems related to the time-lag in the system's goals. If a Goal object is disseminating juju, and the Goal has an intrinsic time scale of t, then it may be interested in LTI on time-scale t. So when a MA (MindAgent) is acting in pursuit of that goal, it should spend a bunch of its juju on LTI on time-scale t.

Complex goals may be interested in multiple time-scales (for instance, a goal might place greater value on things that occur in the next hour, but still have nonzero interest in things that occur in a week), and hence may have different levels of interest in LTI on multiple time-scales.

23.2.4.2 Formalizing Burst LTI

Regarding burst LTI, two approaches to formalization seem to be the threshold version

$$LTI_{burst,thresh}(A) = P(A \text{ will receive a total of at least } s_{threshold} \text{ amount of normalized stimulus during some time interval of length } t_{short} \text{ in the next } t_{long} \text{ time steps})$$

and the fuzzy version,

$$LTI_{burst,fuzzy}(A,t) = \sum_{s=0}^{\infty} J(A, t+s, t+s+t_{short}) f(s, t_{long})$$

where $f(t, t_{long}) : R^+ \times R^+ \to R^+$ is a nonincreasing function that remains roughly constant in t up till a point t_{long} steps in the future, and then begins slowly decaying.

23.2.4.3 Formalizing Continuous LTI

The threshold version of continuous LTI is quite simply:

$$LTI_{cont,thresh}(A, t_{long}) = STI_{thresh}(A, t_{long})$$

That is, smooth threshold LTI is just like smooth threshold STI, but the time-scale involved is longer.

On the other hand, the fuzzy version of smooth LTI is:

$$LTI_{cont,fuzzy}(A,t) = \sum_{s=0}^{\infty} J(A, t+s) f(s, t_{long})$$

using the same decay function f that was introduced above in the context of burst LTI.

23.3 Defining Burst LTI in Terms of STI

It is straightforward to define burst LTI in terms of STI, rather than directly in terms of juju. We have

EFTA00624238

$$LTI_{burst,thresh}(A,t) = P(\bigcup_{s=0}^{s=t_{long}} STI_{thresh}(A,t+s))$$

Or, using the fuzzy definitions, we obtain instead the *approximate* equation

$$LTI_{burst,fuzzy}(A,t) \approx \sum_{s=0}^{\infty} \alpha(s) STI_{fuzzy}(A,t+s) f(s,t_{long})$$

where

$$\alpha(s) = \frac{1-c}{1-c^{s+1}}$$

or the more complex exact equation:

$$LTI_{burst,fuzzy}(A,t) = \sum_{s=0}^{\infty} STI_{fuzzy}(A,t+s) \left(f(s,t_{long}) - \sum_{r=1}^{s} (c^{-r} f(s-r,t_{long})) \right)$$

23.4 Valuing LTI and STI in terms of a Single Currency

We now further discuss the approach of defining LTIFund and STIFund in terms of a single currency: juju (which as noted, corresponds in the current ECAN design to normalized stimulus).

In essence, we can think of STIFund and LTIFund as different forms of financial instrument, which are both grounded in juju. Each Atom has two financial instruments attached to it: "STIFund of Atom A" and "LTIFund of Atom A" (or more if multiple versions of LTI are used). These financial instruments have the peculiarity that, although many agents can put juju into any one of them, no record is kept of who put juju in which one. Rather, the MA's are acting so as to satisfy the system's Goals, and are adjusting the STIFund and LTIFund values in a heuristic manner that is expected to approximately maximize the total utility propagated from Goals to Atoms.

Finally, each of these financial instruments has a value that gets updated by a specific update equation.

To understand the logic of this situation better, consider the point of view of a Goal with a certain amount of resources (juju, to be used as reward), and a certain time-scale on which its satisfaction is to be measured. Suppose that the goal has a certain amount of juju to expend on getting itself satisfied.

This Goal clearly should allocate some of its juju toward getting processor time allocated toward the right Atoms to serve its ends in the near future; and some of its juju toward ensuring that, in future, the memory will contain the Atoms it will want to see processor time allocated to. Thus, it should allocate some of its juju toward boosting the STIFund of Atoms that it thinks will (if chosen by appropriate MindAgents) serve its needs in the near future, and some of its juju toward boosting the LTIFund of Atoms that it thinks will serve its need in the future (if they remain in RAM). Thus, when a Goal invokes a MindAgent (giving the MindAgent the juju it needs to access Atoms and carry out its work), it should tell this MindAgent to put some of its juju into LTIFunds and some into STIFunds.

EFTA00624239

If a MindAgent receives a certain amount of juju each cycle, independently of what the system Goals are explicitly telling it, then this should be viewed as reflecting an implicit goal of "ambient cognition", and the balance of STI and LTI associated with this implicit goal must be a system parameter.

In general, the trade-off between STI and LTI boils down to the weighting between near and far future that is intrinsic to a particular Goal. Simplistically: if a Goal values getting processor allocated to the right stuff *immediately* 25 times more than getting processor allocated to the right stuff $20K$ cycles in the future, then it should be willing spend $25\times$ more of its juju on STI than on $LTI_{20K\ cycles}$. (This simplistic picture is complicated a little by the relationship between different time-scales. For instance, boosting $LTI_{10K\ cycles}(A)$ will have an indirect effect of increasing the odds that A will still be in memory $20K$ cycles in the future.)

However, this isn't the whole story, because multiple Goals are setting the importance values of the same set of Atoms. If M_1 pumps all its juju into STI for certain Atoms, then M_2 may decide it's not worthwhile for it to bother competing with M_1 in the STI domain, and to spend its juju on LTI instead.

Note that the current system doesn't allow a MA to change its mind about LTI allocations. One can envision a system where a MindAgent could in January pay juju to have Atom A kept around for a year, but then change its mind in June 6 months later, and ask for some of the money back. But this would require an expensive accounting procedure, keeping track of how much of each Atom's LTI had been purchased by which MindAgent; so it seems a poor approach.

A more interesting alternative would be to allow MA's to retain adjustable "reserve funds" of juju. This would mean that a MindAgent would never see a purpose to setting $LTI_{one\ year}(A)$ instead of repeatedly setting $LTI_{one\ minute}$, unless a substantial transaction cost were incurred with each transaction of adjusting an Atom's LTI. Introducing a transaction cost plus an adjustable per-MindAgent juju reserve fund, and LTI's on multiple time scales, would give the LTI framework considerable flexibility. (To prevent MA's from hoarding their juju, one could place a tax rate on reserve juju.)

The conversion rate between STI and LTI becomes an interesting matter; though it seems not a critical one, since in the practical dynamics of the system it's juju that is used to produce STI and LTI. In the current design there is no apparent reason to spread STI of one Atom to LTI of another Atom, or convert the STI of an Atom into LTI of that same Atom, etc. – but such an application might come up. (For the rest of this paragraph, let's just consider LTI with one time scale, for simplicity.) Each Goal will have its own preferred conversion rate between STI and LTI, based on its own balancing of different time scales. But, each Goal will also have a limited amount of juju, hence one can only trade a certain amount of STI for LTI, if one is trading with a specific goal G. One could envision a centralized STI-for-LTI market where different MA's would trade with each other, but this seems overcomplicated, at least at the present stage.

As a simpler software design point, this all suggests a value for associating each Goal with a parameter telling how much of its juju it wants to spend on STI versus LTI. Or, more subtly, how much of its juju it wants to spend on LTI on various time-scales. On the other hand, in a simple ECAN implementation this balance may be assumed constant across all Goals.

EFTA00624240

23.5 Economic Attention Networks

Economic Attention Networks (ECANs) are dynamical systems based on the propagation of STI and LTI values. They are similar in many respects to Hopfield nets, but are based on a different conceptual foundation involving the propagation of amounts of (conserved) currency rather than neural-net activation. Further, ECANs are specifically designed for integration with a diverse body of cognitive processes as embodied in integrative AI designs such as CogPrime. A key aspect of the CogPrime design is the imposition of ECAN structure on the CogPrime AtomSpace.

Specifically, ECANs have been designed to serve two main purposes within CogPrime: to serve as an associative memory for the network, and to facilitate effective allocation of the attention of other cognitive processes to appropriate knowledge items.

An ECAN is simply a graph, consisting of un-typed nodes and links, and also "Hebbian" links that may have types such as HebbianLink, InverseHebbianLink, or SymmetricHebbianLink. Each node and link in an ECAN is weighted with two currency values, called STI (short-term importance) and LTI (long-term importance); and each Hebbian link is weighted with a probabilistic truth value.

The equations of an ECAN explain how the STI, LTI and Hebbian link weights values get updated over time. As alluded to above, the metaphor underlying these equations is the interpretation of STI and LTI values as (separate) artificial currencies. The fact that STI and LTI are currencies means that, except in unusual instances where the ECAN controller decides to introduce inflation or deflation and explicitly manipulate the amount of currency in circulation, the total amounts of STI and LTI in the system are conserved. This fact makes the dynamics of an ECAN dramatically different than that of an attractor neural network.

In addition to STI and LTI as defined above, the ECAN equations also contain the notion of an Attentional Focus (AF), consisting of those Atoms in the ECAN with the highest STI values (and represented by the $s_{threshold}$ value in the above equations). These Atoms play a privileged role in the system and, as such, are treated using an alternate set of equations.

23.5.1 Semantics of Hebbian Links

Conceptually, the probability value of a HebbianLink from A to B is the odds that if A is in the AF, so is B; and correspondingly, the InverseHebbianLink from A to B is weighted with the odds that if A is in the AF, then B is not. An ECAN will often be coupled with a "Forgetting" process that removes low-LTI Atoms from memory according to certain heuristics. A critical aspect of the ECAN equations is that Atoms periodically spread their STI and LTI to other Atoms that connect to them via Hebbian and InverseHebbianLinks; this is the ECAN analogue of activation spreading in neural networks.

Multiple varieties of HebbianLink may be constructed, for instance

- Asymmetric HebbianLinks, whose semantics are as mentioned above: the truth value of *HebbianLink A B* denotes the probability that if A is in the AF, so is B
- Symmetric HebbianLinks, whose semantics are that: the truth value of *SymmetricHebbianLink A B* denotes the probability that if one of A or B is in the AF, both are

EFTA00624241

It is also worth noting that one can combine ContextLinks with HebbianLinks and express contextual association such that in context C, there is a strong HebbianLink between A and B.

23.5.2 Explicit and Implicit Hebbian Relations

In addition to explicit HebbianLinks, it can be useful to treat other links implicitly as HebbianLinks. For instance, if ConceptNodes A and B are found to connote similar concepts, and a SimilarityLink is formed between them, then this gives reason to believe that maybe a SymmetricHebbianLink between A and B should exist as well. One could incorporate this insight in CogPrime in at least three ways:

- creating HebbianLinks paralleling other links (such as SimilarityLinks)
- adding "Hebbian weights" to other links (such as SimilarityLinks)
- implicitly interpreting other links (such as SimilarityLinks) as HebbianLinks

Further, these strategies may potentially be used together.

There are some obvious semantic relationships to be used in interpreting other link types implicitly as HebbianLinks: for instance, Similarity maps into SymmetricHebbian, and *Inheritance A B* maps into *Hebbian A B*. One may express these as inference rules, e.g.

```
SimilarityLink A B <tv_1>
|-
SymmetricHebbianLink A B <tv_2>
```

where $tv_2.s = tv_1.s$. Clearly, $tv_2.c < tv_1.c$; but the precise magnitude of $tv2.c$ must be determined by a heuristic formula. One option is to set $tv_2.c = \alpha tv_1.c$ where the constant α is set empirically via data mining the System Activity Tables to be described below.

23.6 Dynamics of STI and LTI Propagation

We now get more specific about how some of these ideas are implemented in the currently implemented ECAN subsystem of CogPrime. We'll discuss mostly STI here because in the current design LTI works basically the same way.

MindAgents send out stimulus to Atoms whenever they use them (or else, sometimes, just for the purpose of increasing the Atom's STI); and before these stimulus values are used to update the STI levels of the receiving Atom, they are normalized by: the total amount of stimulus sent out by the MindAgent in that cycle, multiplied by the total amount of STI currency that the MindAgent decided to spend in that cycle. The normalized stimulus is what has above been called juju. This normalization preserves fairness among MA's, and conservation of currency.

(The reason "stimuli" exist, separately from STI, is that stimulus-sending needs to be very computationally cheap, as in general it's done frequently by each MA each cycle, and we don't want each action a MA takes to invoke some costly importance-updating computation.)

Then, Atoms exchange STI according to certain equations (related to HebbianLinks and other links), and have their STI values updated according to certain equations (which involve, among other operations, transferring STI to the "central bank").

EFTA00624242

23.6.1 ECAN Update Equations

The CogServer is understood to maintain a kind of central bank of STI and LTI funds. When a non-ECAN MindAgent finds an Atom valuable, it sends that Atom a certain amount of Stimulus, which results in that Atom's STI and LTI values being increased (via equations to be presented below, that transfer STI and LTI funds from the CogServer to the Atoms in question). Then, the ECAN ImportanceUpdating MindAgent carries out multiple operations, including some that transfer STI and LTI funds from some Atoms back to the CogServer.

There are multiple ways to embody this process equationally; here we briefly describe two variants.

23.6.1.1 Definition and Analysis of Variant 1

We now define a specific set of equations in accordance with the ECAN conceptual framework described above. We define $H_{STI} = [s_1, \cdots, s_n]$ to be the vector of STI values, and $C = \begin{bmatrix} c_{11}, \cdots, c_{1n} \\ \vdots \ddots \vdots \\ c_{n1}, \cdots, c_{nn} \end{bmatrix}$ to be the connection matrix of Hebbian probability values, where it is assumed that the existence of a HebbianLink or InverseHebbianLink between A and B are mutually exclusive possibilities. We also define $C_{LTI} = \begin{bmatrix} g_{11}, \cdots, g_{1n} \\ \vdots \ddots \vdots \\ g_{n1}, \cdots, g_{nn} \end{bmatrix}$ to be the matrix of LTI values for each of the corresponding links.

We assume an updating scheme in which, periodically, a number of Atoms are allocated Stimulus amounts, which causes the corresponding STI values to change according to the equations

$$\forall i : s_i = s_i - \text{rent} + \text{wages},$$

where rent and wages are given by

$$\text{rent} = \begin{cases} \langle \text{Rent} \rangle \cdot \max\left(0, \frac{\log\left(\frac{20 s_i}{\text{recentMaxSTI}}\right)}{2}\right), & \text{if } \ s_i > 0 \\ 0, & \text{if } \ s_i \geq e \geq 0 \end{cases}$$

$$rent = 0, if s_i \leq e$$

and

$$\text{wages} = \begin{cases} \frac{\langle \text{Wage} \rangle \langle \text{Stimulus} \rangle}{\sum_{i=1}^{n} p_i}, \text{if } \ p_i = 1 \\ \frac{\langle \text{Wage} \rangle \langle \text{Stimulus} \rangle}{n - \sum_{i=1}^{n} p_i}, \text{if } \ p_i = 0 \end{cases},$$

where $P = [p_1, \cdots, p_n]$, with $p_i \in \{0, 1\}$ is the cue pattern for the pattern that is to be retrieved. All quantities enclosed in angled brackets are system parameters, and LTI updating is accomplished using a completely analogous set of equations.

EFTA00624243

The changing STI values then cause updating of the connection matrix, according to the "conjunction" equations. First define

$$\text{norm}_i = \begin{cases} \frac{s_i}{\text{recentMaxSTI}}, \text{if } s_i \geq 0 \\ \frac{s_i}{\text{recentMinSTI}}, \text{if } s_i < 0 \end{cases}.$$

Next define

$$\text{conj} = \text{Conjunction}\,(\text{s}_\text{i}, s_j) = \text{norm}_\text{i} \times \text{norm}_\text{j}$$

and

$$c'_{ij} = \langle \text{ConjDecay} \rangle \,\text{conj} + (1 - \text{conj})\, c_{ij}.$$

Finally update the matrix elements by setting

$$c_{ij} = \begin{cases} c_{ji} = c'_{ij}, \text{if } \; c'_{ij} \geq 0 \\ c'_{ij}, \qquad \text{if } \; c'_{ij} < 0 \end{cases}.$$

We are currently also experimenting with updating the connection matrix in accordance with the equations suggested by Storkey (1997, 1998, 1999.)

A key property of these equations is that both wages paid to, and rent paid by, each node are positively correlated to their STI values. That is, the more important nodes are paid more for their services, but they also pay more in rent.

A fixed percentage of the links with the lowest LTI values is then forgotten (which corresponds equationally to setting the LTI to 0).

Separately from the above, the process of Hebbian probability updating is carried out via a diffusion process in which some nodes "trade" STI utilizing a diffusion matrix D, a version of the connection matrix C normalized so that D is a left stochastic matrix. D acts on a similarly scaled vector v, normalized so that v is equivalent to a probability vector of STI values.

The decision about which nodes diffuse in each diffusion cycle is carried out via a decision function. We currently are working with two types of decision functions: a standard threshold function, by which nodes diffuse if and only if the nodes are in the AF; and a stochastic decision function in which nodes diffuse with probability $\frac{\tanh(\text{shape}(s_i - \text{FocusBoundary}))+1}{2}$, where shape and FocusBoundary are parameters.

The details of the diffusion process are as follows. First, construct the diffusion matrix from the entries in the connection matrix as follows:

$$\begin{array}{l} \text{If } c_{ij} \geq 0, \text{ then } d_{ij} = c_{ij}, \\ \text{else, set } d_{ji} = -c_{ij}. \end{array}$$

Next, we normalize the columns of D to make D a left stochastic matrix. In so doing, we ensure that each node spreads no more than a $\langle \text{MaxSpread} \rangle$ proportion of its STI, by setting

$$\text{if } \sum_{\text{i}=1}^{\text{n}} d_{ij} > \langle \text{MaxSpread} \rangle :$$

$$d_{ij} = \begin{cases} d_{ij} \times \frac{\langle \text{MaxSpread} \rangle}{\sum_{i=1}^{n} d_{ij}}, \;\; \text{for } i \neq j \\ d_{jj} = 1 - \langle \text{MaxSpread} \rangle \end{cases}$$

EFTA00624244

else:

$$d_{jj} = 1 - \sum_{\substack{i=1 \\ i \neq j}}^{n} d_{ij}$$

Now we obtain a scaled STI vector v by setting

$$\text{minSTI} = \min_{i \in \{1,2,\cdots,n\}} s_i \text{ and maxSTI} = \max_{i \in \{1,2,\cdots,n\}} s_i$$

$$v_i = \frac{s_i - \min STI}{\max \text{STI} - \min \text{STI}}$$

The diffusion matrix is then used to update the node STIs

$$v' = Dv$$

and the STI values are rescaled to the interval [minSTI,maxSTI].

In both the rent and wage stage and in the diffusion stage, the total STI and LTI funds of the system each separately form a conserved quantity: in the case of diffusion, the vector v is simply the total STI times a probability vector. To maintain overall system funds within homeostatic bounds, a mid-cycle tax and rent-adjustment can be triggered if necessary; the equations currently used for this are

1. $\langle \text{Rent} \rangle = \frac{\text{recent stimulus awarded before update} \times \langle \text{Wage} \rangle}{\text{recent size of AF}}$;
2. $\text{tax} = \frac{x}{n}$, where x is the distance from the current AtomSpace bounds to the center of the homeostatic range for AtomSpace funds;
3. $\forall i: s_i = s_i - \text{tax}$

23.6.1.2 Investigation of Convergence Properties of Variant 1

Now we investigate some of the properties that the above ECAN equations display when we use an ECAN defined by them as an associative memory network in the manner of a Hopfield network.

We consider a situation where the ECAN is supplied with memories via a "training" phase in which one imprints it with a series of binary patterns of the form $P = [p_1, \cdots, p_n]$, with $p_i \in \{0, 1\}$. Noisy versions of these patterns are then used as cue patterns during the retrieval process.

We obviously desire that the ECAN retrieve the stored pattern corresponding to a given cue pattern. In order to achieve this goal, the ECAN must converge to the correct fixed point.

Theorem 23.1. *For a given value of e in the STI rent calculation, there is a subset of hyperbolic decision functions for which the ECAN dynamics converge to an attracting fixed point.*

Proof. Rent is zero whenever $e \leq s_i \leq \frac{\text{recentMaxSTI}}{20}$, so we consider this case first. The updating process for the rent and wage stage can then be written as $f(s) = s + \text{constant}$. The next stage is governed by the hyperbolic decision function

EFTA00624245

$$g(s) = \frac{\tanh(\text{shape}(\text{s} - \text{FocusBoundary})) + 1}{2}.$$

The entire updating sequence is obtained by the composition $(g \circ f)(s)$, whose derivative is then

$$(g \circ f)' = \frac{\text{sech}^2(\text{f}(\text{s})) \cdot \text{shape}}{2} \cdot (1),$$

which has magnitude less than 1 whenever -2 < shape < 2. We next consider the case $s_i > \frac{\text{recentMaxSTI}}{20} \geq e$. The function f now takes the form

$$f(s) = s - \frac{\log(20s/\text{recentMaxSTI})}{2} + \text{constant},$$

and we have

$$(g \circ f)' = \frac{\text{sech}^2(\text{f}(\text{s})) \cdot \text{shape}}{2} \cdot \left(1 - \frac{1}{2s}\right).$$

which has magnitude less than 1 whenever $|\text{shape}| < \left|\frac{4e}{2e-1}\right|$. Choosing the shape parameter to satisfy $0 < \text{shape} < \min\left(2, \left|\frac{4e}{2e-1}\right|\right)$ then guarantees that $\left|(g \circ f)'\right| < 1$. Finally, g ∘ f maps the closed interval $[recentMinStI, recentMaxSTI]$ into itself, so applying the Contraction Mapping Theorem completes the proof.

23.6.1.3 Definition and Analysis of Variant 2

The ECAN variant described above has performed completely acceptably in our experiments so far; however we have also experimented with an alternate variant, with different convergence properties. In Variant 2, the dynamics of the ECAN are specifically designed so that a certain conceptually intuitive function serves as a Liapunov function of the dynamics.

At a given time t, for a given Atom indexed i, we define two quantities: $OUT_i(t)$ = the total amount that Atom i pays in rent and tax and diffusion during the time-t iteration of ECAN ; $IN_i(t)$ = the total amount that Atom i receives in diffusion, stimulus and welfare during the time-t iteration of ECAN. Note that welfare is a new concept to be introduced below. We then define $DIFF_i(t) = IN_i(t) - OUT_i(t)$; and define $AVDIFF(t)$ as the average of $DIFF_i(t)$ over all i in the ECAN.

The design goal of Variant 2 of the ECAN equations is to ensure that, if the parameters are tweaked appropriately, AVDIFF can serve as a (deterministic or stochastic, depending on the details) Lyapunov function for ECAN dynamics. This implies that with appropriate parameters the ECAN dynamics will converge toward a state where AVDIFF=0, meaning that no Atom is making any profit or incurring any loss. It must be noted that this kind of convergence is not always desirable, and sometimes one might want the parameters set otherwise. But if one wants the STI components of an ECAN to converge to some specific values, as for instance in a classic associative memory application, Variant 2 can guarantee this easily.

In Variant 2, each ECAN cycle begins with rent collection and welfare distribution, which occurs via collecting rent via the Variant 1 equation, and then performing the following two steps:

EFTA00624246

- **Step A**: calculate X, defined as the positive part of the total amount by which AVDIFF has been increased via the overall rent collection process.
- **Step B**: redistribute X to needy Atoms as follows: *For each Atom z, calculate the positive part of $OUT - IN$, defined as deficit(z). Distribute $X + e$ wealth among all Atoms z, giving each Atom a percentage of X that is proportional to deficit(z), but never so much as to cause OUT<IN for any Atom (the welfare being given counts toward IN).* Here $e > 0$ ensures AVDIFF decrease; $e = 0$ may be appropriate if convergence is not required in a certain situation.

Step B is the welfare step, which guarantees that rent collection will decrease AVDIFF. Step A calculates the amount by which the rich have been made poorer, and uses this to make the poor richer. In the case that the sum of *deficit*(z) over all nodes z is less than X, a mid-cycle rent adjustment may be triggered, calculated so that step B will decrease AVDIFF. (I.e. we cut rent on the rich, if the poor don't need their money to stay out of deficit.)

Similarly, in each Variant 2 ECAN cycle, there is a wage-paying process, which involves the wage-paying equation from Variant 1 followed by two steps. Step A: calculate Y, defined as the positive part of the total amount by which AVDIFF has been increased via the overall wage payment process. Step B: exert taxation based on the surplus Y as follows: *For each Atom z, calculate the positive part of $IN - OUT$, defined as surplus(z). Collect $Y + e1$ wealth from all Atom z, collecting from each node a percentage of Y that is proportional to surplus(z), but never so much as to cause $IN < OUT$ for any node (the new STI being collected counts toward OUT).*

In case the total of *surplus*(z) over all nodes z is less than Y, one may trigger a mid-cycle wage adjustment, calculated so that step B will decrease AVDIFF. I.e. we cut wages since there is not enough surplus to support it.

Finally, in the Variant 2 ECAN cycle, diffusion is done a little differently, via iterating the following process: If AVDIFF has increased during the diffusion round so far, then choose a random node whose diffusion would decrease AVDIFF, and let it diffuse; if AVDIFF has decreased during the diffusion round so far, then choose a random node whose diffusion would increase AVDIFF, and let it diffuse. In carrying out these steps, we avoid letting the same node diffuse twice in the same round. This algorithm does not let all Atoms diffuse in each cycle, but it stochastically lets a lot of diffusion happen in a way that maintains AVDIFF constant. The iteration may be modified to bias toward an average decrease in AVDIFF.

The random element in the diffusion step, together with the logic of the rent/welfare and wage/tax steps, combines to yield the result that for Variant 2 of ECAN dynamics, AVDIFF is a stochastic Lyapunov function. The details of the proof of this will be omitted but the outline of the argument should be clear from the construction of Variant 2. And note that by setting the e and e_1 parameter to 0, the convergence requirement can be eliminated, allowing the network to evolve more spontaneously as may be appropriate in some contexts; these parameters allow one to explicitly adjust the convergence rate.

One may also derive results pertaining to the meaningfulness of the attractors, in various special cases. For instance, if we have a memory consisting of a set M of m nodes, and we imprint the memory on the ECAN by stimulating m nodes during an interval of time, then we want to be able to show that the condition where precisely those m nodes are in the AF is a fixed-point attractor. However, this is not difficult, because one must only show that if these m nodes and none others are in the AF, this condition will persist.

EFTA00624247

23.6.2 ECAN as Associative Memory

We have carried out experiments gauging the performance of Variant 1 of ECAN as an associative memory, using the implementation of ECAN within CogPrime, and using both the conventional and Storkey Hebbian updating formulas.

As with a Hopfield net memory, the memory capacity (defined as the number of memories that can be retrieved from the network with high accuracy) depends on the sparsity of the network, with denser networks leading to greater capacity. In the ECAN case the capacity also depends on a variety of parameters of the ECAN equations, and the precise unraveling of these dependencies is a subject of current research. However, one interesting dependency has already been uncovered in our preliminary experimentation, which has to do with the size of the AF versus the size of the memories being stored.

Define the size of a memory (a pattern being imprinted) as the number of nodes that are stimulated during imprinting of that memory. In a classical Hopfield net experiment, the mean size of a memory is usually around, say, .2-.5 of the number of neurons. In typical CogPrime associative memory situations, we believe the mean size of a memory will be one or two orders of magnitude smaller than that, so that each memory occupies only a relatively small portion of the overall network.

What we have found is that the memory capacity of an ECAN is generally comparable to that of a Hopfield net with the same number of nodes and links, if and only if the ECAN parameters are tuned so that the memories being imprinted can fit into the AF. That is, the AF threshold or (in the hyperbolic case) shape parameter must be tuned so that the size of the memories is not so large that the active nodes in a memory cannot stably fit into the AF. This tuning may be done adaptively by testing the impact of different threshold/shape values on various memories of the appropriate size; or potentially a theoretical relationship between these quantities could be derived, but this has not been done yet. This is a reasonably satisfying result given the cognitive foundation of ECAN: in loose terms what it means is that ECAN works best for remembering things that fit into its focus of attention during the imprinting process.

23.7 Glocal Economic Attention Networks

In order to transform ordinary ECANs into glocal ECANs, one may proceed in essentially the same manner as with glocal Hopfield nets as discussed in Chapter 13 of Part 1. In the language normally used to describe CogPrime, this would be termed a "map encapsulation" heuristic. As with glocal Hopfield nets, one may proceed most simply via creating a fixed pool of nodes intended to provide locally-representative keys for the maps formed as attractors of the network. Links may then be formed to these key nodes, with weights and STI and LTI values adapted by the usual ECAN algorithms.

EFTA00624248

23.7.1 Experimental Explorations

To compare the performance of glocal ECANs with glocal Hopfield networks in a simple context, we ran experiments using ECAN in the manner of a Hopfield network. That is, a number of nodes take on the equivalent role of the neurons that are presented patterns to be stored. These patterns are imprinted by setting the corresponding nodes of active bits to have their STI within the AF, whereas nodes corresponding to inactive bits of the pattern are below the AF threshold. Link weight updating then occurs, using one of several update rules, but in this case the update rule of [SV99] was used. Attention was spread using a diffusion approach by representing the weights of Hebbian links between pattern nodes within a left stochastic Markov matrix, and multiplying it by the vector of normalised STI values to give a vector representing the new distribution of STI.

To explore the effects of key nodes on ECAN Hopfield networks, in [Goe08b] we used the palimpsest testing scenario of [SV99], where all the local neighbours of the imprinted pattern, within a single bit change, are tested. Each neighbouring pattern is used as input to try and retrieve the original pattern. If all the retrieved patterns are the same as the original (within a given tolerance) then the pattern is deemed successfully retrieved and recall of the previous pattern is attempted via its neighbours. The number of patterns this can repeat for successfully is called the palimpsest storage of the network.

As an example, consider one simple experiment that was run with recollection of 10×10 pixel patterns (so, 100 nodes, each corresponding to a pixel in the grid), a Hebbian link density of 30%, and with 1% of links being forgotten before each pattern is imprinted. The results demonstrated that, when the mean palimpsest storage is calculated for each of 0, 1, 5 and 10 key nodes we find that the storage is 22.6, 22.4, 24.9, and 26.0 patterns respectively, indicating that key nodes do improve memory recall on average.

23.8 Long-Term Importance and Forgetting

Now we turn to the forgetting process (carried out by the Forgetting MindAgent), which is driven by LTI dynamics, but has its own properties as well.

Overall, the goal of the "forgetting" process is to maximize the total utility of the Atoms in the AtomSpace throughout the future. The most basic heuristic toward this end is to remove the Atoms with the lowest LTI, but this isn't the whole story. Clearly, the decision to remove an Atom from RAM should depend on factors beyond just the LTI of the Atom. For example, one should also take into account the expected difficulty in reconstituting the given Atom from other Atoms. Suppose the system has the relations:

```
''dogs are animals''

''animals are cute''

''dogs are cute''
```

and the strength of the third relation is not dissimilar from what would be obtained by deduction and revision from the first two relations and others in the system. Then, even if the system judges it will be very useful to know *dogs are cute* in the future, it may reasonably choose to remove *dogs are cute* from memory anyway, because it knows it can be so easily reconstituted,

EFTA00624249

by a few inference steps for instance. Thus, as well as removing the lowest-LTI Atoms, the Forgetting MindAgent should also remove Atoms meeting certain other criteria such as the combination of:

- low STI
- easy reconstitutability in terms of other Atoms that have LTI not less than its own

23.9 Attention Allocation via Data Mining on the System Activity Table

In this section we'll discuss an object called the System Activity Table, which contains a number of subtables recording various activities carried out by the various objects in the CogPrime system. These tables may be used for sophisticated attention allocation processes, according to an approach in which importance values and HebbianLink weight values are calculated via direct data mining of a centralized knowledge store (the System Activity Table). This approach provides highly accurate attention allocation but at the cost of significant computational effort.

The System Activity Table is actually a set of tables, with multiple components. The precise definition of the tables will surely be adapted based on experience as the work with CogPrime progresses; what is described here is a reasonable first approximation.

First, there is a MindAgent Activity Table, which includes, for each MindAgent in the system, a table such as Table 23.1 (in which the time-points recorded are the last T system cycles, and the Atom-lists recorded are lists of Handles for Atoms).

System Cycle	Effort Spent	Memory Used	Atom Combo 1 Utilized	Atom Combo 2 Utilized	...
Now	3.3	4000	Atom21, Atom44	Atom 44, Atom 47, Atom 345	...
Now -1	0.4	6079	Atom123, Atom33	Atom 345	...
...	...	...	...	...	...

Table 23.1: Example MindAgent Table

The MindAgent's activity table records, for that MindAgent and for each system cycle, which Atom-sets were acted on by that MindAgent at that point in time.

Similarly, a table of this nature must be maintained for each Task-type, e.g. InferenceTask, MOSESCategorizationTask, etc. The Task tables are used to estimate Effort values for various Tasks, which are used in the procedure execution process. If it can be estimated how much spatial and temporal resources a Task is likely to use, via comparison to a record of previous similar tasks (in the Task table), then a MindAgent can decide whether it is appropriate to carry out this Task (versus some other one, or versus some simpler process not requiring a Task) at a given point in time, a process to be discussed in a later chapter.

In addition to the MindAgent and Task-type tables, it is convenient if tables are maintained corresponding to various goals in the system (as shown in Table ??), including the Ubergoals but also potentially derived goals of high importance.
For each goal, at minimum, the degree of achievement of the goal at a given time must be recorded. Optionally, at each point in time, the degree of achievement of a goal relative to some

EFTA00624250

System Cycle	Total Achievement	Achievement for Atom44	Achievement for set {Atom44, Atom 233}	...
Now	.8	.4	.5	...
Now-1	.9	.5	.55	...
...	...	...	...	...

Table 23.2: Example Goal Table

particular Atoms may be recorded. Typically the list of Atom-specific goal-achievements will be short and will be different for different goals and different time points. Some goals may be applied to specific Atoms or Atom sets, others may only be applied more generally.

The basic idea is that attention allocation and credit assignment may be effectively carried out via datamining on these tables.

23.10 Schema Credit Assignment

And, how do we apply a similar approach to clarifying the semantics of schema credit assignment?

From the above-described System Activity Tables, one can derive information of the form

```
Achieve(G,E,T) = ``Goal G was achieved to extent E at time T''
```

which may be grounded as, for example:

```
Similarity
    E
    ExOut
        GetTruthValue
        Evaluation
            atTime
                T
                HypLink G
```

and more refined versions such as

```
Achieve(G,E,T,A,P) = ``Goal G was achieved to extent E using
                     Atoms A (with parameters P) at time T''

Enact(S,I,$T_1$,O,$T_2$) = ``Schema S was enacted on inputs I
                           at time $T_1$, producing outputs O
                           at time $T_2$''
```

The problem of schema credit assignment is then, in essence: Given a goal G and a distribution of times $\mathcal{D}$, figure out what schema to enact in order to cause G's achievement at some time in the future, where the desirability of times is weighted by $\mathcal{D}$.

The basic method used is the learning of predicates of the form

```
ImplicationLink
```

$F(C, P_1, ..., P_n)$

$\mathcal{G}$

where

EFTA00624251

- the P_i are *Enact*() statements in which the T_1 and T_2 are variable, and the S, I and O may be concrete or variable
- C is a predicate representing a *context*
- $\mathcal{G}$ is an *Achieve*() statement, whose arguments may be concrete or abstract
- F is a Boolean function

Typically, the variable expressions in the T_1 and T_2 positions will be of the form $T + offset$, where offset is a constant value and T is a time value representing the time of inception of the whole compound schema. T may then be defined as $T_G - offset_1$, where $offset_1$ is a constant value and T_G is a variable denoting the time of achievement of the goal.

In CogPrime, these predicates may be learned by a combination of statistical pattern mining, PLN inference and MOSES or hill-climbing procedure learning.

The choice of what action to take at a given point in time is then a probabilistic decision. Based on the time-distribution $\mathcal{D}$ given, the system will know a certain number of expressions $\mathcal{C} = F(C, P_1, ..., P_n)$ of the type described above. Each of these will be involved in an ImplicationLink with a certain estimated strength. It may select the "compound schema" $\mathcal{C}$ with the highest strength.

One might think to introduce other criteria here, e.g. to choose the schema with the highest strength but the lowest cost of execution. However, it seems better to include all pertinent criteria in the goal, so that if one wants to consider cost of execution, one assumes the existence of a goal that incorporates cost of execution (which may be measured in multiple ways, of course) as part of its internal evaluation function.

Another issue that arises is whether to execute multiple $\mathcal{C}$ simultaneously. In many cases this won't be possible because two different $\mathcal{C}$'s will contradict each other. It seems simplest to assume that $\mathcal{C}$'s that can be fused together into a single plan of action, are presented to the schema execution process as a single fused $\mathcal{C}$. In other words, the fusion is done during the schema learning process rather than the execution process.

A question emerges regarding how this process deals with false causality, e.g. with a schema that, due to the existence of a common cause, often happens to occur immediately prior to the occurrence of a given goal. For instance, roosters crowing often occurs prior to the sun rising. This matter is discussed in more depth in the PLN book and *The Hidden Pattern*; but in brief, the answer is: In the current approach, if roosters crowing often causes the sun to rise, then if the system wants to cause the sun to rise, it may well cause a rooster to crow. Once this fails, then the system will no longer hold the false belief, and afterwards will choose a different course of action. Furthermore, if it holds background knowledge indicating that roosters crowing is not likely to cause the sun to rise, then this background knowledge will be invoked by inference to discount the strength of the ImplicationLink pointing from rooster-crowing to sun-rising, so that the link will never be strong enough to guide schema execution in the first place.

The problem of credit assignment thus becomes a problem of creating appropriate heuristics to guide inference of ImplicationLinks of the form described above. Assignment of credit is then implicit in the calculation of truth values for these links. The difficulty is that the predicates F involved may be large and complex.

EFTA00624252

23.11 Interaction between ECANs and other CogPrime Components

We have described above a number of interactions between attention allocation and other aspects of CogPrime; in this section we gather a few comments on these interactions, and some additional ones.

23.11.1 Use of PLN and Procedure Learning to Help ECAN

MOSES or hillclimbing may be used to help mine the SystemActivityTable for patterns of usefulness, and create HebbianLinks reflecting these patterns.

PLN inference may be carried out on HebbianLinks by treating (HebbianLink A B) as a virtual predicate evaluation relationship, i.e. as

```
EvaluationLink Hebbian_predicate (A, B)
```

PLN inference on HebbianLinks may then be used to update node importance values, because node importance values are essentially *node probabilities* corresponding to HebbianLinks. And similarly, MindAgent-relative node importance values are node probabilities corresponding to MindAgent-relative HebbianLinks.

Note that conceptually, the nature of this application of PLN is different from most other uses of PLN in CogPrime. Here, the purpose of PLN is not to draw conclusions about the outside world, but rather about what the system should focus its resources on in what context. PLN, used in this context, effectively constitutes a nonlinear-dynamical iteration governing the flow of *attention* through the CogPrime system.

Finally, inference on HebbianLinks leads to the emergence of maps, via the recognition of clusters in the graph of HebbianLinks.

23.11.2 Use of ECAN to Help Other Cognitive Processes

First of all, associative-memory functionality is directly important in CogPrime because it is used to drive concept creation. The CogPrime heuristic called "map formation" creates new Nodes corresponding to prominent attractors in the ECAN, a step that (according to our preliminary results) not only increases the memory capacity of the network beyond what can be achieved with a pure ECAN but also enables attractors to be explicitly manipulated by PLN inference.

Equally important to associative memory is the capability of ECANs to facilitate effective allocation of the attention of other cognitive processes to appropriate knowledge items (Atoms). For example, one key role of ECANs in CogPrime is to guide the forward and backward chaining processes of PLN (Probabilistic Logic Network) inference. At each step, the PLN inference chainer is faced with a great number of inference steps (branches) from which to choose; and a choice is made using a statistical "bandit problem" mechanism that selects each possible inference step with a probability proportional to its expected "desirability." In this context, there is considerable appeal in the heuristic of weighting inference steps using probabilities

EFTA00624253

proportional to the STI values of the Atoms they contain. One thus arrives at a combined PLN/ECAN dynamic as follows:

1. An inference step is carried out, involving a choice among multiple possible inference steps, which is made using STI-based weightings (and made among Atoms that LTI weightings have deemed valuable enough to remain in RAM)
2. The Atoms involved in the inference step are rewarded with STI and LTI proportionally to the utility of the inference step (how much it increases the confidence of Atoms in the system's memory)
3. The ECAN operates, and multiple Atom's importance values are updated
4. Return to Step 1 if the inference isn't finished

An analogous interplay may occur between ECANs and MOSES.

It seems intuitively clear that the same attractor-convergence properties highlighted in the above analysis of associative-memory behavior, will also be highly valuable for the application of ECANs to attention allocation. If a collection of Atoms is often collectively useful for some cognitive process (such as PLN), then the associative-memory-type behavior of ECANs means that once a handful of the Atoms in the collection are found useful in a certain inference process, the other Atoms in the collection will get their STI significantly boosted, and will be likely to get chosen in subsequent portions of that same inference process. This is exactly the sort of dynamics one would like to see occur. Systematic experimentation with these interactions between ECAN and other CogPrime processes is one of our research priorities going forwards.

23.12 MindAgent Importance and Scheduling

So far we have discussed economic transactions between Atoms and Atoms, and between Atoms and Units. MindAgents have played an indirect role, via spreading stimulation to Atoms which causes them to get paid wages by the Unit. Now it is time to discuss the explicit role of MindAgents in economic transactions. This has to do with the integration of economic attention allocation with the Scheduler that schedules the core MindAgents involved in the basic cognitive cycle.

This integration may be done in many ways, but one simple approach is:

1. When a MindAgent utilizes an Atom, this results in sending stimulus to that Atom. (Note that we don't want to make MindAgents pay for using Atoms individually; that would penalize MA's that use more Atoms, which doesn't really make much sense.)
2. MindAgents then get currency from the Lobe (as defined in Chapter 19) periodically, and get extra currency based on usefulness for goal achievement as determined by the credit assignment process. The Scheduler then gives more processor time to MindAgents with more STI.
3. However, any MindAgent with LTI above a certain minimum threshold will get some minimum amount of processor time (i.e. get scheduled at least once each N cycles).

As a final note: In a multi-Lobe Unit, the Unit may use the different LTI values of MA's in different Lobes to control the distribution of MA's among Lobes: e.g. a very important (LTI) MA might get cloned across multiple Lobes.

EFTA00624254

23.13 Information Geometry for Attention Allocation

Appendix ?? outlines some very broad ideas regarding the potential utilization of information geometry and related ideas for modeling cognition. In this section, we present some more concrete and detailed experiments inspired by the same line of thinking. We model CogPrime's Economic Attention Networks (ECAN) component using information geometric language, and then use this model to propose a novel information geometric method of updating ECAN networks (based on an extension of Amari's ANGL algorithm). Tests on small networks suggest that information-geometric methods have the potential to vastly improve ECAN's capability to shift attention from current preoccupations to desired preoccupations. However, there is a high computational cost associated with the simplest implementations of these methods, which has prevented us from carrying out large-scale experiments so far. We are exploring the possibility of circumventing these issues via using sparse matrix algorithms on GPUs.

23.13.1 Brief Review of Information Geometry

"Information geometry" is a branch of applied mathematics concerned with the application of differential geometry to spaces of probability distributions. In [GI11] we have suggested some extensions to traditional information geometry aimed at allowing it to better model general intelligence. However for the concrete technical work in this Chapter, the traditional formulation of information geometry will suffice.

One of the core mathematical constructs underlying information geometry, is the Fisher Information, a statistical quantity which has a a variety of applications ranging far beyond statistical data analysis, including physics [Fri98], psychology and AI [AN00]. Put simply, FI is a formal way of measuring the amount of information that an observable random variable X carries about an unknown parameter θ upon which the probability of X depends. FI forms the basis of the Fisher-Rao metric, which has been proved the only Riemannian metric on the space of probability distributions satisfying certain natural properties regarding invariance with respect to coordinate transformations. Typically θ in the FI is considered to be a real multidimensional vector; however, [Dab99] has presented a FI variant that imposes basically no restrictions on the form of θ. Here the multidimensional FI will suffice, but the more general version is needed if one wishes to apply FI to AGI more broadly, e.g. to declarative and procedural as well as attentional knowledge.

In the set-up underlying the definition of the ordinary finite-dimensional Fisher information, the probability function for X, which is also the likelihood function for $\theta \in R^n$, is a function $f(X;\theta)$; it is the probability mass (or probability density) of the random variable X conditional on the value of θ. The partial derivative with respect to θ_i of the log of the likelihood function is called the *score* with respect to θ_i. Under certain regularity conditions, it can be shown that the first moment of the score is 0. The second moment is the Fisher information:

$$\mathcal{I}(\theta)_i = \mathcal{I}_X(\theta)_i = E\left[\left(\left(\frac{\partial}{\partial\theta_i}\ln f(X;\theta)\right)^2\right)|\theta\right]$$

where, for any given value of θ_i, the expression $E[..|\theta]$ denotes the conditional expectation over values for X with respect to the probability function $f(X;\theta)$ given θ. Note that $0 \leq \mathcal{I}(\theta)_i <$

EFTA00624255

∞. Also note that, in the usual case where the expectation of the score is zero, the Fisher information is also the variance of the score.

One can also look at the whole Fisher information matrix

$$\mathcal{I}(\theta)_{i,j} = E\left[\left(\frac{\partial ln f(X,\theta)}{\partial \theta_i}\frac{\partial ln f(X,\theta)}{\partial \theta_j}\right)|\theta\right]$$

which may be interpreted as a metric g_{ij}, that provably is the only "intrinsic" metric on probability distribution space. In this notation we have $\mathcal{I}(\theta)_i = \mathcal{I}(\theta)_{i,i}$.

Dabak [Dab99] has shown that the geodesic between two parameter vectors θ and θ' is given by the exponential weighted curve $(\gamma(t))(x) = \frac{f(x,\theta)^{1-t}f(x,\theta')^t}{\int f(y,\theta)^{1-t}f(y,\theta')^t dy}$, under the weak condition that the log-likelihood ratios with respect to $f(X,\theta)$ and $f(X,\theta')$ are finite. Also, along this sort of curve, the sum of the Kullback-Leibler distances between θ and θ', known as the J-divergence, equals the integral of the Fisher information along the geodesic connecting θ and θ'. This suggests that if one is attempting to learn a certain parameter vector based on data, and one has a certain other parameter vector as an initial value, it may make sense to use algorithms that try to follow the Fisher-Rao geodesic between the initial condition and the desired conclusion. This is what Amari [Ama85] [AN00] calls "natural gradient" based learning, a conceptually powerful approach which subtly accounts for dependencies between the components of θ.

23.13.2 Information-Geometric Learning for Recurrent Networks: Extending the ANGL Algorithm

Now we move on to discuss the practicalities of information-geometric learning within CogPrime's ECAN component. As noted above, Amari [Ama85, AN00] introduced the natural gradient as a generalization of the direction of steepest descent on the space of loss functions of the parameter space. Issues with the original implementation include the requirement of calculating both the Fisher information matrix and its inverse. To resolve these and other practical considerations, Amari [Ama98] proposed an adaptive version of the algorithm, the Adaptive Natural Gradient Learning (ANGL) algorithm. Park, Amari, and Fukumizu [PAF00] extended ANGL to a variety of stochastic models including stochastic neural networks, multi-dimensional regression, and classification problems.

In particular, they showed that, assuming a particular form of stochastic feedforward neural network and under a specific set of assumptions concerning the form of the probability distributions involved, a version of the Fisher information matrix can be written as

$$G(\theta) = E_\xi\left[\left(\frac{r'}{r}\right)^2\right] E_x\left[\nabla H (\nabla H)^T\right].$$

Although Park et al considered only feedforward neural networks, their result also holds for more general neural networks, including the ECAN network. What is important is the decomposition of the probability distribution as

$$p(\mathbf{y}|\mathbf{x};\theta) = \prod_{i=1}^{L} r_i(y_i - H_i(\mathbf{x},\theta))$$

EFTA00624256

where

$$\mathbf{y} = \mathbf{H}(\mathbf{x};\theta) + \xi,\ \mathbf{y} = (y_1, \cdots, y_L)^T,\ \mathbf{H} = (H_1, \cdots, H_L)^T,\ \xi = (\xi_1, \cdots, \xi_L)^T,$$

where ξ is added noise. If we assume further that each r_i has the same form as a Gaussian distribution with zero mean and standard deviation σ, then the Fisher information matrix simplifies further to

$$G(\theta) = \frac{1}{\sigma^2} E_x \left[\nabla H \left(\nabla H \right)^T \right].$$

The adaptive estimate for $\hat{G}_{t+1}^{-1}$ is given by

$$\hat{G}_{t+1}^{-1} = (1 + \epsilon_t)\hat{G}_t^{-1} - \epsilon_t (\hat{G}_t^{-1} \nabla H)(\hat{G}_t^{-1} \nabla H)^T.$$

and the loss function for our model takes the form

$$l(\mathbf{x}, \mathbf{y}; \theta) = -\sum_{\mathbf{i=1}}^{\mathbf{L}} \log \mathbf{r}(\mathbf{y_i} - \mathbf{H_i}(\mathbf{x}, \theta)).$$

The learning algorithm for our connection matrix weights θ is then given by

$$\theta_{t+1} = \theta_t - \eta_t \hat{G}_t^{-1} \nabla l(\theta_t).$$

23.13.3 Information Geometry for Economic Attention Allocation: A Detailed Example

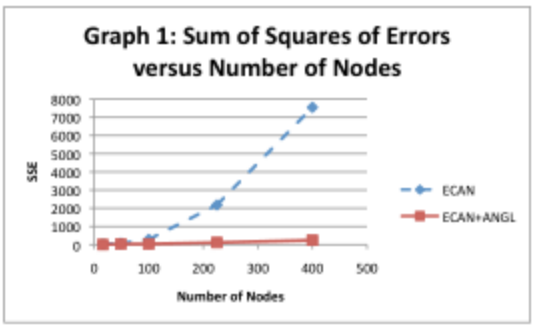


Fig. 23.3: Results from Experiment 1

We now present the results of a series of small-scale, exploratory experiments comparing the original ECAN process running alone with the ECAN process coupled with ANGL. We are interested in determining which of these two lines of processing result in focusing attention more accurately.

EFTA00624257

The experiment started with base patterns of various sizes to be determined by the two algorithms. In the training stage, noise was added, generating a number of instances of noisy base patterns. The learning goal is to identify the underlying base patterns from the noisy patterns as this will identify how well the different algorithms can focus attention on relevant versus irrelevant nodes.

Next, the ECAN process was run, resulting in the determination of the connection matrix C. In order to apply the ANGL algorithm, we need the gradient, ∇H, of the ECAN training process, with respect to the input $\mathbf{x}$. While calculating the connection matrix C, we used Monte Carlo simulation to simultaneously calculate an approximation to ∇H.

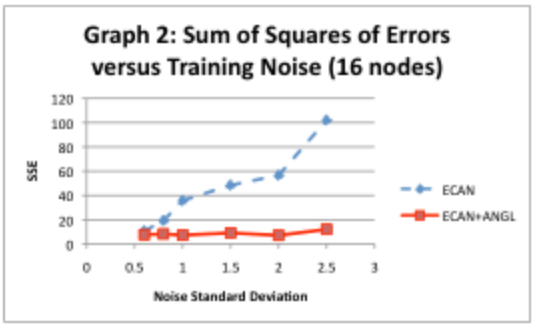


Fig. 23.4: Results from Experiment 2

After ECAN training was completed, we bifurcated the experiment. In one branch, we ran fuzzed cue patterns through the retrieval process. In the other, we first applied the ANGL algorithm, optimizing the weights in the connection matrix, prior to running the retrieval process on the same fuzzed cue patterns. At a constant value of $\sigma = 0.8$ we ran several samples through each branch with pattern sizes of 4×4, 7×7, 10×10, 15×15, and 20×20. The results are shown in Figure 23.3. We also ran several experiments comparing the sum of squares of the errors to the input training noise as measured by the value of σ.; see Figures 23.4 and **??**.

These results suggest two major advantages of the ECAN+ANGL combination compared to ECAN alone. Not only was the performance of the combination better in every trial, save for one involving a small number of nodes and little noise, but the combination clearly scales significantly better both as the number of nodes increases, and as the training noise increases.

EFTA00624258

Chapter 24
Economic Goal and Action Selection

24.1 Introduction

A significant portion of CogPrime's dynamics is explicitly goal-driven – that is, based on trying (inasmuch as possible within the available resources) to figure out which actions will best help the system achieve its goals, given the current context. A key aspect of this explicit activity is guided by the process of "goal and action selection" – prioritizing goals, and then prioritizing actions based on these goals. We have already outlined the high-level process of action selection, in Chapter 22 above. Now we dig into the specifics of the process, showing how action selection is dynamically entwined with goal prioritization, and how both processes are guided by economic attention allocation as described in Chapter 23.

While the basic structure of CogPrime's action selection aspect is fairly similar to MicroPsi (due to the common foundation in Dorner's Psi model), the dynamics are less similar. MicroPsi's dynamics are a little closer to being a formal neural net model, whereas ECAN's economic foundation tends to push it in different directions. The CogPrime goal and action selection design involves some simple simulated financial mechanisms, building on the economic metaphor of ECAN, that are different from, and more complex than, anything in MicroPsi.

The main actors (apart from the usual ones like the AtomTable, economic attention allocation, etc.) in the tale to be told here are as follows:

- Structures:
  - UbergoalPool
  - ActiveSchemaPool
- MindAgents:
  - GoalBasedSchemaSelection
  - GoalBasedSchemaLearning
  - GoalAttentionAllocation
  - FeasibilityUpdating
  - SchemaActivation

The Ubergoal Pool contains the Atoms that the system considers as top-level goals. These goals must be treated specially by attention allocation: they must be given funding by the Unit so that they can use it to pay for getting themselves achieved. The weighting among different

EFTA00624260

top-level goals is achieved via giving them different amounts of currency. STICurrency is the key kind here, but of course ubergoals must also get some LTICurrency so they won't be forgotten. (Inadvertently deleting your top-level supergoals from memory is generally considered to be a bad thing ... it's in a sense a sort of suicide...)

24.2 Transfer of STI "Requests for Services" Between Goals

Transfer of "attentional funds" from goals to subgoals, and schema modules to other schema modules in the same schema, take place via a mechanism of promises of funding (or 'requests for service,' to be called 'RFS's' from here on). This mechanism relies upon and interacts with ordinary economic attention allocation but also has special properties. Note that we will sometimes say that an Atom "issues" an RFS or "transfers" currency while what we really mean is that some MindAgent working on that Atom issues an RFS or transfers currency.

The logic of these RFS's is as follows. If agent A issues an RFS of value x to agent B, then

1. When B judges it appropriate, B may redeem the note and ask A to transfer currency of value x to B.
2. A may withdraw the note from B at any time.

(There is also a little more complexity here, in that we will shortly introduce the notion of RFS's whose value is defined by a set of constraints. But this complexity does not contradict the two above points.) The total value of the of RFS's possessed by an Atom may be referred to as its "promise."

A rough schematic depiction of this RFS process is given in Figure 24.1.

Now we explain how RFS's may be passed between goals. Given two predicates A and B, if A is being considered as a goal, then B may be considered as a subgoal of A (and A the supergoal of B) if there exists a Link of the form

```
PredictiveImplication B A
```

I.e., achieving B may help to achieve A. Of course, the strength of this link and the temporal characteristics of this link are important in terms of quantifying how strongly and how usefully B is a subgoal of A.

Supergoals (not only top-level ones, aka ubergoals) allocate RFS's to subgoals as follows. Supergoal A may issue a RFS to subgoal B if it is judged that achievement (i.e., predicate satisfaction) of B implies achievement of A. This may proceed recursively: subgoals may allocate RFS's to subsubgoals according to the same justification.

Unlike actual currency, RFS's are not conserved. However, the actual payment of real currency upon redemption of RFS's obeys the conservation of real currency. This means that agents need to be responsible in issuing and withdrawing RFS's. In practice this may be ensured by having agents follow a couple simple rules in this regard.

1. If B and C are two alternatives for achieving A, and A has x units of currency, then A may promise both B and C x units of currency. Whoever asks for a redemption of the promise first, will get the money, and then the promise will be rescinded from the other one.
2. On the other hand, if the achievement of A requires both B and C to be achieved, then B and C may be granted RFS's that are defined by constraints. If A has x units of currency, then B and C receive an RFS tagged with the constraint (B+C<=x). This means that in

EFTA00624261

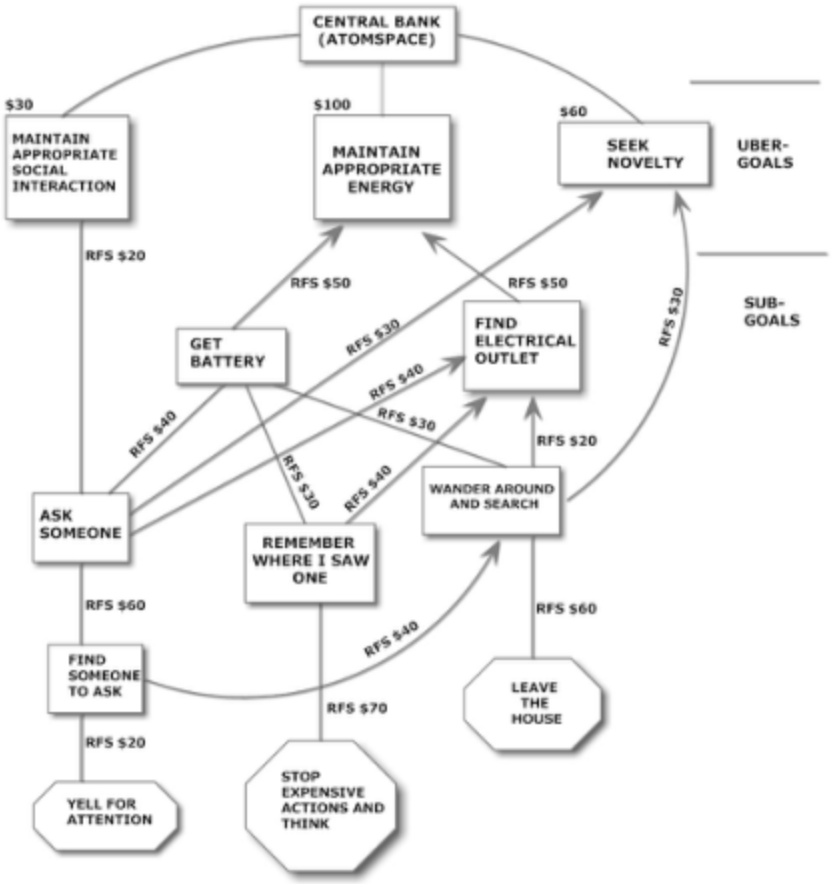


Fig. 24.1: **The RFS Propagation Process**. An illustration of the process via which RFS's propagate from goals to abstract procedures, and finally must get cashed out to pay for the execution of actual concrete procedures that are estimated relatively likely to lead to goal fulfillment.

order to redeem the note, either one of B or C must confer with the other one, so that they can simultaneously request constraint-consistent amounts of money from A.

As an example of the role of constraints, consider the goal of playing fetch successfully (a subgoal of "get reward").... Then suppose it is learned that:

```
ImplicationLink
    SequentialAND
         get_ball
         deliver_ball
    play_fetch
```

where *SequentialAND A B* is the conjunction of A and B but with B occurring after A in time. Then, if play_fetch has \$10 in STICurrency, it may know it has \$10 to spend on a combination

EFTA00624262

of get_ball and deliver_ball. In this case both get_ball and deliver_ball would be given RFS's labeled with the contraint:

```
RFS.get_ball + RFS.deliver_ball <= 10
```

The issuance of RFS's embodying constraints is different from (and generally carried out prior to) the evaluation of whether the constraints can be fulfilled.

An ubergoal may rescind offers of reward for service at any time. And, generally, if a subgoal gets achieved and has not spent all the money it needed, the supergoal will not offer any more funding to the subgoal (until/unless it needs that subgoal achieved again).

As there are no ultimate sources of RFS in OCP besides ubergoals, promise may be considered as a measure of "goal-related importance."

Transfer of RFS's among Atoms is carried out by the GoalAttentionAllocation MindAgent.

24.3 Feasibility Structures

Next, there is a numerical data structure associated with goal Atoms, which is called the feasibility structure. The feasibility structure of an Atom G indicates the feasibility of achieving G as a goal using various amounts of effort. It contains triples of the form (t,p,E) indicating the truth value t of achieving goal G to degree p using effort E. Feasibility structures must be updated periodically, via scanning the links coming into an Atom G; this may be done by a FeasibilityUpdating MindAgent. Feasibility may be calculated for any Atom G for which there are links of the form:

```
Implication
    Execution S
    G
```

for some S. Once a schema has actually been executed on various inputs, its cost of execution on other inputs may be empirically estimated. But this is not the only case in which feasibility may be estimated. For example, if goal G inherits from goal G1, and most children (e.g. subgoals) of G1 are achievable with a certain feasibility, then probably G is achievable with a similar feasibility as well. This allows feasibility estimation even in cases where no plan for achieving G yet exists, e.g. if the plan can be produced via predicate schematization, but such schematization has not yet been carried out.

Feasibility then connects with importance as follows. Important goals will get more STICurrency to spend, thus will be able to spawn more costly schemata. So, the GoalBasedSchemaSelection MindAgent, when choosing which schemata to push into the ActiveSchemaPool, will be able to choose more costly schemata corresponding to goals with more STICurrency to spend.

24.4 Goal Based Schema Selection

Next, the GoalBasedSchemaSelection (GBSS) selects schemata to be placed into the ActiveSchemaPool. It does this by choosing goals G, and then choosing schemata that are alleged to be useful for achieving these goals. It chooses goals via a fitness function that combines promise and feasibility. This involves solving an optimization problem: figuring out how to

EFTA00624263

maximize the odds of getting a lot of goal-important stuff done within the available amount of (memory and space) effort. Potentially this optimization problem can get quite subtle, but initially some simple heuristics are satisfactory. (One subtlety involves handling dependencies between goals, as represented by constraint-bearing RFS's.)

Given a goal, the GBSS MindAgent chooses a schema to achieve that goal via the heuristic of selecting the one that maximizes a fitness function balancing the estimated effort required to achieve the goal via executing the schema, with the estimated probability that executing the schema will cause the goal to be achieved.

When searching for schemata to achieve G, and estimating their effort, one factor to be taken into account is the set of schemata already in the ActiveSchemaPool. Some schemata S may simultaneously achieve two goals; or two schemata achieving different goals may have significant overlap of modules. In this case G may be able to get achieved using very little or no effort (no additional effort, if there is already a schema S in the ActiveSchemaPool that is going to cause G to be achieved). But if G "decides" it can be achieved via a schema S already in the ActiveSchemaPool, then it should still notify the ActiveSchemaPool of this, so that G can be added to S's index (see below). If the other goal G1 that placed S in the ActiveSchemaPool decides to withdraw S, then S may need to hit up G1 for money, in order to keep itself in the ActiveSchemaPool with enough funds to actually execute.

24.4.1 A Game-Theoretic Approach to Action Selection

Min Jiang has observed that, mathematically, the problem of action selection (represented in CogPrime as the problem of goal-based schema selection) can be modeled in terms of game theory, as follows:

- the intelligent agent is one player, the world is another player
- the agent's model of the world lets it make probabilistic predictions of how the world may respond to what the agent does (i.e. to estimate what mixed strategy the world is following, considering the world as a game player)
- the agent itself chooses schema probabilistically, so it's also following a mixed strategy
- so, in principle the agent can choose schema that it thinks will lead to a mixed Nash equilibrium[1]

But the world's responses are very high-dimensional, which means that finding a mixed Nash equilibrium even approximately is a very hard computational problem. Thus, in a sense, the crux of the problem seems to come down to **feature identification**. If the world's response (real or predicted) can be represented as a low-dimensional set of features, then these features can be considered as the world's "move" in the game ... and the game theory problem becomes tractable via approximation schemes. But without the reduction of the world to a low-dimensional set of features, finding the mixed Nash equilbrium even approximately will not be computationally tractable...

Some AI theorists would argue that this division into "feature identification" versus "action selection" is unnecessarily artificial; for instance, Hawkins [HB06] or Arel [ARC09b] might suggest to use a single hierarchical neural network to do both of them. But the brain after all

[1] in game theory, a Nash equilibrium is when no player can do better by unilaterally changing its strategy

EFTA00624264

contains many different regions, with different architectures and dynamics.... In the visual cortex, it seems that feature extraction and object classification are done separately. And it seems that in the brain, action selection has a lot to do with the basal ganglia, whereas feature extraction is done in the cortex. So the neural analogy provides some inspiration for an architecture in which feature identification and action selection are separated.

There is literature discussing numerical methods for calculating approximate Nash equilibria; however, this is an extremely tricky topic in the CogPrime context because action selection must generally be done in real-time. Like perception processing, this may be an area calling for the use of parallel processing hardware. For instance, a neural network algorithm for finding mixed Nash equilibria could be implemented on a GPU supercomputer, enabling rapid real-time action selection based on a reduced-dimensionality model of the world produced by intelligent feature identification.

Consideration of the application of game theory in this context brings out an important point, which is that to do reasonably efficient and intelligent action selection, the agent needs some rapidly-evaluable model of the world, i.e. some way to rapidly evaluate **the predicted response of the world to a hypothetical action by the agent**. In the game theory approach (or any other sufficiently intelligent approach), for the agent to evaluate fitness of a schema-set S for achieving certain goals in a certain context, it has to (explicitly or implicitly) estimate

- how the world will respond if the agent does S
- how the agent could usefully respond to the world's response (call this action-set S_1)
- how the world will respond to the agent doing S_1
- etc.

and so to rapidly evaluate the fitness of S, the agent needs to be able to quickly estimate how the world will respond. This may be done via simulation, or it may be done via inference (which however will rarely be fast enough, unless with a very accurate inference control mechanism), or it may be done by learning some compacted model of the world as represented for instance in a hierarchical neural network.

24.5 SchemaActivation

And what happens with schemata that are actually in the ActiveSchemaPool? Let us assume that each of these schema is a collection of modules (subprograms), connected via ActivationLinks, which have semantics: (ActivationLink A B) means that if the schema that placed module A in the schema pool is to be completed, then after A is activated, B should be activated. (We will have more to say about schemata, and their modularization, in Chapter 25.)

When a goal places a schema in the ActiveSchemaPool, it grants that schema an RFS equal in value to the total or some fraction of the promissory+real currency it has in its possession. The heuristics for determining how much currency to grant may become sophisticated; but initially we may just have a goal give a schema all its promissory currency; or in the case of a top-level supergoal, all its actual currency.

When a module within a schema actually executes, then it must redeem some of its promissory currency to turn it into actual currency, because executing costs money (paid to the Lobe).

EFTA00624265

Once a schema is done executing, if it hasn't redeemed all its promissory currency, it gives the remainder back to the goal that placed it in the ActiveSchemaPool.

When a module finishes executing, it passes promissory currency to the other modules to which it points with ActivationLinks.

The network of modules in the ActiveSchemaPool is a digraph (whose links are ActivationLinks), because some modules may be shared within different overall schemata. Each module must be indexed via which schemata contain it, and each schema must be indexed via which goal(s) want it in the ActiveSchemaPool.

24.6 GoalBasedSchemaLearning

Finally, we have the process of trying to figure out how to achieve goals, i.e. trying to learn links between ExecutionLinks and goals G. This process should be focused on goals that have a high importance but for which feasible achievement-methodologies are not yet known. Predicate schematization is one way of achieving this; another is MOSES procedure evolution.

EFTA00624266

Chapter 25
Integrative Procedure Evaluation

25.1 Introduction

Procedural knowledge must be learned, an often subtle and difficult process – but it must also be enacted. Procedure enaction is not as tricky a topic as procedure learning, but still is far from trivial, and involves the real-time interaction of procedures, during the course of execution, with other knowledge. In this brief chapter we explain how this process may be most naturally and flexibly carried out, in the context of CogPrime's representation of procedures as programs ("Combo trees").

While this may seem somewhat of a "mechanical", implementation-level topic, it also involves some basic conceptual points, on which CogPrime as an AGI design does procedure evaluation fundamentally differently from narrow-AI systems or conventional programming language interpreters. Basically, what makes CogPrime Combo tree evaluation somewhat subtle is due to the interfacing between the Combo evaluator itself and the rest of the CogPrime system.

In the CogPrime design, Procedure objects (which contain Combo trees, and are associated with ProcedureNodes) are evaluated by ProcedureEvaluator objects. Different ProcedureEvaluator objects may evaluate the same Combo tree in different ways. Here we explain these various sorts of evaluation – how they work and what they mean.

25.2 Procedure Evaluators

In this section we will mention three different ProcedureEvaluators:

- Simple procedure evaluation
- Effort-based procedure evaluation, which is more complex but is required for integration of inference with procedure evaluation
- Adaptive evaluation order based procedure evaluation

In the following section we will delve more thoroughly into the interactions between inference and procedure evaluation.

Another related issue is the modularization of procedures. This issue however is actually orthogonal to the distinction between the three ProcedureEvaluators mentioned above. Modularity simply requires that particular nodes within a Combo tree be marked as "module roots",

EFTA00624268

so that they may be extracted from the Combo tree as a whole and treated as separate modules (called differently, sub-routines), if the ExecutionManager judges this appropriate.

25.2.1 Simple Procedure Evaluation

The SimpleComboTreeEvaluator simply does Combo tree evaluation as described earlier. When an Atom is encountered, it looks into the AtomTable to evaluate the object.

In the case that a Schema refers to an ungrounded SchemaNode (that is not defined by a ComboTree as defined in Chapter 19), and an appropriate EvaluationLink value isn't in the AtomTable, there's an evaluation failure, and the whole procedure evaluation returns the truth value $\langle .5, 0\rangle$: i.e., a zero-weight-of-evidence truth value, which is equivalent essentially to returning no value.

In the case that a Predicate refers to an ungrounded PredicateNode, and an appropriate EvaluationLink isn't in the AtomTable, then some very simple "default thinking" is done, and it is assigned the truth value of the predicate on the given arguments to be the TruthValue of the corresponding PredicateNode (which is defined as the mean truth value of the predicate across all arguments known to CogPrime.)

25.2.2 Effort Based Procedure Evaluation

The next step is to introduce the notion of "effort" the amount of effort that the CogPrime system must undertake in order to carry out a procedure evaluation. The notion of effort is encapsulated in Effort objects, which may take various forms. The simplest Effort objects measure only elapsed processing time; more advanced Effort objects take into consideration other factors such as memory usage.

An effort-based Combo tree evaluator keeps a running total of the effort used in evaluating the Combo tree. This is necessary if inference is to be used to evaluate Predicates, Schema, Arguments, etc. Without some control of effort expenditure, the system could do an arbitrarily large amount of inference to evaluate a single Atom.

The matter of evaluation effort is nontrivial because in many cases a given node of a Combo tree may be evaluated in more than one way, with a significant effort differential between the different methodologies. If a Combo tree Node refers to a predicate or schema that is very costly to evaluate, then the ProcedureEvaluator managing the evaluation of the Combo tree must decide whether to evaluate it directly (expensive) or estimate the result using inference (cheaper but less accurate). This decision depends on how much effort the ProcedureEvaluator has to play with, and what percentage of this effort it finds judicious to apply to the particular Combo tree Node in question.

In the relevant prototypes we built within OpenCog, this kind of decision was made based on some simple heuristics inside ProcedureEvaluator objects. However, it's clear that, in general, more powerful intelligence must be applied here, so that one needs to have ProcedureEvaluators that - in cases of sub-procedures that are both important and highly expensive - use PLN inference to figure out how much effort to assign to a given subproblem.

EFTA00624269

The simplest useful kind of effort-based Combo tree evaluator is the EffortIntervalComboTreeEvaluator, which utilizes an Effort object that contains three numbers (yes, no, max). The yes parameter tells it how much effort should be expended to evaluate an Atom if there is a ready answer in the AtomTable. The no parameter tells it how much effort should be expended in the case that there is not a ready answer in the AtomTable. The max parameter tells it how much effort should be expended, at maximum, to evaluate all the Atoms in the Combo tree, before giving up. Zero effort, in the simplest case, may be heuristically defined as simply looking into the AtomTable – though in reality this does of course take effort, and a more sophisticated treatment would incorporate this as a factor as well.

Quantification of amounts of effort is nontrivial, but a simple heuristic guideline is to assign one unit of effort for each inference step. Thus, for instance,

- (yes, no, max) = (0,5,1000) means that if an Atom can be evaluated by AtomTable lookup, this is done, but if AtomTable lookup fails, a minimum of 5 inference steps are done to try to do the evaluation. It also says that no more than 1000 evaluations will be done in the course of evaluating the Combo tree.
- (yes, no, max) = (3,5,1000) says the same thing, but with the change that even if evaluation could be done by direct AtomTable lookup, 3 inference steps are tried anyway, to try to improve the quality of the evaluation.

25.2.3 Procedure Evaluation with Adaptive Evaluation Order

While tracking effort enables the practical use of inference within Combo tree evaluation, if one has truly complex Combo trees, then a higher degree of intelligence is necessary to guide the evaluation process appropriately. The order of evaluation of a Combo tree may be determined adaptively, based on up to three things:

- The history of evaluation of the Combo tree
- Past history of evaluation of other Combo trees, as stored in a special AtomTable consisting only of relationships about Combo tree-evaluation-order probabilities
- New information entering into CogPrime's primary AtomTable during the course of evaluation

ProcedureEvaluator objects may be selected at runtime by cognitive schemata, and they may also utilize schemata and MindAgents internally. The AdaptiveEvaluationOrderComboTreeEvaluator is more complex than the other ProcedureEvaluators discussed, and will involve various calls to CogPrime MindAgents, particularly those concerned with PLN inference. WIKISOURCE:ProcedureExecutionDetails

25.3 The Procedure Evaluation Process

Now we give a more thorough treatment of the procedure evaluation process, as embodied in the effort-based or adaptive-evaluation-order evaluators discussed above. The process of procedure evaluation is somewhat complex, because it encompasses three interdependent processes:

EFTA00624270

- The mechanics of procedure evaluation, which in the CogPrime design involves traversing Combo trees in an appropriate order. When a Combo tree node referring to a predicate or schema is encountered during Combo tree traversal, the process of predicate evaluation or schema execution must be invoked.
- The evaluation of the truth values of predicates - which involves a combination of inference and (in the case of grounded predicates) procedure evaluation.
- The computation of the truth values of schemata - which may involve inference as well as procedure evaluation.

We now review each of these processes.

25.3.1 Truth Value Evaluation

What happens when the procedure evaluation process encounters a Combo tree Node that represents a predicate or compound term? The same thing as when some other CogPrime process decides it wants to evaluate the truth value of a PredicateNode or CompoundTermNode: the generic process of predicate evaluation is initiated.

This process is carried out by a TruthValueEvaluator object. There are several varieties of TruthValueEvaluator, which fall into the following hierarchy:

```
TruthValueEvaluator
   DirectTruthValueEvaluator (abstract)
       SimpleDirectTruthValueEvaluator
   InferentialTruthValueEvaluator (abstract)
       SimpleInferentialTruthValueEvaluator
   MixedTruthValueEvaluator
```

A DirectTruthValueEvaluator evaluates a grounded predicate by directly executing it on one or more inputs; an InferentialTruthValueEvaluator evaluates via inference based on the previously recorded, or specifically elicited, behaviors of other related predicates or compound terms. A MixedTruthValueEvaluator contains references to a DirectTruthValueEvaluator and an InferentialTruthValueEvaluator, and contains a weight that tells it how to balance the outputs from the two.

Direct truth value evaluation has two cases. In one case, there is a given argument for the predicate; then, one simply plugs this argument in to the predicate's internal Combo tree, and passes the problem off to an appropriately selected ProcedureEvaluator. In the other case, there is no given argument, and one is looking for the truth value of the predicate *in general*. In this latter case, some estimation is required. It is not plausible to evaluate the truth value of a predicate on every possible argument, so one must sample a bunch of arguments and then record the resulting probability distribution. A greater or fewer number of samples may be taken, based on the amount of effort that's been allocated to the evaluation process. It's also possible to evaluate the truth value of a predicate in a given context (information that's recorded via embedding in a ContextLink); in this case, the random sampling is restricted to inputs that lie within the specified context.

On the other hand, the job of an InferentialTruthValueEvaluator is to use inference rather than direct evaluation to guess the truth value of a predicate (sometimes on a particular argument, sometimes in general). There are several different control strategies that may be applied here, depending on the amount of effort allocated. The simplest strategy is to rely on analogy,

EFTA00624271

simply searching for similar predicates and using their truth values as guidance. (In the case where a specific argument is given, one searches for similar predicates that have been evaluated on similar arguments.) If more effort is available, then a more sophisticated strategy may be taken. Generally, an InferentialTruthValueEvaluator may invoke a SchemaNode that embodies an inference control strategy for guiding the truth value estimation process. These SchemaNodes may then be learned like any others.

Finally, a MixedTruthValueEvaluator operates by consulting a DirectTruthValueEvaluator and/or an InferentialTruthValueEvaluator as necessary, and merging the results. Specifically, in the case of an ungrounded PredicateNode, it simply returns the output of the InferentialTruthValueEvaluator it has chosen. But in the case of a GroundedPredicateNode, it returns a weighted average of the directly evaluated and inferred values, where the weight is a parameter. In general, this weighting may be done by a SchemaNode that is selected by the MixedTruthValueEvaluator; and these schemata may be adaptively learned.

25.3.2 Schema Execution

Finally, schema execution is handled similarly to truth value evaluation, but it's a bit simpler in the details. Schemata have their outputs evaluated by SchemaExecutor objects, which in turn invoke ProcedureEvaluator objects. We have the hierarchy:

```
SchemaExecutor
   DirectSchemaExecutor (abstract)
       SimpleDirectSchemaExecutor
   InferentialSchemaExecutor (abstract)
       SimpleInferentialSchemaExecutor
    MixedSchemaExecutor
```

A DirectSchemaExecutor evaluates the output of a schema by directly executing it on some inputs; an InferentialSchemaExecutor evaluates via inference based on the previously recorded, or specifically elicited, behaviors of other related schemata. A MixedSchemaExecutor contains references to a DirectSchemaExecutor and an InferentialSchemaExecutor, and contains a weight that tells it how to balance the outputs from the two (not always obvious, depending on the output type in question).

Contexts may be used in schema execution, but they're used only indirectly, via being passed to TruthValueEvaluators used for evaluating truth values of PredicateNodes or CompoundTermNodes that occur internally in schemata being executed.

EFTA00624272

Section III
Perception and Action

EFTA00624274

Chapter 26
Perceptual and Motor Hierarchies

26.1 Introduction

Having discussed declarative, attentional, intentional and procedural knowledge, we are left only with sensorimotor and episodic knowledge to complete our treatment of the basic CogPrime "cognitive cycle" via which a CogPrime system can interact with an environment and seek to achieve its goals therein.

The cognitive cycle in its most basic form leaves out the most subtle and unique aspects of CogPrime, which all relate to learning in various forms. But nevertheless it is the foundation on which CogPrime is built, and within which the various learning processes dealing with the various forms of memory all interact. The CogPrime cognitive cycle is more complex in many respects than it would need to be if not for the need to support diverse forms of learning. And this learning-driven complexity is present to some extent in the contents of the present chapter as well. If learning weren't an issue, perception and actuation could more likely be treated as wholly (or near-wholly) distinct modules, operating according to algorithms and structures independent of cognition. But our suspicion is that this sort of approach is unlikely to be adequate for achieving high levels of perception and action capability under real-world conditions. Instead, we suspect, it's necessary to create perception and action processes that operate fairly effectively on their own, but are capable of cooperating with cognition to achieve yet higher levels of functionality.

And the benefit in such an approach goes both ways. Cognition helps perception and actuation deal with difficult cases, where the broad generalization that is cognition's specialty is useful for appropriately biasing perception and actuation based on subtle environmental regularities. And, the patterns involved in perception and actuation help cognition, via supplying a rich reservoir of structures and processes to use as analogies for reasoning and learning at various levels of abstraction. The prominence of visual and other sensory metaphors in abstract cognition is well known [Arn69, Gar00]; and according to Lakoff and Nunez [LN00] even pure mathematics is grounded in physical perception and action in very concrete ways.

We begin by discussing the perception and action mechanisms required to interface CogPrime with an agent operating in a virtual world. We then turn to the more complex mechanisms needed to effectively interface CogPrime with a robot possessing vaguely humanoid sensors and actuators, focusing largely on vision processing. This discussion leads up to deeper discussions in Chapters 27, 28 and 29 where we describe in detail the strategy that would be used to integrate

EFTA00624276

CogPrime with the DeSTIN framework for AGI perception/action (which was described in some detail in Chapter 4 of Part 1).

In terms of the integrative cognitive architecture presented in Chapter 5, the material presented in the chapters in this section has mostly to do with the perceptual and motor hierarchies, also touching on the pattern recognition and imprinting processes that play a role in the interaction between these hierarchies and the conceptual memory. The commitment to a hierarchical architecture for perception and action is not critical for the CogPrime design as a whole – one could build a CogPrime with non-hierarchical perception and action modules, and the rest of the system would be about the same. The role of hierarchy here is a reflection of the obvious hierarchical structure of the everyday human environment, and of the human body. In a world marked by hierarchical structure, a hierarchically structured perceptual system is advantageous. To control a body marked by hierarchical structure, an hierarchically structured action system is advantageous. It would be possible to create a CogPrime system without this sort of in-built hierarchical structure, and have it gradually self-adapt in such a way as to grow its own internal hierarchical structure, based its experience in the world. However, this might be a case of pushing the "experiential learning" perspective too far. The human brain definitely has hierarchical structure built into it; it doesn't need to learn to experience the world in hierarchical terms; and there seems to be no good reason to complicate an AGI's early development phase by forcing it to learn the basic facts of the world's and its body's hierarchality.

26.2 The Generic Perception Process

We have already discussed the generic action process of CogPrime, in Chapter 25 on procedure evaluation. Action sequences are generated by Combo programs, which execute primitive actions, including those corresponding to actuator control signals as well as those corresponding to, say, mathematical or cognitive operations. In some cases the actuator control signals may directly dictate movements; in other cases they may supply inputs and/or parameters to other software (such as DeSTIN, in the integrated CogBot architecture to be described below).

What about the generic perception process? We distinguish *sensation* from *perception*, in a CogPrime context, by defining

- *perception* as what occurs when some signal from the outside world registers itself in either: a CogPrime Atom, or some other sort of node (e.g. a DeSTIN node) that is capable of serving as the target of a CogPrime Link.
- *sensation* as any "preprocessing" that occurs between the impact of some signal on some sensor, and the creation of a corresponding perception

Once perceptual Atoms have been created, various perceptual MindAgents comes into play, taking perceptual schemata (schemata whose arguments are perceptual nodes or relations therebetween) and applying them to Atoms recently created (creating appropriate ExecutionLinks to store the results). The need to have special, often modality-specific perception MindAgents to do this, instead of just leaving it to the generic SchemaExecution MindAgent, has to do with computational efficiency, scheduling and parameter settings. Perception MindAgents are doing schema execution urgently, and doing it with parameter settings tuned for perceptual processing. This means that, except in unusual circumstances, newly received stimuli will be processed immediately by the appropriate perceptual schemata.

EFTA00624277

Some newly formed perceptual Atoms will have links to existing atoms, ready-made at their moment of creation. CharacterInstanceNodes and NumberInstanceNodes are examples; they are born linked to the appropriate CharacterNodes and NumberNodes. Of course, atoms representing perceived relationships, perceived groupings, etc., will not have ready-made links and will have to grow such links via various cognitive processes. Also, the ContextFormation MindAgent looks at perceptual atom creation events and creates Context Nodes accordingly; and this must be timed so that the Context Nodes are entered into the system rapidly, so that they can be used by the processes doing initial-stage link creation for new perceptual Atoms.

In a full CogPrime configuration, newly created perceptual nodes and perceptual schemata may reside in a special perception-oriented Units, so as to ensure that perceptual processes occur rapidly, not delayed by slower cognitive processes.

26.2.1 The ExperienceDB

Separate from the ordinary perception process, it may also valuable for there to be a direct route from the system's sensory sources to a special "ExperienceDB" database that records all of the system's experience. This does not involve perceptual schemata at all, nor is it left up to the sensory source; rather, it is carried out by the CogPrime server at the point where it receives input from the sensory source. This experience database is a record of what the system has seen in the past, and may be mined by the system in the future for various purposes. The creation of new perceptual atoms may also be stored in the experience database, but this must be handled with care as it can pose a large computational expense; it will often be best to store only a subset of these.

Obviously, such an ExperienceDB is something that has no correlate in the human mind/brain. This is a case where CogPrime takes advantage of the non-brainlike properties of its digital computer substrate. The CogPrime perception process is intended to work perfectly well without access to the comprehensive database of experiences potentially stored in the ExperienceDB. However, a complete record of a mind's experience is a valuable thing, and there seems no reason for the system not to exploit it fully. Advantages like this allow the CogPrime system to partially compensate for its lack of some of the strengths of the human brain as an AI platform, such as massive parallelism.

26.3 Interfacing CogPrime with a Virtual Agent

We now discuss some of the particularities of connecting CogPrime to a virtual world (such as Second Life, Multiverse, or Unity3D, to name some of the virtual world / gaming platforms to which OpenCog has already been connected in practice).

EFTA00624278

26.3.1 Perceiving the Virtual World

The most complex, high-bandwidth sensory data coming in from a typical virtual world is visual data, so that will be our focus here. We consider three modes in which a virtual world may present visual data to CogPrime (or any other system):

- *Object vision*: CogPrime receives information about polygonal objects and their colors, textures and coordinates (each object is a set of contiguous polygons, and sometimes objects have "type" information, e.g. cube or sphere)
- *Polygon vision*: CogPrime receives information about polygons and their colors, textures and coordinates
- *Pixel vision*: CogPrime receives information about pixels and their colors and coordinates

In each case, coordinates may be given either in "world coordinates" or in "relative coordinates" (relative to the gaze). This distinction is not a huge deal since within an architecture like CogPrime, supplying schemata for coordinate transformation is trivial; and, even if treated as a machine learning task, this sort of coordinate transformation is not very difficult to learn. Our current approach is to prefer relative coordinates, as this approach is more natural in terms of modern Western human psychology; but we note that in some other cultures world coordinates are preferred and considered more psychologically natural.

Currently we have not yet done any work with pixel vision in virtual worlds. We have been using object vision for most of our experiments, and consider a combination of polygon vision and object vision as the "right" approach for early AGI experiments in a virtual worlds context. The problem with pure object vision is that it removes the possibility for CogPrime to understand object segmentation. If, for instance, CogPrime perceives a person as a single object, then how can it recognize a head as a distinct sub-object? Feeding the system a pre-figured hierarchy of objects, sub-objects and so forth seems inappropriate in the context of an experiential learning system. On the other hand, the use of polygon vision instead of pixel vision seems to meet no such objections. This may take different forms in different platforms. For instance, in our work with a Minecraft-like world in the Unity3D environment, we have relied heavily on virtual objects made of blocks, in which case the polygons of most interest are the faces of the blocks.

Momentarily sticking with the object vision case for simplicity, examples of the perceptions emanating from the virtual world perceptual preprocessor into CogPrime are things like:

1. I am at world-coordinates $W
2. Object with metadata $M is at world-coordinates $W
3. Part of object with metadata $M is at world-coordinates $W
4. Avatar with metadata $M is at world-coordinates $W
5. Avatar with metadata $M is carrying out animation $A
6. Statements in natural language, from the pet owner

The perceptual preprocessor takes these signals and translates them into Atoms, making use of the special Atomspace mechanisms for efficiently indexing spatial and temporal information (the and) as appropriate.

EFTA00624279

26.3.1.1 Transforming Real-World Vision into Virtual Vision

One approach to enabling CogPrime to handle visual data coming from the real world is to transform this data into data of the type CogPrime sees in the virtual world. While this is not the approach we are taking in our current work, we do consider it a viable strategy, and we briefly describe it here.

One approach along these lines would involve multiple phases:

- Use a camera eye and a LiDAR (Light Detection And Ranging, used for high-resolution topographic mapping) sensor in tandem, so as to avoid having to deal with stereo vision
- Using the above two inputs, create a continuous 3D contour map of the perceived visual world
- Use standard mathematical transforms to polygon-ize the 3D contour map into a large set of small polygons
- Use heuristics to merge together the small polygons, obtaining a smaller set of larger polygons (but retaining the large set of small polygons for the system to reference in cases where a high level of detail is necessary)
- Feed the polygons into the perceptual pattern mining subsystem, analogously to the polygons that come in from virtual-world

In this approach, preprocessing is used to make the system see the physical world in a manner analogous to how it sees the virtual-world world. This is quite different from the DeSTIN-based approach to CogPrime vision that we will discuss in Chapter 28, but may well also be feasible.

26.3.2 Acting in the Virtual World

Complementing the perceptual preprocessor is the action postprocessor: code that translates the actions and action-sequences generated by CogPrime into instructions the virtual world can understand (such as "launch thus-and-thus animation"). Due to the particularities of current virtual world architectures, the current OpenCogPrime system carries out actions via executing pre-programmed high-level procedures, such as "move forward one step", "bend over forward" and so forth. Example action commands are:

1. Move ($D, $S) : $D is a distance, $S is a speed
2. Turn ($A, $S) : $A is an angle, $S is a speed
3. Pitch ($A, $S) : turn vertically up/down... [for birds only]
4. Jump ($D, $H, $S) : $H is a maximum height, at the center of the jump
5. Say ($T), $T is text : for agents with linguistic capability, which is not enabled in the current version
6. pick up($O) : $O is an object
7. put down($O)

This is admittedly a crude approach, and if a robot simulator rather than a typical virtual world were used, it would be possible for CogPrime to emanate detailed servomotor control commands rather than high-level instructions such as these. However, as noted in Chapter 16 of Part 1, at the moment there is no such thing as a "massive multiplayer robot simulator," and so the choice is between a multi-participant virtual environment (like the Multiverse environment

EFTA00624280

currently used with the PetBrain) or a small-scale robot simulator. Our experiments with virtual worlds so far have used the high-level approach described here; but we are also experimenting with using physical robots and corresponding simulators, as will be described below.

26.4 Perceptual Pattern Mining

Next we describe how perceptual pattern mining may be carried out, to recognize meaningful structures in the stream of data produced via perceiving a virtual or physical world.

In this subsection we discuss the representation of knowledge, and then in the following subsection we discuss the actual mining. We discuss the process in the context of virtual-world perception as outlined above, but the same processes apply to robotic perception, whether one takes the "physical world as virtual world" approach described above or a different sort of approach such as the DeSTIN hybridization approach described below.

26.4.1 Input Data

First, we may assume that each perception is recorded as set of "transactions", each of which is of the form

```
Time, 3D coordinates, object type
```

or

```
Time, 3D coordinates, action type
```

Each transaction may also come with an additional list of (attribute, value) pairs, where the list of attributes is dependent upon the object or action type. Transactions are represented as Atoms, and don't need to be a specific Atom type - but are referred to here by the special name *transactions* simply to make the discussion clear.

Next, define a transaction template as a transaction with location and time information set to *wild cards* - and potentially, some other attributes set to wild cards. (These are implemented in terms of Atoms involving VariableNodes.)

For instance, some transaction templates in the current virtual-world might be informally represented as:

- Reward
- Red cube
- kick
- move_forward
- Cube
- Cube, size 5
- me
- Teacher

EFTA00624281

26.4.2 Transaction Graphs

Next we may conceive a transaction graph, whose nodes are transactions and whose links are labeled with labels like after, SimAND, SSeqAND (short for SimultaneousSequentialAND), near, in_front_of, and so forth (and whose links are weighted as well).

We may also conceive a transaction template graph, whose nodes are transaction templates, and whose links are the same as in the transaction graph. Examples of transaction template graphs are

```
near(Cube, Teacher)
```

```
SSeqAND(move_forward, Reward)
```

where Cube, Teacher, etc are transaction templates since Time and 3D coordinates are left unspecified.

And finally, we may conceive a transaction template relationship graph (TTRG), whose nodes may be any of: transactions; transaction templates; basic spatiotemporal predicates evaluated at tuples of transactions or transaction templates. For instance

```
SimAND(near(Cube, Teacher), above(Cube, Chair))
```

26.4.3 Spatiotemporal Conjunctions

Define a temporal conjunction as a conjunction involving SimultaneousAND and SequentialAND operators (including SSeqAND as a special case of SeqAND: the special case that interests us in the short term). The conjunction is therefore ordered, e.g.

```
A SSeqAND B SimAND C SSeqAND D
```

We may assume that the order of operations favors SimAND, so that no parenthesizing is necessary.

Next, define a basic spatiotemporal conjunction as a temporal conjunction that conjoins terms that are either

- transactions, or
- transaction templates, or
- basic spatiotemporal predicates applied to tuples of transactions or transaction templates

I.e. a basic spatiotemporal conjunction is a temporal conjunction of nodes from the transaction template relationship graph.

An example would be:

```
(hold ball) SimAND ( near(me, teacher) ) SSeqAND Reward
```

This assumes that the *hold* action has an attribute that is the type of object held, so that

```
hold ball
```

in the above temporal conjunction is a shorthand for the transaction template specified by

```
action type: hold
```

```
object_held_type: ball
```

This example says that if the agent is holding the ball and is near the teacher then shortly after that, the agent will get a reward.

EFTA00624282

26.4.4 The Mining Task

The perceptual mining task, then, is to find basic spatiotemporal conjunctions that are *interesting*. What constitutes interestingness is multifactorial, and includes.

- involves important Atoms (e.g. Reward)
- has a high temporal cohesion (i.e. the strength of the time relationships embodied in the SimAND and SeqAND links is high)
- has a high spatial cohesion (i.e. the near() relationships have high strength)
- has a high frequency
- has a high surprise value (its frequency is far from what would be predicted by its component sub-conjunctions)

Note that a conjunction can be interesting without satisfying all these criteria; e.g. if it involves something important and has a high temporal cohesion, we want to find it regardless of its spatial cohesion.

In preliminary experiments we have worked with a provisional definition of"interestingness" as the combination of frequency and temporal cohesion, but this must be extended; and one may even wish to have the combination function optimized over time (slowly) where the fitness function is defined in terms of the STI and LTI of the concepts generated.

26.4.4.1 A Mining Approach

One tractable approach to perceptual pattern mining is greedy and iterative, involving the following steps:

1. Build an initial transaction template graph G
2. Greedily mine some interesting basic spatiotemporal conjunctions from it, adding each interesting conjunction found as a new node in G (so that G becomes a transaction template relationship graph), repeating step 2 until boredom results or time runs out

The same TTRG may be maintained over time, but of course will require a robust forgetting mechanism once the history gets long or the environment gets nontrivially complex.

The greedy mining step may involve simply grabbing SeqAND or SimAND links with probability determined by the (importance and/or interestingness) of their targets, and the probabilistic strength and temporal strength of the temporal AND relationship, and then creating conjunctions based on these links (which then become new nodes in the TTRG, so they can be built up into larger conjunctions).

26.5 The Perceptual-Motor Hierarchy

The perceptual pattern mining approach described above is "flat," in the sense that it simply proposes to recognize patterns in a stream of perceptions, without imposing any kind of explicitly hierarchical structure on the pattern recognition process or the memory of perceptual patterns. This is different from how the human visual system works, with its clear hierarchical

EFTA00624283

structure, and also different from many contemporary vision architectures, such as DeSTIN or Hawkins' Numenta system which also utilizes hierarchical neural networks.

However, the approach described above may be easily made hierarchical within the CogPrime architecture, and this is likely the most effective way to deal with complex visual scenes. Most simply, in this approach, a hierarchy may be constructed corresponding to different spatial regions, within the visual field. The RegionNodes at the lowest level of the hierarchy correspond to small spatial regions, the ones at the next level up correspond to slightly larger spatial regions, and so forth. Each RegionNode also correspond to a certain interval of time, and there may be different RegionNodes corresponding to the same spatial region but with different time-durations attached to them. RegionNodes may correspond to overlapping rather than disjoint regions.

Within each region mapped by a RegionNode, then, perceptual pattern mining as defined in the previous section may occur. The patterns recognized in a region are linked to the corresponding RegionNode - and are then fed as inputs to the RegionNodes corresponding to larger, encompassing regions; and as suggestions-to-guide-pattern-recognition to nearby RegionNodes on the same level. This architecture involves the fundamental hierarchical structure/dynamic observed in the human visual cortex. Thus, the hierarchy incurs a dynamic of patterns-within-patterns-within-patterns, and the heterarchy incurs a dynamic of patterns-spawning-similar-patterns.

Also, patterns found in a RegionNode should be used to bias the pattern-search in the RegionNodes corresponding to smaller, contained regions: for instance, if many of the sub-regions corresponding to a certain region have revealed parts of a face, then the pattern-mining processes in the remaining sub-regions may be instructed to look for other face-parts.

This architecture permits the hierarchical dynamics utilized in standard hierarchical vision models, such as Jeff Hawkins' and other neural net models, but within the context of CogPrime's pattern-mining approach to perception. It is a good example of the flexibility intrinsic to the CogPrime architecture.

Finally, why have we called it a perceptual-*motor* hierarchy above? This is because, due to the embedding of the perceptual hierarchy in CogPrime's general Atom-network, the percepts in a certain region will automatically be linked to actions occurring in that region. So, there may be some perception-cognition-action interplay specific to a region, occurring in parallel with the dynamics in the hierarchy of multiple regions. Clearly this mirrors some of the complex dynamics occurring in the human brain, and is also reflected in the structure of sophisticated perceptual-motor approaches like DeSTIN, to be discussed below.

26.6 Object Recognition from Polygonal Meshes

Next we describe a more specific perceptual pattern recognition algorithm – a strategy for identifying objects in a visual scene that is perceived as a set of polygons. It is not a thoroughly detailed algorithmic approach, but rather a high-level description of how this may be done effectively within the CogPrime design. It is offered here largely as an illustration of how specialized perceptual data processing algorithms may be designed and implemented within the CogPrime framework.

EFTA00624284

We deal here with an agent whose perception of the world, at any point in time, is understood to consist of a set of polygons, each one described in terms of a list of corners. The corners may be assumed to be described in coordinates relative to the viewing eye of the agent.

What we mean by "identifying objects" here is something very simple. We don't mean identifying that a particular object is a chair, or is Ben's brown chair, or anything like that - we simply mean identifying that a given collection of polygons is meaningfully grouped into an object. That is the task considered here. The object could be a single block, it could be a person, or it could be a tower of blocks (which appears as a single object until it is taken apart).

Of course, not all approaches to polygon-based vision processing would require this sort of phase: it would be possible, as an alternative, to simply compare the set of polygons in the visual field to a database of prior experience and then do object identification (in the present sense) based on this database-comparison. But in the approach described in this section, one begins instead with an automated segmentation of the set of perceived polygons into a set of objects.

26.6.1 Algorithm Overview

The algorithm described here falls into three stages:

1. Recognizing PersistentPolygonNodes (PPNodes) from PolygonNodes.
2. Creating Adjacency Graphs from PPNodes.
3. Clustering in the Adjacency Graph.

Each of these stages involves a bunch of details, not all of which have been fully resolved: this section just gives a conceptual overview.

We will speak in terms of objects such as PolygonNode, PPNode and so forth, because inside the CogPrime AI engine, observed and conceived entities are represented as nodes in an graph. However, this terminology is not very important here, and what we call a PolygonNode here could just as well be represented in a host of other ways, within the overall CogPrime framework.

26.6.2 Recognizing PersistentPolygonNodes (PPNodes) from PolygonNodes

A PolygonNode represents a polygon observed at a point in time. A PPNode represents a series of PolygonNodes that are heuristically guessed to represent the same PolygonNode at different moments in time.

Before "object permanence" is learned, the heuristics for recognizing PPNodes will only work in the case of a persistent polygon that, over an interval of time, is experiencing relative motion within the visual field, but is never leaving the visual field. For example some reasonable heuristics are: If P1 occurs at time t, P2 occurs at time s where s is very close to t, and P1 are similar in shape, size and color and position, then P1 and P2 should be grouped together into the same PPNode.

EFTA00624285

More advanced heuristics would deal carefully with the case where some of these similarities did not hold, which would allow us to deal e.g. with the case where an object was rapidly changing color.

In the case where the polygons are coming from a simulation world like OpenSim, then from our positions as programmers and world-masters, we can see that what a PPNode is supposed to correspond to is a certain side of a certain OpenSim object; but it doesn't appear immediately that way to CogPrime when controlling an agent in OpenSim since CogPrime isn't perceiving OpenSim objects, it's perceiving polygons. On the other hand, in the case where polygons are coming from software that postprocesses the output of a LiDAR based vision system, then the piecing together of PPNodes from PolygonNodes is really necessary.

26.6.3 Creating Adjacency Graphs from PPNodes

Having identified PPNodes, we may then draw a graph between PPNodes, a PPGraph (also called an "Adjacency Graph"), wherein the links are AdjacencyLinks (with weights indicating the degree to which the two PPNodes tend to be adjacent, over time). A more refined graph might also involve SpatialCoordinationLinks (with weights indicating the degree to which the vector between the centroids of the two PPNodes tends to be consistent over time).

We may then use this graph to do object identification:

- First-level objects may be defined as clusters in the graph of PPNodes.
- One may also make a graph between first-level objects, an ObjectGraph with the same kinds of links as in the PPGraph. Second-level objects may be defined as clusters in the ObjectGraph.

The "strength" of an identified object may be assigned as the "quality" of the cluster (measured in terms of how tight the cluster is, and how well separated from other clusters.)

As an example, consider a robot with two parts: a body and a head. The whole body may have a moderate strength as a first-level object, but the head and body individually will have significantly greater strengths as first-level objects. On the other hand, the whole body should have a pretty strong strength as a second-level object.

It seems convenient (though not necessary) to have a PhysicalObjectNode type to represent the objects recognized via clustering; but the first versus second level object distinction should not need to be made on the Atom type level.

Building the adjacency graph requires a mathematical formula defining what it means for two PPNodes to be adjacent. Creating this formula may require a little tuning. For instance, the adjacency between two PPNodes PP1 and PP2 may be defined as the average over time of the adjacency of the PolygonNodes PP1(t) and PP2(t) observed at each time t. (A $p'th$ power average[1] may be used here, and different values of p may be tried.) Then, the adjacency between two (simultaneous) PolygonNodes P1 and P2 may be defined as the average over all x in P1 of the minimum over all y in P2 of sim(x,y), where sim(,) is an appropriately scaled similarity function. This latter average could arguably be made a maximum; or perhaps even better a $p'th$ power average with large p, which approximates a maximum.

[1] the $p'th$ power average is defined as $\sqrt[p]{\sum X^p}$

EFTA00624286

26.6.4 Clustering in the Adjacency Graph.

As noted above, the idea is that objects correspond to clusters in the adjacency graph. This means we need to implement some hierarchical clustering algorithm that is tailored to find clusters in symmetric weighted graphs. Probably some decent algorithms of this character exist, if not it would be fairly easy to define one, e.g. by mapping some standard hierarchical clustering algorithm to deal with graphs rather than vectors.

Clusters will then be mapped into PhysicalObjectNodes, interlinked appropriately via PhysicalPartLinks and AdjacencyLinks. (E.g. there would be a PhysicalPartLink between the PhysicalObjectNode representing a head and the PhysicalObjectNode representing a body [where the body is considered as including the head]).

26.6.5 Discussion

It seems probable that, for simple scenes consisting of a small number of simple objects, clustering for object recognition will be fairly unproblematic. However, there are two cases that are potentially tricky:

- Sub-objects: e.g. the head and torso of a body, which may move separately; or the nose of the head, which may wiggle; or the legs of a walking dog; etc.
- Coordinated objects: e.g. if a character's hat is on a table, and then later on his head, then when it's on his head we basically want to consider him and his hat as the same object, for some purposes.

These examples show that partitioning a scene into *objects* is a borderline-cognitive rather than purely lower-level-perceptual task, which cannot be hard-wired in any very simple way.

We also note that, for complex scenes, clustering may not work perfectly for object recognition and some reasoning may be needed to aid with the process. Intuitively, these may correspond to scenes that, in human perceptual psychology, require conscious attention and focus in order to be accurately and usefully perceived.

26.7 Interfacing the Atomspace with a Deep Learning Based Perception-Action Hierarchy

We have discussed how one may do perception processing such as object recognition within the Atomspace, and this is indeed a viable strategy. But an alternate approach is also interesting, and likely more valuable in the case of robotic perception/action: build a separate perceptual-motor hierarchy, and link it in with the Atomspace. This approach is appealing in large part because a lot of valuable and successful work has already been done using neural networks and related architectures for perception and actuation. And it is not necessarily contradictory to doing perception processing in the Atomspace – obviously, one may have complementary, synergetic perception processing occurring in two different parts of the architecture.

EFTA00624287

This section reviews some general ideas regarding the interfacing of CogPrime with deep learning hierarchies for perception and action; the following chapter then discusses one example of this in detail, involving the DeSTIN deep learning architecture.

26.7.1 Hierarchical Perception Action Networks

CogPrime could be integrated with a variety of different hierarchical perception/action architectures. For the purpose of this section, however, we will consider a class of architectures that is neither completely general nor extremely specific. Many of the ideas to be presented here are in fact more broadly applicable beyond the architecture described here.

The following assumptions will be made about the HPANs (Hierarchical Perception/Action Network) to be hybridized with CogPrime. It may be best to use multiple HPANs, at least one for declarative/sensory/episodic knowledge (we'll call this the "primary HPAN") and one for procedural knowledge. A HPAN for intentional knowledge (a goal hierarchy; in DeSTIN called the "critic hierarchy") may be valuable as well. We assume that each HPAN has the properties:

1. It consists of a network of nodes, endowed with a learning algorithm, whose connectivity pattern is largely but not entirely hierarchical (and whose hierarchy contains both feedback, feedforward and lateral connections)
2. It contains a set of input nodes, receiving perceptual inputs, at the bottom of the hierarchy
3. It has a set of output nodes, which may span multiple levels of the hierarchy. The "output nodes" indicate informational signals to cognitive processes lying outside the HPAN, or else control signals to actuators, which may be internal or external.
4. Other nodes besides I/O nodes may potentially be observed or influenced by external processes; for instance they may receive stimulation
5. Link weights in the HPAN get updated via some learning algorithm that is roughly speaking "statistically Hebbian," in the sense that on the whole when a set of nodes get activated together for a period of time, they will tend to become attractors. By an attractor we mean: a set S of nodes such that the activation of a subset of S during a brief interval tends to lead to the activation of the whole set S during a reasonably brief interval to follow
6. As an approximate but not necessarily strict rule, nodes higher in the hierarchy tend to be involved in attractors corresponding to events or objects localized in larger spacetime regions

Examples of specific hierarchical architectures broadly satisfying these requirements are the visual pattern recognition networks constructed by Hawkins [HB06] and [PCP00], and Arel's DeSTIN system discussed earlier (and in more depth in following chapters). The latter appears to fit the requirements particularly snugly due to having dynamics very well suited to the formation of a complex array of attractors, and a richer methodology for producing outputs. These are all not only HPANs but have a more particular structure that in Chapter 27 is called a Compositional Spatiotemporal Deep Learning Network or CSDLN.

The particulars of the use of HPANs with OpenCog are perhaps best explained via enumeration of memory types and control operations.

EFTA00624288

26.7.2 Declarative Memory

The key idea here is linkage of primary HPAN attractors to CogPrime

ConceptNodes via MemberLinks. This is in accordance with the notion of glocal memory, in the language of which the HPAN attractors are the maps and the corresponding ConceptNodes are the keys. Put simply, when a HPAN attractor is recognized, MemberLinks are created between the HPAN nodes comprising the main body of the attractor, and a ConceptNode in the AtomTable representing the attractor. MemberLink weights may be used to denote fuzzy attractor membership. Activation may spread from HPAN nodes to ConceptNodes, and STI may spread from ConceptNodes to HPAN nodes; a conversion rate between HPAN activation and STI currency must be maintained by the CogPrime central bank (see Chapter 23), for ECAN purposes.

Both abstract and concrete knowledge may be represented in this way. For instance, the Eiffel Tower would correspond to one attractor, the general shape of the Eiffel Tower would correspond to another, and the general notion of a "tower" would correspond to yet another. As these three examples are increasingly abstract, the corresponding attractors would be weighted increasingly heavily on the upper levels of the hierarchy.

26.7.3 Sensory Memory

CogPrime may also use its primary HPAN to store memories of sense-perceptions and low-level abstractions therefrom. MemberLinks may join concepts in the AtomTable to percept-attractors in the HPAN. If the HPAN is engineered to associate specific neural modules to specific spatial regions or specific temporal intervals, then this may be accounted for by automatically indexing ConceptNodes corresponding to attractors centered in those modules in the AtomTable's TimeServer and SpaceServer objects, which index Atoms according to time and space.

An attractor representing something specific like the Eiffel Tower, or Bob's face, would be weighted largely in the lower levels of the hierarchy, and would correspond mainly to sensory rather than conceptual memory.

26.7.4 Procedural Memory

The procedural HPAN may be used to learn procedures such as low-level motion primitives that are more easily learned using HPAN training than using more abstract procedure learning methods. For example, a Combo tree learned by MOSES in CogPrime might contain a primitive corresponding to the predicate-argument relationship *pick_up(ball)*; but the actual procedure for controlling a robot hand to pick up a ball, might be expressed as an activity pattern within the low-level procedural HPAN. A procedure P stored in the low-level procedural HPAN would be represented in the AtomTable as a ConceptNode C linked to key nodes in the HPAN attractor corresponding to P. The invocation of P would be accomplished by transferring STI currency to C and then allowing ECAN to do its work.

On the other hand, CogPrime's interfacing of the high-level procedural HPAN with the CogPrime ProcedureRepository is intimately dependent on the particulars of the MOSES proce-

EFTA00624289

dure learning algorithm. As will be outlined in more depth in Chapter 33, MOSES is a complex, multi-stage process that tries to find a program maximizing some specified fitness function, and that involves doing the following within each "deme" (a deme being an island of roughly-similar programs)

1. casting program trees into a hierarchical normal form
2. evaluating the program trees on a fitness function
3. building a model distinguishing fit versus unfit program trees, which involves: 3a. figuring out what program tree features the model should include; 3b. building the model using a learning algorithm
4. generating new program trees that are inferred likely to give high fitness, based on the model
5. return to step 1 with these new program trees

There is also a system for managing the creation and deletion of demes.

The weakest point in CogPrime's current MOSES-based approach to procedure learning appears to be step 3. And the main weakness is conceptual rather than algorithmic; what is needed is to replace the current step 3 with something that uses long-term memory to do model-building and feature-selection, rather than (like the current code) doing these things in a manner that's restricted to the population of program trees being evolved to optimize a particular fitness function.

One promising approach to resolving this issue is via replacing step 3b (and, to a limited extent, 3a) with an interconnection between MOSES and a procedural HPAN. A HPAN can do supervised categorization, and can be designed to handle feature selection in a manner integrated with categorization, and also to integrate long-term memory into its categorization decisions.

26.7.5 Episodic Memory

In a hybrid CogPrime /HPAN architecture, episodic knowledge may be handled via a combination of:

1. using a traditional approach to store a large ExperienceDB of actual experienced episodes (including sensory inputs and actions; and also the states of the most important items in memory during the experience)
2. using the Atomspace (with its TimeServer and SpaceServer components) to store declarative knowledge about experiences
3. using dimensional embedding to index the AtomSpace's episodic knowledge in a spatiotemporally savvy way, as described in Chapter 40
4. training a large HPAN to summarize the scope of experienced episodes (this could be the primary HPAN used for declarative and sensory memory, or could potentially be a separate episodic HPAN)

Such a network should be capable of generating imagined episodes based on cues, as well recalling real episodes. The HPAN would serve as a sort of index into the memory of episodes, There would be HebbianLinks from the AtomTable into the episodic HPAN.

EFTA00624290

For instance, suppose that once the agent built an extremely tall tower of blocks, taller than any others in its memory. Perhaps it wants to build another very tall tower again, so it wants to summon up the memory of that previous occasion, to see if there is possibly guidance therein. It then proceeds by thinking about tallness and towerness at the same time, which stimulates the relevant episode, because at the time of building the extremely tall tower, the agent was thinking a lot about tallness (so thoughts of tallness are part of the episodic memory).

26.7.6 Action Selection and Attention Allocation

CogPrime's action selection mechanism chooses procedures based on which ones are estimated most likely to achieve current goals given current context, and places these in an "active procedure pool" where an ExecutionManager object mediates their execution.

Attention allocation spans all components of CogPrime, including an HPAN if one is integrated. Attention flows between the two components due to the conversion of STI to and from HPAN activation. And in this manner assignment of credit flows from GoalNodes into the HPAN, because this kind of simultaneous activation may be viewed as "rewarding" a HPAN link. So, the HPAN may reward signals from GoalNodes via ECAN, because when a ConceptNode gets rewarded, if the ConceptNode points to a set of nodes, these nodes get some of the reward.

26.8 Multiple Interaction Channels

Now we discuss a broader issue regarding the interfacing between CogPrime and the external world. The only currently existing embodied OpenCog applications, PetBrain and CogBot, are based on a loosely human model of perception and action, in which a single CogPrime instance controls a single mobile body, but this of course is not the only way to do things. More generally, what we can say is that a variety of external-world events come into a CogPrime system from physical or virtual world sensors, plus from other sources such as database interfaces, Web spiders, and/or other sources. The external systems providing CogPrime with data may be generically referred to as *sensory sources* (and in the terminology we adopt here, once Atoms have been created to represent external data, then one is dealing with *perceptions* rather than sensations). The question arises how to architect a CogPrime system, in general, for dealing with a variety of sensory sources.

We introduce the notion of an "interaction channel": a collection of sensory sources that is intended to be considered as a whole as a synchronous stream, and that is also able to receive CogPrime actions - in the sense that when CogPrime carries out actions relative to the interaction channel, this directly affects the perceptions that CogPrime receives from the interaction channel. A CogPrime meant to have conversations with 10 separate users at once might have 10 interaction channels. A human mind has only one interaction channel in this sense (although humans may become moderately adept at processing information from multiple external-world sources, coming in through the same interaction channel).

Multiple-interaction-channel digital psychology may become extremely complex - and hard for us, with our single interaction channels, to comprehend. This is one among many cases

EFTA00624291

where a digital mind, with its more flexible architecture, will have a clear advantage over our human minds with their fixed and limited neural architectures. For simplicity, however, in the following chapters we will often focus on the single-interaction-channel case.

Events coming in through an interaction channel are presented to the system as new perceptual Atoms, and relationships amongst these. In the multiple interaction channel case, the AttentionValues of these newly created Atoms require special treatment. Not only do they require special rules, they require additional fields to be added to the AttentionValue object, beyond what has been discussed so far.

We require newly created perceptual Atoms to be given a high initial STI. And we also require them to be given a high amount of a quantity called "interaction-channel STI." To support this, the AttentionValue objects of Atoms must be expanded to contain interaction-channel STI values; and the ImportanceUpdating MindAgent must compute interaction-channel importance separately from ordinary importance.

And, just as we have channel-specific AttentionValues, we may also have channel-specific TruthValues. This allows the system to separately account for the frequency of a given perceptual item in a given interaction channel. However, no specific mechanism is needed for these, they are merely contextual truth values, to be interpreted within a Context Node associated with the interaction channel.

EFTA00624292

Chapter 27
Integrating CogPrime with a Compositional Spatiotemporal Deep Learning Network

27.1 Introduction

Many different approaches to "low-level" perception and action processing are possible within the overall CogPrime framework. We discussed several in the previous chapter, all elaborations of the general hierarchical pattern recognition approach. Here we describe one sophisticated approach to hierarchical pattern recognition based perception in more detail: the tight integration of CogPrime with a sophisticated hierarchical perception/action oriented learning system such as the DeSTIN architecture reviewed in Chapter 4 of Part 1.

We introduce here the term "Compositional Spatiotemporal Deep Learning Network" (CSDLN), to refer to deep learning networks whose hierarchical structure directly mirrors the hierarchical structure of spacetime. In the language of Chapter 26, a CSDLN is a special kind of HPAN (hierarchical perception action network), which has the special property that each of its nodes refers to a certain spatiotemporal region and is concerned with predicting what happens inside that region. Current exemplifications of the CSDLN paradigm include the DeSTIN architecture that we will focus on here, along with Jeff Hawkins' Numenta "HTM" system [HB06] [1], Itamar Arel's DeSTIN [ARC09a], Itamar Arel's HDRN [2] system (the proprietary, closed-source sibling of DeSTIN), Dileep George's spin-off from Numenta [3], and work by Mohamad Tarifi [TSH11], Bundzel and Hashimoto [BH10], and others. CSDLNs are reasonably well proven as an approach to intelligent sensory data processing, and have also been hypothesized as a broader foundation for artificial general intelligence at the human level and beyond [HB06] [ARC09a].

While CSDLNs have been discussed largely in the context of perception, the specific form of CSDLN we will pursue here goes beyond perception processing, and involves the coupling of three separate hierarchies, for perception, action and goals/reinforcement [GLdG+10]. The "action" CSDLNs discussed here correspond to the procedural HPAN discussed in Chapter 26. Abstract learning and self-understanding are then hypothesized as related to systems of attractors emerging from the close dynamic coupling of the upper levels of the three hierarchies.

[1] While the Numenta system is the best-known CSDLN architecture, other CSDLNs appear more impressively functional in various respects; and many CSDLN-related ideas existed in the literature well before Numenta's advent.

[2] http:\binatix.com

[3] http:\vicarioussystems.com

EFTA00624294

DeSTIN is our paradigm case of this sort of CSDLN, but most of the considerations given here would apply to any CSDLN of this general character.

CSDLNs embody a certain conceptual model of the nature of intelligence, and to integrate them appropriately with a broader architecture, one must perform the integration not only on the level of software code but also on the level of conceptual models. Here we focus here on the problem of integrating an extended version of the DeSTIN CSDLN system with the CogPrime integrative AGI (artificial general intelligence) system. The crux of the issue here is how to map DeSTIN's attractors into CogPrime's more abstract, probabilistic "weighted, labeled hypergraph" representation (called the Atomspace). The main conclusion reached is that in order to perform this mapping in a conceptually satisfactory way, one requires a system of hierarchies involving **the structure of DeSTIN's network but the semantic structures of the Atomspace**. The DeSTIN perceptual hierarchy is augmented by motor and goal hierarchies, leading to a tripartite "extended DeSTIN". In this spirit, three "semantic-perceptual" hierarchies are proposed, corresponding to the three extended-DeSTIN CSDLN hierarchies and explicitly constituting an intermediate level of representation between attractors in DeSTIN and the habitual cognitive usage of CogPrime Atoms and Atom-networks. For simple reference we refer to this as the "Semantic CSDLN" approach.

A "tripartite semantic CSDLN" consisting of interlinked semantic perceptual, motoric and goal hierarchies could be coupled with DeSTIN or another CSDLN architecture to form a novel AGI approach; or (our main focus here) it may be used as a glue between an CSDLN and and a more abstract semantic network such as the cognitive Atoms in CogPrime's Atomspace.

One of the core intuitions underlying this integration is that, in order to achieve the desired level of functionality for tasks like picture interpretation and assembly of complex block structures, a convenient route is to perform a fairly *tight* integration of a highly capable CSDLN like DeSTIN with other CogPrime components. For instance, we believe it's necessary to go deeper than just using DeSTIN as an input/output layer for CogPrime, by building associative links between the nodes inside DeSTIN and those inside the Atomspace.

This "tightly linked integration" approach is obviously an instantiation of the general cognitive synergy principle, which hypothesizes particular properties that the interactions between components in an integrated AGI system should display, in order for the overall system to display significant general intelligence using limited computational resources. Simply piping output from an CSDLN to other components, and issuing control signals from these components to the CSDLN, is likely an inadequate mode of integration, incapable of leveraging the full potential of CSDLNs; what we are suggesting here is a much tighter and more synergetic integration.

In terms of the general principle of mind-world correspondence, the conceptual justification for CSDLN/CogPrime integration would be that the everyday human world contains many compositional spatiotemporal structures relevant to human goals, but also contains many relevant patterns that are not most conveniently cast into a compositional spatiotemporal hierarchy. Thus, in order to most effectively perceive, remember, represent, manipulate and enact the full variety of relevant patterns in the world, it is sensible to have a cognitive structure containing a CSDLN as a significant component, but not the only component.

EFTA00624295

27.2 Integrating CSDLNs with Other AI Frameworks

CSDLNs represent knowledge as attractor patterns spanning multiple levels of hierarchical networks, supported by nonlinear dynamics and (at least in the case of the overall DeSTIN design) involving cooperative activity of perceptual, motor and control networks. These attractors are learned and adapted via a combination of methods including localized pattern recognition algorithms and probabilistic inference. Other AGI paradigms represent and learn knowledge in a host of other ways. How then can CSDLNs be integrated with these other paradigms?

A very simple form of integration, obviously, would be to use a CSDLN as a sensorimotor cortex for another AI system that's focused on more abstract cognition. In this approach, the CSDLN would stream state-vectors to the abstract cognitive system, and the abstract cognitive system would stream abstract cognitive inputs to the CSDLN (which would then consider them together with its other inputs). One thing missing in this approach is the possibility of the abstract cognitive system's insights biasing the judgments inside the CSDLN. Also, abstract cognition systems aren't usually well prepared to handle a stream of quantitative state vectors (even ones representing intelligent compressions of raw data).

An alternate approach is to build a richer intermediate layer, which in effect translates between the internal language of the CSDLN and the internal language of the other AI system involved. The particulars, and the viability, of this will depend on the particulars of the other AI system. What we'll consider here is the case where the other AI system contains explicit symbolic representations of patterns (including patterns abstracted from observations that may have no relation to its prior knowledge or any linguistic terms). In this case, we suggest, a viable approach may be to construct a "semantic CSDLN" to serve as an intermediary. The semantic CSDLN has the same hierarchical structure as an CSDLN, but inside each node it contains abstract patterns rather than numerical vectors. This approach has several potential major advantages: the other AI system is not presented with a large volume of numerical vectors (which it may be unprepared to deal with effectively); the CSDLN can be guided by the other AI system, without needing to understand symbolic control signals; and the intermediary semantic CSDLN can serve as a sort of "blackboard" which the CSDLN and the other AI system can update in parallel, and be guided by in parallel, thus providing a platform encouraging "cognitive synergy".

The following sections go into more detail on the concept of semantic CSDLNs. The discussion mainly concerns the specific context of DeSTIN/CogPrime integration, but the core ideas would apply to the integration of any CSDLN architecture with any other AI architecture involving uncertain symbolic representations susceptible to online learning.

27.3 Semantic CSDLN for Perception Processing

In the standard perceptual CSDLN hierarchy, a node N on level k (considering level 1 as the bottom) corresponds to a spatiotemporal region S with size s_k (s_k increasing monotonically and usually exponentially with k); and, has children on level $k-1$ corresponding to spatiotemporal regions that collectively partition S. For example, a node on level 3 might correspond to a $16x16$ pixel region S of 2D space over a time period of 10 seconds, and might have 4 level 2 children corresponding to disjoint $4x4$ regions of 2D space over 10 seconds, collectively composing S.

EFTA00624296

This kind of hierarchy is very effective for recognizing certain types of visual patterns. However it is cumbersome for recognizing some other types of patterns, e.g. the pattern that a face typically contains two eyes beside each other, but at variable distance from each other.

One way to remedy this deficiency is to extend the definition of the hierarchy, so that nodes do not refer to fixed spatial or temporal positions, but only to *relative* positions. In this approach, the internals of a node are basically the same as in an CSDLN, and the correspondence of the nodes on level k with regions of size s_k is retained, but the relationships between the nodes are quite different. For instance, a variable-position node of this sort could contain several possible 2D pictures of an eye, but be nonspecific about where the eye is located in the 2D input image.

Figure 27.1 depicts this "semantic-perceptual CSDLN" idea heuristically, showing part of a semantic-perceptual CSDLN indicating the parts of a face, and also the connections between the semantic-perceptual CSDLN, a standard perceptual CSDLN, and a higher-level cognitive semantic network like CogPrime's Atomspace. [4]

More formally, in the suggested "semantic-perceptual CSDLN" approach, a node N on level k, instead of pointing to a set of level $k-1$ children, points to a small (but not necessarily connected) *semantic network* , such that the nodes of the semantic network are (variable-position) level $k-1$ nodes; and the edges of the semantic network possess labels representing spatial or temporal relationships, for example *horizontally_ aligned*, *vertically_ aligned*, *right_ side*, *left_ side*, *above*, *behind*, *immediately_ right*, *immediately_ left*, *immediately_ above*, *immediately_ below*, *after*, *immediately_ after*. The edges may also be weighted either with numbers or probability distributions, indicating the quantitative weight of the relationship indicated by the label.

So for example, a level 3 node could have a child network of the form horizontally_aligned(N_1, N_2) where N_1 and N_2 are variable-position level 2 nodes. This would mean that N_1 and N_2 are along the same horizontal axis in the 2D input but don't need to be immediately next to each other. Or one could say, e.g. on_axis_perpendicular_to(N_1, N_2, N_3, N_4), meaning that N_1 and N_2 are on an axis perpendicular to the axis between N_3 and N_4. It may be that the latter sort of relationship is fundamentally better in some cases, because *horizontally_aligned* is still tied to a specific orientation in an absolute space, whereas *on_axis_perpendicular_to* is fully relative. But it may be that both sorts of relationship are useful.

Next, development of learning algorithms for semantic CSDLNs seems a tractable research area. First of all, it would seem that, for instance, the DeSTIN learning algorithms could straightforwardly be utilized in the semantic CSDLN case, once the local semantic networks involved in the network are known. So at least for some CSDLN designs, the problem of learning the semantic networks may be decoupled somewhat from the learning occurring inside the nodes. DeSTIN nodes deal with clustering of their inputs, and calculation of probabilities based on these clusters (and based on the parent node states). The difference between the semantic CSDLN and the traditional DeSTIN CSDLN has to do with what the inputs are.

Regarding learning the local semantic networks, one relatively straightforward approach would be to *data mine them from a standard CSDLN*. That is, if one runs a standard CS-

[4] The perceptual CSDLN shown is unrealistically small for complex vision processing (only 4 layers), and only a fragment of the semantic-perceptual CSDLN is shown (a node corresponding to the category face, and then a child network containing nodes corresponding to several components of a typical face). In a real semantic-perceptual CSDLN, there would be many other nodes on the same level as the face node, many other parts to the face subnetwork besides the eyes, nose and mouth depicted here; the eye, nose and mouth nodes would also have child subnetworks; there would be link from each semantic node to centroids within a large number of perceptual nodes; and there would also be many nodes not corresponding clearly to any single English language concept like eye, nose, face, etc.

EFTA00624297

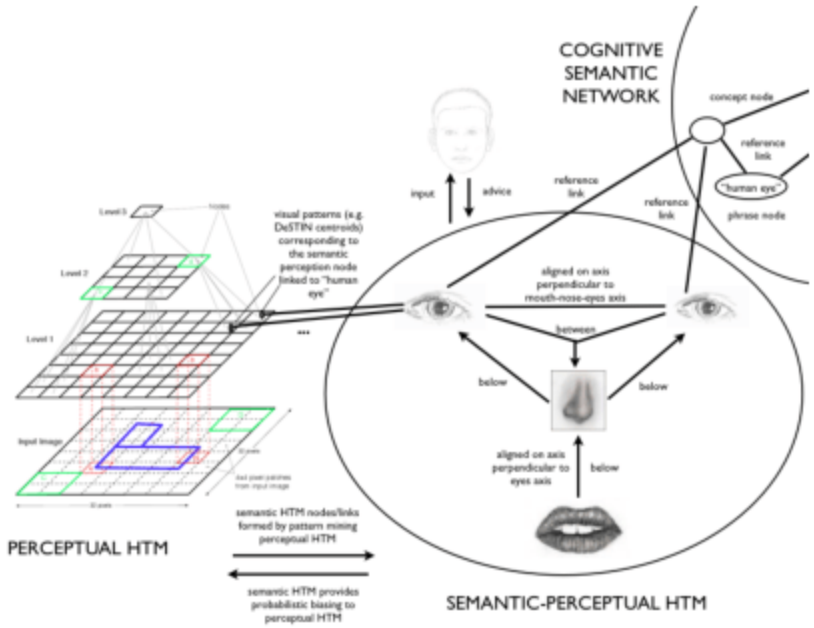


Fig. 27.1: Simplified depiction of the relationship between a semantic-perceptual CSDLN, a traditional perceptual CSDLN (like DeSTIN), and a cognitive semantic network (like CogPrime's AtomSpace).

DLN on a stream of inputs, one can then run a frequent pattern mining algorithm to find semantic networks (using a given vocabulary of semantic relationships) that occur frequently in the CSDLN as it processes input. A subnetwork that is identified via this sort of mining, can then be grouped together in the semantic CSDLN, and a parent node can be created and pointed to it.

Also, the standard CSDLN can be searched for frequent patterns involving the clusters (referring to DeSTIN here, where the nodes contain clusters of input sequences) inside the nodes in the semantic CSDLN. Thus, in the "semantic DeSTIN" case, we have a feedback interaction wherein: 1) the standard CSDLN is formed via processing input;

2) frequent pattern mining on the standard CSDLN is used to create subnetworks and corresponding parent nodes in the semantic CSDLN;

3) the newly created nodes in the semantic CSDLN get their internal clusters updated via standard DeSTIN dynamics;

4) the clusters in the semantic nodes are used as seeds for frequent pattern mining on the standard CSDLN, returning us to Step 2 above.

After the semantic CSDLN is formed via mining the perceptual CSDLN, it may be used to bias the further processing of the perceptual CSDLN. For instance, in DeSTIN each node carries out probabilistic calculations involving knowledge of the prior probability of the "observation" coming into that node over a given interval of time. In the current DeSTIN version, this prior probability is drawn from a uniform distribution, but it would be more effective to draw the prior probability from the semantic network – observations matching things represented in the semantic network would get a higher prior probability. One could also use subtler strategies

EFTA00624298

such as using imprecise probabilities in DeSTIN [Goe11b], and assigning a greater confidence to probabilities involving observations contained in the semantic network.

Finally, we note that the nodes and networks in the semantic CSDLN may either

- be linked into the nodes and links in a semantic network such as CogPrime's AtomSpace
- actually be implemented in terms of an abstract semantic network language like CogPrime's AtomSpace (the strategy to be suggested in Chapter 29).

This allows us to think of the semantic CSDLN as a kind of bridge between the standard CSDLN and the cognitive layer of an AI system. In an advanced implementation, the cognitive network may be used to suggest new relationships between nodes in the semantic CSDLN, based on knowledge gained via inference or language.

27.4 Semantic CSDLN for Motor and Sensorimotor Processing

Next we consider a semantic CSDLN that focuses on movement rather than sensation. In this case, rather than a 2D or 3D visual space, one is dealing with an n-dimensional *configuration space* (C-space). This space has one dimension for each degree of freedom of the agent in question. The more joints with more freedom of movement an agent has, the higher the dimensionality of its configuration space.

Using the notion of configuration space, one can construct a *semantic-motoric CSDLN hierarchy* analogous to the semantic-perceptual CSDLN hierarchy. However, the curse of dimensionality demands a thoughtful approach here. A square of side 2 can be tiled with 4 squares of side 1, but a 50-dimensional cube of side 2 can be tiled with 2^{50} 50-dimensional cubes of side 1. If one is to build a CSDLN hierarchy in configuration space analogous to that in perceptual space, some sort of sparse hierarchy is necessary.

There are many ways to build a sparse hierarchy of this nature, but one simple approach is to build a hierarchy where the nodes on level k represent motions that combine the motions represented by nodes on level $k-1$. In this case the most natural semantic label predicates would seem to be things like *simultaneously*, *after*, *immediately_after*, etc. So a level k node represents a sort of "motion plan" corresponded by chaining together (serially and/or in parallel) the motions encoded in level $k-1$ nodes. Overlapping regions of C-space correspond to different complex movements that share some of the same component movements, e.g. if one is trying to slap one person while elbowing another, or run while kicking a soccer ball forwards. Also note, the semantic CSDLN approach reveals perception and motor control to have essentially similar hierarchical structures, more so than with the traditional CSDLN approach and its fixed-position perceptual nodes.

Just as the semantic-perceptual CSDLN is naturally aligned with a traditional perceptual CSDLN, similarly a semantic-motoric CSDLN may be naturally aligned with a "motor CSDLN". A typical motoric hierarchy in robotics might contain a node corresponding to a robot arm, with children corresponding to the hand, upper arm and lower arm; the hand node might then contain child nodes corresponding to each finger, etc. This sort of hierarchy is intrinsically spatiotemporal because each individual action of each joint of an actuator like an arm is intrinsically bounded in space and time. Perhaps the most ambitious attempt along these lines is [AM01], which shows how perceptual and motoric hierarchies are constructed and aligned in an architecture for intelligent automated vehicle control.

EFTA00624299

Figure 27.2 gives a simplified illustration of the potential alignment between a semantic-motoric CSDLN and a purely motoric hierarchy (like the one posited above in the context of extended DeSTIN). [5] In the figure, the motoric hierarchy is assumed to operate somewhat like DeSTIN, with nodes corresponding to (at the lowest level) individual servomotors, and (on higher levels) natural groupings of servomotors. The node corresponding to a set of servos is assumed to contain centroids of clusters of trajectories through configuration space. The task of choosing an appropriate action is then executed by finding the appropriate centroids for the nodes. Note an asymmetry between perception and action here. In perception the basic flow is bottom-up, with top-down flow used for modulation and for "imaginative" generation of percepts. In action, the basic flow is top-down, with bottom-up flow used for modulation and for imaginative, "fiddling around" style generation of actions. The semantic-motoric hierarchy then contains abstractions of the C-space centroids from the motoric hierarchy – i.e., actions that bind together different C-space trajectories that correspond to the same fundamental action carried out in different contexts or under different constraints. Similarly to in the perceptual case, the semantic hierarchy here serves as a glue between lower-level function and higher-level cognitive semantics.

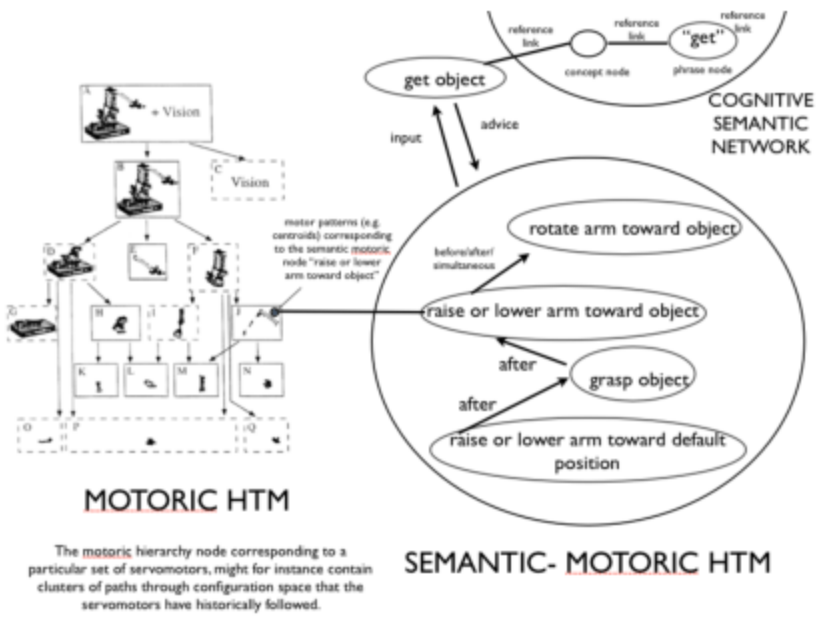


Fig. 27.2: Simplified depiction of the relationship between a semantic-motoric CSDLN, a motor control hierarchy (illustrated by the hierarchy of servos associated with a robot arm), and a cognitive semantic network (like CogPrime's AtomSpace).

[5] In the figure, only a fragment of the semantic-motoric CSDLN is shown (a node corresponding to the "get object" action category, and then a child network containing nodes corresponding to several components of the action). In a real semantic-motoric CSDLN, there would be many other nodes on the same level as the get-object node, many other parts to the get-object subnetwork besides the ones depicted here; the subnetwork nodes would also have child subnetworks; there would be link from each semantic node to centroids within a large number of motoric nodes; and there might also be many nodes not corresponding clearly to any single English language concept like "grasp object" etc.

EFTA00624300

27.5 Connecting the Perceptual and Motoric Hierarchies with a Goal Hierarchy

One way to connect perceptual and motoric CSDLN hierarchies is using a "semantic-goal CSDLN" bridging the semantic-perceptual and semantic-motoric CSDLNs. The semantic-goal CSDLN would be a "semantic CSDLN" loosely analogous to the perceptual and motor semantic CSDLNs – and could optionally be linked into the reinforcement hierarchy of a tripartite CSDLN like extended DeSTIN. Each node in the semantic-goal CSDLN would contain implications of the form "Context & Procedure → Goal", where Goal is one of the AI system's overall goals or a subgoal thereof, and Context and Procedure refer to nodes in the perceptual and motoric semantic CSDLNs respectively.

For instance, a semantic-goal CSDLN node might contain an implication of the form "I perceive my hand is near object X & I grasp object X → I possess object X." This would be useful if "I possess object X" were a subgoal of some higher-level system goal, e.g. if X were a food object and the system had the higher-level goal of obtaining food.

To the extent that the system's goals can be decomposed into hierarchies of progressively more and more spatiotemporally localized subgoals, this sort of hierarchy will make sense, leading to a tripartite hierarchy as loosely depicted in Figure 27.3. [6] One could attempt to construct an overall AGI approach based on a tripartite hierarchy of this nature, counting on the upper levels of the three hierarchies to come together dynamically to form an integrated cognitive network, yielding abstract phenomena like language, self, reasoning and mathematics. On the other hand, one may view this sort of hierarchy as a portion of a larger integrative AGI architecture, containing a separate cognitive network, with a less rigidly hierarchical structure and less of a tie to the spatiotemporal structure of physical reality. The latter view is the one we are primarily taking within the CogPrime AGI approach, viewing perceptual, motoric and goal hierarchies as "lower level" subsystems connected to a "higher level" system based on the CogPrime AtomSpace and centered on its abstract cognitive processes.

Learning of the subgoals and implications in the goal hierarchy is of course a complex matter, which may be addressed via a variety of algorithms, including online clustering (for subgoals or implications) or supervised learning (for implications, the "supervision" being purely internal and provided by goal or subgoal achievement).

[6] The diagram is simplified in many ways, e.g. only a handful of nodes in each hierarchy is shown (rather than the whole hierarchy), and lines without arrows are used to indicate bidirectional arrows, and nearly all links are omitted. The purpose is just to show the general character of interaction between the components in a simplified context.

EFTA00624301

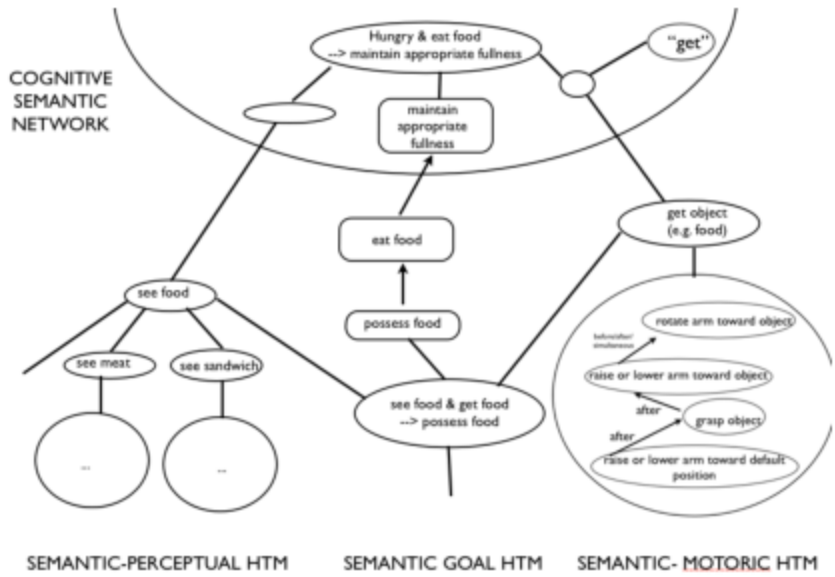


Fig. 27.3: Simplified illustration of the proposed interoperation of perceptual, motoric and goal semantic CSDLNs.

EFTA00624302

Chapter 28
Making DeSTIN Representationally Transparent

Co-authored with Itamar Arel

28.1 Introduction

In this chapter and the next we describe one particular incarnation of the above ideas on semantic CSDLNs in more depth: the integration of CogPrime with the DeSTIN architecture reviewed in Chapter 4 of Part 1.

One of the core intuitions underlying this integration is that, in order to achieve the desired level of functionality for tasks like picture interpretation and assembly of complex block structures, it will be necessary to integrate DeSTIN (or some similar system) and CogPrime components fairly tightly – going deeper than just using DeSTIN as an input/output layer for CogPrime, by building a number of explicit linkages between the nodes inside DeSTIN and CogPrime respectively.

The general DeSTIN design has been described in talks as comprising three crosslinked hierarchies, handling perception, action and reinforcement; but so far only the perceptual hierarchy (also called the "spatiotemporal inference network") has been implemented or described in detail in publications. In this chapter we will focus on DeSTIN's perception hierarchy. We will explain DeSTIN's perceptual dynamics and representations as we understand them, more thoroughly than was done in the brief review above; and we will describe a series of changes to the DeSTIN design, made in the spirit of easing DeSTIN/OpenCog integration. In the following chapter we will draw action and reinforcement into the picture, deviating somewhat in the details from the manner in which these things would be incorporated into a standalone DeSTIN, but pursuing the same concepts in an OpenCog integration context.

What we describe here is a way to make a "Uniform DeSTIN", in which the internal representation of perceived visual forms is independent of affine transformations (translation, scaling, rotation and shear). This "representational transparency" means that, when Uniform DeSTIN perceives a pattern: no matter how that pattern is shifted or linearly transformed, the way Uniform DeSTIN represents that pattern internally is going to be basically the same. This makes it easy to look at a collection of DeSTIN states, obtained by exposing a DeSTIN perception network to the world at different points in time, and see the commonalities in what they are perceiving and how they are interpreting it. By contrast, in the original version of DeSTIN (here called "classic DeSTIN"), it may take significant effort to connect the internal representation of a visual pattern and the representation of its translated or linearly transformed versions. The uniformity of Uniform DeSTIN makes it easier for humans to inspect DeSTIN's state and understand what's going on, and also (more to the point) makes it easier for other

EFTA00624304

AI components to recognize patterns in sets of DeSTIN states. The latter fact is critical for the DeSTIN/OpenCog integration

28.2 Review of DeSTIN Architecture and Dynamics

The hierarchical architecture of DeSTIN's spatiotemporal inference network comprises an arrangement into multiple layers of "nodes" comprising multiple instantiations of an identical processing unit. Each node corresponds to a particular spatiotemporal region, and uses a statistical learning algorithm to characterize the sequences of patterns that are presented to it by nodes in the layer beneath it.

More specifically, at the very **lowest layer** of the hierarchy nodes receive as input raw data (e.g. pixels of an image) and continuously construct a belief state that attempts to characterize the sequences of patterns viewed. The **second layer**, and all those above it, receive as input the belief states of nodes at their corresponding lower layers, and attempt to construct belief states that capture regularities in their inputs. Each node also receives as input the belief state of the node above it in the hierarchy (which constitutes "contextual" information, utilized in the node's prediction process).

Inside each node, an online clustering algorithm is used to identify regularities in the sequences received by that node. The centroids of the clusters learned are stored in the node and comprise the basic visual patterns recognized by that node. The node's "belief" regarding what it is seeing, is then understood as a probability density function defined over the centroids at that node. The equations underlying this centroid formation and belief updating process are identical for every node in the architecture, and were given in their original form in [ARC09a], though the current open-source DeSTIN codebase reflects some significant improvements not yet reflected in the publication record.

In short, the way DeSTIN represents an item of knowledge is as a probability distribution over "network activity patterns" in its hierarchical network. An activity pattern, at each point in time, comprises an indication of which centroids in each node are most active, meaning they have been identified as most closely resembling what that node has perceived, as judged in the context of the perceptions of the other nodes in the system. Based on this methodology, the DeSTIN perceptual network serves the critical role of building and maintaining a model of the state of the world as visually perceived.

This methodology allows for powerful unsupervised classification. If shown a variety of real-world scenes, DeSTIN will automatically form internal structures corresponding to the various natural categories of objects shown in the scenes, such as trees, chairs, people, etc.; and also to the various natural categories of events it sees, such as reaching, pointing, falling. In order to demonstrate the informativeness of these internal structures, experiments have been done using DeSTIN's states as input feature vectors for supervised learning algorithms, enabling high-accuracy supervised learning of classification models from labeled image data [KAR10]. A closely related algorithm developed by the same principal researcher (Itamar Arel) has proven extremely successful at audition tasks such as phoneme recognition [ABS+11].

EFTA00624305

28.2.1 Beyond Gray-Scale Vision

The DeSTIN approach may easily be extended to other senses beyond gray-scale vision. For color vision, it suffices to replace the one-dimensional signals coming into DeSTIN's lower layer with 3D signals representing points in the color spectrum; the rest of the DeSTIN process may be carried over essentially without modification. Extension to further senses is also relatively straightforward on the mathematical and software structure level, though they may of course require significant additional tuning and refinement of details.

For instance, olfaction does not lend itself well to hierarchical modeling, but audition and haptics (touch) do:

- for auditory perception, one could use a DeSTIN architecture in which each layer is one-dimensional rather than two-dimensional, representing a certain pitch. Or one could use two dimensions for pitch and volume. This results in a system quite similar to the DeSTIN-like system shown to perform outstanding phoneme recognition in [ABS+11], and is conceptually similar to Hierarchical Hidden Markov Models (HHMMs), which have proven quite successful in speech recognition and which Ray Kurzweil has argued are the central mechanism of human intelligence [Kur12]. Note also recent results published by Microsoft Research, showing dramatic improvements over prior speech recognition results based on use of a broadly HHMM-like deep learning system [HDY+12].
- for haptic perception, one could use a DeSTIN architecture in which the lower layer of the network possesses a 2D topology reflecting the topology of the surface of the body. Similar to the somatosensory cortex in the human brain, the map could be distorted so that more "pixels" are used for regions of the body from which more data is available (e.g. currently this might be the fingertips, if these were implemented using Syntouch technology [FL12], which has proved excellent at touch-based object identification). Input could potentially be multidimensional if multiple kinds of haptic sensors were available, e.g temperature, pressure and movement as in the Syntouch case.

Augmentation of DeSTIN to handle action as well as perception is also possible, and will be discussed in Chapter 29

28.3 Uniform DeSTIN

It would be possible to integrate DeSTIN in its original form with OpenCog or other AI systems with symbolic aspects, via using an unsupervised machine learning algorithm to recognize patterns in sets of states of the DeSTIN network as originally defined. However, this pattern recognition task becomes much easier if one suitably modifies DeSTIN, so as to make the commonalities between semantically similar states more obviously perceptible. This can be done by making the library of patterns recognized within each DeSTIN node invariant with respect to translation, scale, rotation and shear – a modification we call "Uniform DeSTIN." This "uniformization" decreases DeSTIN's degree of biological mimicry, but eases integration of DeSTIN with symbolic AI methods.

EFTA00624306

28.3.1 Translation-Invariant DeSTIN

The first revision to the "classic DeSTIN" to be suggested here is: All the nodes on the same level of the DeSTIN hierarchy should share the same library of patterns. In the context of classic DeSTIN (i.e. in the absence of further changes to DeSTIN to be suggested below, which extend the type of patterns usable by DeSTIN), this means: the nodes on the same level should share the same list of centroids. This makes DeSTIN's pattern recognition capability translation-invariant. This translation invariance can be achieved without any change to the algorithms for updating centroids and matching inputs to centroids.

In this approach, it's computationally feasible to have a much larger library of patterns utilized by each node, as compared to in classic DeSTIN. Suppose we have a $n \times n$ pixel grid, where the lowest level has nodes corresponding to 4×4 squares. Then, there are $(\frac{n}{4})^2$ nodes on the lowest level, and on the k'th level there are $(\frac{n}{4^k})^2$ nodes. This means that, without increasing computational complexity (actually decreasing it, under reasonable assumptions), in translation-invariant Uniform DeSTIN we can have a factor of $(\frac{n}{4^k})^2$ more centroids on level k.

One can achieve a much greater decrease in computational complexity (with the same amount of centroid increase) via use of a clever data structure like a cover tree [BKL06] to store the centroids at each level. Then the nearest-neighbor matching of input patterns to the library (centroid) patterns would be very rapid, much faster than linearly comparing the input to each pattern in the list.

28.3.1.1 Conceptual Justification for Uniform DeSTIN

Generally speaking, one may say that: **if** the class of images that the system will see is invariant with respect to linear translations, **then** without loss of generality, we can assume that the library of patterns at each node on the same level is the same.

In reality this assumption isn't quite going to hold. For instance, for an eye attached to a person or humanoid robot, the top of the pixel grid will probably look at a person's hair more often than the bottom ... because the person stands right-side-up more often than they stand upside-down, and because they will often fixate the center of their view on a person's face, etc. For this reason, we can recognize our friend's face better if we're looking at them directly, with their face centered in our vision.

However, we suggest that this kind of peculiarity is not really essential to vision processing for general intelligence. There's no reason you can't have an intelligent vision system that recognizes a face just as well whether it's centered in the visual field or not. (In fact you could straightforwardly explicitly introduce this kind of bias within a translation-invariant DeSTIN, but it's not clear this is a useful direction.)

By and large, in almost all cases, it seems to us that in a DeSTIN system exposed to a wide variety of real-world inputs in complex situations, the library of patterns in the different nodes at the same level would turn out to be substantially the same. Even if they weren't exactly the same, they would be close to the same, embodying essentially the same regularities. But of course, this sameness would be obscured, because centroid 7 in a certain node X on level 4 might actually be the same as centroid 18 in some other node Y on level 4 ... and there would be no way to tell that centroid 7 in node X and centroid 18 and node Y were actually referring to the same pattern, without doing a lot of work.

EFTA00624307

28.3.1.2 Comments on Biological Realism

Translation-invariant DeSTIN deviates further from human brain structure than classic DeSTIN, but this is for good reason.

The brain has a lot of neurons, since adding new neurons was fairly easy and cheap for evolution; and tends to do things in a massively parallel manner, with great redundancy. For the brain, it's not so problematically expensive to have the functional equivalent of a lot of DeSTIN nodes on the same level, all simultaneously using and learning libraries of patterns that are essentially identical to each other. Using current computer technology, on the other hand, this sort of strategy is rather inefficient.

In the brain, messaging between separated regions is expensive, whereas replicating function redundantly is cheap. In most current computers (with some partial exceptions such as GPUs), messaging between separated regions is fairly cheap (so long as those regions are stored on the same machine), whereas replicating function redundantly is expensive. Thus, even in cases where the same concept and abstract mathematical algorithm can be effectively applied in both the brain and a computer, the specifics needed for efficient implementation may be quite different.

28.3.2 Mapping States of Translation-Invariant DeSTIN into the Atomspace

Mapping classic DeSTIN's states into a symbolic pattern-manipulation engine like OpenCog is possible, but relatively cumbersome. Doing the same thing with Uniform DeSTIN is much more straightforward.

In Uniform DeSTIN, for example, Cluster 7 means the same thing in ANY node on level 4. So after a Uniform DeSTIN system has seen a fair number of images, you can be pretty sure its library of patterns is going to be relatively stable. Some clusters may come and go as learning progresses, but there's going to be a large and solid library of clusters at each level that persists, because all of its member clusters occur reasonably often across a variety of inputs.

Define a DeSTIN state-tree as a (quaternary) tree with one node for each DeSTIN node; and living at each node, a small list of (integer pattern_code, float weight) pairs. That is, at each node, the state-tree has a short-list of the patterns that closely match a given state at that node. The weights may be assumed between 0 and 1. The integer pattern codes have the same meaning for every node on the same level.

As you feed DeSTIN inputs, at each point in time it will have a certain state, representable as a state-tree. So, suppose you have a large database of DeSTIN state-trees, obtained by showing various inputs to DeSTIN over a long period of time. Then, you can do various kinds of pattern recognition on this database of state-trees.

More formally, define a state-subtree as a (quaternary) tree with a single integer at each node. Two state-subtrees may have various relationships with each other within a single state-tree – for instance they may be adjacent to each other, or one may appear atop or below the other, etc. In these terms, one interesting kind of pattern recognition to do is: Recognize frequent state-subtrees in the stored library of state-trees; and then recognize frequent relationships between these frequent state-subtrees. The latter relationships will form a kind of "image grammar," conceptually similar and formally related to those described in [ZM06]. Further, temporal pat-

EFTA00624308

terns may be recognized in the same way as spatial ones, as part of the state-subtree grammar (e.g. state-subtree A often occurs right before state-subtree B; state-subtree C often occurs right before and right below state-subtree D; etc.).

The flow of activation from OpenCog back down to DeSTIN is also fairly straightforward in the context of translation-invariant DeSTIN. If relationships have been stored between concepts in OpenCogPrime's memory and grammatical patterns between state-subtrees, then whenever concept C becomes important in OpenCogPrime's memory, this can cause a top-down increase in the probability of matching inputs to DeSTIN node centroids, that would cause the DeSTIN state-tree to contain the grammatical patterns corresponding to concept C.

28.3.3 Scale-Invariant DeSTIN

The next step, moving beyond translation invariance, is to make DeSTIN's pattern recognition mostly (not wholly) scale invariant. We will describe a straightforward way to map centroids on one level of DeSTIN, into centroids on the other levels of DeSTIN. This means that when a centroid has been learned on one level, it can be experimentally ported to all the other levels, to see if it may be useful there too.

To make the explanation of this mapping clear, we reiterate some DeSTIN basics in slightly different language:

- A centroid on Level N is: a spatial arrangement (e.g. $k \times k$ square lattice) of beliefs of Level $N-1$. (More generally it is a spatiotemporal arrangement of such beliefs, but we will ignore this for the moment.)
- A belief on Level N is: a probability distribution over centroids on Level N. For heuristic purposes one can think about this as a mixture of Gaussians, though this won't always be the best model.
- Thus, a belief on Level N is: a probability distribution over spatial (or more generally, spatiotemporal) arrangements of beliefs on Level $N-1$

On Level 1, the role of centroids is played by simple $k \times k$ squares of pixels. Level 1 beliefs are probability distributions over these small pixel squares. Level 2 centroids are hence spatial arrangements of probability distributions over small pixel-squares; and Level 2 beliefs are probability distributions over spatial arrangements of probability distributions over small pixel-squares.

A small pixel-square S may be mapped into a single pixel P via a heuristic algorithm such as:

- if S has more black than white pixels, then P is black
- is S has more white than black pixels, then P is white
- if S has an equal number of white and black pixels, then use some heuristic. For instance if S is 4×4 you could look at the central 2×2 square and assign P to the color that occurs most often there. If that is also a tie, then you can just arbitrarily assign P to the color that occurs in the upper left corner of S.

A probability distribution over small pixel-squares may then be mapped into a probability distribution over pixel values (B or W). A probability distribution over the two values B and Wmay be approximatively mapped into a single pixel value – the one that occurs most often

EFTA00624309

in the distribution, with a random choice made to break a tie. This tells us how to map Level 2 beliefs into spatial arrangements of pixels; and thus, it tells us how to map Level 2 beliefs into Level 1 beliefs.

But this tells us how to map Level N beliefs into Level $N-1$ beliefs, inductively. Remember, a Level N belief is a probability distribution (pdf for short) over spatial arrangements of beliefs on Level $N-1$. For example: A Level 3 belief if a pdf over arrangements of Level 2 beliefs. But since we can map Level 2 beliefs into Level 1 beliefs, this means we can map a Level 3 belief into a pdf over arrangements of Level 1 beliefs – which means we can map a Level 3 belief into a Level 2 belief. Etc.

Of course, this also tells us how to map Level N centroids into Level $N-1$ centroids. A Level N centroid is a pdf over arrangements of Level $N-1$ beliefs; a Level $N-1$ centroid is a pdf over arrangements of Level $N-2$ beliefs. But Level $N-1$ beliefs can be mapped into Level $N-2$ beliefs, so Level N centroids can be represented as pdfs over arrangements of Level N beliefs, and hence mapped into Level $N-1$ centroids.

In practice, one can implement this idea by moving from the bottom up. Given the mapping from Level 1 "centroids" to pixels, one can iterate through the Level 1 beliefs and identify which pixels they correspond to. Then one can iterate through the Level 2 beliefs and identify which Level 1 beliefs they correspond to. Etc. Each Level N belief can be explicitly linked to a corresponding level $N-1$ belief. Synchronously, as one moves up the hierarchy, Level N centroids can be explicitly linked to corresponding Level $N-1$ centroids.

Since there are in principle more possible Level Nbeliefs than Level $N-1$ beliefs, the mapping from level Nbeliefs to level $N-1$ beliefs is many-to-one. This is a reason not to simply maintain a single centroid pool across levels. However, when a new centroid C is added to the Level N pool, it can be mapped into a Level $N-1$ centroid to be added to the Level $N-1$ pool (if not there already). And, it can also be used to spawn a Level $N+1$ centroid, drawn randomly from the set of possible Level $N+1$ centroids that map into C.

Also, note that it is possible to maintain a single centroid *numbering system* across levels, so that a reference like "centroid # 175" has only one meaning in an entire DeSTIN network, even though some of these centroid may only be meaningful above a certain level in the network.

28.3.4 Rotation Invariant DeSTIN

With a little more work, one can make DeSTIN rotation and shear invariant as well [1]. Considering rotation first:

- When comparing an input A to a Level N node with a Level N centroid B, consider various rotations of A, and see which rotation gives the closest match.
- When you match a centroid to an input observation-or-belief, record the rotation angle corresponding to the match.

The second of these points implies the tweaked definitions

- A centroid on Level N is: a spatial arrangement (e.g. $k \times k$ square lattice) of beliefs of Level $N-1$
- A belief on Level N is: a probability distribution over (angle, centroid) pairs on Level N.

[1] The basic idea in this section, in the context of rotation, is due to Jade O'Neill (private communication)

EFTA00624310

From these it follows that a belief on Level N is: a probability distribution over (angle, spatial arrangement of beliefs) pairs on Level $N-1$

An additional complexity here is that two different (angle, centroid) pairs (on the same level) could be (exactly or approximately) equal to each other. This necessitates an additional step of "centroid simplification", in which ongoing checks are made to see if there are any two centroids C_1, C_2 on the same level so that: There exist angles A_1, A_2 so that (A_1, C_1) is very close to (A_2, C_2). In this case the two centroids may be merged into one.

To apply these same ideas to shear, one may simply replace "rotation angle" in the above by "(rotation angle, shear factor) pair."

28.3.5 Temporal Perception

Translation and scale invariant DeSTIN can be applied perfectly well if the inputs to DeSTIN, at level 1, are movies rather than static images. Then, in the simplest version, Level 1 consists of pixel cubes instead of pixel squares, etc. (the third dimension in the cube representing time). The scale invariance achieved by the methods described above would then be scale invariance in time as well as in space.

In this context, one may enable rectangular shapes as well as cubes. That is, one can look at a Level N centroid consisting of m time-slices of a $k \times k$ arrangement of Level $N-1$ beliefs – without requiring that $m = k$ .... This would make the centroid learning algorithm a little more complex, because at each level one would want to consider centroids with various values of m, from $m = 1, ..., k$ (and potentially $m > k$ also).

28.4 Interpretation of DeSTIN's Activity

Uniform DeSTIN constitutes a substantial change in how DeSTIN does its business of recognizing patterns in the world – conceptually as well as technically. To explicate the meaning of these changes, we briefly present our favored interpretation of DeSTIN's dynamics.

The centroids in the DeSTIN library represent points in "spatial pattern space", i.e. they represent exemplary spatial patterns. DeSTIN's beliefs, as probability distributions over centroids, represent guesses as to which of the exemplary spatial patterns are the best models of what's currently being seen in a certain space-time region.

This matching between observations and centroids might seem to be a simple matter of "nearest neighbor matching"; but the subtle point is, it's not immediately obvious how to best measure the distance between observations and centroids. The optimal way of measuring distance is going to depend on context; that is to say, on the actual distribution of observations in the system's real environment over time.

DeSTIN's algorithm for calculating the belief at a node, based on the observation and centroids at that node **plus** the beliefs at other nearby nodes, is essentially a way of tweaking the distance measurement between observations and centroids, so that this measurement accounts for the context (the historical distribution of observations). There are many possible ways of doing this tweaking. Ideally one could use probability theory explicitly, but that's not always

EFTA00624311

going to be computationally feasible, so heuristics may be valuable, and various versions of DeSTIN have contained various heuristics in this regard.

The various ways of "uniformizing" DeSTIN described above (i.e. making its pattern recognition activity approximately invariant with respect to affine transformations), don't really affect this story – they just improve the algorithm's ability to learn based on small amounts of data (and its rapidity at learning from data in general), by removing the need for the system to repeatedly re-learn transformed versions of the same patterns. So the uniformization just lets DeSTIN carry out its basic activity faster and using less data.

28.4.1 DeSTIN's Assumption of Hierarchical Decomposability

Roughly speaking, DeSTIN will work well to the extent that: The average distance between each part of an actually observed spatial pattern, and the closest centroid pattern, is not too large (note: the choice of distance measure in this statement is potentially subtle). That is: DeSTIN's set of centroids is supposed to provide a compact model of the probability distribution of spatial patterns appearing in the experience of the cognitive system of which DeSTIN is a part.

DeSTIN's effective functionality relies on the assumption that this probability distribution is hierarchically decomposable – i.e. that the distribution of spatial patterns appearing over a $k \times k$ region can be compactly expressed, to a reasonable degree of approximation, as a spatial combination of the distributions of spatial patterns appearing over $(k/4) \times (k/4)$ regions. This assumption of hierarchical decomposability greatly simplifies the search problem that DeSTIN faces, but also restricts DeSTIN's capability to deal with more general spatial patterns that are not easily hierarchically decomposable. However, the benefits of this approach seem to outweigh the costs, given that visual patterns in the environments humans naturally encounter do seem (intuitively at least) to have this hierarchical property.

28.4.2 Distance and Utility

Above we noted that choice of distance measure involved in the assessment of DeSTIN's effective functionality is subtle. Further above, we observed that the function of DeSTIN's belief assessment is basically to figure out the contextually best way to measure the distance between the observation and the centroids at a node. These comments were both getting at the same point.

But what is the right measure of distance between two spatial patterns? Ultimately, the right measure is: the probability that the two patterns A and B can be used in the same way. That is: the system wants to identify observation A with centroid B if it has useful action-patterns involving B, and it can substitute A for B in these patterns without loss.

This is difficult to calculate in general, though – a rough proxy, which it seems will often be acceptable, is to measure the distance between A and B in terms of both

- the basic (extensional) distance between the physical patterns they embody (e.g. pixel by pixel distance)
- the contextual (intensional) distance, i.e. the difference between the contexts in which they occur

EFTA00624312

Via enabling the belief in a node's parent to play a role in modulating a certain node's belief, DeSTIN's core algorithm enables contextual/intensional factors to play a role in distance assessment.

28.5 Benefits and Costs of Uniform DeSTIN

We now summarize the main benefits and costs of Uniform DeSTIN a little more systematically. The key point we have made here regarding Uniform DeSTIN and representational transparency may be summarized as follows:

- Define an "affine perceptual equivalence class" as a set of percepts that are equivalent to each other, or nearly so, under affine transformation. An example would be views of the same object from different perspectives or distances.
- Suppose one has an embodied agent using DeSTIN for visual perception, whose perceptual stream tends to include a lot of reasonably large affine perceptual equivalence classes.
- Then, supposing the "mechanics" of DeSTIN can be transferred to the Uniform DeSTIN case without dramatic loss of performance, Uniform DeSTIN should be able to recognize patterns based on many fewer examples than classic DeSTIN.

As soon as Uniform DeSTIN has learned to recognize one element of a given affine perceptual equivalence class, it can recognize all of them. Whereas, classic DeSTIN must learn each element of the equivalence class separately. So, roughly speaking, the number of cases required for unsupervised training of Uniform DeSTIN will be less than that for classic DeSTIN, by a ratio equal to the average size of the affine perceptual equivalence classes in the agent's perceptual stream.

Counterbalancing this, we have the performance cost of comparing the input to each node against a much larger set of centroids (in Uniform DeSTIN as opposed to classic DeSTIN). However, if a cover tree or other efficient data structure is used, this cost is not so onerous. The cost of nearest neighbor queries in a cover tree storing n items (in this case, n centroids) is $O(c^{12}logn)$, where the constant c represents the "intrinsic dimensionality" of the data; and in practice the cover tree search algorithm seems to perform quite well. So, the added time cost for online clustering in Uniform DeSTIN as opposed to DeSTIN, is a factor on the order of the log of the number of nodes in the DeSTIN tree. We believe this moderate added time cost is well worth paying, to gain a significant decrease in the number of training examples required for unsupervised learning.

Beyond increases in computational cost, there is also the risk that the online clustering may just not work as well when one has so many clusters in each node. This is the sort of problem that can really only be identified, and dealt with, during extensive practice – since the performance of any clustering algorithm is largely determined by the specific distribution of the data it's dealing with. It may be necessary to improve DeSTIN's online clustering in some way to make Uniform DeSTIN work optimally, e.g. improving its ability to form clusters with markedly non-spherical shapes. This ties in to a point raised in chapter 29 – the possibility of supplementing traditional clusters with predicates learned by CogPrime, which may live inside DeSTIN nodes alongside centroids. Each such predicate in effect defines a (generally nonconvex) "cluster".

EFTA00624313

28.6 Imprecise Probability as a Tool for Linking CogPrime and DeSTIN

One key aspect of vision processing is the ability to preferentially focus attention on certain positions within a perceived visual scene. In this section we describe a novel strategy for enabling this in a hybrid CogPrime /DeSTIN system, via use of imprecise probabilities. In fact the basic idea suggested here applies to any probabilistic sensory system, whether deep-learning-based or not, and whether oriented toward vision or some other sensory modality. However, for sake of concreteness, we will focus here on the case of DeSTIN/CogPrime integration.

28.6.1 Visual Attention Focusing

Since visual input streams contain vast amounts of data, it's beneficial for a vision system to be able to focus its attention specifically on the most important parts of its input. Sometimes knowledge of what's important will come from cognition and long-term memory, but sometimes it may come from mathematical heuristics applied to the visual data itself.

In the human visual system the latter kind of "low level attention focusing" is achieved largely in the context of the eye changing its focus frequently, looking preferentially at certain positions in the scene [Cha09]. This works because the center of the eye corresponds to a greater density of neurons than the periphery.

So for example, consider a computer vision algorithm like SIFT (Scale-Invariant Feature Extraction) [Low99], which (as shown in Figure 28.1) mathematically isolates certain points in a visual scene as "keypoints" which are particularly important for identifying what the scene depicts (e.g. these may be corners, or easily identifiable curves in edges). The human eye, when looking at a scene, would probably spend a greater percentage of its time focusing on the SIFT keypoints than on random points in the image.

The human visual system's strategy for low-level attention focusing is obviously workable (at least in contexts similar to those in which the human eye evolved), but it's also somewhat complex, requiring the use of subtle temporal processing to interpret even static scenes. We suggest here that there may be a simpler way to achieve the same thing, in the context of vision systems that are substantially probabilistic in nature, via using imprecise probabilities. The crux of the idea is to represent the most important data, e.g. keypoints, using imprecise probability values with greater confidence.

Similarly, cognition-guided visual attention-focusing occurs when a mind's broader knowledge of the world tells it that certain parts of the visual input may be more interesting to study than others. For example, in a picture of a person walking down a dark street, the contours of the person may not be tremendously striking visually (according to SIFT or similar approaches); but even so, if the system as a whole knows that it's looking at a person, it may decide to focus extra visual attention on anything person-like. This sort of cognition guided visual attention focusing, we suggest, may be achieved similarly to visual attention focusing guided on lower-level cues – by increasing the confidence of the imprecise probabilities associated with those aspects of the input that are judged more cognitively significant.

EFTA00624314

28.6.2 Using Imprecise Probabilities to Guide Visual Attention Focusing

Suppose one has a vision system that internally constructs probabilistic values corresponding to small local regions in visual input (these could be pixels or voxels, or something a little larger), and then (perhaps via a complex process) assigns probabilities to different interpretations of the input based on combinations of these input-level probabilities. For this sort of vision system, one may be able to achieve focusing of attention via appropriately replacing the probabilities with imprecise probabilities. Such an approach may be especially interesting in hierarchical vision systems, that also involve the calculation of probabilities corresponding to larger regions of the visual input. Examples of the latter include deep learning based vision systems like HTM or DeSTIN, which construct nested hierarchies corresponding to larger and larger regions of the input space, and calculate probabilities associated with each of the regions on each level, based in part on the probabilities associated with other related regions.

In this context, we now state the basic suggestion of the section:

1. Assign higher confidence to the low-level probabilities that the vision system creates corresponding to the local visual regions that one wants to focus attention on (based on cues from visual preprocessing or cognitive guidance)
2. Carry out the vision system's processing using imprecise probabilities rather than single-number probabilities
3. Wherever the vision system makes a decision based on "the most probable choice" from a number of possibilities, change the system to make a decision based on "the choice maximizing the product (expectation * confidence)".

28.6.3 Sketch of Application to DeSTIN

Internally to DeSTIN, probabilities are assigned to clusters associated with local regions of the visual input. If a system such as SIFT is run as a preprocessor to DeSTIN, then those small regions corresponding to SIFT keypoints may be assumed semantically meaningful, and internal DeSTIN probabilities associated with them can be given a high confidence. A similar strategy may be taken if a cognitive system such as OpenCog is run together with DeSTIN, feeding DeSTIN information on which portions of a partially-processed image appear most cognitively relevant. The probabilistic calculations inside DeSTIN can be replaced with corresponding calculations involving imprecise probabilities. And critically, there is a step in DeSTIN where, among a set of beliefs about the state in each region of an image (on each of a set of hierarchical levels), the one with the highest probability is selected. In accordance with the above recipe, this step should be modified to select the belief with the highest probability * confidence.

28.6.3.1 Conceptual Justification

What is the conceptual justification for this approach?

One justification is obtained by assuming that each percept has a certain probability of being erroneous, and those percepts that appear to more closely embody the semantic meaning of the

EFTA00624315

visual scene are less likely to be erroneous. This follows conceptually from the assumption that the perceived world tends to be patterned and structured, so that being part of a statistically significant pattern is (perhaps weak) evidence of being real rather than artifactual. Under this assumption, the proposed approach will maximize the accuracy of the system's judgments.

A related justification is obtained by observing that this algorithmic approach follows from the consideration of the perceived world as mutable. Consider a vision system that has the capability to modify even the low-level percepts that it intakes – i.e. to use what it thinks and knows, to modify what it sees. The human brain certainly has this potential [Cha09]. In this case, it will make sense for the system to place some constraints regarding which of its percepts it is more likely to modify. Confidence values semantically embody this – a higher confidence being sensibly assigned to percepts that the system considers should be less likely to be modified based on feedback from its higher (more cognitive) processing levels. In that case, a higher confidence should be given to those percepts that seem to more closely embody the semantic meaning of the visual scene – which is exactly what we're suggesting here.

28.6.3.2 Enabling Visual Attention Focusing in DeSTIN via Imprecise Probabilities

We now refer back to the mathematical formulation of DeSTIN summarized in Section 4.3.1 of Chapter 4 above, in the context of which the application of imprecise probability based attention focusing to DeSTIN is almost immediate.

The probabilities $P(o|s)$ may be assigned greater or lesser confidence depending on the assessed semantic criticality of the observation o in question. So for instance, if one is using SIFT as a preprocessor to DeSTIN, then one may assign probabilities $P(o|s)$ higher confidence if they correspond to observations o of SIFT keypoints, than if they do not.

These confidence levels may then be propagated throughout DeSTIN's probabilistic mathematics. For instance, if one were using Walley's interval probabilities, then one could carry out the probabilistic equations using interval arithmetic.

Finally, one wishes to replace Equation 4.3.1.2 in Chapter 4 with

$$c = \arg\max_s \left((b_p(s)).\text{strength} * (b_p(s)).\text{confidence} \right), \tag{28.1}$$

or some similar variant. The effect of this is that hypotheses based on high-confidence observations are more likely to be chosen, which of course has a large impact on the dynamics of the DeSTIN network.

EFTA00624316

Fig. 28.1: The SIFT algorithm finds keypoints in an image, i.e. localized features that are particularly useful for identifying the objects in an image. The top row shows images that are matched against the image in the middle row. The bottom-row image shows some of the keypoints used to perform the matching (i.e. these keypoints demonstrate the same features in the top-row images and their transformed middle-row counterparts). SIFT keypoints are identified via a staged filtering approach. The first stage identifies key locations in scale space by looking for locations that are maxima or minima of a difference-of-Gaussian function. Each point is used to generate a feature vector that describes the local image region sampled relative to its scale-space coordinate frame. The features achieve partial invariance to local variations, such as affine or 3D projections, by blurring image gradient locations.

EFTA00624317

Chapter 29
Bridging the Symbolic/Subsymbolic Gap

29.1 Introduction

While it's widely accepted that human beings carry out both *symbolic* and *subsymbolic* processing, as integral parts of their general intelligence, the precise definition of "symbolic" versus "subsymbolic" is a subtle issue, which different AI researchers will approach in different ways depending on their differing overall perspectives on AI. Nevertheless, the intuitive meaning of the concepts is commonly understood:

- **"subsymbolic"** refers to things like pattern recognition in high-dimensional quantitative sensory data, and real-time coordination of multiple actuators taking multidimensional control signals
- **"symbolic"** refers to things like natural language grammar and (certain or uncertain) logical reasoning, that are naturally modeled in terms of manipulation of symbolic tokens in terms of particular (perhaps experientially learned) rules

Views on the relationship between these two aspects of intelligence in human and artificial cognition are quite diverse, including perspectives such as

1. Symbolic representation and reasoning are the core of human-level intelligence; subsymbolic aspects of intelligence are of secondary importance and can be thought of as pre- or post-processors to symbolic representation and reasoning
2. Subsymbolic representation and learning are the core of human intelligence; symbolic aspects of intelligence
    a. emerge from the subsymbolic aspects as needed; or,
    b. arise via a relatively simple, thin layer on top of subsymbolic intelligence, that merely applies subsymbolic intelligence in a slightly different way
3. Symbolic and subsymbolic aspects of intelligence are best considered as different subsystems, which
    a. have a significant degree of independent operation, but also need to coordinate closely together; or,
    b. operate largely separately and can be mostly considered as discrete modules

EFTA00624318

In evolutionary terms, it is clear that subsymbolic intelligence came first, and that most of the human brain is concerned with the subsymbolic intelligence that humans share with other animals. However, this observation doesn't have clear implications regarding the relationship between symbolic and subsymbolic intelligence in the context of everyday cognition.

In the history of the AI field, the symbolic/subsymbolic distinction was sometimes aligned with the dichotomy between logic-based and rule-based AI systems (on the symbolic side) and neural networks (on the subsymbolic side) [PJ88b]. However, this dichotomy has become much blurrier in the last couple decades, with developments such as neural network models of language parsing [GH11] and logical reasoning [LBH10], and symbolic approaches to perception and action [SR04]. Integrative approaches have also become more common, with one of the major traditional symbolic AI systems, ACT-R, spawning a neural network version [LA93] with parallel structures and dynamics to the traditional explicitly symbolic version and a hybridization with a computational neuroscience model [JL08]; and another one, SOAR, incorporating perception processing components as separate modules [Lai12]. The field of "neural-symbolic computing" has emerged, covering the emergence of symbolic rules from neural networks, and the hybridization of neural networks with explicitly symbolic systems [HH07].

Our goal here is not to explore the numerous deep issues involved with the symbolic/subsymbolic dichotomy, but rather to describe the details of a particular approach to symbolic/subsymbolic integration, inspired by Perspective 3a in the above list: the consideration of symbolic and subsymbolic aspects of intelligence as different subsystems, which have a significant degree of independent operation, but also need to coordinate closely together. We believe this kind of integration can serve a key role in the quest to create human-level general intelligence. The approach presented here is at the beginning rather than end of its practical implementation; what we are describing here is the initial design intention of a project in progress, which is sure to be revised in some respects as implementation and testing proceed. We will focus mainly on the tight integration of a subsymbolic system enabling gray-scale vision processing into a cognitive architecture with significant symbolic aspects, and will then briefly explain how the same ideas can be used for color vision, and multi-sensory and perception-action integration.

The approach presented here begins with two separate AI systems, OpenCog (introduced in Chapter 6.3) and DeSTIN (introduced in Chapter 4.3.1) - both currently implemented in open-source software. Here are the relevant features of each as they pertain to our current effort of bridging the symbolic/subsymbolic gap::

- **OpenCog**, is centered on a "weighted, labeled hypergraph" knowledge representation called the Atomspace, and features a number of different, sophisticated cognitive algorithms acting on the Atomspace. Some of these cognitive algorithms are heavily symbolic in focus (e.g. a probabilistic logic engine); others are more subsymbolic in nature (e.g. a neural net like system for allocating attention and assigning credit). However, OpenCog in its current form cannot deal with high-dimensional perceptual input, nor with detailed real-time control of complex actuators. OpenCog is now being used to control intelligent characters in an experimental virtual world, where the perceptual inputs are the 3D coordinate locations of objects or small blocks; and the actions are movement commands like "step forward", "turn head to the right."
- **DeSTIN** is a deep learning system consisting of a hierarchy of processing nodes, in which the nodes on higher levels correspond to larger regions of space-time, and each node carries out prediction regarding events in the space-time region to which it corresponds. Feedback and feedforward dynamics between nodes combine with the predictive activity within nodes, to create a complex nonlinear dynamical system whose state self-organizes to reflect the

EFTA00624319

state of the world being perceived. However, the specifics of DeSTIN's dynamics have been designed in what we consider a particularly powerful way, and the system has shown good results on small-scale test problems [KAR10]. So far DeSTIN has been utilized only for vision processing, but a similar proprietary system has been used for auditory data as well; and DeSTIN was designed to work together with an accompanying action hierarchy.

These two systems were not originally designed to work together, but we will describe a method for achieving their tight integration via

1. Modifying DeSTIN in several ways, so that
   a. the patterns in its states over time will have more easily recognizable regularities
   b. its nodes are able to scan their inputs not only for simple statistical patterns (DeSTIN "centroids"), but also for patterns recognized by routines supplied to it by an external source (e.g. another AI system such as OpenCog)
2. Utilizing one of OpenCogPrime's cognitive processes (the "Fishgram" frequent subhypergraph mining algorithm) to recognize patterns in sets of DeSTIN states, and then recording these patterns in OpenCogPrime's Atomspace knowledge store
3. Utilizing OpenCogPrime's other cognitive processes to abstract concepts and draw conclusions from the patterns recognized in DeSTIN states by Fishgram
4. Exporting the concepts and conclusions thus formed to DeSTIN, so that its nodes can explicitly scan for their presence in their inputs, thus allowing the results of symbolic cognition to explicitly guide subsymbolic perception
5. Creating an action hierarchy corresponding closely to DeSTIN's perceptual hierarchy, and also corresponding to the actuators of a particular robot. This allows action learning to be done via an optimization approach ([LKP+05], [YKL+04]), where the optimization algorithm uses DeSTIN states corresponding to perceived actuator states as part of its inputs.

The ideas presented here are compatible with those described in [Goe11a], but different in emphasis. That paper described a strategy for integrating OpenCog and DeSTIN via creating an intermediate "semantic CSDLN" hierarchy to translate between OpenCog and DeSTIN, in both directions. In the approach suggested here, this semantic CSDLN hierarchy exists conceptually but not as a separate software object: it exists as the combination of

- OpenCog predicates exported to DeSTIN and used alongside DeSTIN centroids, inside DeSTIN nodes
- OpenCog predicates living in the OpenCog knowledge repository (AtomSpace), and interconnected in a hierarchical way using OpenCog nodes and links (thus reflecting DeSTIN's hierarchical structure within the AtomSpace).

This hierarchical network of predicates, spanning the two software systems, plays the role of a semantic CSDLN as described in [Goe11a].

29.2 Simplified OpenCog Workflow

The dynamics inside an OpenCog system may be highly complex, defying simple flowcharting, but from the point of view of OpenCog-DeSTIN integration, one important pattern of information flow through the system is as follows:

EFTA00624320

1. Perceptions come into the Atomspace. In the current OpenCog system, these are provided via a proxy to the game engine where the OpenCog controlled character interacts. In an OpenCog-DeSTIN hybrid, these will be provided via DeSTIN.
2. Hebbian learning builds HebbianLinks between perceptual Atoms representing percepts that have frequently co-occurred
3. PLN inference, concept blending and other methods act on these perceptual Atoms and their HebbianLinks, forming links between them and linking them to other Atoms stored in the Atomspace reflecting prior experience and generalizations therefrom
4. Attention allocation gives higher short and long term importance values to those Atoms that appear likely to be useful based on the links they have obtained
5. Based on the system's current goals and subgoals (the latter learned from the top-level goals using PLN), and the goal-related links in the Atomspace, the OpenPsi mechanism triggers the PLN-based planner, which chooses a series of high-level actions that are judged likely to help the system achieve its goals in the current context
6. The chosen high-level actions are transformed into series of lower-level, directly executable actions. In the current OpenCog system, this is done by a set of hand-coded rules based on the specific mechanics of the game engine where the OpenCog controlled character interacts. In an OpenCog-DeSTIN hybrid, the lower-level action sequence will be chosen by an optimization method acting based on the motor control and perceptual hierarchies.

This pattern of information flow omits numerous aspects of OpenCog cognitive dynamics, but gives the key parts of the picture in terms of the interaction of OpenCog cognition with perception and action. Most of the other aspects of the dynamics have to do with the interaction of multiple cognitive processes acting on the Atomspace, and the interaction between the Atomspace and several associated specialized memory stores, dealing with procedural, episodic, temporal and spatial aspects of knowledge. From the present point of view, these additional aspects may be viewed as part of Step 3 above, wrapped up in the phrase "and other methods act on these perceptual Atoms." However, it's worth noting that in order to act appropriately on perceptual Atoms, a lot of background cognition regarding more abstract conceptual Atoms (often generalized from previous perceptual Atoms) may be drawn on. This background inference incorporates both symbolic and subsymbolic aspects, but goes beyond the scope of the present discussion, as its particulars do not impinge on the particulars of DeSTIN-OpenCog integration.

OpenCog also possesses a specialized facility for natural language comprehension and generation [LGE10] [Goe10b], which may be viewed as a parallel perception/action pathway, bypassing traditional human-like sense perception and dealing with text directly. Integrating OpenCog-Prime's current linguistics processes with DeSTIN-based auditory and visual processing is a deep and important topic, but one we will bypass here, for sake of brevity and because it's not our current research priority.

29.3 Integrating DeSTIN and OpenCog

The integration of DeSTIN and OpenCog involves two key aspects:

- recognition of patterns in sets of DeSTIN states, and exportation of these patterns into the OpenCog Atomspace

EFTA00624321

- use of OpenCog-created concepts within DeSTIN nodes, alongside statistically-derived "centroids"

From here on, unless specified otherwise, when we mention "DeSTIN" we will refer to "Uniform DeSTIN" as presented in Chapter 28 and an extension of "classic DeSTIN" as defined in [ARK09].

29.3.1 Mining Patterns from DeSTIN States

The first step toward using OpenCog tools to mine patterns from sets of DeSTIN states, is to represent these states in Atom form in an appropriate way. A simple but workable approach, restricting attention for the moment to purely spatial patterns, is to use the six predicates:

- $hasCentroid(node\ N, int\ k)$
- $hasParentCentroid(node\ N, int\ k)$
- $hasNorthNeighborCentroid(node\ N, int\ k)$
- $hasSouthNeighborCentroid(node\ N, int\ k)$
- $hasEastNeighborCentroid(node\ N, int\ k)$
- $hasWestNeighborCentroid(node\ N, int\ k)$

For instance

$$hasNorthNeighborCentroid(N, 3)$$

means that N's north neighbor has centroid #3

One may consider also the predicates

- $hasParent(node\ N, Node\ M)$
- $hasNorthNeighbor(node\ N, Node\ M)$
- $hasSouthNeighbor(node\ N, Node\ M)$
- $hasEastNeighbor(node\ N, Node\ M)$
- $hasWestNeighbor(node\ N, Node\ M)$

Now suppose we have a stored set of DeSTIN states, saved from the application of DeSTIN to multiple different inputs. What we want to find are predicates P that are *conjunctions* of instances of the above 10 predicates, which occur frequently in the stored set of DeSTIN states. A simple example of such a predicate would be the conjunction of

- $hasNorthNeighbor(\$N, \$M)$
- $hasParentCentroid(\$N, 5)$
- $hasParentCentroid(\$M, 5)$
- $hasNorthNeighborCentroid(\$N, 6)$
- $hasWestNeighborCentroid(\$M, 4)$

This predicate could be evaluated at any pair of nodes $(\$N, \$M)$ on the same DeSTIN level. If it is true for atypically many of these pairs, then it's a "frequent pattern", and should be detected and stored.

EFTA00624322

OpenCogPrime's pattern mining component, Fishgram, exists precisely for the purpose of mining this sort of conjunction from sets of relationships that are stored in the Atomspace. It may be applied to this problem as follows:

- Translate each DeSTIN state into a set of relationships drawn from: hasNorthNeighbor, hasSouthNeighbor, hasEastNeighbor, hasWestNeighbor, hasCentroid, hasParent
- Import these relationships, describing each DeSTIN state, into the OpenCog Atomspace
- Run pattern mining on this AtomSpace.

29.3.2 Probabilistic Inference on Mined Hypergraphs

Patterns mined from DeSTIN states can then be reasoned on by OpenCogPrime's PLN inference engine, allowing analogy and generalization.

Suppose centroids 5 and 617 are estimated to be similar – either via DeSTIN's built-in similarity metric, or, more interestingly via OpenCog inference on the Atom representations of these centroids. As an example of the latter, consider: 5 could represent a person's nose and 617 could represent a rabbit's nose. In this case, DeSTIN might not judge the two centroids particularly similar on a purely visual level, but, OpenCog may know that the images corresponding to both of these centroids are are called "noses" (e.g. perhaps via noticing people indicate these images in association with the word "nose"), and may thus infer (using a simple chain of PLN inferences) that these centroids seem probabilistically similar.

If 5 and 617 are estimated to be similar, then a predicate like

```
ANDLink
   EvaluationLink
      hasNorthNeighbor
      ListLink $N  $M
   EvaluationLink
      hasParentCentroid
      ListLink $N  5
   EvaluationLink
      hasParentCentroid
      ListLink $M  5
   EvaluationLink
      hasNorthNeighborCentroid
      ListLink $N 6
   EvaluationLink
      hasWestNeighborCentroid
      ListLink $M 4
```

mined from DeSTIN states, could be extended via PLN analogical reasoning to

```
ANDLink
   EvaluationLink
      hasNorthNeighbor
      ListLink $N $M
   EvaluationLink
```

EFTA00624323

```
      hasParentCentroid
      ListLink $N  617
   EvaluationLink
      hasParentCentroid
      ListLink $M  617
   EvaluationLink
      hasNorthNeighborCentroid
      ListLink $N 6
   EvaluationLink
      hasWestNeighborCentroid
      ListLink $M 4
```

29.3.3 Insertion of OpenCog-Learned Predicates into DeSTIN's Pattern Library

Suppose one has used Fishgram, as described in the earlier part of this chapter, to recognize predicates embodying frequent or surprising patterns in a set of DeSTIN states or state-sequences. The next natural step is to add these frequent or surprising patterns to DeSTIN's pattern library, so that the pattern library contains not only classic DeSTIN centroids, but also these corresponding "image grammar" style patterns. Then, when a new input comes into a DeSTIN node, in addition to being compared to the centroids at the node, it can be fed as input to the predicates associated with the node.

What is the advantage of this approach, compared to DeSTIN without these predicates? The capability for more compact representation of a variety of spatial patterns. In many cases, a spatial pattern that would require a large number of DeSTIN centroids to represent, can be represented by a single, fairly compact predicate. It is an open question whether these sorts of predicates are really critical for human-like vision processing. However, our intuition is that they do have a role in human as well as machine vision. In essence, DeSTIN is based on a fancy version of nearest-neighbor search, applied in a clever way on multiple levels of a hierarchy, using context-savvy probabilities to bias the matching. But we suspect there are many visual patterns that are more compactly and intuitively represented using a more flexible language, such as OpenCog predicates formed by combining elementary predicates involving appropriate spatial and temporal relations.

For example, consider the archetypal spatial pattern of a face as: either two eyes that are next to each other, or sunglasses, above a nose, which is in turn above a mouth. (This is an oversimplified toy example, but we're positing it for illustration only. The same point applies to more complex and realistic patterns.) One could represent this in OpenCogPrime's Atom language as something like:

```
AND
  InheritanceLink N B_nose
  InheritanceLink M B_mouth
  EvaluationLink
      above
      ListLink E N
  EvaluationLink
```

EFTA00624324

```
      above
      ListLink N M
   OR
     AND
        MemberLink E1  E
        MemberLink E2  E
        EvaluationLink
           next_to
           ListLink E1 E2
        InheritanceLink E1 B_eye
     AND
        InheritanceLink E  B_sunglasses
```

where e.g. *B_eye* is a DeSTIN belief that corresponds roughly to recognition of the spatial pattern of a human eye. To represent this using ordinary DeSTIN centroids, one couldn't represent the OR explicitly; instead one would need to split it into two different sets of centroids, corresponding to the eye case and the sunglasses case – unless the DeSTIN pattern library contained a belief corresponding to "eyes or sunglasses." But the question then becomes: how would classic DeSTIN actually learn a belief like this? In the suggested architecture, pattern mining on the database of DeSTIN states is proposed as an algorithm for learning such beliefs.

This sort of predicate-enhanced DeSTIN will have advantages over the traditional version, only if the actual distribution of images observed by the system contains many (reasonably high probability) images modeled accurately by predicates involving disjunctions and/or negations as well as conjunctions. If the system's perceived world is simpler than this, then good old DeSTIN will work just as well, and the OpenCog-learned predicates are a needless complication.

Without these sorts of predicates, how might DeSTIN be extended to include beliefs like "eyes or sunglasses"? One way would be to couple DeSTIN with a reinforcement learning subsystem, that reinforced the creation of beliefs that were useful for the system as a whole. If reasoning in terms of faces (independent of whether they have eyes or sunglasses) got the system reward, presumably it could learn to form the concept "eyes or sunglasses." We believe this would also be a workable approach, but that given the strengths and weaknesses of contemporary computer hardware, the proposed DeSTIN-OpenCog approach will prove considerably simpler and more effective.

29.4 Multisensory Integration, and Perception-Action Integration

In Chapter 28.2.1 we have briefly indicated how DeSTIN could be extended beyond vision to handle other senses such as audition and touch. If one had multiple perception hierarchies corresponding to multiple senses, the easiest way to integrate them within an OpenCog context would be to use OpenCog as the communication nexus – representing DeSTIN centroids in the various modality-specific hierarchies as OpenCog Atoms (PerceptualCentroidNodes), and building HebbianLinks in OpenCogPrime's Atomspace between these PerceptualCentroidNodes as appropriate based on their association. So for instance the sound of a person's footsteps would correspond to a certain belief (probability distribution over centroids) in the auditory DeSTIN network, and the sight of a person's feet stepping would correspond to a certain belief (probability distribution over centroids) in the visual DeSTIN network; and the OpenCog

EFTA00624325

Atomspace would contain links between the sets of centroids assigned high weights between these two belief distributions. Importance spreading between these various PerceptualCentroidNodes would cause a dynamic wherein seeing feet stepping would bias the system to think it was hearing footsteps, and hearing footsteps would bias it to think it was seeing feet stepping.

And, suppose there are similarities between the belief distributions for the visual appearance of dogs, and the visual appearance of cats. Via the intermediary of the Atomspace, this would bias the auditory and haptic DeSTIN hierarchies to assume a similarity between the auditory and haptic characteristics of dogs, and the analogous characteristics of cats. Because: PLN analogical reasoning would extrapolate from, e.g.

- HebbianLinks joining cat-related visual PerceptualCentroidNodes and dog-related visual PerceptualCentroidNodes
- HebbianLinks joining cat-related visual PerceptualCentroidNodes to cat-related haptic PerceptualCentroidNodes; and others joining dog-related visual PerceptualCentroidNodes to dog-related haptic PerceptualCentroidNodes

to yield HebbianLinks joining cat-related haptic PerceptualCentroidNodes and dog-related haptic PerceptualCentroidNodes. This sort of reasoning would then cause the system DeSTIN to, for example, upon touching a cat, vaguely expect to maybe hear dog-like things. This sort of simple analogical reasoning will be right sometimes and wrong sometimes – a cat walking sounds a fair bit like a dog walking, and cat and dog growls sound fairly similar, but a cat meowing doesn't sound that much like a dog barking. More refined inferences of the same basic sort may be used to get the details right as the system explores and understands the world more accurately.

29.4.1 Perception-Action Integration

While experimentation with DeSTIN has so far been restricted to perception processing, the system was designed from the beginning with robotics applications in mind, involving integration of perception with action and reinforcement learning. As OpenCog already handles reinforcement learning on a high level (via OpenPsi), our approach to robot control using DeSTIN and OpenCog involves creating a control hierarchy parallel to DeSTIN's perceptual hierarchy, and doing motor learning using optimization algorithms guided by reinforcement signals delivered from OpenPsi and incorporating DeSTIN perceptual states as part of their input information.

Our initial research goal, where action is concerned, is not to equal the best purely control-theoretic algorithms at fine-grained control of robots carrying out specialized tasks, but rather to achieve basic perception / control / cognition integration in the rough manner of a young human child. A two year old child is not particularly well coordinated, but is capable of coordinating actions involving multiple body parts using an integration of perception and action with unconscious and deliberative reasoning. Current robots, in some cases, can carry out specialized actions with great accuracy, but they lack this sort of integration, and thus generally have difficulty effectively carrying out actions in unforeseen environments and circumstances.

We will create an action hierarchy with nodes corresponding to different parts of the robot body, where e.g. the node corresponding to an arm would have child nodes corresponding to a shoulder, elbow, wrist and hand; and the node corresponding to a hand would have child nodes corresponding to the fingers of the hand; etc. Physical self-perception is then achieved

EFTA00624326

by creating a DeSTIN "action-perception" hierarchy with nodes corresponding to the states of body components. In the simplest case this means the lowest-level nodes will correspond to individual servomotors, and their inputs will be numerical vectors characterizing servomotor states. If one is dealing with a robot endowed with haptic technology, e.g. Syntouch [FL12] fingertips, then numerical vectors characterizing haptic inputs may be used alongside these.

The configuration space of an action-perception node, corresponding to the degrees of freedom of the servomotors of the body part the node represents, may be approximated by a set of "centroid" vectors. When an action is learned by the optimization method used for this purpose, this involves movements of the servomotors corresponding to many different nodes, and thus creates a series of "configuration vectors" in each node. These configuration vector series may be subjected to online clustering, similar to percepts in a DeSTIN perceptual hierarchy. The result is a library of "codewords", corresponding to discrete trajectories of movement, associated with each node. The libraries may be shared by identical body parts (e.g. shared among legs, shared among fingers), but will be distinct otherwise. Each coordinated whole-body action thus results in a series of (node, centroid) pairs, which may be mined for patterns, similarly to the perception case.

The set of predicates needed to characterize states in this action-perception hierarchy is simpler than the one described for visual perception above; here one requires only

- $hasCentroid(node\ N, int\ k)$
- $hasParentCentroid(node\ N, int\ k)$
- $hasParent(node\ N, Node\ M)$
- $hasSibling(node\ N, Node\ M)$

and most of the patterns will involve specific nodes rather than node variables. The different nodes in a DeSTIN vision hierarchy are more interchangeable (in terms of their involvement in various patterns) than, say, a leg and a finger.

In a pure DeSTIN implementation, the visual and action-perception hierarchies would be directly linked. In the context of OpenCog integration, it is simplest to link the two via OpenCog, in a sense using cognition as a bridge between action and perception. It is unclear whether this strategy will be sufficient in the long run, but we believe it will be more than adequate for experimentation with robotic perceptual-motor coordination in a variety of everyday tasks. OpenCogPrime's Hebbian learning process can be used to find common associations between action-perception states and visual-perception states, via mining a data store containing time-stamped state records from both hierarchies.

Importance spreading along the HebbianLinks learned in this way can then be used to bias the weights in the belief states of the nodes in both hierarchies. So, for example, the action-perception patterns related to clenching the fist, would be Hebbianly correlated with the visual-perception patterns related to seeing a clenched fist. When a clenched fist was perceived via servomotor data, importance spreading would increase the weighting of visual patterns corresponding to clenched fists, within the visual hierarchy. When a clenched fist was perceived via visual data, importance spreading would increase the weighting of servomotor data patterns corresponding to clenched fists, within the action-perception hierarchy.

EFTA00624327

29.4.2 Thought-Experiment: Eye-Hand Coordination

For example, how would DeSTIN-OpenCog integration as described here carry out a simple task of eye-hand coordination? Of course the details of such a feat, as actually achieved, would be too intricate to describe in a brief space, but it still is meaningful to describe the basic ideas. Consider the case of a robot picking up a block, in plain sight immediately in front of the robot, via pinching it between two fingers and then lifting it. In this case,

- The visual scene, including the block, is perceived by DeSTIN; and appropriate patterns in various DeSTIN nodes are formed
- Predicates corresponding to the distribution of patterns among DeSTIN nodes are activated and exported to the OpenCog Atomspace
- Recognition that a block is present is carried out, either by
  - PLN inference within OpenCog, drawing the conclusion that a block is present from the exported predicates, using ImplicationLinks comprising a working definition of a "block"
  - A predicate comprising the definition of "block", previously imported into DeSTIN from OpenCog and utilized within DeSTIN nodes as a basic pattern to be scanned for. This option would obtain only if the system had perceived many blocks in the past, justifying the automation of block recognition within the perceptual hierarchy.
- OpenCog, motivated by one of its higher-level goals, chooses "picking up the block" as subgoal. So it allocates effort to finding a procedure whose execution, in the current context, has a reasonable likelihood of achieving the goal of picking up the block. For instance, the goal could be curiosity (which might make the robot want to see what lies under the block), or the desire to please the agent's human teacher (in case the human teacher likes presents, and will reward the robot for giving it a block as a present), etc.
- OpenCog, based on its experience, uses PLN to reason that "grabbing the block" is a subgoal of "picking up the block"
- OpenCog utilizes a set of predicates corresponding to the desired state of "grabbing the the block" as a target for an optimization algorithm, designed to figure out a series of servomotor actions that will move the robot's body from the current state to the target state. This is a relatively straightforward control theory problem.
- Once the chosen series of servomotor actions has been executed, the robot has its fingers poised around the block, ready to pick it up. At this point, the action-perception hierarchy perceives what is happening in the fingers. If the block is really being grabbed properly, then the fingers are reporting some force, due to the feeling of grabbing the block (haptic input is another possibility and would be treated similarly, but we will leave that aside for now). Importance spreads from these action-perception patterns into the Atomspace, and back down into the visual perception hierarchy, stimulating concepts and percepts related to "something is being grabbed by the fingers."
- If the fingers aren't receiving enough force, because the agent is actually only poking the block with one finger and grabbing the air with another finder, then the "something is being grabbed by the fingers" stimulation doesn't happen, and the agent is less sure it's actually grabbing anything. In that case it may withdraw its hand a bit, so that it can more easily assess its hand's state visually, and try the optimization-based movement planning again.

EFTA00624328

- Once the robot estimates the goal of grabbing the block has been successfully achieved, it proceeds to the next sub-subgoal, and asks the action-sequence optimizer to find a sequence of movements that will likely cause the predicates corresponding to "hold the block up" to obtain. It then executes this movement series and picks the block up in the air.

This simple example is a far cry from the perceptual-motor coordination involved in doing embroidery, juggling or serving a tennis ball. But we believe it illustrates, in a simple way, the same basic cognitive structures and dynamics used in these more complex instances.

29.5 A Practical Example: Using Subtree Mining to Bridge the Gap Between DeSTIN and PLN

In this section we describe some relatively simple practical experiments we have run, exploring the general ideas described above. The core idea of these experiments is to apply Yun Chi's Frequent Subtree Mining software [CXYM05] [1] to mine frequent patterns from a data-store of trees representing DeSTIN states. In this application, each frequent subtree represents a "common visual pattern." These patterns may then be reasoned about using PLN. This approach may also be extended to include additional quality metrics besides frequency, e.g. interaction information [Bel03] which lets one measure how "surprising" a subtree is.

Figure **??** illustrates the overall architecture into which the use of frequent subtree mining to bridge DeSTIN and PLN is intended to fit. This architecture is not yet fully implemented, but is a straightforward extension of the current OpenCog architecture for processing data from game worlds [GEA08], and is scheduled for implementation later in 2013 in the course of a funded project involving the use of DeSTIN and OpenCog for humanoid robot control.

The components intervening between DeSTIN and OpenCog, in this architecture, are:

- **DeSTIN State DB**: Stores all DeSTIN states the system has experienced, indexed by time of occurrence
- **Frequent Subtree Miner**: Recognizes frequent subtrees in the database of DeSTIN states, and can also filter the frequent subtrees by other criteria such as information-theoretic surprisingness. These subtrees may sometimes span multiple time points.
- **Frequent Subtree Recognizer**: Scans DeSTIN output, and recognizes frequent subtrees therein. These subtrees are the high level "visual patterns" that make their way from DeSTIN to OpenCog.
- **Perception Collector**: Linearly normalizes the spatial coordinates associated with its input subtrees, to compensate for movement of the camera. Filters out perceptions that didn't change recently (e.g. a static white wall), so that only new visual information is passed along to OpenCog. Translates the subtrees into Scheme files representing OpenCog logical Atoms.
- **Experience DB**: Stores all the normalized subtrees that have actually made their way into OpenCog

[1] available for download at http://www.nec-labs.com/~ychi/publication/software.html

EFTA00624329

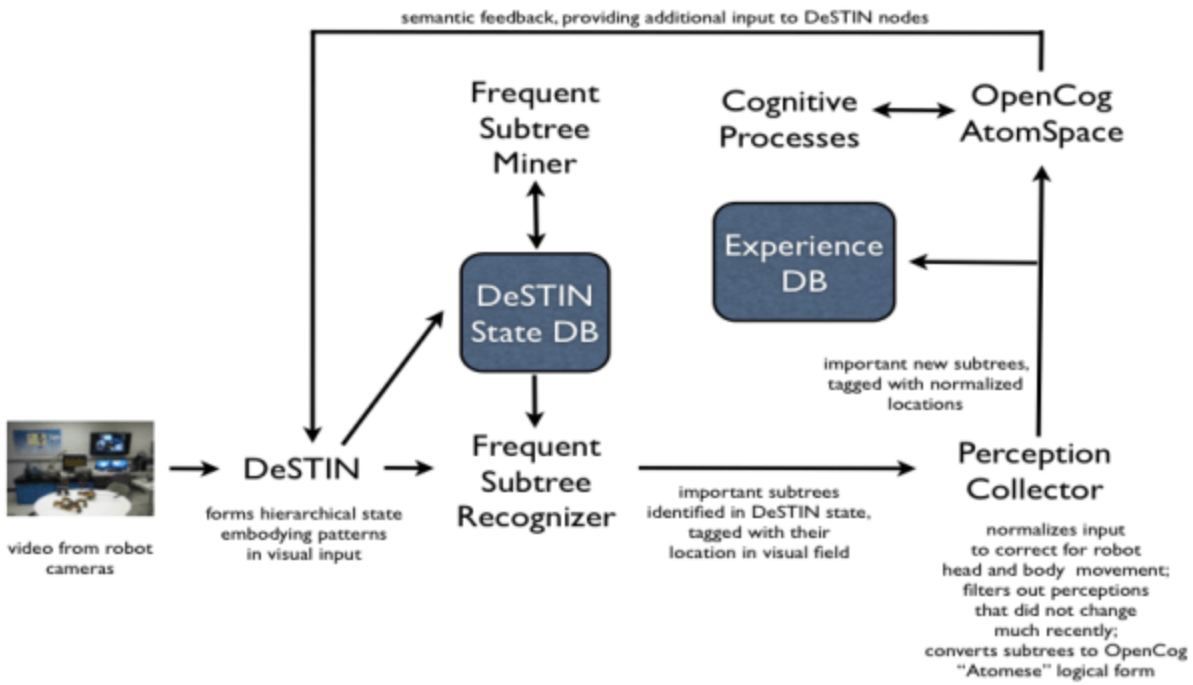


Fig. 29.1: Graphical depiction of the architecture for DeSTIN/OpenCog integration using frequent subtree mining as a bridge. Semantic feedback is not yet implemented; and sophisticated processing of DeSTIN or other visual input is not yet handled by OpenCog's Perception Collector. The experiments presented here utilized a simplified, preliminary version of the architecture.

- **Semantic Feedback**: Allows the semantic associations OpenCog makes to a subtree, to be fed back into DeSTIN as additional inputs to the nodes involved in the subtree. This allows perception to make use of cognitive information.

EFTA00624330

29.5.1 The Importance of Semantic Feedback

One aspect of the above architecture not yet implemented, but worthy of note, is semantic feedback. Without the semantic feedback, we expect to be able to emulate human object and event recognition insofar as they are done by the human brain in a span of less than 500ms or so. In this time frame, the brain cannot do much sophisticated cognitive feedback, and processes perceptual data in an essentially feedforward manner. On the other hand, properly tuned semantic feedback along with appropriate symbolic reasoning in OpenCog, may allow us to emulate human object and event recognition as the human brain does it when it has more time available, and can use its cognitive understanding to guide vision processing.

A simple example of this sort of symbolic reasoning is analogical inference. Given a visual scene, OpenCog can reason about what the robot has seen in similar situations before, where its notion of "similarity" draws not only on visual cues but on other contextual information: what time it is, what else is in the room (even if not currently visible), who has been seen in the room recently, etc.

For instance, recognizing familiar objects that are largely occluded and in dim light, may be something requiring semantic feedback, and not achievable via the feedforward dynamics alone. This can be tested in a robot vision context via showing the robot various objects and events in various conditions of lighting and occlusion, and observing its capability at recognizing the objects and events with and without semantic feedback, in each of the conditions.

If the robot sees an occluded object in a half-dark area on a desk, and it knows that a woman was recently sitting at that desk and then got up and left the room, its symbolic analogical inference may make it more likely to conclude that the object is a purse. Without this symbolic inference, it might not be able to recognize the object as a purse based on bottom-up visual clues alone.

29.6 Some Simple Experiments with Letters

To illustrate the above ideas in an elementary context, we now present results of an experiment using DeSTIN, subtree mining and PLN together to recognize patterns among a handful of black and white images comprising simple letter-forms. This is a "toy" example, but exemplifies the key processes reviewed above. During the next year we will be working on deploying these same processes in the context of robot vision.

29.6.1 Mining Subtrees from DeSTIN States Induced via Observing Letterforms

Figure 29.2 shows the 7 input images utilized; Figure 29.3 shows the centroids found on each of the layers of DeSTIN (note that translation invariance was enabled for these experiments); and Figure 29.4 shows the most frequent subtrees recognized among the DeSTIN states induced by observing the 7 input images.

EFTA00624331

The centroid images shown in Figure 29.3 were generated as follows. For the bottom layer, centroids were directly represented as 4x4 grayscale images (ignoring the previous and parent belief sections of the centroid). For higher-level centroids, we proceeded as follows:

- Divide the centroid into 4 sub-arrays. An image is generated for each sub-array by treating the elements of the sub-array as weights in a weighted sum of the child centroid images. This weighted sum is used superpose / blend the child images into 1 image.
- Then these 4 sub-array images are combined in a square to create the whole centroid image.
- Repeat the process recursively, till one reaches the top level.

The weighted averaging used a p-power approach, i.e. replacing each weight w_i with $w_i^p/(w_1^p + ... + w_n^p)$ for a given exponent $p > 0$. The parameter p toggles how much attention is paid to nearby versus distant centroids. In generating Figure 29.3 we used $p = 4$.

The relation between subtrees and input images, in this example, was directly given via the subtree miner as:

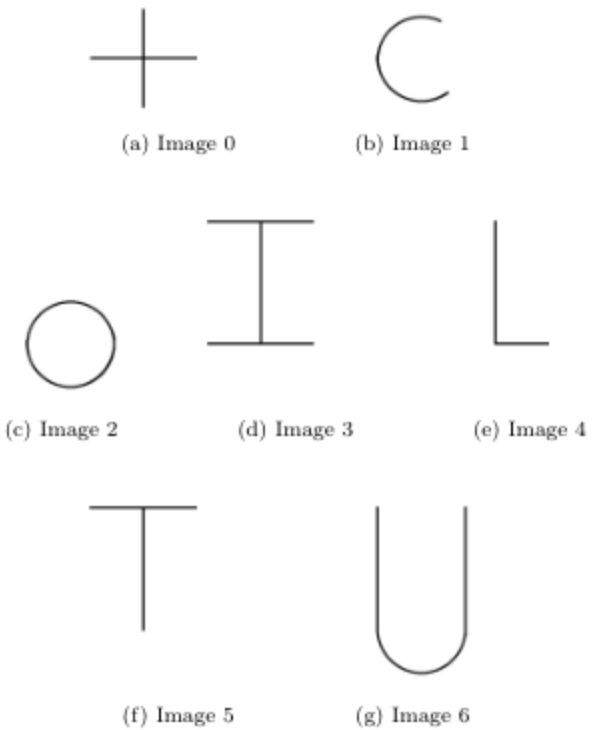
```
tree #0 matches input image: 4 6
tree #1 matches input image: 1 2
tree #2 matches input image: 3 5
tree #3 matches input image: 0 1 2 4
tree #4 matches input image: 3 5
tree #5 matches input image: 4 5
tree #6 matches input image: 1 2
tree #7 matches input image: 0 3 4
```

29.6.2 Mining Subtrees from DeSTIN States Induced via Observing Letterforms

The subtree-image relationships listed above may be most directly expressed in PLN syntax/semantics via

```
Evaluation contained_in (Tree0 Image4)
Evaluation contained_in (Tree0 Image6)
Evaluation contained_in (Tree1 Image1)
Evaluation contained_in (Tree1 Image2)
Evaluation contained_in (Tree2 Image3)
Evaluation contained_in (Tree2 Image5)
Evaluation contained_in (Tree3 Image0)
Evaluation contained_in (Tree3 Image1)
Evaluation contained_in (Tree3 Image2)
Evaluation contained_in (Tree3 Image4)
Evaluation contained_in (Tree4 Image3)
Evaluation contained_in (Tree4 Image5)
Evaluation contained_in (Tree5 Image4)
Evaluation contained_in (Tree5 Image5)
Evaluation contained_in (Tree6 Image1)
Evaluation contained_in (Tree6 Image2)
Evaluation contained_in (Tree7 Image0)
Evaluation contained_in (Tree7 Image3)
Evaluation contained_in (Tree7 Image4)
```

EFTA00624332



(a) Image 0 (b) Image 1

(c) Image 2 (d) Image 3 (e) Image 4

(f) Image 5 (g) Image 6

Fig. 29.2: Simple input images fed to DeSTIN for the experiment reported here.

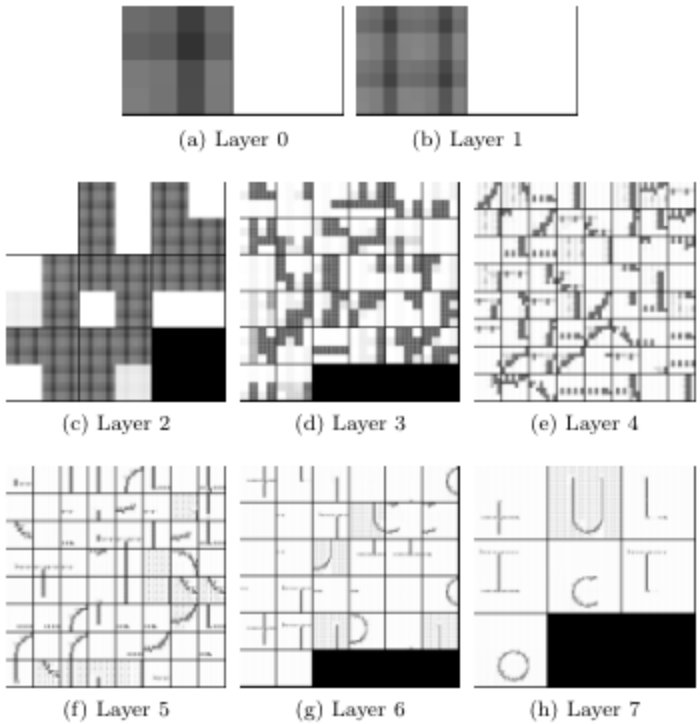
But while this is a perfectly natural way to import such relationships into OpenCog, it is not necessarily the most convenient form for PLN to use to manipulate them. For some useful inference chains, it is most convenient for PLN to translate these into the more concise form

```
Inheritance Image4 hasTree0
Inheritance Image6 hasTree0
...
Inheritance Image3 hasTree7
Inheritance Image4 hasTree7
```

PLN performs the translation from Evaluation into Inheritance form via the inference steps

```
Evaluation contains (Tree0 Image4)
==> \\ definition of SatisfyingSet
Member Image4 (SatisfyingSet (Evaluation contains(Tree0 *)) )
== \\ definition of hasTree0
Member Image4 hasTree0
==> \\ M2I, Member to Inheritance inference
Inheritance Image4 hasTree0
```

EFTA00624333



(a) Layer 0 (b) Layer 1

(c) Layer 2 (d) Layer 3 (e) Layer 4

(f) Layer 5 (g) Layer 6 (h) Layer 7

Fig. 29.3: Example visualization of the centroids on the 7 layers of the DeSTIN network. Each picture shows multiple centroids at the corresponding level. Higher level centroids are visualized as p'th power averages of lower level centroids, with p=4.

Finally, given the Inheritance relations listed above, PLN can draw some simple conclusions fairly directly, such as:

```
Similarity Image1 Image2 <1, .375>
Similarity Image3 Image5 <.5, .444>
```

The PLN truth values above are given in the form " <strength, confidence> ", where strength is in this case effectively a probability, and confidence represents a scaling into the interval $[0, 1]$ of the amount of evidence on which that strength value is based. The confidence is calculated using a "personality parameter" of $k = 5$ (k may vary between 1 and ∞, with higher numbers indicating less value attached to each individual piece of evidence. For example the truth value strength of 1 attached to "Similarity Image1 Image2" indicates that according to the evidence provided by these subtrees (and ignoring all other evidence), Image1 and Image2 are the same. Of course they are not the same – one is a C and another is an O – and once more evidence is given to PLN, it will decrease the strength value of this SimilarityLink. The confidence value of .375 indicates that PLN is not very certain of the sameness of these two letters.

What conclusion can we draw from this toy example, practically speaking? The conclusions drawn by the PLN system are not useful in this case – PLN thinks C and O are the same, as a provisional hypothesis based on this data. But this is not because DeSTIN and PLN are stupid. Rather, it's because they have not been fed enough data. The hypothesis that a C is an

EFTA00624334

occluded O is actually reasonably intelligent. If we fed these same systems many more pictures, then the subtree miner would recognize many more frequent subtrees in the larger corpus of DeSTIN states, and PLN would have a lot more information to go on, and would draw more conmmonsensically clever conclusions. We will explore this in our future work.

We present this toy example not as a useful practical achievement, but rather as a very simple illustration of the process via which subsymbolic knowledge (as in the states of the DeSTIN deep learning architecture) can be mapped into abstract logical knowledge, which can then be reasoned on via a probabilistic logical reasoning engine (such as PLN). We believe that the same process illustrated so simplistically in this example, will also generalize to more realistic and interesting examples, involving more complex images and inferences. The integration of DeSTIN and OpenCog described here is being pursued in the context of a project aimed at the creation of a humanoid robot capable of perceiving interpreting and acting in its environment with a high level of general intelligence.

29.7 Conclusion

We have described, at a high level, a novel approach to bridging the symbolic / subsymbolic gap, via very tightly integrating DeSTIN with OpenCog. We don't claim that this is the only way to bridge the gap, but we do believe it is a viable way. Given the existing DeSTIN and OpenCog designs and codebases, the execution of the ideas outlined here seems to be relatively straightforward, falling closer to the category of "advanced development" than that of blue-sky research. However, fine-tuning all the details of the approach will surely require substantial effort.

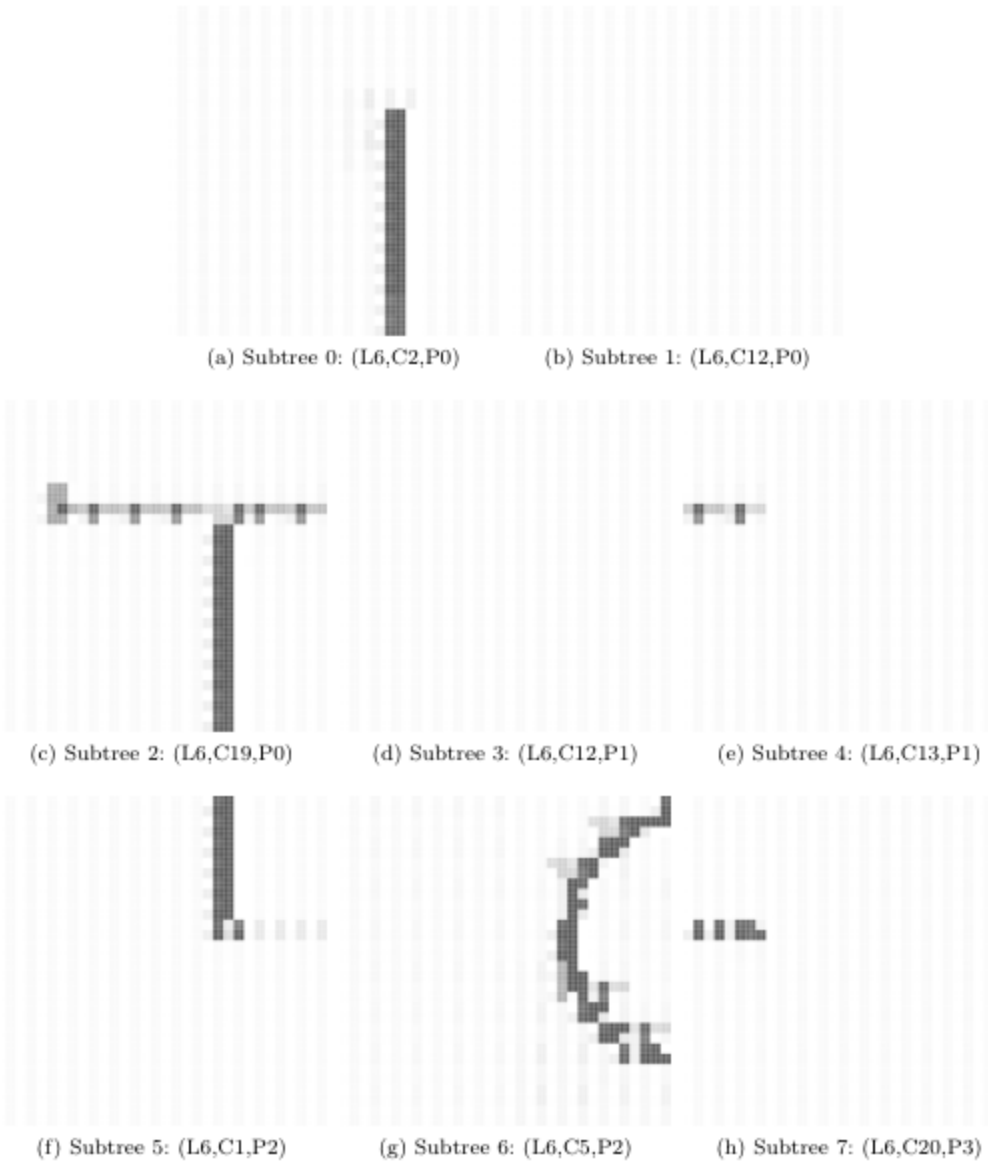
While we have focused on robotics applications here, the basic ideas described could be implemented and evaluated in a variety of other contexts as well, for example the identification of objects and events in videos, or intelligent video summarization. Our interests are broad, however, we feel that robotics is the best place to start – partly due to a general intuition regarding the deep coupling between human-like intelligence and human-like embodiment; and partly due to a more specific intuition regarding the value of action for perception, as reflected in Heinz von Foerster's dictum "if you want to see, learn how to act". We suspect there are important cognitive reasons why perception in the human brain centrally involves premotor regions. The coupling of a perceptual deep learning hierarchy and a symbolic AI system doesn't intrinsically solve the combinatorial explosion problem intrinsic in looking for potential conceptual patterns in masses of perceptual data. However, a system with particular goals and the desire to act in such a way as to achieve them, possesses a very natural heuristic for pruning the space of possible perceptual/conceptual patterns. It allows the mind to focus in on those percepts and concepts that are useful for action. Of course, there are other ways besides integrating action to enforce effective pruning, but the integration of perception and action has a variety of desirable properties that might be difficult to emulate via other methods, such as the natural alignment of the hierarchical structures of action and reward with that of perception.

The outcome of any complex research project is difficult to foresee in detail. However, our intuition – based on our experience with OpenCog and DeSTIN, and our work with the mathematical and conceptual theories underlying these two systems – is that the hybridization of OpenCog and DeSTIN as described here will constitute a major step along the path to human-level AGI. It will enable the creation of an OpenCog instance endowed with the capability of

EFTA00624335

flexibly interacting with a rich stream of data from the everyday human world. This data will not only help OpenCog to guide a robot in carrying out everyday tasks, but will also provide raw material for OpenCogPrime's cognitive processes to generalize from in various ways – e.g. to use as the basis for the formation of new concepts or analogical inferences.

EFTA00624336



(a) Subtree 0: (L6,C2,P0) (b) Subtree 1: (L6,C12,P0)

(c) Subtree 2: (L6,C19,P0) (d) Subtree 3: (L6,C12,P1) (e) Subtree 4: (L6,C13,P1)

(f) Subtree 5: (L6,C1,P2) (g) Subtree 6: (L6,C5,P2) (h) Subtree 7: (L6,C20,P3)

Fig. 29.4: Example subtrees extracted from the set of DeSTIN states corresponding to the input images given above. Each subtree is associated with a triple (level, centroid, position). The position is one of the four level n squares making up a level $n-1$ centroid. In this simple exmaple, all these frequent subtrees happen to be from Level 6, but this is not generally the case for more complex images. some of the centroids look like whitespace, but this is because a common region of whitespace was recognized among multiple input images.

EFTA00624337

Section IV
Procedure Learning

Chapter 30
Procedure Learning as Program Learning

30.1 Introduction

Broadly speaking, the learning of predicates and schemata (executable procedures) is done in CogPrime via a number of different methods, including for example PLN inference and concept predicatization (to be discussed in later chapters). Most of these methods, however, merely extrapolate procedures directly from other procedures or concepts in the AtomSpace, in a *local* way – a new procedure is derived from a small number of other procedures or concepts. General intelligence also requires a method for deriving new procedures that are more "fundamentally new." This is where CogPrime makes recourse to explicit procedure learning algorithms such as hillclimbing and MOSES, discussed in Chapters 32 and 33 below.

In this brief chapter we formulate the procedure learning problem as a program learning problem in a general way, and make some high-level observations about it. Conceptually, this chapter is a follow-up to Chapter 21, which discussed the choice to represent procedures as programs; here we make some simple observations regarding the implications of this choice for procedure learning, and the formal representation of procedure learning with CogPrime.

30.1.1 Program Learning

An optimization problem may be defined as follows: a solution space $\boldsymbol{S}$ is specified, together with some fitness function on solutions, where "solving the problem" corresponds to discovering a solution in $\boldsymbol{S}$ with a sufficiently high fitness.

In this context, we may define **program learning** as follows: given a program space $\boldsymbol{P}$, a behavior space $\boldsymbol{B}$, an execution function $\boldsymbol{exec : P \mapsto B}$, and a fitness function on *behaviors*, "solving the problem" corresponds to discovering a program $\boldsymbol{p}$ in $\boldsymbol{P}$ whose corresponding behavior, $\boldsymbol{exec(p)}$, has a sufficiently high fitness.

In evolutionary learning terms, the program space is the space of **genotypes**, and the behavior space is the space of **phenotypes**.

This formalism of procedure learning serves well for explicit procedure learning CogPrime, not counting cases like procedure learning within other systems (like DeSTIN) that may be hybridized with CogPrime.

EFTA00624340

Of course, this extended formalism can of be entirely vacuous – the behavior space could be identical to the program space, and the execution function simply identity, allowing any optimization problem to be cast as a problem of program learning. The utility of this specification arises when we make interesting assumptions regarding the program and behavior spaces, and the execution and fitness functions (thus incorporating additional inductive bias):

1. **Open-endedness** – $\boldsymbol{P}$ has a natural "program size" measure – programs may be enumerated from smallest to largest, and there is no obvious problem-independent upper bound on program size.
2. **Over-representation** – $\boldsymbol{exec}$ often maps many programs to the same behavior.
3. **Compositional hierarchy** – programs themselves have an intrinsic hierarchical organization, and may contain subprograms which are themselves members of $\boldsymbol{P}$ or some related program space. This provides a natural family of distance measures on programs, in terms of the the number and type of compositions / decompositions needed to transform one program into another (i.e., edit distance).
4. **Chaotic Execution** – very similar programs (as conceptualized in the previous item) may have very different behaviors.

Precise mathematical definitions could be given for all of these properties but would provide little insight – it is more instructive to simply note their ubiquity in symbolic representations; human programming languages (LISP, C, etc.), Boolean and real-valued formulae, pattern-matching systems, automata, and many more. The crux of this line of thought is that the combination of these four factors conspires to *scramble* fitness functions – even if the mapping from behaviors to fitness is separable or nearly decomposable, the complex* program space and chaotic execution function will often quickly lead to intractability as problem size grows. These properties are not superficial inconveniences that can be circumvented by some particularly clever encoding. On the contrary, they are the essential characteristics that give programs the power to compress knowledge and generalize correctly, in contrast to flat, inert representations such as lookup tables (see Baum [Bau04] for a full treatment of this line of argument).

The consequences of this particular kind of complexity, together with the fact that most program spaces of interest are combinatorially very large, might lead one to believe that competent program learning is impossible. Not so: real-world program learning tasks of interest have a compact structure† – they are not "needle in haystack" problems or uncorrelated fitness landscapes, although they can certainly be encoded as such. The most one can definitively state is that algorithm ***foo***, methodology ***bar,*** or representation ***baz*** is unsuitable for expressing and exploiting the regularities that occur across interesting program spaces. Some of these regularities are as follows:

1. **Simplicity prior** – our prior assigns greater probability mass to smaller programs.
2. **Simplicity preference** – given two programs mapping to the same behavior, we prefer the smaller program (this can be seen as a secondary fitness function).
3. **Behavioral decomposability** – the mapping between behaviors and fitness is separable or nearly decomposable. Relatedly, fitness are more than scalars – there is a partial ordering corresponding to behavioral dominance, where one behavior dominates another if it exhibits

* Here "complex" means open-ended, over-representing, and hierarchical.

† Otherwise, humans could not write programs significantly more compact than lookup tables.

EFTA00624341

a strict superset of the latter's desideratum, according to the fitness function.[‡] This partial order will never contradict the total ordering of scalar fitnesss.

4. **White box execution** – the mechanism of program execution is known *a priori*, and remains constant across many problems.

How these regularities may be exploited will be discussed in later sections and chapters. Another fundamental regularity of great interest for artificial general intelligence is patterns across related problems that may be solvable with similar programs (e.g., involving common modules).

30.2 Representation-Building

One important issue in achieving competent program learning is *representation building*. In an ideally encoded optimization problem, all prespecified variables would exhibit complete *separability*, and could be optimized independently. Problems with hierarchical dependency structure cannot be encoded this way, but are still tractable by dynamically learning the problem decomposition (as is done by the BOA and hBOA, described in Chapter 33). For complex problems with interacting subcomponents, finding an accurate problem decomposition is often tantamount to finding a solution. In an idealized run of a competent optimization algorithm, the problem decomposition evolves along with the set of solutions being considered, with parallel convergence to the correct decomposition and the global solution optima. However, this is certainly contingent on the existence of some compact[§] and reasonably correct decomposition in the space (of decompositions, not solutions) being searched.

Difficulty arises when no such decomposition exists, or when a more effective decomposition exists that cannot be formulated as a probabilistic model over representational parameters. Accordingly, one may extend current approaches via either: a more general modeling language for expressing problem decompositions; or *additional mechanisms* that modify the representations on which modeling operates (introducing additional inductive bias). In CogPrime we have focused on the latter – the former would appear to require qualitatively more computational capacity than will be available in the near future. If one ignores this constraint, such a "universal" approach to general problem-solving is indeed possible, e.g. $AIXI^{tl}$ as discussed in Chapter 7.3.

We refer to these additional mechanisms as "representation-building" because they serve the same purpose as the pre-representational mechanisms employed (typically by humans) in setting up an optimization problem – to present an optimization algorithm with the salient parameters needed to build effective problem decompositions and vary solutions along meaningful dimensions. We return to this issue in detail in Chapter 33 in the context of MOSES, the most powerful procedure-learning algorithm provided in CogPrime.

[‡] For example, in supervised classification one rule dominates another if it correctly classifies all of the items that second rule classifies correctly, as well as some which the second rule gets wrong.

[§] The decomposition must be compact because in practice only a fairly small sampling of solutions may be evaluated (relative to the size of the total space) at a time, and the search mechanism for exploring decomposition-space is greedy and local. This is in also accordance with the general notion of learning corresponding to compression.

EFTA00624342

30.3 Specification Based Procedure Learning

Now we explain how procedure learning fits in with the declarative and intentional knowledge representation in the Atomspace.

The basic method that CogPrime uses to learn procedures that appear fundamentally new from the point of view of the AtomSpace at a given point in time is "specification-based procedure learning". This involves taking a PredicateNode with a ProcedureNode input type as a specification, and searching for ProcedureNodes that fulfill this specification (in the sense of making the specification PredicateNode as true as possible). In evolutionary computing lingo, the specification predicate is a *fitness function*.

Searching for PredicateNodes that embody patterns in the AtomSpace as a whole is a special case of this kind of learning, where the specification PredicateNode embodies a notion of what constitutes an "interesting pattern". The quantification of interestingness is of course an interesting and nontrivial topic in itself.

Finding schemata that are likely to achieve goals important to the system is also a special case of this kind of learning. In this case, the specification predicate is of the form:

```
F(S) = PredictiveImplicationLink (ExOutLink S) G
```

This measures the extent to which executing schema S is positively correlated with goal-predicate G being achieved shortly later.

Given a PredicateNode interpretable as a specification, how do we find a ProcedureNode satisfying the specification? Lacking prior knowledge sufficient to enable an incremental approach like inference, we must search the space of possible ProcedureNodes, using an appropriate search heuristic, hopefully one that makes use of the system's existing knowledge as fully as possible.

EFTA00624343

Chapter 31
Learning Procedures via Imitation, Reinforcement and Correction

Co-authored with Moshe Looks, Samir Araujo and Welter Silva

31.1 Introduction

In procedure learning as elsewhere in cognition, it's not enough to use the right algorithm, one has to use it in the right way based on the data and context and affordances available. While Chapters **??** and 33 focus on procedure learning algorithms, this one focuses on procedure learning *methodology*. We will delve into the important special case of procedure learning in which the fitness function involves reinforcement and imitation supplied by a teacher and/or an environment, and look at examples of this in the context of teaching behaviors to virtual pets controlled by OpenCogPrime. While this may seem a very narrow context, many of the lessons learned are applicable more broadly; and the discussion has the advantage of being grounded in actual experiments done with OpenCogPrime's predecessor system, the Novamente Cognition Engine, and with an early OpenCog version as well, during the period 2007-2008.

We will focus mainly on learning from a teacher, and then common on the very similar case where the environment, rather than some specific agent, is the teacher.

31.2 IRC Learning

Suppose one intelligent agent (the "teacher") has knowledge of how to carry out a certain behavior, and wants to transfer this knowledge to another intelligent agent (the "student"). But, suppose the student agent lacks the power of language (which might be, for example, because language is the thing being taught!). How may the knowledge be transferred? At least three methodologies are possible:

1. **Imitative learning**: The teacher acts out the behavior, showing the student by example
2. **Reinforcement learning**: The student tries to do the behavior himself, and the teacher gives him feedback on how well he did
3. **Corrective learning**: As the student attempts the behavior, the teacher actively corrects (i.e. changes) the student's actions, guiding him toward correct performance

Obviously, these three forms of instruction are not exclusive. What we describe here, and call IRC learning, is a pragmatic methodology for instructing AGI systems that combines these

EFTA00624344

three forms of instruction. We believe this combination is a potent one, and is certainly implicit in the way human beings typically teach young children and animals.

For sake of concreteness, we present IRC learning here primarily in the context of virtually embodied AGI systems – i.e., AGI systems that control virtual agents living in virtual worlds. There is an obvious extension to physical robots living in the real world and capable of flexible interaction with humans. In principle, IRC learning is applicable more broadly as well, and could be explored in various non-embodied context such as (for instance) automated theorem-proving. In general, the term "IRC learning" may be used to describe any teacher/student interaction that involves a combination of reinforcement, imitation and correction. While we have focused in our practical work so far on the use of IRC to teach simple "animal-like" behaviors, the application that interests us more in the medium term is language instruction, to which we will return in later chapters.

Harking back to Chapter 9, it is clear that an orientation toward effective IRC learning will be valuable for any system attempting to achieve complex goals in an environment heavily populated by other intelligences possessing significant goal-relevant knowledge. Everyday human environments possess this characteristic, and we suggest the best way to create human-level AGIs will be to allow them to develop in environments possessing this characteristic as well.

31.2.1 A Simple Example of Imitation/Reinforcement Learning

Perhaps the best way to introduce the essential nature of the IRC teaching protocol is to give a brief snippet from a script that was created to guide the actual training of the virtual animals controlled by the PetBrain. This snippet involves only I and R; the C will be discussed afterwards.

This snippet demonstrates a teaching methodology that involves two avatars: Bob who is being the teacher, and Jill who is being an "imitation animal," showing the animal what to do by example.

1. *Bob wants to teach the dog Fido a trick. He calls his friend Jill over. "Jill, can you help me teach Fido a trick?"*
2. *Jill comes over. "How much will you pay me for it?*
3. *Bob gives her a kiss.*
4. *"All right," says Jill, "what do you want to teach him?"*
5. *"Let's start with fetching stuff," replies Bob.*
6. *So Bob and Jill start teaching Fido to fetch using the Pet language....*
7. *Bob says: "Fido, I'm going to teach you to play fetch with Jill."*
8. *Fido sits attentively, looking at Bob.*
9. *Bob says: "OK, I'm playing fetch now."*
10. *Bob picks up a stick from the ground and throws it. Jill runs to get the stick and brings it back to Bob.*
11. *Bob says: "I'm done fetching.*
12. *Bob says, "You try it."*
13. *Bob throws a stick. Fido runs to the stick, gets it, and brings it back.*
14. *Bob says "Good dog!"*
15. *Fido looks happy.*
16. *Bob says: "Ok, we're done with that game of fetch.*

EFTA00624345

17. *Bob says, "Now, let's try playing fetch again."*
18. *This time, Bob throws a stick in a different direction, where there's already a stick lying on the ground (call the other stick Stick 2).*
19. *Fido runs and retrieves Stick 2. As soon as he picks it up, Bob says "No." But Fido keeps on running and brings the stick back to Bob.*
20. *Bob says "No, that was wrong. That was the wrong stick. Stop trying!"*
21. *Jill says, "Furry little moron!"*
22. *Bob says to Jill, "Have some patience, will you? Let's try again."*
23. *Fido is slowly wandering around, sniffing the ground.*
24. *Bob says "Fido, stay." Fido returns near Bob and sits.*
25. *Bob throws Stick 2. Fido starts to get up and Bob repeats "Fido, stay."*
26. *Bob goes and picks up Stick 1, and walks back to his original position.*
27. *Bob says "Fido, I'm playing fetch with Jill again."*
28. *Bob throws the first stick in the direction of stick 2.*
29. *Jill goes and gets stick 1 and brings it back to Bob.*
30. *Bob says "I'm done playing fetch with Jill."*
31. *Bob says "Try playing fetch with me now." He throws stick 1 in another direction, where stick 3 and stick 4 are lying on the ground, along with some other junk.*
32. *Fido runs and gets stick 1 and brings it back.*
33. *Bob and Jill both jump up and down smiling and say "Good dog! Good dog, Fido!! Good dog!!"*
34. *Fido smiles and jumps up and licks Jill on the face.*
35. *Bob says, "Fido, we're done practicing fetch."*

In the above transcript, Line 7 initiates a formal training session, and Line 33 terminates this session. The training session is broken into "exemplar" intervals during which exemplars are being given, and "trial" intervals during which the animal is trying to imitate the exemplars, following which is receives reinforcement on its success or otherwise. For instance line 9 initiates the presentation of an exemplar interval, and line 11 indicates the termination of this interval. Line 12 indicates the beginning of a trial interval, and line 16 indicates the termination of this interval.

The above example of combined imitative/reinforcement learning involves two teachers, but, this is of course not the only way things can be done. Jill could be eliminated from the above teaching example. The result of this would be that, in figuring out how to imitate the exemplars, Fido would have to figure out which of Bob's actions were "teacher" actions and which were "simulated student" actions. This is not a particularly hard problem, but it's harder than the case where Jill carries out all the simulated-student actions. So in the case of teaching fetch with only one teacher avatar, on average, more reinforcement trials will be required.

31.2.2 A Simple Example of Corrective Learning

Another interesting twist on the imitative/reinforcement teaching methodology described above is the use of explicit correctional instructions from the teacher to the animal. This is not shown in the above example but represents an important addition to the methodology show there. One good example of the use of corrections would be the problem of teaching would be teaching an animal to sit and wait until the teacher says "Get Up," using only a single teacher. Obviously,

EFTA00624346

using two teachers, this is a much easier problem. Using only one teacher, it's still easy, but involves a little more subtlety, and becomes much more tractable when corrections are allowed.

One way that human dog owners teach their dogs this sort of behavior is as follows:

1. Tell the dog "sit"
2. tell the dog "stay"
3. Whenever the dog tries to get up, tell him "no" or "sit", and then he sits down again
4. eventually, tell the dog to "get up"

The real dog understands, in its own way, that the "no" and "sit" commands said after the "stay" command are meta-commands rather than part of the "stay" behavior.

In our virtual-pet case, this would be more like

1. tell the dog "I'm teaching you to stay"
2. Tell the dog "sit"
3. Whenever the dog tries to get up, tell him "no" or "sit", and then he sits down again
4. eventually, tell the dog to "get up"
5. tell the dog "I'm done teaching you to stay"

One easy way to do this, which deviates from the pattern of humanlike interaction, would be to give the agent knowledge about how to interpret an explicit META flag in communications directed toward it. In this case, the teaching would look like

1. tell the dog "I'm teaching you to stay"
2. Tell the dog "META: sit"
3. Whenever the dog tries to get up, tell him "META: no" or "META:sit", and then he sits down again
4. eventually, tell the dog to "get up"
5. tell the dog "I'm done teaching you to stay"

Even without the META tag, this behavior (and other comparable ones) is learnable via CogPrime's learning algorithms within a modest number of reinforcement trials. So we have not actually implemented the META approach. But it well illustrates the give-and-take relationship between the sophistication of the teaching methodology and the number of reinforcement trials required. In many cases, the best way to reduce the number of reinforcement trials required to learn a behavior is not to increase the sophistication of the learning algorithm, but rather to increase the information provided during the instruction process. No matter how advanced the learning algorithm, if the teaching methodology only gives a small amount of information, it's going to take a bunch of reinforcement trials to go through the search space and find one of the right procedures satisfying the teacher's desires. One of the differences between the real-world learning that an animal or human child (or adult) experiences, and the learning "experienced" by standard machine-learning algorithms, is the richness and diversity of information that the real world teaching environment provides, beyond simple reinforcement signals. Virtual worlds provide a natural venue in which to experiment with providing this sort of richer feedback to AI learning systems, which is one among the many reasons why we feel that virtual worlds are an excellent venue for experimentation with and education of early-stage AGI systems.

EFTA00624347

31.3 IRC Learning in the PetBrain

Continuing the theme of the previous section, we now discuss "trick learning" in the PetBrain, as tested using OpenCog and the Multiverse virtual world during 2007-2008. The PetBrain constitutes a specific cognitive infrastructure implementing the IRC learning methodology in the virtual-animal context, with some extensibility beyond this context as well.

In the PetBrain, learning itself is carried out by a variety of hillclimbing as described in Chapter 32, which is a fast learning algorithm but may fail on harder behaviors (in the sense of requiring an unacceptably large number of reinforcement trials). For more complex behaviors, MOSES (Chapter 33) would need to be integrated as an alternative. Compared to hillclimbing, MOSES is much smarter but slower, and may take a few minutes to solve a problem. The two algorithms (as implemented for the PetBrain) share the same Combo knowledge representation and some other software components (e.g. normalization rules for placing procedures in an appropriate hierarchical normal form, as described in Chapter 21).

The big challenge involved in designing the PetBrain system, AI-wise, was that these learning algorithms, used in a straightforward way with feedback from a human-controlled avatar as the fitness function, would have needed an excessive number of reinforcement trials to learn relatively simple behaviors. This would bore the human beings involved with teaching the animals. This is not a flaw of the particular learning algorithms being proposed, but is a generic problem that would exist with any AI algorithms. To choose an appropriate behavior out of the space of all possible behaviors satisfying reasonable constraints, requires more bits of information that is contained in a handful of reinforcement trials.

Most "animal training" games (e.g. Nintendogs may be considered as a reference case) work around this "hard problem" by not allowing teaching of novel behaviors. Instead, a behavior list is made up front by the game designers. The animals have preprogrammed procedures for carrying out the behaviors on the list. As training proceeds they make fewer errors, till after enough training they converge "miraculously" on the pre-programmed plan

This approach only works, however, if all the behaviors the animals will ever learn have been planned and scripted in advance.

The first key to making learning of non-pre-programmed behaviors work, without an excessive number of reinforcement trials, is in "fitness estimation" – code that guesses the fitness of a candidate procedure at fulfilling the teacher's definition of a certain behavior, without actually having to try out the procedure and see how it works. This is where the I part of IRC learning comes in.

At an early stage in designing the PetBrain application, we realized it would be best if the animals were instructed via a methodology where the same behaviors are defined by the teacher both by demonstration *and* by reinforcement signals. Learning based on reinforcement signals only can also be handled, but learning will be much slower.

In evolutionary programming lingo, we have

1. Procedures = genotypes
2. Demonstrated exemplars, and behaviors generated via procedures = phenotypes
3. Reinforcement signals from pet owner = fitness

EFTA00624348

One method of imitation-based fitness estimation used in the PetBrain involves an internal simulation world which we'll call CogSim, as discussed in Chapter 40 *. CogSim can be visualized using a simple testing UI, but in the normal course of operations it doesn't require a user interface; it is an internal simulation world, which allows the PetBrain to experiment and see what a certain procedure would be likely to do if enacted in the SL virtual world. Of course, the accuracy of this kind of simulation depends on the nature of the procedure. For procedures that solely involve moving around and interacting with inanimate objects, it can be very effective. For procedures involving interaction with human-controlled avatars, other animals, or other complex objects, it may be unreliable – and making it even moderately reliable would require significant work that has not yet been done, in terms of endowing CogSim with realistic simulations of other agents and their internal motivational structures and so forth. But short of this, CogSim has nonetheless proved useful for estimating the fitness of simple behavioral procedures.

When a procedure is enacted in CogSim, this produces an object called a "behavior description" (BD), which is represented in the AtomSpace knowledge representation format. The BD generated by the procedure is then compared with the BD's corresponding to the "exemplar" behaviors that the teacher has generated, and that the student is trying to emulate. Similarities are calculated, which is a fairly subtle matter that involves some heuristic inferences. An estimate of the likelihood that the procedure, if executed in the world, will generate a behavior adequately similar to the exemplar behaviors.

Furthermore, this process of estimation may be extended to make use of the animal's long-term episodic memory. Suppose a procedure P is being evaluated in the context of exemplar-set E. Then

1. The episodic memory is mined for pairs (P', E') that are similar to (P,E)
2. The fitness of these pairs (P', E') is gathered from the experience base
3. An estimate of the fitness of (P,E) is then formed

Of course, if a behavior description corresponding to P has been generated via CogSim, this may also be used in the similarity matching against long-term memory. The tricky part here, of course, is the similarity measurement itself, which can be handled via simple heuristics, but if taken sufficiently seriously becomes a complex problem of uncertain inference.

One thing to note here is that in the PetBrain context, although learning is done by each animal individually, this learning is subtly guided by collective knowledge within the fitness estimation process. Internally, we have a "borg mind" with multiple animal bodies, and an architecture designed to ensure the maintenance of unique personalities on the part of the individual animals in spite of the collective knowledge and learning underneath.

At time of writing, we have just begun to experiment with the learning system as described above, and are using it to learn simple behaviors such as playing fetch, basic soccer skills, doing specific dances as demonstrated by the teacher, and so forth. We have not yet done enough experimentation to get a solid feel for the limitations of the methodology as currently implemented.

Note also the possibility of using CogPrime's PLN inference component to allow generalization of learned behaviors. For instance, with inference deployed appropriately, a pet that had learned how to play tag would afterwards have a relatively easy time learning to play "freeze tag." A pet that had learned how to hunt for Easter eggs would have a relatively easy time

* The few readers familiar with obscure OpenCog documentation may remember that CogSim was previously called "Third Life", in reference to the Second Life virtual world that was being used to embody the OpenCog virtual pets at the time

EFTA00624349

learning to play hide-and-seek. Episodic memory can be very useful for fitness estimation here, but explicit use of inference may allow much more rapid and far-reaching inference capabilities.

31.3.1 Introducing Corrective Learning

Next, how may corrections be utilized in the learning process we have described? Obviously, the corrected behavior description gets added into the knowledge base as an additional exemplar. And, the fact of the correction acts as a partial reinforcement (up until the time of the correction, what the animal was doing was correct). But beyond this, what's necessary is to propagate the correction backward from the BD level to the procedure level. For instance, if the animal is supposed to be staying in one place, and it starts to get up but is corrected by the teacher (who says "sit" or physically pushes the animal back down), then the part of the behavior-generating procedure that directly generated the "sit" command needs to be "punished." How difficult this is to do, depends on how complex the procedure is. It may be as simple as providing a negative reinforcement to a specific "program tree node" within the procedure, thus disincentivizing future procedures generated by the procedure learning algorithm from containing this node. Or it may be more complex, requiring the solution of an inference problem of the form "Find a procedure P" that is as similar as possible to procedure P, but that does not generate the corrected behavior, but rather generates the behavior that the teacher wanted instead." This sort of "working backwards from the behavior description to the procedure" is never going to be perfect except in extremely simple cases, but it is an important part of learning. We have not yet experimented with this extensively in our virtual animals, but plan to do so as the project proceeds.

There is also an interesting variant of correction in which the agent's own memory serves implicitly as the teacher. That is, if a procedure generates a behavior that seems wrong based on the history of successful behavior descriptions for similar exemplars, then the system may suppress that particular behavior or replace it with another one that seems more appropriate – inference based on history thus serving the role of a correcting teacher.

31.4 Applying A Similar IRC Methodology to Spontaneous Learning

We have described the IRC teaching/learning methodology in the context of learning from a teacher – but in fact a similar approach can be utilized for purely unsupervised learning. In that case, the animal's intrinsic goal system acts implicitly as a teacher.

For instance, suppose the animal wants to learn how to better get itself fed. In this case,

1. Exemplars are provided by instances in the animal's history when it has successfully gotten itself fed
2. Reinforcement is provided by, when it is executing a certain procedure, whether or not it actually gets itself fed or not
3. Correction as such doesn't apply, but implicit correction may be used via deploying history-based inference. If a procedure generates a behavior that seems wrong based on the history of successful behavior descriptions for the goal of getting fed, then the system may suppress that particular behavior.

EFTA00624350

The only real added complexity here lies in identifying the exemplars. In surveying its own history, the animal must look at each previous instance in which it got fed (or same sample thereof), and for each one recollect the series of N actions that it carried out prior to getting fed. It then must figure out how to set N – i.e. which of the actions prior to getting fed were part of the behavior that led up to getting fed, and which were just other things the animal happened to be doing a while before getting fed. To the extent that this exemplar mining problem can be solved adequately, innate-goal-directed spontaneous learning becomes closely analogous to teacher-driven learning as we've described it. Or in other words: Experience, as is well known, can serve as a very effective teacher.

EFTA00624351

Chapter 32
Procedure Learning via Adaptively Biased Hillclimbing

Primary author: Nil Geisweiller

32.1 Introduction

Having chosen to represent procedures as programs, explicit procedure learning then becomes a matter of automated program learning. In its most general incarnation, automated program learning is obviously an intractable problem; so the procedure learning design problem then boils down to finding procedure learning algorithms that are effective on the class of problems relevant to CogPrime systems in practice. This is a subtle matter because there is no straightforward way to map from the vaguely-defined category of real-world "everyday human world like" goals and environments to any formally-defined class of relevant objective functions for a program learning algorithm.

However, this difficulty is not a particular artifact of the choice of programs to represent procedures; similar issues would arise with any known representational mechanism of suitable power. For instance, if procedures were represented as recurrent neural nets, there would arise similar questions of how many layers to give the networks, how to determine the connection statistics, what sorts of neurons to use, which learning algorithm, etc. One can always push such problems to the meta level and use automated learning to determine which variety of learning algorithm to use – but then one has to make some decisions on the metalearning level, based on one's understanding of the specific structure of the space of relevant program learning algorithms. In the fictitious work of unbounded computational resources no such judicious choices are necessary, but that's not the world we live in, and it's not relevant to the design of human-like AGI systems.

At the moment, in CogPrime, we utilize two different procedure learning systems, which operate on the same knowledge representation and rely on much of the same internal code. One, which we roughly label "hill climbing", is used for problems that are sufficiently "easy" in the sense that it's possible for the system to solve them using feasible resources without (implicitly or explicitly) building any kind of sophisticated model of the space of solutions to the problem. The other, MOSES, is used for problems that are sufficiently difficult that the right way to solve them is to progressively build a model of the program space as one tries out various solutions, and then use this model to guide ongoing search for better and better solutions. Hillclimbing is treated in this chapter; MOSES in the next.

EFTA00624352

32.2 Hillclimbing

"Hillclimbing," broadly speaking, is not a specific algorithm but a category of algorithms. It applies in general to any search problem where there is a large space of possible solutions, which can be compared as to their solution quality. Here we are interested in applying it specifically to problems of search through spaces of *programs*.

In hillclimbing, one starts with a candidate solution to a problem (often a random one, which may be very low-quality), and iteratively makes small changes to the candidate to generate new possibilities, hoping one of them will be a better solution. If a new possibility is better than the current candidate, then the algorithm adopts the new possibility as its new candidate solution. When the current candidate solution can no longer be improved via small changes, the algorithm terminates. Ideally, at that point the current candidate solution is close to optimal – but this is not guaranteed!

Various tweaks to hillclimbing exist, including "restart" which means that one starts hillclimbing over and over again, taking the best solution from multiple trials; and "backtracking", which means that if the algorithm terminates at a solution that seems inadequate, then the search can "backtrack" to a previously considered candidate solution, and try to make different small changes to that candidate solution, trying previously unexplored possibilities in search of a new candidate. The value of these and other tweaks depends on the specific problem under consideration.

In the specific approach to hillclimbing described here, we use a hillclimber with backtracking, applied to programs that are represented in the same hierarchical normal form used with MOSES (based on the program normalization ideas presented in Chapter 21). The basic pseudocode for the hillclimber may be given as:

Let L be the list (initially empty) of programs explored so far in decreasing order with respect to their fitness. Let N_p be the neighbors of program p.

1. Take the best program $p \in L$
2. Evaluate all programs of N_p
3. Merge N_p in L
4. Move p from L to the set of best programs found so far and repeat from step 1 until time runs out

In the following sections of this chapter, we show how to speed up the hillclimbing search for learning procedures via four optimizations, which have been tested fairly extensively. For concreteness we will refer often to the specific case of using the hill climbing algorithm to control a virtual agent in a virtual world – and especially the case of teaching a virtual pet tricks via imitation learning (as in Chapter 31) but the ideas have more general importance. The four optimizations are:

- reduce candidates to normal form to minimize over-representation and increase the syntactic semantic correlation (Chapter 21),
- filter perceptions using an entropy criterion to avoid building candidates that involve nodes unlikely to be contained in the solution (Section 32.3),
- use sequences of agent actions, observed during the execution of the program, as building blocks (Section 32.4),
- choose and calibrate a simplicity measure to focus on simpler solutions (the "Occam bias") first (Section 32.5).

EFTA00624353

32.3 Entity and Perception Filters

The number of program candidates of a given size increases exponentially with the alphabet of the language; therefore it is important to narrow that alphabet as much as possible. This is the role of the two filters explained below, the Entity and the Entropy filter.

32.3.1 Entity filter

This filter is in charge of selecting the entities in the scene the pet should take into account during an imitation learning session. These entities can be any objects, avatars or other pets.

In general this is a very hard problem, for instance if a bird is flying near the owner while teaching a trick, should the pet ignore it? Perhaps the owner wants to teach the pet to bark at them; if so they should not be ignored.

In our current and prior work with OpenCog controlling virtual world agents, we have used some fairly crude heuristics for entity filtering, which must be hand-tuned depending on the properties of the virtual world. However, our intention is to replace these heuristics with entity filtering based on Economic Attention Networks (ECAN) as described in Chapter 23.

32.3.2 Entropy perception filter

The perception filter is in charge of selecting all perceptions in the scene that are reasonably likely to be part of the solution to the program learning problem posed. A "perception" in the virtual world context means the evaluation of one of a set of pre-specified perception predicates, with an argument consisting of one of the entities in the observed environment.

Given N entities (provided by the Entity filter), there are usually $O(N^2)$ potential perceptions in the Atomspace due to binary perceptions like

```
near(owner bird)
inside(toy box)
...
```

The perception filter proceeds by computing the entropy of any potential perceptions happening during a learning session. Indeed if the entropy of a given perception `P` is high that means that a conditional `if(P B1 B2)` has a rather balanced probability of taking Branch `B1` or `B2`. On the other hand if the entropy is low then the probability of taking these branches is unbalanced, for instance the probability of taking `B1` may be significantly higher than the probability of taking `B2`, and therefore `if(P B1 B2)` could reasonably be substituted by `B1`.

For example, assume that during the teaching sessions, the predicate `near(owner bird)` is false 99% percent of the time; then `near(owner bird)` will have a low entropy and will possibly be discarded by the filter (depending on the threshold). If the bird is always far from the owner then it will have entropy 0 and will surely be discarded, but if the bird comes and goes it will have a high entropy and will pass the filter. Let P be such a perception and P_t returns 1 when the perception is true at time t or 0 otherwise, where t ranges over the set of instants, of size N, recorded between the beginning and the end of the demonstrated trick. The calculation goes as follows

EFTA00624354

$$Entropy(P) = H\left(\frac{\sum_t P_t}{N}\right)$$

where $H(p) = -p\,log(p) - (1-p)log(1-p)$. There are additional subtleties when the perception involves random operators, like `near(owner random_object)` that is the entropy is calculated by taking into account a certain distribution over entities grouped under the term random_object. The calculation is optimized to ignore instants when the perception relates to object that have not moved which makes the calculation efficient enough, but there is room to improve it in various ways; for instance it could be made to choose perceptions based not only on entropy but also inferred relevancy with respect to the context using PLN.

32.4 Using Action Sequences as Building Blocks

A heuristic that has been shown to work, in the "virtual pet trick" context, is to consider sequences of actions that are compatible with the behavior demonstrated by the avatar showing the trick as building blocks when defining the neighborhood of a candidate. For instance if the trick is to fetch a ball, compatible sequences would be

`goto(ball), grab(ball), goto(owner), drop`
`goto(random_object), grab(nearest_object), goto(owner), drop`
...

Sub-sequences can be considered as well – though too many building blocks also increase the neighborhood exponentially, so one has to be careful when doing that. In practice using the set of whole compatible sequences worked well. This for instance can speed up many fold the learning of the trick triple_kick as shown in Section 32.6.

32.5 Automatically Parametrizing the Program Size Penalty

A common heuristic for program learning is an "Occam penalty" that penalizes large programs, hence biasing search toward compact programs. The function we use to penalize program size is inspired by Ray Solomonoff's theory of optimal inductive inference [Sol64a, Sol64b]; simply said, a program is penalized exponentially with respect to its size. Also, one may say that since the number of program candidates grows exponentially with their size, exploring solutions with higher size must be exponentially worth the cost.

In the next subsections we describe the particular penalty function we have used and how to tune its parameters.

32.5.1 Definition of the complexity penalty

Let p be a program candidate and $penalty(p)$ a function with domain [0,1] measuring the complexity of p. If we consider the complexity penalty function $penalty(p)$ as if it denotes the prior probability of p, and $score(p)$ (the quality of p as utilized within the hill climbing algorithm) as denoting the conditional probability of the desired behavior knowing p, then

EFTA00624355

Bayes rule* tells us that

$$fitness(p) = score(p) \times penalty(p)$$

denotes the conditional probability of p knowing the right behavior to imitate, the fitness function that we want to maximize.

It happens that in the pet trick learning context which is our main example in this chapter, $score(p)$ does not denote such a probability; instead it measures how similar the behavior generated by p and the behavior to imitate are. However, we utilize the above formula anyway, with a heuristic interpretation. One may construct assumptions under which $score(p)$ does represent a probability but this would take us too far afield.

The penalty function we use is then given by:

$$penalty(p) = exp(-a \times log(b \times |A| + e) \times |p|)$$

where $|p|$ is the program size, $|A|$ its alphabet size and $e = exp(1)$. The reason $|A|$ enters into the equation is because the alphabet size varies from one problem to another due to the perception and action filters. Without that constraint the term $log(b \times |A| + e)$ could simply be included in a. The higher a the more intense the penalty is. The parameter b controls how that intensity varies with the alphabet size.

It is important to remark the difference between such a penalty function and lexicographic parsimony pressure (literally said *everything being equal, choose the shortest program*). Because of the use of sequences as building blocks, without such a penalty function the algorithm may rapidly reach an optimal program (a mere long sequence of actions) and remain stuck in an apparent optimum while missing the very logic of the action sequence that the human wants to convey.

32.5.2 Parameterizing the complexity penalty

Due to the nature of the search algorithm (hill climbing with restart), the choice of the candidate used to restart the search is crucial. In our case we restart with the candidate with the best fitness so far which has not been yet used to restart. The danger of such an approach is that when the algorithm enters a region with local optima (like a plateau), it may basically stay there as long as there exist better candidates in that region than outside of it non used yet for restart. Longer programs tend to generate larger regions of local optima (because they have exponentially more syntactic variations that lead to close behaviors), so if the search enters such region via an overly complex program it is likely to take a very long time to get out of it. Introducing probability in the choice of the restart may help avoiding that sort of trap but having experimented with that it turned out not to be significantly better on average for learning relatively simple things (indeed although the restart choice is more diverse it still tends to occur in large region of local optima).

However, a significant improvement we have found is to carefully choose the size penalty function so that the search will tend to restart on simpler programs even if they do not exhibit

* Bayes rule as used here is $P(M|D) = \frac{P(M)P(D|M)}{P(D)}$ where M denotes the Model (the program) and D denotes the data (the behavior to imitate), here $P(D)$ is ignored, that is the data is assumed to be distributed uniformly

EFTA00624356

the best behaviors, but will still be able to reach the optimal solution even if it is a complex one.

The solution we suggest is to choose a and b such that $penalty(p)$ is:

1. as penalizing as possible, to focus on simpler programs first (although that constraint may possibly be lightened as the experimentation shows),
2. but still correct in the sense that the optimal solution p maximizes $fitness(p)$.

And we want that to work for all problems we are interested in. That restriction is an important point because it is likely that in general the second constraint will be too strict to produce a good penalty function.

We will now formalize the above problem. Let i be an index that ranges over the set of problems of interest (in our case pet tricks to learn), $score_i$ and $fitness_i$ denotes the score and fitness functions of the i^{th} problem. Let $\Theta_i(s)$ denote the set of programs of score s

$$\Theta_i(s) = \{p | score(p) = s\}$$

Define a family of partial functions

$$f_i : [0,1] \mapsto \mathbb{N}$$

so that

$$f_i(s) = \underset{p \in \Theta_i(p)}{\text{argmin}} \ |p|$$

What this says is that for any given score s we want the size of the shortest program p with that score. And f_i is partial because there may not be any program returning a given score.

Let be the family of partial functions

$$g_i : [0,1] \mapsto [0,1]$$

parametrized by a and b such that

$$g_i(s) = s \times exp(-a \times (log(b \times |A| + e) \times f_i(s))$$

That is: given a score s, $g_i(s)$ returns the fitness $fitness(p)$ of the shortest program p that marks that score.

32.5.3 Definition of the Optimization Problem

Let s_i be the highest score obtained for fitness function i (that is the score of the program chosen as the current best solution of i). Now the optimization problem consists of finding some a and b such that

$$\forall i \ \underset{s}{\text{argmax}} \ g_i(s) = s_i$$

that is the highest score has also the highest fitness. We started by choosing a and b as high as possible, it is a good heuristic but not the best, the best one would be to choose a and b so

EFTA00624357

that they minimize the number of iterations (number of restarts) to reach a global optimum, which is a harder problem.

Also, regarding the resolution of the above equation, it is worth noting we do not need the analytical expression of $score(p)$. Using past learning experiences we can get a partial description of the fitness landscape of each problem just by looking at the traces of the search.

Overall we have found this optimization works rather well; that is, tricks that would otherwise take several hours or days of computation can be learned in seconds or minutes. And the method also enables fast learning for new tricks, in fact all tricks we have experimented with so far could be learned reasonably fast (seconds or minutes) without the need to retune the penalty function.

In the current CogPrime codebase, the algorithm in charge of calibrating the parameters of the penalty function has been written in Python. It takes in input the log of the imitation learning engine that contains the score, the size, the penalty and the fitness of all candidates explored for all tricks taken in consideration for the parameterizing. The algorithm proceeds in 2 steps:

1. Reconstitute the partial functions f_i for all fitness functions i already attempted, based on the traces of these previously optimized fitness functions.
2. Try to find the highest a and b so that

$$\forall i \ \underset{s}{\text{argmax}}\ g_i(s) = s_i$$

For step 2, since there are only two parameters to tune, we have used a 2D grid, enumerating all points (a, b) and zooming when necessary. So the speed of the process depends largely on the resolution of the grid but (on an ordinary 2009 PC processor) usually it does not require more than 20 minutes to both extract f_i and find a and b with a satisfactory resolution.

32.6 Some Simple Experimental Results

To test the above ideas in a simple context, we initially used them to enable an OpenCog powered virtual world agent to learn a variety of simple "dog tricks" based on imitation and reinforcement learning in the Multiverse virtual world. We have since deployed them on a variety of other applications in various domains.

We began these experiments by running learning on two tricks, fetch_ball and triple_kick to be described below, in order to calibrate the size penalty function:

1. fetch_ball, which corresponds to the Combo program

```
and_seq(goto(ball)
        grab(ball)
        goto(owner)
        drop)
```

2. triple_kick, if the stick is near the ball then kick 3 times with the left leg and otherwise 3 times with the right leg. So for that trick the owner had to provide 2 exemplars, one for kickL (with the stick near the ball) and one for kickR, and move away the ball from the stick before showing the second exemplar. Below is the Combo program of triple_kick

```
if(near(stick ball)
   and_seq(kickL kickL kickL)
   and_seq(kickR kickR kickR))
```

EFTA00624358

Before choosing an exponential size penalty function and calibrating it fetch_ball would be learned rather rapidly in a few seconds, but triple_kick would take more than an hour. After calibration both fetch_ball and triple_kick would be learned rapidly, the later in less than a minute.

Then we experimented with a new few tricks, some simpler, like sit_under_tree

```
and_seq(goto(tree) sit)
```

and others more complex like double_dance, where the trick consists of dancing until the owner emits the message "stop dancing", and changing the dance upon owner's actions

```
while(not(says(owner ''stop dancing''))
      if(last_action(owner ''kickL'')
         tap_dance
         lean_rock_dance))
```

That is the pet performs a tap_dance when the last action of the owner is kickL, and otherwise performs a lean_rock_dance.

We tested learning for 3 tricks, fetch_ball, triple_kick and double_dance. Each trick was tested in 7 settings denoted $conf_1$ to $conf_{10}$ summarized in Table 32.1.

- $conf_1$ is the default configuration of the system, the parameters of the size penalty function are $a = 0.03$ and $b = 0.34$, which is actually not what is returned by the calibration technique but close to. That is because in practice we have found that on average learning is working slightly faster with these values.
- $conf_2$ is the configuration with the exact values returned by the calibration, that is $a = 0.05$, $b = 0.94$.
- $conf_3$ has the reduction engine disabled.
- $conf_4$ has the entropy filter disabled (threshold is null so all perceptions pass the filter).
- $conf_5$ has the intensity of the penalty function set to 0.
- $conf_6$ has the penalty function set with low intensity.
- $conf_7$ and $conf_8$ have the penalty function set with high intensity.
- $conf_9$ has the action sequence building block enabled
- $conf_{10}$ has the action sequence building block enabled but with a slightly lower intensity of the size penalty function than normal.

Reduct	ActSeq	Entropy	a	b	Setting
On	Off	0.1	0.03	0.34	$conf_1$
On	Off	0.1	0.05	0.94	$conf_2$
Off	Off	0.1	0.03	0.34	$conf_3$
On	Off	0	0.03	0.34	$conf_4$
On	Off	0.1	0	0.34	$conf_5$
On	Off	0.1	0.0003	0.34	$conf_6$
On	Off	0.1	0.3	0.34	$conf_7$
On	Off	0.1	3	0.34	$conf_8$
On	On	0.1	0.03	0.34	$conf_9$
On	On	0.1	0.025	0.34	$conf_{10}$

Table 32.1: Settings for each learning experiment

EFTA00624359

Setting	Percep	Restart	Eval	Time
$conf_1$	3	3	653	5s18
$conf_2$	3	3	245	2s
$conf_3$	3	3	1073	8s42
$conf_4$	136	3	28287	4mn7s
$conf_5$	3	>700	>500000	>1h
$conf_6$	3	3	653	5s18
$conf_7$	3	8	3121	23s42
$conf_8$	3	147	65948	8mn10s
$conf_9$	3	0	89	410ms
$conf_{10}$	3	0	33	161ms

Table 32.2: Learning time for fetch_ball

Setting	Percep	Restart	Eval	Time
$conf_1$	1	18	2783	21s47
$conf_2$	1	110	11426	1mn53s
$conf_3$	1	49	15069	2mn15s
$conf_4$	124	∞	∞	∞
$conf_5$	1	>800	>200K	>1h
$conf_6$	1	7	1191	9s67
$conf_7$	1	>2500	>200K	>1h
$conf_8$	1	>2500	>200K	>1h
$conf_9$	1	0	107	146ms
$conf_{10}$	1	0	101	164ms

Table 32.3: Learning time for triple_kick

Setting	Percep	Restart	Eval	Time
$conf_1$	5	1	113	4s
$conf_2$	5	1	113	4s
$conf_3$	5	1	150	6s20ms
$conf_4$	139	>4	>60K	>1h
$conf_5$	5	1	113	4s
$conf_6$	5	1	113	4s
$conf_7$	5	1	113	4s
$conf_8$	5	>1000	>300K	>1h
$conf_9$	5	1	138	4s191ms
$conf_{10}$	5	181	219K	56mn3s

Table 32.4: Learning time for double_dance

Tables 32.2, 32.3 and 32.4 contain the results of the learning experiment for the three tricks, fetch_ball, triple_kick and double_dance. In each table the column Percept gives the number perceptions which is taken into account for the learning. Restart gives the number of restarts hill climbing had to do before reaching the solution. Eval gives the number of evaluations and Time the search time.

In Table 32.2 and 32.4 we can see that fetch_ball or double_dance are learned in a few seconds both in $conf_1$ and $conf_2$. In 32.3 however learning is about five times faster with $conf_1$ than with $conf_2$, which was the motivation to go with $conf_2$ as default configuration, but $conf_2$ still performs well.

EFTA00624360

As Tables 32.2, 32.3 and 32.4 demonstrate for setting $conf_3$, the reduction engine speeds the search up by less than twice for fetch_ball and double_dance, and many times for triple_kick.

The results for $conf_4$ shows the importance of the filtering function, learning is dramatically slowed down without it. A simple trick like fetch_ball took few minutes instead of seconds, double_dance could not be learned after an hour, and triple_kick might never be learned because it did not focus on the right perception from the start.

The results for $conf_5$ shows that without any kind of complexity penalty learning can be dramatically slowed down, for the reasons explained in *Section 32.5* , that is the search loses itself in large regions of sub-optima. Only double_dance was not affected by that, which is probably explained by the fact that only one restart occurred in double_dance and it happened to be the right one.

The results for $conf_6$ show that when action sequence building-block is disabled the intensity of the penalty function could be set even lower. For instance triple_kick is learned faster (9s67 instead of 21s47 for $conf_1$). Conversely the results for $conf_7$ show that when action sequence building-block is enabled, if the Occam's razor is too weak it can dramatically slow down the search. That is because in this circumstance the search is misled by longer candidates that fit and takes a very cut before it can reach the optimal more compact solution.

32.7 Conclusion

In our experimentation with hillclimbing for learning pet tricks in a virtual world, we have shown that the combination of

1. candidate reduction into normal form,
2. filtering operators to narrow the alphabet,
3. using action sequences that are compatible with the shown behavior as building blocks,
4. adequately choosing and calibrating the complexity penalty function,

can speed up imitation learning so that moderately complex tricks can be learned within seconds to minutes instead of hours, using a simple "hill climbing with restarts" learning algorithm.

While we have discussed these ideas in the context of pet tricks, they have of course been developed with more general applications in mind, and have been applied in many additional contexts. Combo can be used to represent any sort of procedure, and both the hillclimbing algorithm and the optimization heuristics described here appear broad in their relevance.

Natural extensions of the approach described here include the following directions:

1. improving the Entity and Entropy filter using ECAN and PLN so that filtering is not only based on entropy but also relevancy with respect to the context and background knowledge,
2. using transfer learning (see Section 33.5 of Chapter 33) to tune the parameters of algorithm using contextual and background knowledge.

Indeed these improvements are under active investigation at time of writing, and some may well have been implemented and tested by the time you read this.

EFTA00624361

Chapter 33
Probabilistic Evolutionary Procedure Learning

Co-authored with Moshe Looks*

33.1 Introduction

The CogPrime architecture fundamentally requires, as one of its components, some powerful algorithm for automated program learning. This algorithm must be able to solve procedure learning problems relevant to achieving human-like goals in everyday human environments, relying on the support of other cognitive processes, and providing them with support in turn. The requirement is not that complex human behaviors need to be learnable via program induction alone, but rather that when the best way for the system to achieve a certain goal seems to be the acquisition of a chunk of *procedural* knowledge, the program learning component should be able to carry out the requisite procedural knowledge.

As CogPrime is a fairly broadly-defined architecture overall, there are no extremely precise requirements for its procedure learning component. There could be variants of CogPrime in which procedure learning carried more or less weight, relative to other components.

Some guidance here may be provided by looking at which tasks are generally handled by humans primarily using procedural learning, a topic on which cognitive psychology has a fair amount to say, and which is also relatively amenable to commonsense understanding based on our introspective and social experience of being human. When we know how to do something, but can't explain very clearly to ourselves or others how we do it, the chances are high that we have acquired this knowledge using some form of "procedure learning" as opposed to declarative learning. This is especially the case if we can do this same sort of thing in many different contexts, each time displaying a conceptually similar series of actions, but adapted to the specific situation. We would like CogPrime to be able to carry out procedural learning in roughly the same situations ordinary humans can (and potentially other situations as well: maybe even at the start, and definitely as development proceeds), largely via action of its program learning component.

In practical terms, our intuition (based on considerable experience with automated program learning, in OpenCog and other contexts) is that one requires a program learning component capable of learning programs with between dozens and hundreds of program tree nodes, in Combo or some similar representation – not able to learn *arbitrary* programs of this size, but rather able to solve problems arising in everyday human situations in which the simplest acceptable solutions involve programs of this size. We also suggest that the majority of procedure

* First author

EFTA00624362

learning problems arising in everyday human situation can be solved via program with hierarchical structure, so that it likely suffices to be able to learn programs with between dozens and hundreds of program tree nodes, where the programs have a modular structure, consisting of modules each possessing no more than dozens of program tree nodes. Roughly speaking, with only a few dozen Combo tree nodes, complex behaviors seem only achievable via using very subtle algorithmic tricks that aren't the sort of thing a human-like mind in the early stages of development could be expected to figure out; whereas, getting beyond a few hundred Combo tree nodes, one seems to get into the domain where an automated program learning approach is likely infeasible without rather strong restrictions on the program structure, so that a more appropriate approach within CogPrime would be to use PLN, concept creation or other methods to fuse together the results of multiple smaller procedure learning runs.

While simple program learning techniques like hillclimbing (as discussed in Chapter 32 above) can be surprisingly powerful, they do have fundamental limitations, and our experience and intuition both indicate that they are not adequate for serving as CogPrime's primary program learning component. This chapter describes an algorithm that we do believe is thus capable – CogPrime's most powerful and general procedure learning algorithm, MOSES, an integrative probabilistic evolutionary program learning algorithm that was briefly overviewed in Chapter 6 of Part 1.

While MOSES as currently designed and implemented embodies a number of specific algorithmic and structural choices, at bottom it embodies two fundamental insights that are critical to generally intelligent procedure learning:

- *Evolution is the right approach* to the learning of difficult procedures
- *Enhancing evolution with probabilistic methods is necessary.* Pure evolution, in the vein of the evolution of organisms and species, is too slow for broad use within cognition; so what is required is a hybridization of evolutionary and probabilistic methods, where probabilistic methods provide a more directed approach to generating candidate solutions than is possible with typical evolutionary heuristics like crossover and mutation

We summarize these insights in the phrase *Probabilistic Evolutionary Program Learning* (PEPL); MOSES is then one particular PEPL algorithm, and in our view a very good one. We have also considered other related algorithms such as the PLEASURE algorithm [Goe08a] (which may also be hybridized with MOSES), but for the time being it appears to us that MOSES satisfies CogPrime's needs.

Our views on the fundamental role of evolutionary dynamics in intelligence were briefly presented in Chapter 3 of Part 1. Terrence Deacon said it even more emphatically: "At every step the design logic of brains is a Darwinian logic: overproduction, variation, competition, selection ... it should not come as a surprise that this same logic is also the basis for the normal millisecond-by-millisecond information processing that continues to adapt neural software to the world." [Dea98] He has articulated ways in which, during neurodevelopment, different computations compete with each other (e.g., to determine which brain regions are responsible for motor control). More generally, he posits a kind of continuous flux as control shifts between competing brain regions, again, based on high-level "cognitive demand."

Deacon's intuition is similar to the one that led Edelman to propose Neural Darwinism [Ede93], and Calvin and Bickerton [CB00] to pose the notion of mind as a "Darwin Machine". The latter have given plausible neural mechanisms ("Darwin Machines") for synthesizing short "programs". These programs are for tasks such as rock throwing and sentence generation, which are represented as coherent firing patterns in the cerebral cortex. A population of such patterns,

EFTA00624363

competing for neurocomputational territory, replicates with variations, under selection pressure to conform to background knowledge and constraints.

To incorporate these insights, a system is needed that can recombine existing solutions in a non-local synthetic fashion, learning nested and sequential structures, and incorporate background knowledge (e.g. previously learned routines). MOSES is a particular kind of *program evolution* intended to satisfy these goals, using a combination of probability theory with ideas drawn from genetic programming, and also incorporating some ideas we have seen in previous chapters such as program normalization.

The main conceptual assumption about CogPrime's world, implicit in the suggestion of MOSES as the primary program learning component, is that the goal-relevant knowledge that cannot effectively be acquired by the other methods at CogPrime's disposal (PLN, ECAN, etc.), forms a body of knowledge that can effectively be induced across via probabilistic modeling on the space of programs for controlling a CogPrime agent. If this is not true, then MOSES will provide no advantage over simple methods like well-tuned hillclimbing as described in Chapter 32. If it is true, then the effort of deploying a complicated algorithm like MOSES is worthwhile. In essence, the assumption is that there are relatively simple regularities among the programs implementing those procedures that are most critical for a human-like intelligence to acquire via procedure learning rather than other methods.

33.1.1 Explicit versus Implicit Evolution in CogPrime

Of course, the general importance of evolutionary dynamics for intelligence does not imply the need to use explicit evolutionary algorithms in one's AGI system. Evolution can occur in an intelligent system whether or not the low-level *implementation layer* of the system involves any explicitly evolutionary processes. For instance it's clear that the human mind/brain involves evolution in this sense on the emergent level – we create new ideas and procedures by varying and combining ones that we've found useful in the past, and this occurs on a variety of levels of abstraction in the mind. In CogPrime, however, we have chosen to implement evolutionary dynamics explicitly, as well as encouraging them to occur implicitly.

CogPrime is intended to display evolutionary dynamics on the derived-hypergraph level, and this is intended to be a consequence of both explicitly-evolutionary and not-explicitly-evolutionary dynamics. Cognitive processes such as PLN inference may lead to emergent evolutionary dynamics (as useful logical relationships are reasoned on and combined, leading to new logical relationships in an evolutionary manner); even though PLN in itself is not explicitly *evolutionary* in character, it becomes emergently evolutionary via its coupling with CogPrime's attention allocation subsystem, which gives more cognitive attention to Atoms with more importance, and hence creates an evolutionary dynamic with importance as the fitness criterion and the whole constellation of MindAgents as the novelty-generation mechanism. However, MOSES *explicitly* embodies evolutionary dynamics for the learning of new patterns and procedures that are too complex for hillclimbing or other simple heuristics to handle. And this evolutionary learning subsystem naturally also contributes to the creation of evolutionary patterns on the emergent, derived-hypergraph level.

EFTA00624364

33.2 Estimation of Distribution Algorithms

There is a long history in AI of applying evolution-derived methods to practical problem-solving; John Holland's genetic algorithm [Hol75], initially a theoretical model, has been adapted successfully to a wide variety of applications (see e.g. the proceedings of the GECCO conferences). Briefly, the methodology applied is as follows:

1. generate a random population of solutions to a problem
2. evaluate the solutions in the population using a predefined fitness function
3. select solutions from the population proportionate to their fitnesss
4. recombine/mutate them to generate a new population
5. go to step 2

Holland's paradigm has been adapted from the case of fixed-length strings to the evolution of variable-sized and shaped trees (typically Lisp S-expressions), which in principle can represent arbitrary computer programs [Koz92, Koz94].

Recently, replacements-for/extensions-of the genetic algorithm have been developed (for fixed-length strings) which may be described as *estimation-of-distribution* algorithms (see [Pel05] for an overview). These methods, which outperform genetic algorithms and related techniques across a range of problems, maintain centralized probabilistic models of the population learned with sophisticated datamining techniques. One of the most powerful of these methods is the *Bayesian optimization algorithm* (BOA) [Pel05].

The basic steps of the BOA are:

1. generate a random population of solutions to a problem
2. evaluate the solutions in the population using a predefined fitness function
3. from the promising solutions in the population, learn a generative model
4. create new solutions using the model, and merge them into the existing population
5. go to step 2.

The neurological implausibility of this sort of algorithm is readily apparent – yet recall that in CogPrime we are attempting to roughly emulate human cognition on the level of behavior not structure or dynamics.

Fundamentally, the BOA and its ilk (the *competent* adaptive optimization algorithms) differ from classic selecto-recombinative search by attempting to dynamically learn a problem decomposition, in terms of the variables that have been pre-specified. The BOA represents this decomposition as a Bayesian network (directed acyclic graph with the variables as nodes, and an edge from x to y indicating that y is probabilistically dependent on x). An extension, the hierarchical Bayesian optimization algorithm (hBOA) , uses a Bayesian network with local structure to more accurately represent hierarchical dependency relationships. The BOA and hBOA are scalable and robust to noise across the range of nearly decomposable functions. They are also effective, empirically, on real-world problems with unknown decompositions, which may or may not be effectively representable by the algorithms; robust, high-quality results have been obtained for Ising spin glasses and MaxSAT, as well as a variety of real-world problems.

EFTA00624365

33.3 Competent Program Evolution via MOSES

In this section we summarize *meta-optimizing semantic evolutionary search* (MOSES), a system for competent program evolution, described more thoroughly in [Loo06]. Based on the viewpoint developed in the previous section, MOSES is designed around the central and unprecedented capability of competent optimization algorithms such as the hBOA, to generate new solutions that simultaneously combine sets of promising assignments from previous solutions according to a dynamically learned problem decomposition. The novel aspects of MOSES described herein are built around this core to exploit the unique properties of program learning problems. This facilitates effective problem decomposition (and thus competent optimization).

33.3.1 Statics

The basic goal of MOSES is to exploit the regularities in program spaces outlined in the previous section, most critically *behavioral decomposability* and *white box execution*, to dynamically construct representations that limit and transform the program space being searched into a relevant subspace with a compact problem decomposition. These representations will evolve as the search progresses.

33.3.1.1 An Example

Let's start with an easy example. What knobs (meaningful parameters to vary) exist for the family of programs depicted in Figure **??** on the left? We can assume, in accordance with the principle of white box execution, that all symbols have their standard mathematical interpretations, and that $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ are real-valued variables.

In this case, all three programs correspond to variations on the behavior represented graphically on the right in the figure. Based on the principle of behavioral decomposability, good knobs should express plausible evolutionary variation and recombination of features in behavior space, regardless of the nature of the corresponding changes in program space. It's worth repeating once more that this goal cannot be meaningfully addressed on a syntactic level - it requires us to leverage background knowledge of what the symbols in our vocabulary (***cos***, $+$, ***0.35***, etc.) actually *mean*.

A good set of knobs will also be *orthogonal*. Since we are searching through the space of combinations of knob settings (not a single change at a time, but a set of changes), any knob whose effects are equivalent to another knob or combination of knobs is undesirable.[†] Correspondingly, our set of knobs should *span* all of the given programs (i.e., be able to represent them as various knob settings).

A small *basis* for these programs could be the 3-dimensional parameter space, $x_1 \in \{x, z, 0\}$ (left argument of the root node), $x_2 \in \{y, x\}$ (argument of ***cos***), and $x_3 \in [0.3, 0.4]$ (multiplier for the ***cos***-expression). However, this is a very limiting view, and overly tied to the particulars

[†] First because this will increase the number of samples needed to effectively model the structure of knob-space, and second because this modeling will typically be quadratic with the number of knobs, at least for the BOA or hBOA.

EFTA00624366

of how these three programs happen to be encoded. Considering the space *behaviorally* (right of Figure ??), a number of additional knobs can be imagined which might be turned in meaningful ways, such as:

1. numerical constants modifying the phase and frequency of the cosine expression,
2. considering some weighted average of $\boldsymbol{x}$ and $\boldsymbol{y}$ instead of one or the other,
3. multiplying the entire expression by a constant,
4. adjusting the relative weightings of the two arguments to $+$.

33.3.1.2 Syntax and Semantics

This kind of representation-building calls for a correspondence between syntactic and semantic variation. The properties of program spaces that make this difficult are over-representation and chaotic execution, which lead to *non-orthogonality*, *oversampling of distant behaviors*, and *undersampling of nearby behaviors*, all of which can directly impede effective program evolution.

Non-orthogonality is caused by over-representation. For example, based on the properties of commutativity and associativity, $a_1 + a_2 + ... + a_n$ may be expressed in exponentially many different ways, if $+$ is treated as a non-commutative and non-associative binary operator. Similarly, operations such as addition of zero and multiplication by one have no effect, the successive addition of two constants is equivalent to the addition of their sum, etc. These effects are not quirks of real-valued expressions; similar redundancies appear in Boolean formulae ($\boldsymbol{x}$ ***AND*** $\boldsymbol{x} \equiv \boldsymbol{x}$), list manipulation (***cdr(cons(x, L))*** $\equiv$ ***L***), and conditionals (***if x then y else*** $\boldsymbol{z} \equiv$ ***if NOT x then z else y***).

Without the ability to exploit these identities, we are forced to work in a greatly expanded space which represents equivalent expression in many different ways, and will therefore be very far from orthogonality. Completely eliminating redundancy is infeasible, and typically NP-hard (in the domain of Boolean formulae it is reducible to the satisfiability problem, for instance), but one can go quite far with a heuristic approach.

Oversampling of distant behaviors is caused directly by chaotic execution, as well as a somewhat subtle effect of over-representation, which can lead to simpler programs being heavily oversampled. Simplicity is defined relative to a given program space in terms of minimal length, the number of symbols in the shortest program that produces the same behavior.

Undersampling of nearby behaviors is the flip side of the oversampling of distant behaviors. As we have seen, syntactically diverse programs can have the same behavior; this can be attributed to redundancy, as well as non-redundant programs that simply compute the same result by different means. For example, ***3*x*** can also be computed as ***x+x+x***; the first version uses less symbols, but neither contains any obvious "bloat" such as addition of zero or multiplication by one. Note however that the nearby behavior of ***3.1*x***, is syntactically close to the former, and relatively far from the latter. The converse is the case for the behavior of ***2*x+y***. In a sense, these two expressions can be said to exemplify differing organizational principles, or points of view, on the underlying function.

Differing organizational principles lead to different biases in sampling nearby behaviors. A superior organizational principle (one leading to higher-fitness syntactically nearby programs for a particular problem) might be considered a *metaptation* (adaptation at the second tier). Since equivalent programs organized according to different principles will have identical fitnesss, some methodology beyond selection for high fitnesss must be employed to search for good organizational principles. Thus, the resolution of undersampling of nearby behaviors revolves

EFTA00624367

around the management of *neutrality* in search, a complex topic beyond the scope of this chapter.

These three properties of program spaces greatly affect the performance of evolutionary methods based solely on syntactic variation and recombination operators, such as local search or genetic programming. In fact, when quantified in terms of various fitness-distance correlation measures, they can be effective predictors of algorithm performance, although they are of course not the whole story. A semantic search procedure will address these concerns in terms of the underlying behavioral effects of and interactions between a language's basic operators; the general scheme for doing so in MOSES is the topic of the next subsection.

33.3.1.3 Neighborhoods and Normal Forms

The procedure MOSES uses to construct a set of knobs for a given program (or family of structurally related programs) is based on three conceptual steps: *reduction to normal form*, *neighborhood enumeration*, and *neighborhood reduction*.

Reduction to normal form

- Redundancy is heuristically eliminated by reducing programs to a *normal form*. Typically, this will be via the iterative application of a series of local rewrite rules (e.g., $\forall x,\ x + 0 \rightarrow x$), until the target program no longer changes. Note that the well-known conjunctive and disjunctive normal forms for Boolean formulae are generally unsuitable for this purpose; they destroy the hierarchical structure of formulae, and dramatically limit the range of behaviors (in this case Boolean functions) that can be expressed compactly. Rather, *hierarchical normal forms* for programs are required.

Neighborhood enumeration

- A set of possible atomic *perturbations* is generated for all programs under consideration (the overall perturbation set will be the union of these). The goal is to heuristically generate new programs that correspond to behaviorally nearby variations on the source program, in such a way that arbitrary sets of perturbations may be *composed* combinatorially to generate novel valid programs.

Neighborhood reduction

- Redundant perturbations are heuristically culled to reach a more orthogonal set. A straightforward way to do this is to exploit the reduction to normal form outlined above; if multiple knobs lead to the same normal forms program, only one of them is actually needed. Additionally, note that the number of symbols in the normal form of a program can be used as a heuristic approximation for its minimal length - if the reduction to normal form of the program resulting from twiddling some knob significantly decreases its size, it can be assumed to be a source of oversampling, and hence eliminated from consideration. A slightly smaller program is typically

EFTA00624368

a meaningful change to make, but a large reduction in complexity will rarely be useful (and if so, can be accomplished through a combination of knobs that individually produce small changes).

At the end of this process, we will be left with a set of knobs defining a subspace of programs centered around a particular region in program space and heuristically centered around the corresponding region in behavior space as well. This is part of the *meta* aspect of MOSES, which seeks not to evaluate variations on existing programs itself, but to construct parameterized program subspaces (representations) containing meaningful variations, guided by background knowledge. These representations are used as search spaces within which an optimization algorithm can be applied.

33.3.2 Dynamics

As described above, the representation-building component of MOSES constructs a parameterized representation of a particular *region* of program space, centered around a single program family of closely related programs. This is consistent with the line of thought developed above, that a representation constructed across an arbitrary region of program space (e.g., all programs containing less than $\boldsymbol{n}$ symbols), or spanning an arbitrary collection of unrelated programs, is unlikely to produce a meaningful parameterization (i.e., one leading to a compact problem decomposition).

A sample of programs within a region derived from representation-building together with the corresponding set of knobs will be referred to herein as a *deme*;[‡] a set of demes (together spanning an arbitrary area within program space in a patchwork fashion) will be referred to as a *metapopulation*.[§] MOSES operates on a metapopulation, adaptively creating, removing, and allocating optimization effort to various demes. Deme management is the second fundamental *meta* aspect of MOSES, after (and above) representation-building; it essentially corresponds to the problem of effectively allocating computational resources to competing regions, and hence to competing programmatic organizational- representational schemes.

33.3.2.1 Algorithmic Sketch

The salient aspects of programs and program learning lead to requirements for competent program evolution that can be addressed via a representation-building process such as the one shown above, combined with effective deme management. The following sketch of MOSES, elaborating Figure 33.1 repeated here from Chapter 8 of Part 1, presents a simple control flow that dynamically integrates these processes into an overall program evolution procedure:

1. Construct an initial set of knobs based on some prior (e.g., based on an empty program) and use it to generate an initial random sampling of programs. Add this deme to the metapopulation.
2. Select a deme from the metapopulation and update its sample, as follows:

[‡] A term borrowed from biology, referring to a somewhat isolated local population of a species.

[§] Another term borrowed from biology, referring to a group of somewhat separate populations (the demes) that nonetheless interact.

EFTA00624369

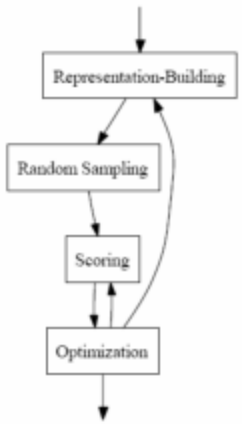


Fig. 33.1: The top-level architectural components of MOSES, with directed edges indicating the flow of information and program control.

   a. Select some promising programs from the deme's existing sample to use for modeling, according to the fitness function.
   b. Considering the promising programs as collections of knob settings, generate new collections of knob settings by applying some (competent) optimization algorithm.
   c. Convert the new collections of knob settings into their corresponding programs, reduce the programs to normal form, evaluate their fitnesss, and integrate them into the deme's sample, replacing less promising programs.

3. For each new program that meets the criteria for creating a new deme, if any:

   a. Construct a new set of knobs (via representation-building) to define a region centered around the program (the deme's *exemplar)*, and use it to generate a *new* random sampling of programs, producing a new deme.
   b. Integrate the new deme into the metapopulation, possibly displacing less promising demes.

4. Repeat from step 2.

The criterion for creating a new deme is *behavioral non-dominance* (programs which are not dominated by the exemplars of any existing demes are used as exemplars to create new demes), which can be defined in a domain-specific fashion. As a default, the fitness function may be used to induce dominance, in which case the set of exemplar programs for demes corresponds to the set of top-fitness programs.

33.3.3 Architecture

The preceding algorithmic sketch of MOSES leads to the top-level architecture depicted in Figure ??. Of the four top-level components, only the fitness function is problem-specific. The

EFTA00624370

representation-building process is domain-specific, while the random sampling methodology and optimization algorithm are domain-general. There is of course the possibility of improving performance by incorporating domain and/or problem-specific bias into random sampling and optimization as well.

33.3.4 Example: Artificial Ant Problem

Let's go through all of the steps that are needed to apply MOSES to a small problem, the artificial ant on the Santa Fe trail [Koz92], and describe the search process. The artificial ant domain is a two-dimensional grid landscape where each cell may or may not contain a piece of food. The artificial ant has a location (a cell) and orientation (facing up, down, left, or right), and navigates the landscape via a primitive sensor, which detects whether or not there is food in the cell that the ant is facing, and primitive actuators *move* (take a single step forward), *right* (rotate 90 degrees clockwise), and *left* (rotate 90 degrees counter-clockwise). The Santa Fe trail problem is a particular 32 x 32 toroidal grid with food scattered on it (Figure 4), and a fitness function counting the number of unique pieces of food the ant eats (by entering the cell containing the food) within 600 steps (movement and 90 degree rotations are considered single steps).

Programs are composed of the primitive actions taking no arguments, a conditional (*if-food-ahead*),[¶] which takes two arguments and evaluates one or the other based on whether or not there is food ahead, and *progn*, which takes a variable number of arguments and sequentially evaluates all of them from left to right. To fitness a program, it is evaluated continuously until 600 time steps have passed, or all of the food is eaten (whichever comes first). Thus for example, the program *if-food-ahead(m, r)* moves forward as long as there is food ahead of it, at which point it rotates clockwise until food is again spotted. It can successfully navigate the first two turns of the placeSanta Fe trail, but cannot cross "gaps" in the trail, giving it a final fitness of 11.

The first step in applying MOSES is to decide what our reduction rules should look like. This program space has several clear sources of redundancy leading to over-representation that we can eliminate, leading to the following reduction rules:

1. Any sequence of rotations may be reduced to either a left rotation, a right rotation, or a reversal, for example:

   progn(left, left, left)
   reduces to
   right

1. Any *if-food-ahead* statement which is the child of an *if-food-ahead* statement may be eliminated, as one of its branches is clearly irrelevant, for example:

   if-food-ahead(m, if-food-ahead(l, r))
   reduces to
   if-food-ahead(m, r)

[¶] This formulation is equivalent to using a general three-argument if-then-else statement with a predicate as the first argument, as there is only a single predicate (food-ahead) for the ant problem.

EFTA00624371

1. Any *progn* statement which is the child of a *progn* statement may be eliminated and replaced by its children, for example:

progn(progn(left, move), move)
reduces to
progn(left, move, move)

The representation language for the ant problem is simple enough that these are the only three rules needed – in principle there could be many more. The first rule may be seen as a consequence of general domain-knowledge pertaining to rotation. The second and third rules are fully general simplification rules based on the semantics of *if-then-else* statements and associative functions (such as *progn*), respectively.

These rules allow us to naturally parameterize a knob space corresponding to a given program (note that the arguments to the *progn* and *if-food-ahead* functions will be recursively reduced and parameterized according to the same procedure). Rotations will correspond to knobs with four possibilities (*left, right, reversal, no rotation).* Movement commands will correspond to knobs with two possibilities (*move, no movement).* There is also the possibility of introducing a new command in between, before, or after, existing commands. Some convention (a "canonical form") for our space is needed to determine how the knobs for new commands will be introduced. A representation consists of a rotation knob, followed by a conditional knob, followed by a movement knob, followed by a rotation knob, etc.[‖]

The structure of the space (how large and what shape) and default knob values will be determined by the "exemplar" program used to construct it. The default values are used to bias the initial sampling to focus around the prototype associated to the exemplar: all of the n direct neighbors of the prototype are first added to the sample, followed by a random selection of n programs at a distance of two from the prototype, n programs at a distance of three, etc., until the entire sample is filled. Note that the hBOA can of course effectively recombine this sample to generate novel programs at any distance from the initial prototype. The empty program *progn* (which can be used as the initial exemplar for MOSES), for example, leads to the following prototype:

```
progn(
        rotate? [default no rotation],
        if-food-ahead(
                        progn(
                                rotate? [default no rotation],
                                move? [default no movement]),
                        progn(
                                rotate? [default no rotation],
                                move? [default no movement])),
        move? [default no movement])
```

There are six parameters here, three which are quaternary (*rotate*), and three which are binary (*move*). So the program

‖ That there is some fixed ordering on the knobs is important, so that two rotation knobs are not placed next to each other (as this would introduce redundancy). In this case, the precise ordering chosen (rotation, conditional, movement) does not appear to be critical.

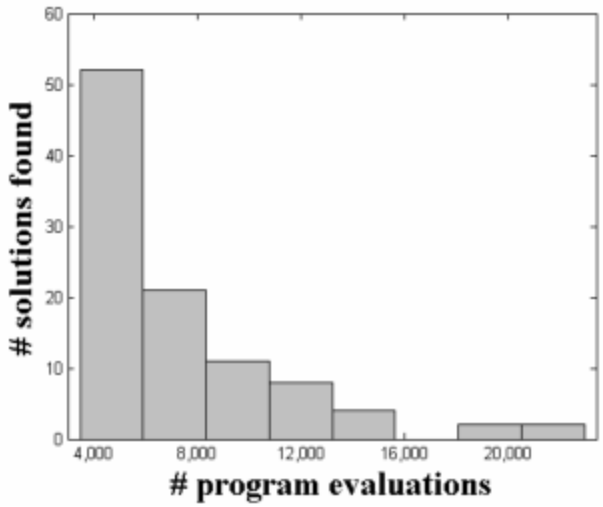
EFTA00624372

progn(left, if-food-ahead(move, left))

would be encoded in the space as

[*left, no rotation, move, left, no movement, no movement*]

with knobs ordered according to a pre-order left-to-right traversal of the program's parse tree (this is merely for exposition; the ordering of the parameters has no effect on MOSES). For a prototype program already containing an *if-food-ahead* statement, nested conditionals would be considered.



Technique	Computational Effort
Genetic Programming	450,000 evaluations
Evolutionary Programming	136,000 evaluations
MOSES	23,000 evaluations

Fig. 33.2: On the top, histogram of the number of global optima found after a given number of program evaluations for 100 runs of MOSES on the artificial ant problem (each run is counted once for the first global optimum reached). On the bottom, computational effort required to find an optimal solution for various techniques with probability p=.99 (for MOSES p=1, since an optimal solution was found in all runs).

A space with six parameters in it is small enough that MOSES can reliably find the optimum (the program *progn(right, if-food-ahead(progn(),left), move)*), with a very small population. After no further improvements have been made in the search for a specified number of generations (calculated based on the size of the space based on a model derived from [23] that is general to the hBOA, and not at all tuned for the artificial ant problem), a new representation is constructed centered around this program.** Additional knobs are introduced "in between" all

** MOSES reduces the exemplar program to normal form before constructing the representation; in this particular case however, no transformations are needed. Similarly, in general neighborhood reduction would be

EFTA00624373

existing ones (e.g., an optional move in between the first rotation and the first conditional), and possible nested conditionals are considered (a nested conditional occurring in a sequence *after* some other action has been taken is not redundant). The resulting space has 39 knobs, still quite tractable for hBOA, which typically finds a global optimum within a few generations. If the optimum were not to be found, MOSES would construct a new (possibly larger or smaller) representation, centered around the best program that *was* found, and the process would repeat.

The artificial ant problem is well-studied, with published benchmark results available for genetic programming as well as evolutionary programming based solely on mutation (i.e., a form of population-based stochastic hill climbing). Furthermore, an extensive analysis of the search space has been carried out by Langdon and Poli [LP02], with the authors concluding:

1. The problem is "deceptive at all levels", meaning that the partial solutions that must be recombined to solve the problem to optimality have lower average fitness than the partial solutions that lead to inferior local optima.
2. The search space contains many symmetries (e.g., between left and right rotations),
3. There is an unusually high density of global optima in the space (relative to other common test problems);
4. even though current evolutionary methods can solve the problem, they are not significantly more effective (in terms of the number of program evaluations require) than random sample.
5. "If real program spaces have the above characteristics (we expect them to do so but be still worse) then it is important to be able to demonstrate scalable techniques on such problem spaces".

33.3.4.1 Test Results

Koza [Koz92] reports on a set of 148 runs of genetic programming with a population size of 500 which had a 16% success rate after 51 generations when the runs were terminated (a total of 25,500 program evaluations per run). The minimal "computational effort" needed to achieve success with 99% probability was attained by processing through generation 14 was 450,000 (based on parallel independent runs). Chellapilla [Che97] reports 47 out of 50 successful runs with a minimal computational effort (again, for success with 99% probability) of 136,000 for his stochastic hill climbing method.

In our experiment with the artificial ant problem, one hundred runs of MOSES were executed. Beyond the domain knowledge embodied in the reduction and knob construction procedure, the only parameter that needed to be set was the population scaling factor, which was set to 30 (MOSES automatically adjusts to generate a larger population as the size of the representation grows, with the base case determined by this factor). Based on these "factory" settings, MOSES found optimal solutions on every run out of 100 trials, within a maximum of 23,000 program evaluations (the computational effort figure corresponding to 100% success). The average number of program evaluations required was 6952, with 95% confidence intervals of ±856 evaluations.

Why does MOSES outperform other techniques? One factor to consider first is that the language programs are evolved in is slightly more expressive than that used for the other

---

used to eliminate any extraneous knobs (based on domain-specific heuristics). For the ant domain however no such reductions are necessary.

EFTA00624374

techniques; specifically, a *progn* is allowed to have no children (if all of its possible children are "turned off"), leading to the possibility of *if-food-ahead* statements which do nothing if food is present (or not present). Indeed, many of the smallest solutions found by MOSES exploit this feature. This can be tested by inserting a "do nothing" operation into the terminal set for genetic programming (for example). Indeed, this reduces the computational effort to 272,000; an interesting effect, but still over an order of magnitude short of the results obtained with MOSES (the success rate after 50 generations is still only 20%).

Another possibility is that the reductions in the search space via simplification of programs alone are responsible. However, the results past attempts at introducing program simplification into genetic programming systems [27, 28] have been mixed; although the system may be sped up (because programs are smaller), there have been no dramatic improvement in results noted. To be fair, these results have been primarily focused on the symbolic regression domain; I am not aware of any results for the artificial ant problem.

The final contributor to consider is the sampling mechanism (knowledge-driven knob-creation followed by probabilistic model-building). We can test to what extent model-building contributes to the bottom line by simply disabling it and assuming probabilistic independence between all knobs. The result here is of interest because model-building can be quite expensive ($O(n^2N)$ per generation, where n is the problem size and N is the population size[††]). In 50 independent runs of MOSES without model-building, a global optimum was still discovered in all runs. However, the variance in the number of evaluations required was much higher (in two cases over 100,000 evaluations were needed). The new average was 26,355 evaluations to reach an optimum (about 3.5 times more than required with model-building). The contribution of model-building to the performance of MOSES is expected to be even greater for more difficult problems.

Applying MOSES without model-building (i.e., a model assuming no interactions between variables) is a way to test the combination of representation-building with an approach resembling the probabilistic incremental program learning (PIPE) algorithm [SS03], which learns programs based on a probabilistic model without any interactions. PIPE has now been shown to provide results competitive with genetic programming on a number of problems (regression, agent control, etc.).

It is additionally possible to look inside the models that the hBOA constructs (based on the empirical statistics of successful programs) to see what sorts of linkages between knobs are being learned.[‡‡] For the 6-knob model given above for instance an analysis of the linkages learned shows that the three most common pairwise dependencies uncovered, occurring in over 90% of the models across 100 runs, are between the rotation knobs. No other individual dependencies occurred in more than 32% of the models. This preliminary finding is quite significant given Landgon and Poli's findings on symmetry, and their observation that "[t]hese symmetries lead to essentially the same solutions appearing to be the opposite of each other. E.g. either a pair of Right or pair of Left terminals at a particular location may be important."

In this relatively simple case, all of the components of MOSES appear to mesh together to provide superior performance – which is promising, though it of course does not prove that these same advantages will apply across the range of problems relevant to human-level AGI.

[††] The fact that reduction to normal tends to reduce the problem size is another synergy between it and the application of probabilistic model-building.

[‡‡] There is in fact even more information available in the hBOA models concerning hierarchy and direction of dependence, but this is difficult to analyze.

EFTA00624375

33.3.5 Discussion

The overall MOSES design is unique. However, it is instructive at this point to compare its two primary unique facets (representation-building and deme management) to related work in evolutionary computation.

Rosca's *adaptive representation* architecture [Ros99] is an approach to program evolution which also alternates between separate representation-building and optimization stages. It is based on Koza's genetic programming, and modifies the representation based on a *syntactic* analysis driven by the fitness function, as well as a modularity bias. The representation-building that takes place consists of introducing new compound operators, and hence modifying the implicit distance function in tree-space. This modification is uniform, in the sense that the new operators can be placed in any context, without regard for semantics.

In contrast to Rosca's work and other approaches to representation-building such as Koza's *automatically defined functions* [KA95], MOSES explicitly addresses the underlying (semantic) structure of program space independently of the search for any kind of modularity or problem decomposition. This preliminary stage critically changes neighborhood structures (syntactic similarity) and other aggregate properties of programs.

Regarding deme management, the embedding of an evolutionary algorithm within a superordinate procedure maintaining a metapopulation is most commonly associated with "island model" architectures [SWM90]. One of the motivations articulated for using island models has been to allow distinct islands to (usually implicitly) explore different regions of the search space, as MOSES does explicitly. MOSES can thus be seen as a very particular kind of island model architecture, where programs never migrate between islands (demes), and islands are created and destroyed dynamically as the search progresses.

In MOSES, optimization does not operate directly on program space, but rather on a subspace defined by the representation-building process. This subspace may be considered as being defined by a sort of template assigning values to some of the underlying dimensions (e.g., it restricts the size and shape of any resulting trees). The messy genetic algorithm [GKD89], an early competent optimization algorithm, uses a similar mechanism - a common "competitive template" is used to evaluate candidate solutions to the optimization problem which are themselves underspecified. Search consequently centers on the template(s), much as search in MOSES centers on the programs used to create new demes (and thereby new representations). The issue of deme management can thus be seen as analogous to the issue of template selection in the messy genetic algorithm.

33.3.6 Conclusion

Competent evolutionary optimization algorithms are a pivotal development, allowing encoded problems with compact decompositions to be tractably solved according to normative principles. We are still faced with the problem of *representation-building* – casting a problem in terms of knobs that can be twiddled to solve it. Hopefully, the chosen encoding will allow for a compact problem decomposition. Program learning problems in particular rarely possess compact decompositions, due to particular features generally present in program spaces (and in the mapping between programs and behaviors). This often leads to intractable problem formulations, even if the mapping between behaviors and fitness has an intrinsic separable or nearly decom-

EFTA00624376

posable structure. As a consequence, practitioners must often resort to manually carrying out the analogue of representation-building, on a problem-specific basis. Working under the thesis that *the properties of programs and program spaces can be leveraged as inductive bias to remove the burden of manual representation-building, leading to competent program evolution*, we have developed the MOSES system, and explored its properties.

While the discussion above has highlighted many of the features that make MOSES uniquely powerful, in a sense it has told only half the story. Part of what makes MOSES valuable for CogPrime is that it's good on its own; and the other part is that it cooperates well with the other cognitive processes within CogPrime. We have discussed aspects of this already in Chapter 8 of Part 1, especially in regard to the MOSES/PLN relationship. In the following section we proceed further to explore the interaction of MOSES with other aspects of the CogPrime system – a topic that will arise repeatedly in later chapters as well.

33.4 Integrating Feature Selection Into the Learning Process

In the typical workflow of applied machine learning, one begins with a large number of features, each applicable to some or all of the entities one wishes to learn about; then one applies some feature selection heuristics to whittle down the large set of features into a smaller one; then one applies a learning algorithm to the reduced set of features. The reason for this approach is that the more powerful among the existing machine learning algorithms tend to get confused when supplied with too many features. The problem with this approach is that sometimes one winds up throwing out potentially very useful information during the feature selection phase. This same sort of problem exists with MOSES in its simplest form, as described above.

The human mind, as best we understand it, does things a bit differently than this standard "feature selection followed by learning" process. It does seem to perform operations analogous to feature selection, and operations analogous to the application of a machine learning algorithm to a reduced feature set – but then it also involves feedback from these "machine learning like" operations to the "feature selection like" operations, so that the intermediate results of learning can cause the introduction into the learning process of features additional to those initially selected, thus allowing the development of better learning results.

Compositional spatiotemporal deep learning (CSDLN) architectures like HTM [HB06] or DeSTIN [ARC09a], as discussed in 27 incorporate this same sort of feedback. The lower levels of such an architecture, in effect, carry out "feature selection" for the upper levels – but then feedback from the upper to the lower levels also occurs, thus in effect modulating the "feature selection like" activity at the lower levels based on the more abstract learning activity on the upper levels. However, such CSDLN architectures are specifically biased toward recognition of certain sorts of patterns – an aspect that may be considered a bug or a feature of this class of learning architecture, depending on the context. For visual pattern recognition, it appears to be a feature, since the hierarchical structure of such algorithms roughly mimics the architecture of visual cortex. For automated learning of computer programs carrying out symbolic tasks, on the other hand, CSDLN architectures are awkward at best and probably generally inappropriate. For cases like language learning or abstract conceptual inference, the jury is out.

In this section we explore the question of: how to introduce an appropriate feedback between feature selection and learning in the case of machine learning algorithms with general scope and without explicit hierarchical structure – such as MOSES. We introduce a specific technique

EFTA00624377

enabling this, which we call LIFES, short for Learning-Incorporated Feature Selection. We argue that LIFES is particularly applicable to learning problems that possess the conjunction of two properties that we call data focusability and feature focusability. We illustrate LIFES in a MOSES context, via describing a specific incarnation of the LIFES technique that does feature selection repeatedly during the MOSES learning process, rather than just doing it initially prior to MOSES learning.

33.4.1 Machine Learning, Feature Selection and AGI

The relation between feature selection and machine learning appears an excellent example of the way that, even when the same basic technique is useful in both narrow AI and AGI, the method of utilization is often quite different. In most applied machine learning tasks, the need to customize feature selection heuristics for each application domain (and in some cases, each particular problem) is not a major difficulty. This need does limit the practical utilization of machine learning algorithms, because it means that many ML applications require an expert user who understands something about machine learning, both to deal with feature selection issues and to interpret the results. But it doesn't stand in the way of ML's fundamental usability. On the other hand, in an AGI context, the situation is different, and the need for human-crafted, context-appropriate feature selection does stand in the way of the straightforward insertion of most ML algorithms into an integrative AGI systems.

For instance, in the OpenCog integrative AGI architecture that we have co-architected [Gea13], the MOSES automated program learning algorithm plays a key role. It is OpenCog's main algorithm for acquiring procedural knowledge, and is used for generating some sorts of declarative knowledge as well. However, when MOSES tasks are launched automatically via the OpenCog scheduler based on an OpenCog agent's goals, there is no opportunity for the clever choice of feature selection heuristics based on the particular data involved. And crude feature selection heuristics based on elementary statistics, are often insufficiently effective, as they rule out too many valuable features (and sometimes rule out the most critical features). In this context, having a variant of MOSES that can sift through the scope of possible features in the course of its learning is very important.

An example from the virtual dog domain pursued in [GEA08] would be as follows. Each procedure learned by the virtual dog combines a number of different actions, such as "step forward", "bark", "turn around", "look right","lift left front leg," etc. In the virtual dog experiments done previously, the number of different actions permitted to the dog was less than 100, so that feature selection was not a major issue. However, this was an artifact of the relatively simplistic nature of the experiments conducted. For a real organism, or for a robot that learns its own behavioral procedures (say, via a deep learning algorithm) rather than using a pre-configured set of "animated" behaviors, the number of possible behavioral procedures to potentially be combined using a MOSES-learned program may be very large. In this case, one must either use some crude feature selection heuristic, have a human select the features, or use something like the LIFES approach described here. LIFES addresses a key problem in moving from the relatively simple virtual dog work done before, to related work with virtual agents displaying greater general intelligence.

As an other example, suppose an OpenCog-controlled agent is using MOSES to learn procedures for navigating in a dynamic environment. The features that candidate navigation pro-

EFTA00624378

cedures will want to pay attention to, may be different in a well-lit environment than in a dark environment. However, if the MOSES learning process is being launched internally via OpenCog's goal system, there is no opportunity for a human to adjust the feature selection heuristics based on the amount of light in the environment. Instead, MOSES has got to figure out what features to pay attention to all by itself. LIFES is designed to allow MOSES (or other comparable learning algorithms) to do this.

So far we have tested LIFES in genomics and other narrow-AI application areas, as a way of initially exploring and validating the technique. As our OpenCog work proceeds, we will explore more AGI-oriented applications of MOSES-LIFES. This will be relatively straightforward on a software level as MOSES is fully integrated with OpenCog.

33.4.2 Data- and Feature- Focusable Learning Problems

Learning-integrated feature selection as described here is applicable across multiple domain areas and types of learning problem – but it is not completely broadly applicable. Rather it is most appropriate for learning problems possessing two properties we call data focusability and feature focusability. While these properties can be defined with mathematical rigor, here we will not be proving any theorems about them, so we will content ourselves with semi-formal definitions, sufficient to guide practical work.

We consider a fitness function Φ, defined on a space of programs f whose inputs are features defined on elements of a reference dataset S, and whose outputs lie in the interval $[0, 1]$. The features are construed as functions mapping elements of S into $[0, 1]$. Where $F(x) = (F^1(x).\ldots, F^n(x))$ is the set of features evaluated on $x \in S$, we use $f(x)$ as a shorthand for $f(F(x))$.

We are specifically interested in Φ which are "data focusable", in the sense that, for a large number of highly fit programs f, there is some subset S_f on which f is highly concentrated (note that S_f will be different for different f). By "concentrated" it is meant that

$$\frac{\sum_{x \in S_f} f(x)}{\sum_{x \in S} f(x)}$$

is large. A simple case is where f is Boolean and $f(x) = 1 \iff x \in S_f$.

One important case is where Φ is "property-based", in the sense that each element $x \in S$ has some Boolean or numeric property $p(x)$, and the fitness function $\Phi(f)$ rewards f for predicting $p(x)$ given x for $x \in S_f$, where S_f is **some** non-trivial subset of S. For example, each element of S might belong to some category, and the fitness function might represent the problem of placing elements of S into the proper category – but with the twist that f gets rewarded if it accurately places some subset S_f of elements in S into the proper category, even if it has nothing to say about all the elements in S but not S_f.

For instance, consider the case where S is a set of images. Suppose the function $p(x)$ indicates whether the image x contains a picture of a cat or not. Then, a suitable fitness function Φ would be one measuring whether there is some non-trivially large set of images S_f so that if $x \in S_f$, then f can accurately predict whether x contains a picture of a cat or not. A key point is that the fitness function doesn't care whether f can accurately predict whether x contains a picture of a cat or not, for x outside S_f.

EFTA00624379

Or, consider the case where S is a discrete series of time points, and $p(x)$ indicates the value of some quantity (say, a person's EEG) at a certain point in time. Then a suitable fitness function Φ might measure whether there is some non-trivially large set of time-points S_f so that if $x \in S_f$, then f can accurately predict whether x will be above a certain level L or not.

Finally, in addition to the property of data-focusability introduced above, we will concern ourselves with the complementary property of "feature-focusability." This means that, while the elements of S are each characterized by a potentially large set of features, there are many highly fit programs f that utilize only a small subset of this large set of features. The case of most interest here is where there are various highly fit programs f, each utilizing a different small subset of the overall large set of features. In this case one has (loosely speaking) a pattern recognition problem, with approximate solutions comprising various patterns that combine various different features in various different ways. For example, this would be the case if there were many different programs for recognizing pictures containing cats, each one utilizing different features of cats and hence applying to different subsets of the overall database of images.

There may, of course, be many important learning problems that are neither data nor feature focusable. However, the LIFES technique presented here for integrating feature selection into learning is specifically applicable to objective functions that are both data and feature focusable. In this sense, the conjunction of data and feature focusability appears to be a kind of "tractability" that allows one to bypass the troublesome separation of feature selection and learning, and straightforwardly combine the two into a single integrated process. Being property-based in the sense described above does not seem to be necessary for the application of LIFES, though most practical problems do seem to be property-based.

33.4.3 Integrating Feature Selection Into Learning

The essential idea proposed here is a simple one. Suppose one has a learning problem involving a fitness function that is both data and feature focusable. And suppose that, in the course of learning according to some learning algorithm, one has a candidate program f, which is reasonably fit but merits improvement. Suppose that f uses a subset F_f of the total set F of possible input features. Then, one may do a special feature selection step, customized just for f. Namely, one may look at the total set F of possible features, and ask **which features or small feature-sets display desirable properties on the set** S_f. This will lead to a new set of features potentially worthy of exploration; let's call it F'_f. We can then attempt to improve f by creating variants of f introducing some of the features in F'_f – either replacing features in F_f or augmenting them. The process of creating and refining these variants will then lead to new candidate programs g, potentially concentrated on sets S_g different from S_f, in which case the process may be repeated. This is what we call LIFES – Learning-Integrated Feature Selection.

As described above the LIFES process is quite general, and applies to a variety of learning algorithms – basically any learning algorithm that includes the capability to refine a candidate solution via the introduction of novel features. The nature of the "desirable properties" used to evaluate candidate features or feature-sets on S_f needs to be specified, but a variety of standard techniques may be used here (along with more advanced ideas) – for instance, in the case where the fitness function is defined in terms of some property mapping p as describe,

EFTA00624380

above, then given a feature F^i, one can calculate the mutual information of F^i with p over S_f. Other measures than mutual information may be used here as well.

The LIFES process doesn't necessarily obviate the need for up-front feature selection. What it does, is prevent up-front feature selection from limiting the ultimate feature usage of the learning algorithm. It allows the initially selected features to be used as a rough initial guide to learning – and for the candidates learned using these initial features, to then be refined and improved using additional features chosen opportunistically along the learning path. In some cases, the best programs ultimately learned via this approach might not end up involving any of the initially selected features.

33.4.4 Integrating Feature Selection into MOSES Learning

The application of the general LIFES process in the MOSES context is relatively straightforward. Quite simply, given a reasonably fit program f produced within a deme, one then isolates the set S_f on which f is concentrated, and identifies a set F'_f of features within F that displays desirable properties relative to S_f. One then creates a new deme f^*, with exemplar f, and with a set of potential input features consisting of $F_f \cup F'_f$.

What does it mean to create a deme $f^*$with a certain set of "potential input features" $F_f \cup F'_f$? Abstractly, it means that $F_{f^*} = F_f \cup F'_f$. Concretely, it means that the knobs in the deme's exemplar f^* must be supplied with settings corresponding to the elements of $F_f \cup F'_f$. The right way to do this will depend on the semantics of the features.

For instance, it may be that the overall feature space F is naturally divided into groups of features. In that case, each new feature F^i in F'_f would be added, as a potential knob setting, to any knob in f corresponding to a feature in the same group as F^i.

On the other hand, if there is no knob in f corresponding to features in F^i's knob group, then one has a different situation, and it is necessary to "mutate" f by adding a new node with a new kind of knob corresponding to F^i, or replacing an existing node with a new one corresponding to F^i.

33.4.5 Application to Genomic Data Classification

To illustrate the effectiveness of LIFES in a MOSES context, we now briefly describe an example application, in the genomics domain. The application of MOSES to gene expression data is described in more detail in [Loo07b], and is only very briefly summarized here. To obtain the results summarized here, we have used MOSES, with and without LIFES, to analyze two different genomics datasets: an Alzheimers SNP (single nucleotide polymorphism) dataset [Mea07] previously analyzed using ensemble genetic programming [CGPH09] . The dataset is of the form "Case vs. Control", where the Case category consists of data from individuals with Alzheimers and Control consists of matched controls. MOSES was used to learn Boolean program trees embodying predictive models that take in a subset of the genes in an individual, and output a Boolean combination of their discretized expression values, that is interpreted as a prediction of whether the individual is in the Case or Control category. Prior to feeding them into MOSES, expression values were first Q-normalized, and then discretized via comparison to the median

EFTA00624381

expression measured across all genes on a per-individual basis (1 for greater than the median, 0 for less than). Fitness was taken as precision, with a penalty factor restriction attention to program trees with recall above a specified minimum level.

These study was carried out, not merely for testing MOSES and LIFES, but as part of a practical investigation into which genes and gene combinations may be the best drug targets for Alzheimers Disease. The overall methodology for the biological investigation, as described in [GCPM06], is to find a (hopefully diverse) ensemble of accurate classification models, and then statistically observe which genes tend to occur most often in this ensemble, and which combinations of genes tend to co-occur most often in the models in the ensemble. These most frequent genes and combinations are taken as potential therapeutic targets for the Case category of the underlying classification problem (which in this case denotes inflammation). This methodology has been biologically validated by follow-up lab work in a number of cases; see e.g. [Gea05] where this approach resulted in the first evidence of a genetic basis for Chronic Fatigue Syndrome. A significant body of unpublished commercial work along these lines has been done by Biomind LLC [http://biomind.com] for its various customers.

Comparing MOSES-LIFES to MOSES with conventional feature selection, we find that the former finds model ensembles combining greater diversity with greater precision, and equivalent recall. This is because conventional feature selection eliminates numerous genes that actually have predictive value for the phenotype of inflammation, so that MOSES never gets to see them. LIFES exposes MOSES to a much greater number of genes, some of which MOSES finds useful. And LIFES enables MOSES to explore this larger space of genes without getting bollixed by the potential combinatorial explosion of possibilities.

Algorithm	Train. Precision	Train. Recall	Test Precision	Test Recall	
MOSES	.81	.51	.65	.42	best training precision
MOSES	.80	.52	.69	.43	best test precision
MOSES-LIFES	.84	.51	.68	.38	best training precision
MOSES-LIFES	.82	.51	.72	.48	best test precision

Table 33.1: Impact of LIFES on MOSES classification of Alzheimers Disease SNP data. Fitness function sought to maximize precision consistent with a constraint of precision being at least 0.5. Precision and recall figures are average figures over 10 folds, using 10-fold cross-validation. The results shown here are drawn from a larger set of runs, and are selected according to two criteria: best training precision (the fair way to do it) and best test precision (just for comparison). We see that use of LIFES increases precision by around 3% in these tests, which is highly statistically significant according to permutation analysis.

The genomics example shows that LIFES makes sense and works in the context of MOSES, broadly speaking. It seems very plausible that LIFES will also work effectively with MOSES in an integrative AGI context, for instance in OpenCog deployments where MOSES is used to drive procedure learning, with fitness functions supplied by other OpenCog components. However, the empirical validation of this plausible conjecture remains for future work.

EFTA00624382

33.5 Supplying Evolutionary Learning with Long-Term Memory

This section introduces an important enhancement to evolutionary learning, which extends the basic PEPL framework, by forming an adaptive hybridization of PEPL optimization with PLN inference (rather than merely using PLN inference within evolutionary learning to aid with modeling).

The first idea here is the use of PLN to supply evolutionary learning with a long-term memory. Evolutionary learning approaches each problem as an isolated entity, but in reality, a CogPrime system will be confronting a long series of optimization problems, with subtle interrelationships. When trying to optimize the function f, CogPrime may make use of its experience in optimizing other functions g.

Inference allows optimizers of g to be analogically transformed into optimizers of f, for instance it allows one to conclude:

```
Inheritance f g
   EvaluationLink f x
   EvaluationLink g x
```

However, less obviously, inference also allows patterns in populations of optimizers of g to be analogically transformed into patterns in populations of optimizers of f. For example, if *pat* is a pattern in good optimizers of f, then we have:

```
InheritanceLink f g

ImplicationLink
   EvaluationLink f x
   EvaluationLink pat x

ImplicationLink
   EvaluationLink g x
   EvaluationLink pat x
```

(with appropriate probabilistic truth values), an inference which says that patterns in the population of f-optimizers should also be patterns in the population of g-optimizers).

Note that we can write the previous example more briefly as:

```
InheritanceLink f g
   ImplicationLink (EvaluationLink f) (EvaluationLink pat)
   ImplicationLink (EvaluationLink g) (EvaluationLink pat)
```

A similar formula holds for SimilarityLinks.

We may also infer:

```
ImplicationLink (EvaluationLink g) (EvaluationLink pat_g)

ImplicationLink (EvaluationLink f) (EvaluationLink pat_f)

ImplicationLink
    (EvaluationLink (g AND f))
    (EvaluationLink (pat_g AND pat_f))
```

and:

```
ImplicationLink (EvaluationLink f) (EvaluationLink pat)
```

EFTA00624383

```
ImplicationLink (EvaluationLink ~f) (EvaluationLink ~pat)
```

Through these sorts of inferences, PLN inference can be used to give evolutionary learning a long-term memory, allowing knowledge about population models to be transferred from one optimization problem to another. This complements the more obvious use of inference to transfer knowledge about specific solutions from one optimization problem to another.

For instance in the problem of finding a compact program generating some given sequences of bits the system might have noticed that when the number of 0 roughly balances the number of 1 (let us call this property STR_BALANCE) successful optimizers tend to give greater biases toward conditionals involving comparisons of the number of 0 and 1 inside the condition, let us call this property over optimizers COMP_CARD_DIGIT_BIAS. This can be expressed in PLN as follows

```
AverageQuantifierLink ⟨tv⟩
    ListLink
        $X
        $Y
    ImplicationLink
        ANDLink
            InheritanceLink
                STR_BALANCE
                $X
            EvaluationLink
                SUCCESSFUL_OPTIMIZER_OF
                ListLink
                    $Y
                    $X
        InheritanceLink
            COMP_CARD_DIGIT_BIAS
            $Y
```

which translates by, if the problem `$X` inherits from STR_BALANCE and `$Y` is a successful optimizer of `$X` then, with probability p calculated according to `tv`, `$Y` tends to be biased according to the property described by COMP_CARD_DIGIT_BIAS.

33.6 Hierarchical Program Learning

Next we discuss hierarchical program structure, and its reflection in probabilistic modeling, in more depth. This is a surprisingly subtle and critical topic, which may be approached from several different complementary angles. To an extent, hierarchical structure is automatically accounted for in MOSES, but it may also be valuable to pay more explicit mind to it.

In human-created software projects, one common approach for dealing with the existence of complex interdependencies between parts of a program is to give the program a hierarchical structure. The program is then a hierarchical arrangement of programs within programs within programs, each one of which has relatively simple dependencies between its parts (however its

EFTA00624384

parts may themselves be hierarchical composites). This notion of hierarchy is essential to such programming methodologies as modular programming and object-oriented design.

Pelikan and Goldberg discuss the hierarchal nature of human problem-solving, in the context of the hBOA (hierarchical BOA) version of BOA. However, the hBOA algorithm does not incorporate hierarchical program structure nearly as deeply and thoroughly as the hierarchical procedure learning approach proposed here. In hBOA the hierarchy is implicit in the models of the evolving population, but the population instances themselves are not necessarily explicitly hierarchical in structure. In hierarchical PEPL as we describe it here, the population consists of hierarchically structured Combo trees, and the hierarchy of the probabilistic models corresponds directly to this hierarchical program structure.

The ideas presented here have some commonalities to John Koza's ADFs and related tricks for putting reusable subroutines in GP trees, but there are also some very substantial differences, which we believe will make the current approach far more effective (though also involving considerably more computational overhead).

We believe that this sort of hierarchically-savvy modeling is what will be needed to get probabilistic evolutionary learning to scale to large and complex programs, just as hierarchy-based methodologies like modular and object-oriented programming are needed to get human software engineering to scale to large and complex programs.

33.6.1 Hierarchical Modeling of Composite Procedures in the AtomSpace

The possibility of hierarchically structured programs is (intentionally) present in the CogPrime design, even without any special effort to build hierarchy into the PEPL framework. Combo trees may contain Nodes that point to PredicateNodes, which may in turn contain Combo trees, etc. However, our current framework for learning Combo trees does not take advantage of this hierarchy. What is needed, in order to do so, is for the models used for instance generation to include events of the form:

Combo tree Node at position x has type PredicateNode; and the PredicateNode at position x contains a Combo tree that possesses property P.

where x is a position in a Combo tree and P is a property that may or may not be true of any given Combo tree. Using events like this, a relatively small program explicitly incorporating only *short-range dependencies*; may implicitly encapsulate long-range dependencies via the properties P.

But where do these properties P come from? These properties should be patterns learned as part of the probabilistic modeling of the Combo tree inside the PredicateNode at position x. For example, if one is using a decision tree modeling framework, then the properties might be of the form *decision tree D evaluates to True*. Note that not all of these properties have to be statistically correlated with the fitness of the PredicateNode at position x (although some of them surely will be).

Thus we have a multi-level probabilistic modeling strategy. The top-level Combo tree has a probabilistic model whose events may refer to patterns that are parts of the probabilistic models of Combo trees that occur within it and so on down.

In instance generation, when a newly generated Combo tree is given a PredicateNode at position x, two possibilities exist:

EFTA00624385

- There is already a model for PredicateNodes at position x in Combo trees in the given population, in which case a population of PredicateNodes potentially living at that position is drawn from the known model, and evaluated.
- There is no such model (because it has never been tried to create a PredicateNode at position x in this population before), in which case a new population of Combo trees is created corresponding to the position, and evaluated.

Note that the fitness of a Combo tree that is not at the top level of the overall process, is assessed indirectly in terms of the fitness of the higher-level Combo tree in which it is embedded, due to the requirement of having certain properties, etc.

Suppose each Combo tree in the hierarchy has on average R adaptable sub-programs (represented as Nodes pointing to PredicateNodes containing Combo trees to be learned). Suppose the hierarchy is K levels deep. Then we will have about $R \times K$ program tree populations in the tree. This suggests that hierarchies shouldn't get too big, and indeed, they shouldn't need to, for the same essential reason that human-created software programs, if well-designed, tend not to require extremely deep and complex hierarchical structures.

One may also introduce a notion of *reusable components* across various program learning runs, or across several portions of the same hierarchical program. Here is one learning patterns of the form:

If property $P1(C, x)$ applies to a Combo tree C and a node x within it, then it is often good for node x to refer to a PredicateNode containing a Combo tree with property $P2$.

These patterns may be assigned probabilities and may be used in instance generation. They are general or specialized *programming guidelines,* which may be learned over time.

33.6.2 Identifying Hierarchical Structure In Combo trees via MetaNodes and Dimensional Embedding

One may also apply the concepts of the previous section to model a population of CTs that doesn't explicitly have a hierarchical structure, via introducing the hierarchical structure during the evolutionary process, through the introduction of special extra Combo tree nodes called MetaNodes. For instance MetaNodes may represent subtrees of Combo trees, which have proved useful enough that it seems justifiable to extract them as "macros." This concept may be implemented in a couple different ways, here we will introduce a simple way of doing this based on dimensional embedding, and then in the next section we will allude to a more sophisticated approach that uses inference instead.

The basic idea is to couple decision tree modeling with dimensional embedding of subtrees, a trick that enables small decision tree models to cover large regions of a CT in an approximate way, and which leads naturally to a form of probabilistically-guided crossover.

The approach as described here works most simply for CTs that have many subtrees that can be viewed as mapping numerical inputs into numerical outputs. There are clear generalizations to other sorts of CTs, but it seems advisable to test the approach on this relatively simple case first.

The first part of the idea is to represent subtrees of a CT as numerical vectors in a relatively low-dimensional space (say N=50 dimensions). This can be done using our existing dimensional embedding algorithm, which maps any metric space of entities into a dimensional space. All

EFTA00624386

that's required is that we define a way of measuring distance between subtrees. If we look at subtrees with numerical inputs and outputs, this is easy. Such a subtree can be viewed as a function mapping Rn into Rm; and there are many standard ways to calculate the distance between two functions of this sort (for instance one can make a Monte Carlo estimate of the L^p metric which is defined as:

```
[Sum(x) (f(x) - g(x))^p] ^ (1/p)
```

Of course, the same idea works for subtrees with non-numerical inputs and outputs, the tuning and implementation are just a little trickier.

Next, one can augment a CT with meta-nodes that correspond to subtrees. Each meta-node is of a special CT node type MetaNode, and comes tagged with an N-dimensional vector. Exactly which subtrees to replace with MetaNodes is an interesting question that must be solved via some heuristics.

Then, in the course of executing the PEPL algorithm, one does decision tree modeling as usual, but making use of MetaNodes as well as ordinary CT nodes. The modeling of MetaNodes is quite similar to the modeling of Nodes representing ConceptNodes and PredicateNodes using embedding vectors. In this way, one can use standard, small decision tree models to model fairly large portions of CTs (because portions of CTs are approximately represented by MetaNodes).

But how does one do instance generation, in this scheme? What happens when one tries to do instance generation using a model that predicts a MetaNode existing in a certain location in a CT? Then, the instance generation process has got to find some CT subtree to put in the place where the MetaNode is predicted. It needs to find a subtree whose corresponding embedding vector is close to the embedding vector stored in the MetaNode. But how can it find such a subtree?

There seem to be two ways:

1. A reasonable solution is to look at the database of subtrees that have been seen before in the evolving population, and choose one from this database, with the probability of choosing subtree X being proportional to the distance between X's embedding vector and the embedding vector stored in the MetaNode.
2. One can simply choose good subtrees, where the goodness of a subtree is judged by the average fitness of the instances containing the target subtree.

One can use a combination of both of these processes during instance generation.

But of course, what this means is that we're in a sense doing a form of crossover, because we're generating new instances that combine subtrees from previous instances. But we're combining subtrees in a judicious way guided by probabilistic modeling, rather than in a random way as in GP-style crossover.

33.6.2.1 Inferential MetaNodes

MetaNodes are an interesting and potentially powerful technique, but we don't believe that they, or any other algorithmic trick, is going to be the solution to the problem of learning hierarchical procedures. We believe that this is a cognitive science problem that probably isn't amenable to a purely computer science oriented solution. In other words, we suspect that the correct way to break a Combo tree down into hierarchical components depends on context, algorithms are of course required, but they're algorithms for relating a CT to its context rather

EFTA00624387

than pure CT-manipulation algorithms. Dimensional embedding is arguably a tool for capturing contextual relationships, but it's a very crude one.

Generally speaking, what we need to be learning are patterns of the form "A subtree meeting requirements X is often fit when linked to a subtree meeting requirements Y, when solving a problem of type Z". Here the context requirements Y will not pertain to absolute tree position but rather to abstract properties of a subtree.

The MetaNode approach as outlined above is a kind of halfway measure toward this goal, good because of its relative computational efficiency, but ultimately too limited in its power to deal with really hard hierarchical learning problems. The reason the MetaNode approach is crude is simply because it involves describing subtrees via points in an embedding space. We believe that the *correct* (but computationally expensive) approach is indeed to use MetaNodes - but with each MetaNode tagged, not with coordinates in an embedding space, but with a set of logical relationships describing the subtree that the MetaNode stands for. A candidate subtree's similarity to the MetaNode may then be determined by inference rather than by the simple computation of a distance between points in the embedding space. (And, note that we may have a hierarchy of MetaNodes, with small subtrees corresponding to MetaNodes, larger subtrees comprising networks of small subtrees also corresponding to MetaNodes, etc.)

The question then becomes which logical relationships one tries to look for, when characterizing a MetaNode. This may be partially domain-specific, in the sense that different properties will be more interesting when studying motor-control procedures than when studying cognitive procedures.

To intuitively understand the nature of this idea, let's consider some abstract but commonsense examples. Firstly, suppose one is learning procedures for serving a ball in tennis. Suppose all the successful procedures work by first throwing the ball up really high, then doing other stuff. The internal details of the different procedures for throwing the ball up really high may be wildly different. What we need is to learn the pattern

```
Implication
    Inheritance X ''throwing the ball up really high''
    ''X then Y'' is fit
```

Here X and Y are MetaNodes. But the question is how do we learn to break trees down into MetaNodes according to the formula "tree='X then Y' where X inherits from 'throwing the ball up really high."'?

Similarly, suppose one is learning procedures to do first-order inference. What we need is to learn a pattern such as:

```
Implication
    AND
        F involves grabbing pairs from the AtomTable
        G involves applying an inference rule to those each pair
        H involves putting the results back in the AtomTable
    ''F ( G (H))'' is fit
```

Here we need MetaNodes for F, G and H, but we need to characterize e.g. the MetaNode F by a relationship such as "involves grabbing pairs from the AtomTable."

Until we can characterize MetaNodes using abstract descriptors like this, one might argue we're just doing "statistical learning" rather than "general intelligence style" procedure learning. But to do this kind of abstraction intelligently seems to require some background knowledge about the domain.

EFTA00624388

In the "throwing the ball up really high" case the assignment of a descriptive relationship to a subtree involves looking, not at the internals of the subtree itself, but at the state of the world after the subtree has been executed.

In the "grabbing pairs from the AtomTable" case it's a bit simpler but still requires some kind of abstract model of what the subtree is doing, i.e. a model involving a logic expression such as "The output of F is a set S so that if P belongs to S then P is a set of two Atoms A1 and A2, and both A1 and A2 were produced via the getAtom operator."

How can this kind of abstraction be learned? It seems unlikely that abstractions like this will be found via evolutionary search over the space of all possible predicates describing program subtrees. Rather, they need to be found via probabilistic reasoning based on the terms combined in subtrees, put together with background knowledge about the domain in which the fitness function exists. In short, integrative cognition is required to learn hierarchically structured programs in a truly effective way, because the appropriate hierarchical breakdowns are contextual in nature, and to search for appropriate hierarchical breakdowns without using inference to take context into account, involves intractably large search spaces.

33.7 Fitness Function Estimation via Integrative Intelligence

If instance generation is very cheap and fitness evaluation is very expensive (as is the case in many applications of evolutionary learning in CogPrime), one can accelerate evolutionary learning via a "fitness function estimation" approach. Given a fitness function embodied in a predicate P, the goal is to learn a predicate Q so that:

1. Q is much cheaper than P to evaluate, and
2. There is a high-strength relationship:

```
Similarity Q P
```

or else

```
ContextLink C (Similarity Q P)
```

where C is a relevant context.

Given such a predicate Q, one could proceed to optimize P by ignoring evolutionary learning altogether and just repeatedly following the algorithm:

- Randomly generate N candidate solutions.
- Evaluate each of the N candidate solutions according to Q.
- Take the k$\ll$N solutions that satisfy Q best, and evaluate them according to P.

improved based on the new evaluations of P that are done. Of course, this would not be as good as incorporating fitness function estimation into an overall evolutionary learning framework.

Heavy utilization of fitness function estimation may be appropriate, for example, if the entities being evolved are schemata intended to control an agent's actions in a real or simulated environment. In this case the specification predicate P, in order to evaluate P(S), has to actually use the schema S to control the agent in the environment. So one may search for Q that do not involve any simulated environment, but are constrained to be relatively small predicates involving only cheap-to-evaluate terms (e.g. one may allow standard combinators, numbers,

EFTA00624389

strings, ConceptNodes, and predicates built up recursively from these). Then Q will be an abstract predictor of concrete environment success.

We have left open the all-important question of how to find the "specification approximating predicate" Q.

One approach is to use evolutionary learning. In this case, one has a population of predicates, which are candidates for Q. The fitness of each candidate Q is judged by how well it approximates P over the set of candidate solutions for P that have already been evaluated. If one uses evolutionary learning to evolve Qs, then one is learning a probabilistic model of the set of Qs, which tries to predict what sort of Qs will better solve the optimization problem of approximating P's behavior. Of course, using evolutionary learning for this purpose potentially initiates an infinite regress, but the regress can be stopped by, at some level, finding Qs using a non-evolutionary learning based technique such as genetic programming, or a simple evolutionary learning based technique like standard BOA programming.

Another approach to finding Q is to use inference based on background knowledge. Of course, this is complementary rather than contradictory to using evolutionary learning for finding Q. There may be information in the knowledge base that can be used to "analogize" regarding which Qs may match P. Indeed, this will generally be the case in the example given above, where P involves controlling actions in a simulated environment but Q does not.

An important point is that, if one uses a certain Q1 within fitness estimation, the evidence one gains by trying Q1 on numerous fitness cases may be utilized in future inferences regarding other Q2 that may serve the role of Q. So, once inference gets into the picture, the quality of fitness estimators may progressively improve via ongoing analogical inference based on the internal structures of the previously attempted fitness estimators.

EFTA00624390

Section V
Declarative Learning

Chapter 34
Probabilistic Logic Networks

Co-authored with Matthew Ikle

34.1 Introduction

Now we turn to CogPrime's methods for handling declarative knowledge – beginning with a series of chapters discussing the Probabilistic Logic Networks (PLN) [GMIH08] approach to uncertain logical reasoning, and then turning to chapters on pattern mining and concept creation. In this first of the chapters on PLN, we give a high-level overview, summarizing material given in the book *Probabilistic Logic Networks* [GMIH08] more compactly and in a somewhat differently-organized way. For a more thorough treatment of the concepts and motivations underlying PLN, the reader is encouraged to read [GMIH08].

PLN is a mathematical and software framework for uncertain inference, operative within the CogPrime software framework and intended to enable the combination of probabilistic truth values with general logical reasoning rules. Some of the key requirements underlying the development of PLN were the following:

- To enable uncertainty-savvy versions of all known varieties of logical reasoning, including for instance higher-order reasoning involving quantifiers, higher-order functions, and so forth
- To reduce to crisp "theorem prover" style behavior in the limiting case where uncertainty tends to zero
- To encompass inductive and abductive as well as deductive reasoning
- To agree with probability theory in those reasoning cases where probability theory, in its current state of development, provides solutions within reasonable calculational effort based on assumptions that are plausible in the context of real-world embodied software systems
- To gracefully incorporate heuristics not explicitly based on probability theory, in cases where probability theory, at its current state of development, does not provide adequate pragmatic solutions
- To provide "scalable" reasoning, in the sense of being able to carry out inferences involving billions of premises.
- To easily accept input from, and send input to, natural language processing software systems

In practice, PLN consists of

- a set of inference rules (e.g. deduction, Bayes rule, variable unification, modus ponens, etc.), each of which takes one or more logical relationships or terms (represented as CogPrime Atoms) as inputs, and produces others as outputs

EFTA00624394

- specific mathematical formulas for calculating the probability value of the conclusion of an inference rule based on the probability values of the premises plus (in some cases) appropriate background assumptions.

PLN also involves a particular approach to estimating the confidence values with which these probability values are held (weight of evidence, or second-order uncertainty). Finally, the implementation of PLN in software requires important choices regarding the structural representation of inference rules, and also regarding "inference control" – the strategies required to decide what inferences to do in what order, in each particular practical situation. Currently PLN is being utilized to enable an animated agent to achieve goals via combining actions in a game world. For example, it can figure out that to obtain an object located on top of a wall, it may want to build stairs leading from the floor to the top of the wall. Earlier PLN applications have involved simpler animated agent control problems, and also other domains, such as reasoning based on information extracted from biomedical text using a language parser.

For all its sophistication, however, PLN falls prey to the same key weakness as other logical inference systems: combinatorial explosion. In trying to find a logical chain of reasoning leading to a desired conclusion, or to evaluate the consequences of a given set of premises, PLN may need to explore an unwieldy number of possible combinations of the Atoms in CogPrime's memory. For PLN to be practical beyond relatively simple and constrained problems (and most definitely, for it to be useful for AGI at the human level or beyond), it must be coupled with a powerful method for "inference tree pruning" – for paring down the space of possible inferences that the PLN engine must evaluate as it goes about its business in pursuing a given goal in a certain context. Inference control will be addressed in Chapter 36.

34.2 A Simple Overview of PLN

The key elements of PLN are its rules and Formulas. In general,a PLN rule has

- Input: A tuple of Atoms (which must satisfy certain criteria, specific to the Rule)
- Output: A tuple of Atoms

Actually, in nearly all cases, the output is a single Atom; and the input is a single Atom or a pair of Atoms.

The prototypical example is the DeductionRule. Its input must look like

```
X_Link A B
X_Link B C
```

And its output then looks like

```
X_Link A C
```

Here, X_Link may be either InheritanceLink, SubsetLink, ImplicationLink or ExtensionalImplicationLink.

A PLN formula goes along with a PLN rule, and tells the uncertain truth value of the output, based on the uncertain truth value of the input. For example, if we have

```
X_Link A B <sAB>
X_Link B C <sBC>
```

then the standard PLN deduction formula tells us

EFTA00624395

```
X_Link A C <sAC>
```

with

$$s_{AC} = s_{AB}s_{BC} + \frac{(1 - s_{AB})(s_C - s_B s_{BC})}{1 - s_B}$$

where e.g. s_A denote the strength of the truth value of node A.

In this example, the uncertain truth value of each Atom is given as a single "strength" number. In general, uncertain truth values in PLN may take multiple forms, such as

- Single strength values like .8, which may indicate probability or fuzzy truth value, depending on the Atom type
- (strength, confidence) pairs like (.8, .4)
- (strength, count) pairs like (.8, 15)
- indefinite probabilities like (.6,. .9, .95) which indicate credible intervals of probabilities

34.2.1 Forward and Backward Chaining

Typical patterns of usage of PLN are forward-chaining and backward-chaining inference.

Forward chaining basically means:

1. Given a pool (a list) of Atoms of interest
2. One applies PLN rules to these Atoms, to generate new Atoms, hopefully also of interest
3. Adding these new Atoms to the pool, one returns to Step 1

EXAMPLE: "People are animals" and "animals breathe" are in the pool of Atoms. These are combined by the Deduction rule to form the conclusion "people breathe".

Backward chaining falls into two cases. First:

- "'Truth value query."' Given a target Atom whose truth value is not known (or is too uncertainly known), plus a pool of Atoms, find a way to estimate the truth value of the target Atom, via combining the Atoms in the pool using the inference Rules

EXAMPLE: The target is "do people breathe?" (InheritanceLink people breathe). The truth value of the target is estimated via doing the inference "People are animals, animals breathe, therefore people breathe."

Second:

- "'Variable fulfillment query"'. Given a target Link (Atoms may be Nodes or Links) with one or more VariableAtoms among its targets, figure out what Atoms may be put in place of these VariableAtoms, so as to give the target Link a high strength* confidence (i.e. a "high truth value").

EXAMPLE: The target is "what breathes?", i.e. "InheritanceLink $X breathe"... Direct lookup into the Atomspace reveals the Atom "InheritanceLink animal breathe", indicating that the slot $X may be filled by "animal". Inference reveals that "Inheritance people breathe" , so that the slot $X may also be filled by "people".

EFTA00624396

EXAMPLE: the target is "what breathes and adds", ie "(InheritanceLink $X breathe) AND (InheritanceLink $X add)". Inference reveals that the slot $X may be filled by "people" but not "cats" or "computers."

Common-sense inference may involve a combination of backward chaining and forward chaining.

The hardest part of inference is "inference control" – that is, knowing which among the many possible inference steps to take, in order to obtain the desired information (in backward chaining) or to obtain interesting new information (in forward chaining). In an Atomspace with a large number of (often quite uncertain) Atoms, there are many, many possibilities and powerful heuristics are needed to choose between them. The best guide to inference control is some sort of induction based on the system's past history of which inferences have been useful. But of course, a young system doesn't have much history to go on. And relying on indirectly relevant history is, itself, an inference problem – which can be solved best by a system with some history to draw on!

34.3 First Order Probabilistic Logic Networks

We now review the essentials of PLN in a more formal way. PLN is divided into first-order and higher-order sub-theories (FOPLN and HOPLN). These terms are used in a nonstandard way drawn conceptually from NARS [Wan06]. We develop FOPLN first, and then derive HOPLN therefrom.

FOPLN is a term logic, involving terms and relationships (links) between terms. It is an uncertain logic, in the sense that both terms and relationships are associated with truth value objects, which may come in multiple varieties ranging from single numbers to complex structures like indefinite probabilities. Terms may be either elementary observations, or abstract tokens drawn from a token-set T.

34.3.1 Core FOPLN Relationships

"Core FOPLN" involves relationships drawn from the set: negation; Inheritance and probabilistic conjunction and disjunction; Member and fuzzy conjunction and disjunction. Elementary observations can have only Member links, while token terms can have any kinds of links. PLN makes clear distinctions, via link type semantics, between probabilistic relationships and fuzzy set relationships. Member semantics are usually fuzzy relationships (though they can also be crisp), whereas Inheritance relationships are probabilistic, and there are rules governing the interoperation of the two types.

Suppose a virtual agent makes an elementary VisualObservation o of a creature named Fluffy. The agent might classify o as belonging, with degree 0.9, to the fuzzy set of furry objects. The agent might also classify o as belonging with degree 0.8 to the fuzzy set of animals. The agent could then build the following links in its memory:

Member o furry $< 0.9 >$
Member o animals $< 0.8 >$

EFTA00624397

The agent may later wish to refine its knowledge, by combining these MemberLinks. Using the minimum fuzzy conjunction operator, the agent would conclude:

fuzzyAND $< 0.8 >$
  Member o furry
  Member o animals

meaning that the observation o is a visual observation of a fairly furry, animal object.

The semantics of (extensional) Inheritance are quite different from, though related to, those of the MemberLink. ExtensionalInheritance represents a purely conditional probabilistic subset relationship and is represented through the Subset relationship. If A is Fluffy and B is the set of cat, then the statement

Subset $< 0.9 >$
  A
  B

means that

$$P(x \text{ is in the set } B | x \text{ is in the set } A) = 0.9.$$

34.3.2 PLN Truth Values

PLN is equipped with a variety of different types of truth-value types. In order of increasing information about the full probability distribution, they are:

- strength truth-values, which consist of single numbers; e.g., $< s >$ or $< .8 >$. Usually strength values denote probabilities but this is not always the case.
- SimpleTruthValues, consisting of pairs of numbers. These pairs come in two forms: $< s,\ w >$, where s is a strength and w is a "weight of evidence" and $< s,\ N >$, where N is a "count." "Weight of evidence" is a qualitative measure of belief, while "count" is a quantitative measure of accumulated evidence.
- IndefiniteTruthValues, which quantify truth-values in terms of an interval $[L, U]$, a credibility level b, and an integer k (called the lookahead). IndefiniteTruthValues quantify the idea that after k more observations there is a probability b that the conclusion of the inference will appear to lie in $[L, U]$.
- DistributionalTruthValues, which are discretized approximations to entire probability distributions.

34.3.3 Auxiliary FOPLN Relationships

Beyond the core FOPLN relationships, FOPLN involves additional relationship types of two varieties. There are simple ones like Similarity, defined by

$$Similarity\ A\ B$$

EFTA00624398

We say a relationship R is simple if the truth value of $R\ A\ B$ can be calculated solely in terms of the truth values of core FOPLN relationships between A and B. There are also complex "auxiliary" relationships like IntensionalInheritance, which as discussed in depth in the Appendix **??**, measures the extensional inheritance between the set of properties or patterns associated with one term and the corresponding set associated with another.

Returning to our example, the agent may observe that two properties of cats are that they are furry and purr. Since the Fluffy is also a furry animal, the agent might then obtain, for example

IntensionalInheritance $< 0.5 >$
  Fluffy
  cat

meaning that the Fluffy shares about 50% of the properties of cat. Building upon this relationship even further, PLN also has a mixed intensional/extensional Inheritance relationship which is defined simply as the disjunction of the Subset and IntensionalInheritance relationships.

As this example illustrates, for a complex auxiliary relationship R, the truth value of $R\ A\ B$ is defined in terms of the truth values of a number of different FOPLN relationships among different terms (others than A and B), specified by a certain mathematical formula.

34.3.4 PLN Rules and Formulas

A distinction is made in PLN between rules and formulas. PLN logical inferences take the form of "syllogistic rules," which give patterns for combining statements with matching terms. Examples of PLN rules include, but are not limited to,

- deduction $((A \to B) \wedge (B \to C) \Rightarrow (A \to C))$,
- induction $((A \to B) \wedge (A \to C) \Rightarrow (B \to C))$,
- abduction $((A \to C) \wedge (B \to C) \Rightarrow (A \to C))$,
- revision, which merges two versions of the same logical relationship that have different truth values
- inversion $((A \to B) \Rightarrow (B \to A))$.

The basic schematic of the first four of these rules is shown in Figure 34.1. We can see that the first three rules represent the natural ways of doing inference on three interrelated terms. We can also see that induction and abduction can be obtained from the combination of deduction and inversion, a fact utilized in PLN's truth value formulas.

Related to each rule is a formula which calculates the truth value resulting from application of the rule. As an example, suppose s_A, s_B, s_C, s_{AB}, and s_{BC} represent the truth values for the terms A, B, C, as well the truth values of the relationships $A \to B$ and $B \to C$, respectively. Then, under suitable conditions imposed upon these input truth values, the formula for the deduction rule is given by:

$$s_{AC} = s_{AB}s_{BC} + \frac{(1 - s_{AB})\,(s_C - s_B s_{BC})}{1 - s_B},$$

EFTA00624399

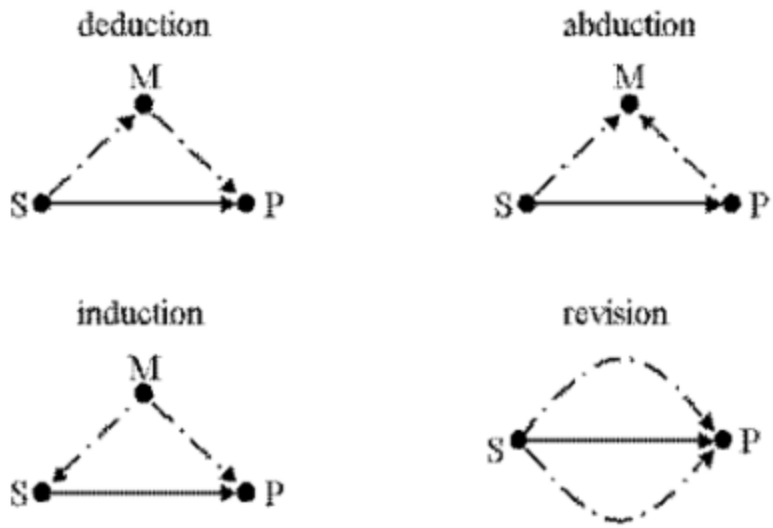


Fig. 34.1: The four most basic first-order PLN inference rules

where s_{AC} represents the truth value of the relationship $A \to C$. This formula is directly derived from probability theory given the assumption that $A \to B$ and $B \to C$ are independent.

For inferences involving solely fuzzy operators, the default version of PLN uses standard fuzzy logic with min/max truth value formulas (though alternatives may also be employed consistently with the overall PLN framework). Finally, the semantics of combining fuzzy and probabilistic operators is hinted at in [GMIH08] but addressed more rigorously in [GL10], which gives a precise semantics for constructs of the form

Inheritance A B

where A and B are characterized by relationships of the form *Member C A*, *Member D B*, etc. It is easy to see that, in the crisp case, where all MemberLinks and InheritanceLinks have strength 0 or 1, FOPLN reduces to standard propositional logic. Where inheritance is crisp but membership isn't, FOPLN reduces to higher-order fuzzy logic (including fuzzy statements about terms or fuzzy statements, etc.).

34.3.5 Inference Trails

Inference trails are a mechanism used in some implementations of PLN, borrowed from the NARS inference engine [Wan06]. In this approach, each Atom contains a Trail structure, which keeps a record of which Atoms were used in deriving the given Atom's TruthValue. In its simplest form, the Trail can just be a list of Atoms. The total set of Atoms involved in a given Trail, in

EFTA00624400

principle, could be very large; but one can in practice cap Trail size at 50 or some other similar number. In a more sophisticated version, one can record the rules used with the Atoms in the Trail as well, allowing recapitulation of the whole inference history producing an Atom's truth value. If the PLN MindAgents store all the inferences they do in some global inference history structure, then Trails are obviated, as the information in the Trail can be found via consulting this history structure.

The purpose of keeping inference trails is to avoid errors due to double-counting of evidence. If links L_1 and L_2 are both derived largely based on link L_0, and L_1 and L_2 both lead to L_4 as a consequence – do we want to count this as two separate, independent pieces of evidence about L_4? Not really, because most of the information involved comes from the single Atom L_0 anyway. If all the Atoms maintain Trails then this sort of overlapping evidence can be identified easily; otherwise it will be opaque to the reasoning system.

While Trails can be a useful tool, there is reason to believe they're not strictly necessary. If one just keeps doing probabilistic inference iteratively without using Trails, eventually the dependencies and overlapping evidence bases will tend to be accounted for, much as in a loopy Bayes net. The key question then comes down to: how long is "eventually" and can the reasoning system afford to wait? A reasonable strategy seems to be

- Use Trails for high-STI Atoms that are being reasoned about intensively, to minimize the amount of error
- For lower-STI Atoms that are being reasoned on more casually in the background, allow the double-counting to exist in the short term, figuring it will eventually "come out in the wash" so it's not worth spending precious compute resources to more rigorously avoid it in the short term

34.4 Higher-Order PLN

Higher-order PLN (HOPLN) is defined as the subset of PLN that applies to predicates (considered as functions mapping arguments into truth values). It includes mechanisms for dealing with variable-bearing expressions and higher-order functions.

A predicate, in PLN, is a special kind of term that embodies a function mapping terms or relationships into truth-values. HOPLN contains several relationships that act upon predicates including Evaluation, Implication, and several types of quantifiers. The relationships can involve constant terms, variables, or a mixture.

The Evaluation relationship, for example, evaluates a predicate on an input term. An agent can thus create a relationship of the form

Evaluation
    near
    (Bob's house, Fluffy)

or, as an example involving variables,

Evaluation
    near
    (X, Fluffy)

EFTA00624401

The Implication relationship is a particularly simple kind of HOPLN relationship in that it behaves very much like FOPLN relationships, via substitution of predicates in place of simple terms. Since our agent knows, for example,

Implication
    is_Fluffy
    AND is_furry purrs

and

Implication
    AND is_furry purrs
    is_cat

the agent could then use the deduction rule to conclude

Implication is_Fluffy is_cat

PLN supports a variety of quantifiers, including traditional crisp and fuzzy quantifiers, plus the AverageQuantifier defined so that the truth value of

$$AverageQuantifier\ X\ F(X)$$

is a weighted average of $F(X)$ over all relevant inputs X. AverageQuantifier is used implicitly in PLN to handle logical relationships between predicates, so that e.g. the conclusion of the above deduction is implicitly interpreted as

AverageQuantifier X
    Implication
        Evaluation is_Fluffy X
        Evaluation is_cat X

We can now connect PLN with the SRAM model (defined in Chapter 7 of Part 1).

Suppose for instance that the agent observes Fluffy from across the room, and that it has previously learned a Fetch procedure that tells it how to obtain an entity once it sees that entity. Then, if the agent has the goal of finding a cat, and it has concluded based on the above deduction that Fluffy is indeed a cat (since it is observed to be furry and purr), the cognitive schematic (knowledge of the form *Context* & *Procedure* → *Goal* as explained in Chapter 8 of Part 1) may suggest it to execute the Fetch procedure.

34.4.1 Reducing HOPLN to FOPLN

In [GMIH08] it is shown that in principle, over any finite observation set, HOPLN reduces to FOPLN. The key ideas of this reduction are the elimination of variables via use of higher-order functions, and the use of the set-theoretic definition of function embodied in the *SatisfyingSet* operator to map function-argument relationships into set-member relationships.

As an example, consider the Implication link. In HOPLN, where X is a variable

EFTA00624402

Implication
    R_1 A X
    R_2 B X

may be reduced to

Inheritance
    SatisfyingSet(R_1 A X)
    SatisfyingSet(R_2 B X)

where e.g. *SatisfyingSet*(R_1 A X) is the fuzzy set of all X satisfying the relationship $R_1(A, X)$.

Furthermore in Appendix **??**, we show how experience-based possible world semantics can be used to reduce PLN's existential and universal quantifiers to standard higher order PLN relationships using AverageQuantifier relationships. This completes the reduction of HOPLN to FOPLN in the SRAM context.

One may then wonder why it makes sense to think about HOPLN at all. The answer is that it provides compact expression of a specific subset of FOPLN expressions, which is useful in cases where agents have limited memory and these particular expressions provide agents practical value (it biases the agent's reasoning ability to perform just as well as in first or higher orders).

34.5 Predictive Implication and Attraction

This section briefly reviews the notions of predictive implication and predictive attraction, which are critical to many aspects of CogPrime dynamics including goal-oriented behavior.

Define

```
Attraction A B <s>
```

as P(B|A) - P(B|¬A) = s, or in node and link terms

```
s = (Inheritance A B).s - (Inheritance ¬A B).s
```

For instance

```
(Attraction fat pig).s =
(Inheritance fat pig).s - (Inheritance ¬fat pig).s
```

Relatedly, in the temporal domain, we have the link type PredictiveImplication, where

```
PredictiveImplication A B <s>
```

roughly means that s is the probability that

```
Implication A B <s>
```

holds and also A occurs before B. More sophisticated versions of PredictiveImplication come along with more specific information regarding the time lag between A and B: for instance a time interval T in which the lag must lie, or a probability distribution governing the lag between the two events.

We may then introduce

```
PredictiveAttraction A B <s>
```

to mean

```
s = (PredictiveImplication A B).s - (PredictiveImplication ¬A B).s
```

For instance

EFTA00624403

```
(PredictiveAttraction kiss_Ben be_happy).s =
  (PredictiveImplication kiss_Ben be_happy).s
  - (PredictiveImplication ¬kiss_Ben be_happy).s
```

This is what really matters in terms of determining whether kissing Ben is worth doing in pursuit of the goal of being happy, not just how likely it is to be happy if you kiss Ben, but how differentially likely it is to be happy if you kiss Ben.

Along with predictive implication and attraction, sequential logical operations are important, represented by operators such as SequentialAND, SimultaneousAND and SimultaneousOR. For instance:

```
PredictiveAttraction
    SequentialAND
        Teacher says 'fetch'
        I get the ball
        I bring the ball to the teacher
    I get a reward
```

combines SequentialAND and PredictiveAttraction. In this manner, an arbitrarily complex system of serial and parallel temporal events can be constructed.

34.6 Confidence Decay

PLN is all about uncertain truth values, yet there is an important kind of uncertainty it doesn't handle explicitly and completely in its standard truth value representations: the decay of information with time.

PLN does have an elegant mechanism for handling this: in the <s,d> formalism for truth values, strength s may remain untouched by time (except as new evidence specifically corrects it), but d may decay over time. So, our confidence in our old observations decreases with time. In the indefinite probability formalism, what this means is that old truth value intervals get wider, but retain the same mean as they had back in the good old days.

But the tricky question is: How fast does this decay happen?

This can be highly context-dependent.

For instance, 20 years ago we learned that the electric guitar is the most popular instrument in the world, and also that there are more bacteria than humans on Earth. The former fact is no longer true (keyboard synthesizers have outpaced electric guitars), but the latter is. And, if you'd asked us 20 years ago which fact would be more likely to become obsolete, we would have answered the former - because we knew particulars of technology would likely change far faster than basic facts of earthly ecology.

On a smaller scale, it seems that estimating confidence decay rates for different sorts of knowledge in different contexts is a tractable data mining problem, that can be solved via the system keeping a record of the observed truth values of a random sampling of Atoms as they change over time. (Operationally, this record may be maintained in parallel with the SystemActivityTable and other tables maintained for purposes of effort estimation, attention allocation and credit assignment.) If the truth values of a certain sort of Atom in a certain context change a lot, then the confidence decay rate for Atoms of that sort should be increased.

This can be quantified nicely using the indefinite probabilities framework.

For instance, we can calculate, for a given sort of Atom in a given context, separate b-level credible intervals for the L and U components of the Atom's truth value at time t-r, centered

EFTA00624404

about the corresponding values at time t. (This would be computed by averaging over all t values in the relevant past, where the relevant past is defined as some particular multiple of r; and over a number of Atoms of the same sort in the same context.)

Since historically-estimated credible-intervals won't be available for every exact value of r, interpolation will have to be used between the values calculated for specific values of r.

Also, while separate intervals for L and U would be kept for maximum accuracy, for reasons of pragmatic memory efficiency one might want to maintain only a single number x, considered as the radius of the confidence interval about both L and U. This could be obtained by averaging together the empirically obtained intervals for L and U.

Then, when updating an Atom's truth value based on a new observation, one performs a revision of the old TV with the new, but before doing so, one first **widens** the interval for the old one by the amounts indicated by the above-mentioned credible intervals.

For instance, if one gets a new observation about A with TV (L_{new}, U_{new}), and the prior TV of A, namely (L_{old}, U_{old}), is 2 weeks old, then one may calculate that L_{old} should really be considered as:

```
$(L_{old} - x, L_{old}+x)$
```

and U_old should really be considered as:

```
$(U_{old} - x, U_{old} + x)$
```

so that (L_new, U_new) should actually be revised with:

```
$(L_{old} - x, U_{old} + x)$
```

to get the total:

```
(L,U)
```

for the Atom after the new observation.

Note that we have referred fuzzily to "sort of Atom" rather than "type of Atom" in the above. This is because Atom type is not really the right level of specificity to be looking at. Rather - as in the guitar vs. bacteria example above - confidence decay rates may depend on semantic categories, not just syntactic (Atom type) categories. To give another example, confidence in the location of a person should decay more quickly than confidence in the location of a building. So ultimately confidence decay needs to be managed by a pool of learned predicates, which are applied periodically. These predicates are mainly to be learned by data mining, but inference may also play a role in some cases.

The ConfidenceDecay MindAgent must take care of applying the confidence-decaying predicates to the Atoms in the AtomTable, periodically.

The ConfidenceDecayUpdater MindAgent must take care of:

- forming new confidence-decaying predicates via data mining, and then revising them with the existing relevant confidence-decaying predicates.
- flagging confidence-decaying predicates which pertain to important Atoms but are unconfident, by giving them STICurrency, so as to make it likely that they will be visited by inference.

34.6.1 An Example

As an example of the above issues, consider that the confidence decay of:

EFTA00624405

```
Inh Ari male
```

should be low whereas that of:

```
Inh Ari tired
```

should be higher, because we know that for humans, being male tends to be a more permanent condition than being tired.

This suggests that concepts should have context-dependent decay rates, e.g. in the context of humans, the default decay rate of maleness is low whereas the default decay rate of tired-ness is high.

However, these defaults can be overridden. For instance, one can say "As he passed through his 80's, Grandpa just got tired, and eventually he died." This kind of tiredness, even in the context of humans, does not have a rapid decay rate. This example indicates why the confidence decay rate of a particular Atom needs to be able to override the default.

In terms of implementation, one mechanism to achieve the above example would be as follows. One could incorporate an *interval* confidence decay rate as an optional component of a truth value. As noted above one can keep two separate intervals for the L and U bounds; or to simplify things one can keep a single interval and apply it to both bounds separately.

Then, e.g., to define the decay rate for tiredness among humans, we could say:

```
ImplicationLink_HOJ
   InheritanceLink $X human
   InheritanceLink $X tired <confidenceDecay = [0,.1]>
```

or else (preferably):

```
ContextLink
  human
  InheritanceLink $X tired <confidenceDecay = [0,.1]>
```

Similarly, regarding maleness we could say:

```
ContextLink
  human
  Inh $X male <confidenceDecay = [0,.00001]>
```

Then one way to express the violation of the default in the case of grandpa's tiredness would be:

```
InheritanceLink grandpa tired <confidenceDecay = [0,.001]>
```

(Another way to handle the violation from default, of course, would be to create a separate Atom:

```
tired_from_old_age
```

and consider this as a separate sense of "tired" from the normal one, with its own confidence decay setting.)

In this example we see that, when a new Atom is created (e.g. *InheritanceLink Ari tired*), it needs to be assigned a confidence decay rate via inference based on relations such as the ones given above (this might be done e.g. by placing it on the queue for immediate attention by the ConfidenceDecayUpdater MindAgent). And periodically its confidence decay rate could be updated based on ongoing inferences (in case relevant abstract knowledge about confidence decay rates changes). Making this sort of inference reasonably efficient might require creating a special index containing abstract relationships that tell you something about confidence decay adjustment, such as the examples given above.

EFTA00624406

34.7 Why is PLN a Good Idea?

We have explored the intersection of the family of conceptual and formal structures that is PLN, with a specific formal model of intelligent agents (SRAM) and its extension using the cognitive schematic. The result is a simple and explicit formulation of PLN as a system by which an agent can manipulate tokens in its memory, thus represent observed and conjectured relationships (between its observations and between other relationships), in a way that assists it in choosing actions according to the cognitive schematic.

We have *not*, however, rigorously answered the question: What is the contribution of PLN to intelligence, within the formal agents framework introduced above? This is a quite subtle question, to which we can currently offer only an intuitive answer, not a rigorous one.

Firstly, there is the question of whether probability theory is really the best way to manage uncertainty, in a practical context. Theoretical results like those of Cox [Cox61] and de Finetti [dF37] demonstrate that probability theory is the optimal way to handle uncertainty, if one makes certain reasonable assumptions. However, these reasonable assumptions don't actually apply to real-world intelligent systems, which must operate with relatively severe computational resource constraints. For example, one of Cox's axioms dictates that a reasoning system must assign the same truth value to a statement, regardless of the route it uses to derive the statement. This is a nice idealization, but it can't be expected of any real-world, finite-resources reasoning system dealing with a complex environment. So an open question exists, as to whether probability theory is actually the best way for practical AGI systems to manage uncertainty. Most contemporary AI researchers assume the answer is yes, and probabilistic AI has achieved increasing popularity in recent years. However, there are also significant voices of dissent, such as Pei Wang [Wan06] in the AGI community, and many within the fuzzy logic community.

PLN is not strictly probabilistic, in the sense that it combines formulas derived rigorously from probability theory with others that are frankly heuristic in nature. PLN was created in a spirit of open-mindedness regarding whether probability theory is actually the optimal approach to reasoning under uncertainty using limited resources, versus merely an approximation to the optimal approach in this case. Future versions of PLN might become either more or less strictly probabilistic, depending on theoretical and practical advances.

Next, aside from the question of the practical value of probability theory, there is the question of whether PLN in particular is a good approach to carrying out significant parts of what an AGI system needs to do, to achieve human-like goals in environments similar to everyday human environments.

Within a cognitive architecture where explicit utilization the cognitive schematic (Context & Procedure $\rightarrow$ Goal) is useful, clearly PLN is useful if it works reasonably well – so this question partially reduces to: what are the environments in which agents relying on the cognitive schematic are intelligent according to formal intelligent measures like those defined in Chapter 7 of Part 1. And then there is the possibility that some uncertain reasoning formalism besides PLN could be even more useful in the context of the cognitive schematic.

In particular, the question arises: What are the unique, peculiar aspects of PLN that makes it more useful in the context of the cognitive schematic, than some other, more straightforward approach to probabilistic inference? Actually there are multiple such aspects that we believe make it particularly useful. One is the indefinite probability approach to truth values, which we believe is more robust for AGI than known alternatives. Another is the clean reduction of higher order logic (as defined in PLN) to first-order logic (as defined in PLN), and the utilization

EFTA00624407

of term logic instead of predicate logic wherever possible – these aspects make PLN inferences relatively simple in most cases where, according to human common sense, they should be simple.

A relatively subtle issue in this regard has to do with PLN intension. The cognitive schematic is formulated in terms of PredictiveExtensionalImplication (or any other equivalent way like PredictiveExtensionalAttraction), which means that intensional PLN links are not required for handling it. The hypothesis of the usefulness of intensional PLN links embodies a subtle assumption about the nature of the environments that intelligent agents are operating in. As discussed in [Goe06] it requires an assumption related to Peirce's philosophical axiom of the "tendency to take habits," which posits that in the real world, entities possessing some similar patterns have a probabilistically surprising tendency to have more similar patterns.

Reflecting on these various theoretical subtleties and uncertainties, one may get the feeling that the justification for applying PLN in practice is quite insecure! However, it must be noted that no other formalism in AI has significantly better foundation, at present. Every AI method involves certain heuristic assumptions, and the applicability of these assumptions in real life is nearly always a matter of informal judgment and copious debate. Even a very rigorous technique like a crisp logic formalism or support vector machines for classification, requires non-rigorous heuristic assumptions to be applied to the real world (how does sensation and actuation get translated into logic formulas, or SVM feature vectors)? It would be great if it were possible to use rigorous mathematical theory to derive an AGI design, but that's not the case right now, and the development of this sort of mathematical theory seems quite a long way off. So for now, we must proceed via a combination of mathematics, practice and intuition.

In terms of demonstrated practical utility, PLN has not yet confronted any really ambitious AGI-type problems, but it has shown itself capable of simple practical problem-solving in areas such as virtual agent control and natural language based scientific reasoning [GIGH08]. The current PLN implementation within CogPrime can be used to learn to play fetch or tag, draw analogies based on observed objects, or figure out how to carry out tasks like finding a cat. We expect that further practical applications, as well as very ambitious AGI development, can be successfully undertaken with PLN without a theoretical understanding of exactly what are the properties of the environments and goals involved that allow PLN to be effective. However, we expect that a deeper theoretical understanding may enable various aspects of PLN to be adjusted in a more effective manner.

EFTA00624408

Chapter 35
Spatiotemporal Inference

35.1 Introduction

Most of the problems and situations humans confront every day involve space and time explicitly and centrally. Thus, any AGI system aspiring to humanlike general intelligence must have some reasonably efficient and general capability to solve spatiotemporal problems. Regarding how this capability might get into the system, there is a spectrum of possibilities, ranging from rigid hard-coding to tabula rasa experiential learning. Our bias in this regard is that it's probably sensible to somehow "wire into" CogPrime some knowledge regarding space and time – these being, after all, very basic categories for any embodied mind confronting the world.

It's arguable whether the explicit insertion of prior knowledge about spacetime is *necessary* for achieving humanlike AGI using feasible resources. As an argument *against* the necessity of this sort of prior knowledge, Ben Kuipers and his colleagues [SMK12] have shown that an AI system can learn via experience that its perceptual stream comes from a world with three, rather than two or four dimensions. There is a long way from learning the number of dimensions in the world to learning the full scope of practical knowledge needed for effectively reasoning about the world – but it does seem plausible, from their work, that a broad variety of spatiotemporal knowledge could be inferred from raw experiential data. On the other hand, it also seems clear that the human brain does not do it this way, and that a rich fund of spatiotemporal knowledge is "hard-coded" into the brain by evolution – often in ways so low-level that we take them for granted, e.g. the way some motion detection neurons fire in the physical direction of motion, and the way somatosensory cortex presents a distorted map of the body's surface. On a psychological level, it is known that some fundamental intuition for space and time is hard-coded into the human infant's brain [Joh05]. So while we consider the learning of basic spatiotemporal knowledge from raw experience a worthy research direction, and fully compatible with the CogPrime vision; yet for our main current research, we have chosen to hard-wire some basic spatiotemporal knowledge.

If one does wish to hard-wire some basic spatiotemporal knowledge into one's AI system, multiple alternate or complementary methodologies may be used to achieve this, including spatiotemporal logical inference, internal simulation, or techniques like recurrent neural nets whose dynamics defy simple analytic explanation. Though our focus in this chapter is on inference, we must emphasize that inference, even very broadly conceived, is not the only way for an intelligent agent to solve spatiotemporal problems occurring in its life. For instance, if the agent has a detailed map of its environment, it may be able to answer some spatiotemporal questions by

EFTA00624410

directly retrieving information from the map. Or, logical inference may be substituted or augmented by (implicitly or explicitly) building a model that satisfies the initial knowledge – either abstractly or via incorporating "visualization" connected to sensory memory – and then interpret new knowledge over that model instead of inferring it. The latter is one way to interpret what DeSTIN and other CSDLNs do; indeed, DeSTIN's perceptual hierarchy is often referred to as a "state inference hierarchy." Any CSDLN contains biasing toward the commonsense structure of space and time, in its spatiotemporal hierarchical structure. It seems plausible that the human mind uses a combination of multiple methods for spatiotemporal understanding, just as we intend CogPrime to do.

In this chapter we focus on spatiotemporal logical inference, addressing the problem of creating a spatiotemporal logic adequate for use within an AGI system that confronts the same sort of real-world problems that humans typically do. The idea is not to fully specify the system's understanding of space and time in advance, but rather to provide some basic spatiotemporal logic rules, with parameters to be adjusted based on experience, and the opportunity for augmenting the logic over time with experientially-acquired rules. Most of the ideas in this chapter are reviewed in more detail, with more explanation, in the book *Real World Reasoning* [GGC+11]; this chapter represent a concise summary, compiled with the AGI context specifically in mind.

A great deal of excellent work has already been done in the areas of spatial, temporal and spatiotemporal reasoning; however, this work does not quite provide an adequate foundation for a logic-incorporating AGI system to do spatiotemporal reasoning, because it does not adequately incorporate uncertainty. Our focus here is to extend existing spatiotemporal calculi to appropriately encompass uncertainty, which we argue is sufficient to transform them into an AGI-ready spatiotemporal reasoning framework. We also find that a simple extension of the standard PLN uncertainty representations, inspired by $\mathcal{P}(\mathcal{Z})$-logic [Yan10], allows more elegant expression of probabilistic fuzzy predicates such as arise naturally in spatiotemporal logic.

In the final section of the chapter, we discuss the problem of **planning**, which has been considered extensively in the AI literature. We describe an approach to planning that incorporates PLN inference using spatiotemporal logic, along with MOSES as a search method, and some record-keeping methods inspired by traditional AI planning algorithms.

35.2 Related Work on Spatio-temporal Calculi

We now review several calculi that have previously been introduced for representing and reasoning about space, time and space-time combined.

Spatial Calculi

Calculi dealing with space usually model three types of relationships between spatial regions: topological, directional and metric.

The most popular calculus dealing with topology is the Region Connection Calculus (RCC) [RCC93], relying on a base relationship `C` (for `Connected`) and building up other relationships from it, like `P` (for `PartOf`), or `O` (for `Overlap`). For instance $\texttt{P}(X, Y)$, meaning X is a part of Y, can be defined using `C` as follows

EFTA00624411

$$\mathtt{P}(X,Y) \text{ iff } \forall Z \in \mathcal{U}, \mathtt{C}(Z,X) \implies \mathtt{C}(Z,Y) \tag{35.1}$$

Where $\mathcal{U}$ is the universe of regions. RCC-8 models eight base relationships, see Figure 35.1. And

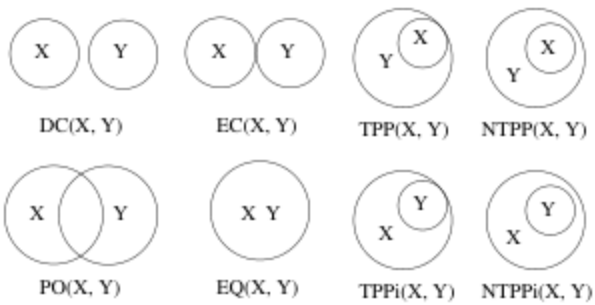


Fig. 35.1: The eight base relationships of RCC-8

it is possible, using the notion of convexity, to model more relationships such as inside, partially inside and outside, see Figure 35.2. For instance RCC-23 [Ben94] is an extension of RCC-8 using relationships based on the notion of convexity. The 9-intersection calculus [Win95, Kur09] is

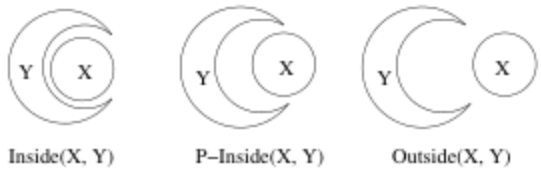


Fig. 35.2: Additional relationships using convexity

another calculus for reasoning on topological relationships, but handling relationships between heterogeneous objects, points, lines, surfaces.

Regarding reasoning about direction, the Cardinal Direction Calculus [GE01, ZLLY08] considers directional relationships between regions, to express propositions such as "region A is to the north of region B".

And finally regarding metric reasoning, spatial reasoning involving qualitative distance (such as close, medium, far) and direction combined is considered in [CFH97].

Some work has also been done to extend and combine these various calculi, such as combining RCC-8 and the Cardinal Direction Calculus [LLR09], or using size [GR00] or shape [Coh95] information in RCC.

Temporal Calculi

The best known temporal calculus is Allen's Interval Algebra [All83], which considers 13 relationships over time intervals, such as `Before`, `During`, `Overlap`, `Meet`, etc. For instance one

EFTA00624412

can express that digestion occurs after or right after eating by

$$\texttt{Before}(Eat, Digest) \vee \texttt{Meet}(Eat, Digest)$$

equivalently denoted $Eat\{\texttt{Before}, \texttt{Meet}\}Digest$. There also exists a generalization of Allen's Interval Algebra that works on semi-intervals [FF92], that are intervals with possibly undefined start or end.

There are modal temporal logics such as LTL and CTL, mostly used to check temporal constraints on concurrent systems such as deadlock or fairness using Model Checking [Mai00].

Calculi with Space and Time Combined

There exist calculi combining space and time, first of all those obtained by "temporizing" spatial calculus, that is tagging spatial predicates with timestamps or time intervals. For instance STCC (for Spatio-temporal Constraint Calculus) [GN02] is basically RCC-8 combined with Allen's Algebra. With STCC one can express spatiotemporal propositions such as

$$\texttt{Meet}(\texttt{DC}(Finger, Key), \texttt{EC}(Finger, Key))$$

which means that the interval during which the finger is away from the key meets the interval during which the finger is against the key.

Another way to combine space and time is by modeling motion; e.g. the Qualitative Trajectory Calculus (QTC) [WKB05] can be used to express whether 2 objects are going forward/backward or left/right relative to each other.

Uncertainty in Spatio-temporal Calculi

In many situations it is worthwhile or even necessary to consider non-crisp extensions of these calculi. For example it is not obvious how one should consider in practice whether two regions are connected or disconnected. A desk against the wall would probably be considered connected to it even if there is a small gap between the wall and the desk. Or if A is not entirely part of B it may still be valuable to consider to which extent it is, rather than formally rejecting $\texttt{PartOf}(A, B)$. There are several ways to deal with such phenomena; one way is to consider probabilistic or fuzzy extensions of spatiotemporal calculi.

For instance in [SDCCK08b, SDCCK08a] the RCC relationship `C` (for `Connected`) is replaced by a fuzzy predicate representing closeness between regions and all other relationships based on it are extended accordingly. So e.g. `DC` (for `Disconnected`) is defined as follows

$$\texttt{DC}(X, Y) = 1 - \texttt{C}(X, Y) \tag{35.2}$$

`P` (for `PartOf`) is defined as

$$\texttt{P}(X, Y) = \inf_{Z \in \mathcal{U}} I(\texttt{C}(Z, X), \texttt{C}(Z, Y)) \tag{35.3}$$

where I is a fuzzy implication with some natural properties (usually $I(x_1, x_2) = \max(1 - x_1, x_2)$). Or, `EQ` (for `Equal`) is defined as

EFTA00624413

$$\texttt{EQ}(X,Y) = \min(\texttt{P}(X,Y), \texttt{P}(Y,X)) \tag{35.4}$$

and so on.

However the inference rules cannot determine the exact fuzzy values of the resulting relationships but only a lower bound, for instance

$$T(\texttt{P}(X,Y), \texttt{P}(Y,Z)) \leq \texttt{P}(X,Z) \tag{35.5}$$

where $T(x_1, x_2) = \max(0, x_1 + x_2 - 1)$. This is to be expected since in order to know the resulting fuzzy value one would need to know the exact spatial configuration. For instance Figure 35.3 depicts 2 possible configurations that would result in 2 different values of $\texttt{P}(X,Z)$.

Fig. 35.3: Depending on where Z is, in dashline, $\texttt{P}(X,Z)$ gets a different value.

One way to address this difficulty is to reason with interval-value fuzzy logic [DP09], with the downside of ending up with wide intervals. For example applying the same inference rule from Equation 35.5 in the case depicted in Figure 35.4 would result in the interval $[0,1]$, corresponding to a state of total ignorance. This is the main reason why, as explained in the next section, we have decided to use distributional fuzzy values for our AGI-oriented spatiotemporal reasoning.

There also exist attempts to use probability with RCC. For instance in [Win00], RCC relationships are extracted from computer images and weighted based on their likelihood as estimated by a shape recognition algorithm. However, to the best of our knowledge, no one has used distributional fuzzy values [Yan10] in the context of spatiotemporal reasoning; and we believe this is important for the adaptation of spatiotemporal calculi to the AGI context.

35.3 Uncertainty with Distributional Fuzzy Values

$\mathcal{P}(\mathcal{Z})$ [Yan10] is an extension of fuzzy logic that considers distributions of fuzzy values rather than mere fuzzy values. That is, fuzzy connectors are extended to apply over probability density functions of fuzzy truth value. For instance the connector $\neg$ (often defined as $\neg x = 1 - x$) is extended such that the resulting distribution $\mu_\neg : [0,1] \mapsto R^+$ is

$$\mu_\neg(x) = \mu(1 - x) \tag{35.6}$$

where μ is the probability density function of the unique argument. Similarly, one can define $\mu_\wedge : [0,1] \mapsto R^+$ as the resulting density function of the connector $x_1 \wedge x_2 = \min(x_1, x_2)$ over the 2 arguments $\mu_1 : [0,1] \mapsto R^+$ and $\mu_2 : [0,1] \mapsto R^+$

EFTA00624414

$$\begin{aligned}\mu_{\wedge}(x) = \mu_1(x) \int_x^1 \mu_2(x_2) dx_2 \\ + \mu_2(x) \int_x^1 \mu_1(x_1) dx_1\end{aligned} \tag{35.7}$$

See [Yan10] for the justification of Equations 35.6 and 35.7.

Besides extending the traditional fuzzy operators, one can also define a wider class of connectors that can fully modulate the output distribution. Let $F : [0,1]^n \mapsto ([0,1] \mapsto \mathbb{R}^+)$ be a n-ary connector that takes n fuzzy values and returns a probability density function. In that case the probability density function resulting from the extension of F over distributional fuzzy values is:

$$\mu_F = \underbrace{\int_0^1 \dots \int_0^1}_{n} F(x_1, \dots, x_n)\mu_1(x_1) \dots \mu_n(x_n) dx_1 \dots dx_n \tag{35.8}$$

where μ_1, ..., μ_n are the n input arguments. That is, it is the average of all density functions output by F applied over all fuzzy input values. Let us call that type of connectors *fuzzy-probabilistic*.

In the following we give an example of such a fuzzy-probabilistic connector.

Example with `PartOf`

Let us consider the RCC relationship `PartOf` (`P` for short as defined in Equation 35.1). A typical inference rule in the crisp case would be:

$$\frac{\mathtt{P}(X,Y) \qquad \mathtt{P}(Y,Z)}{\mathtt{P}(X,Z)} \tag{35.9}$$

expressing the transitivity of `P`. But using a distribution of fuzzy values we would have the following rule

$$\frac{\mathtt{P}(X,Y)\ \langle\mu_1\rangle \qquad \mathtt{P}(Y,Z)\ \langle\mu_2\rangle}{\mathtt{P}(X,Z)\ \langle\mu_{POT}\rangle} \tag{35.10}$$

POT stands for PartOf Transitivity. The definition of μ_{POT} for that particular inference rule may depend on many assumptions like the shapes and sizes of regions X, Y and Z. In the following we will give an example of a definition of μ_{POT} with respect to some oversimplified assumptions chosen to keep the example short.

Let us define the fuzzy variant of $\mathtt{PartOf}(X,Y)$ as the proportion of X which is part of Y (as suggested in [Pal04]). Let us also assume that every region is a unitary circle. In this case, the required proportion depends solely on the distance d_{XY} between the centers of X and Y, so we may define a function f that takes that distance and returns the according fuzzy value; that is, $f(d_{XY}) = \mathtt{P}(X,Y)$

EFTA00624415

$$f(d_{XY}) = \begin{cases} \dfrac{4\alpha - d_{XY}\sin(\alpha)}{2\pi} & \text{if } 0 \leq d_{XY} \leq 2 \\ 0 & \text{if } d_{XY} > 2 \end{cases} \tag{35.11}$$

where $\alpha = \cos^{-1}(d_{XY}/2)$.

For $0 \leq d_{XY} \leq 2$, $f(d_{XY})$ is monotone decreasing, so the inverse of $f(d_{XY})$, that takes a fuzzy value and returns a distance, is a function declared as $f^{-1}(x) : [0,1] \mapsto [0,2]$.

Let be $x_{XY} = \mathtt{P}(X,Y)$, $x_{YZ} = \mathtt{P}(Y,Z)$, $x = \mathtt{P}(X,Z)$, $d_{XY} = f^{-1}(x_{XY})$, $d_{YZ} = f^{-1}(x_{YZ})$, $l = |d_{XY} - d_{YZ}|$ and $u = d_{XY} + d_{YZ}$. For d_{XY} and d_{YZ} fixed, let $g : [0,\pi] \mapsto [l,u]$ be a function that takes as input the angle β of the two lines from the center of Y to X and Y to Z (as depicted in Figure 35.4) and returns the distance d_{XZ}. g is defined as follows

$$g(\beta) = \sqrt{(d_{XY} - d_{YZ}\sin(\beta))^2 + (d_{YZ}\cos(\beta))^2}$$

So $l \leq d_{XZ} \leq u$. It is easy to see that g is monotone increasing and surjective, therefore there exists a function inverse $g^{-1} : [l,u] \mapsto [0,\pi]$. Let $h = f \circ g$, so h takes an angle as input and

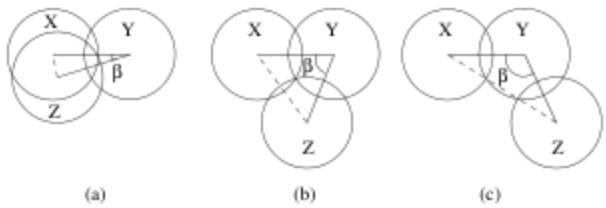


Fig. 35.4: d_{XZ}, in dashline, for 3 different angles

returns a fuzzy value, $h : [0,\pi] \mapsto [0,1]$. Since f is monotone decreasing and g is monotone increasing, h is monotone decreasing. Note that the codomain of h is $[0, f^{-1}(l)]$ if $l < 2$ or $\{0\}$ otherwise. Assuming that $l < 2$, then the inverse of h is a function with the following signature $h^{-1} : [0, f^{-1}(l)] \mapsto [0,\pi]$. Using h^{-1} and assuming that the probability of picking $\beta \in [0,\pi]$ is uniform, we can define the binary connector POT. Let us define $\nu = POT(x_{XY}, x_{YZ})$, recalling that POT returns a density function and assuming $x < f^{-1}(l)$

$$\begin{aligned} \nu(x) &= 2\lim_{\delta \to 0} \frac{\displaystyle\int_{h^{-1}(x+\delta)}^{h^{-1}(x)} \frac{1}{\pi} d\beta}{\delta} \\ &= \frac{2}{\pi} \lim_{\delta \to 0} \frac{h^{-1}(x) - h^{-1}(x+\delta)}{\delta} \\ &= -\frac{2h^{-1'}(x)}{\pi} \end{aligned} \tag{35.12}$$

where $h^{-1'}$ is the derivative of h^{-1}. If $x \geq f^{-1}(l)$ then $\nu(x) = 0$. For sake of simplicity the exact expressions of h^{-1} and $\nu(x)$ have been left out, and the case where one of the fuzzy arguments

EFTA00624416

x_{XY}, x_{YZ} or both are null has not been considered but would be treated similarly assuming some probability distribution over the distances d_{XY} and d_{XZ}.

It is now possible to define μ_{POT} in rule 35.10 (following Equation 35.8)

$$\mu_{POT} = \int_0^1 \int_0^1 POT(x_1, x_2)\mu_1(x_1)\mu_2(x_2)dx_1dx_2 \tag{35.13}$$

Obviously, assuming that regions are unitary circles is crude; in practice, regions might be of very different shapes and sizes. In fact it might be so difficult to chose the right assumptions (and once chosen to define POT correctly), that in a complex practical context it may be best to start with overly simplistic assumptions and then learn POT based on the experience of the agent. So the agent would initially perform spatial reasoning not too accurately, but would improve over time by adjusting POT, as well as the other connectors corresponding to other inference rules.

It may also be useful to have more premises containing information about the sizes (e.g $\texttt{Big}(X)$) and shapes (e.g $\texttt{Long}(Y)$) of the regions, like

$$\frac{\texttt{B}(X)\ \langle\mu_1\rangle \quad \texttt{L}(Y)\ \langle\mu_2\rangle \quad \texttt{P}(X,Y)\ \langle\mu_3\rangle \quad \texttt{P}(Y,Z)\ \langle\mu_4\rangle}{\texttt{P}(X,Z)\ \langle\mu\rangle}$$

where `B` and `L` stand respectively for `Big` and `Long`.

Simplifying Numerical Calculation

Using probability density as described above is computationally expensive, and in many practical cases it's overkill. To decrease computational cost, several cruder approaches are possible, such as discretizing the probability density functions with a coarse resolution, or restricting attention to beta distributions and treating only their means and variances (as in [Yan10]).

The right way to simplify depends on the fuzzy-probabilistic connector involved and on how much inaccuracy can be tolerated in practice.

35.4 Spatio-temporal Inference in PLN

We have discussed the representation of spatiotemporal knowledge, including associated uncertainty. But ultimately what matters is what an intelligent agent can do with this knowledge. We now turn to uncertain reasoning based on uncertain spatiotemporal knowledge, using the integration of the above-discussed calculi into the Probabilistic Logic Networks reasoning system, an uncertain inference framework designed specifically for AGI and integrated into the OpenCog AGI framework.

We give here a few examples of spatiotemporal inference rules coded in PLN. Although the current implementation of PLN incorporates both fuzziness and probability it does not have a built-in truth value to represent distributional fuzzy values, or rather a distribution of distribution of fuzzy value, as this is how, in essence, confidence is represented in PLN. At that point, depending on design choice and experimentation, it is not clear whether we want to use

EFTA00624417

the existing truth values and treat them as distributional truth values or implement a new type of truth value dedicated for that, so for our present theoretical purposes we will just call it *DF Truth Value*.

Due to the highly flexible HOJ formalism (Higher Order Judgment, explained in the PLN book in detail) we can express the inference rule for the relationship `PartOf` directly as Nodes and Links as follows

$$\begin{array}{l}\texttt{ForAllLink \$X \$Y \$Z}\\ \quad\texttt{ImplicationLink_HOJ}\\ \quad\quad\texttt{ANDLink}\\ \quad\quad\quad\texttt{PartOf(\$X, \$Y)}\ \langle \texttt{tv}_1\rangle\\ \quad\quad\quad\texttt{PartOf(\$Y, \$Z)}\ \langle \texttt{tv}_2\rangle\\ \quad\quad\texttt{ANDLink}\\ \quad\quad\quad\texttt{tv}_3 = \mu_{POT}(\texttt{tv}_1, \texttt{tv}_2)\\ \quad\quad\quad\texttt{PartOf(\$X, \$Z)}\ \langle \texttt{tv}_3\rangle\end{array} \tag{35.14}$$

where μ_{POT} is defined in Equation 35.13 but extended over the domain of PLN DF Truth Value instead of $\mathcal{P}(\mathcal{Z})$ distributional fuzzy value. Note that `PartOf(\$X, \$Y)` $\langle$`tv`$\rangle$ is a shorthand for

$$\begin{array}{l}\texttt{EvaluationLink}\ \langle \texttt{tv}\rangle\\ \quad\texttt{PartOf}\\ \quad\texttt{ListLink}\\ \quad\quad\texttt{\$X}\\ \quad\quad\texttt{\$Y}\end{array} \tag{35.15}$$

and `ForAllLink \$X \$Y \$Z` is a shorthand for

$$\begin{array}{l}\texttt{ForAllLink}\\ \quad\texttt{ListLink}\\ \quad\quad\texttt{\$X}\\ \quad\quad\texttt{\$Y}\\ \quad\quad\texttt{\$Z}\end{array} \tag{35.16}$$

Of course one advantage of expressing the inference rule directly in Nodes and Links rather than a built-in PLN inference rule is that we can use OpenCog itself to improve and refine it, or even create new spatiotemporal rules based on its experience. In the next 2 examples the fuzzy-probabilistic connectors are ignored, (so no DF Truth Value is indicated) but one could define them similarly to μ_{POT}.

First consider a temporal rule from Allen's Interval Algebra. For instance "if $\texttt{\$I}_1$ meets $\texttt{\$I}_2$ and $\texttt{\$I}_3$ is during $\texttt{\$I}_2$ then $\texttt{\$I}_3$ is after $\texttt{\$I}_1$" would be expressed as

$$\begin{array}{l}\texttt{ForAllLink \$I}_1\ \texttt{\$I}_2\ \texttt{\$I}_3\\ \quad\texttt{ImplicationLink}\\ \quad\quad\texttt{ANDLink}\\ \quad\quad\quad\texttt{Meet(\$I}_1, \texttt{\$I}_2)\\ \quad\quad\quad\texttt{During(\$I}_3, \texttt{\$I}_2)\\ \quad\quad\texttt{After(\$I}_3, \texttt{\$I}_1)\end{array} \tag{35.17}$$

EFTA00624418

And a last example with a metric predicate could be "if $X is near $Y and $X is far from $Z then $Y is far from $Y"

```
ForAllLink $X $Y $Z
    ImplicationLink_HOJ
        ANDLink
            Near($X, $Y)
            Far($X, $Z)
        Far($Y, $Z)
```
(35.18)

That is only a small and partial illustrative example – for instance other rules may be used to specify that `Near` and `Far` and reflexive and symmetric.

35.5 Examples

The ideas presented here have extremely broad applicability; but for sake of concreteness, we now give a handful of examples illustrating applications to commonsense reasoning problems.

35.5.1 Spatiotemporal Rules

The rules provided here are reduced to the strict minimum needed for the examples:

1. At $T, if $X is inside $Y and $Y is inside $Z then $X is inside $Z

```
ForAllLink $T $X $Y $Z
    ImplicationLink_HOJ
        ANDLink
            atTime($T, Inside($X, $Y))
            atTime($T, Inside($Y, $Z))
        atTime($T, Inside($X, $Z))
```

2. If a small object $X is over $Y and $Y is far from $Z then $X is far from $Z

```
ForAllLink
    ImplicationLink_HOJ
        ANDLink
            Small($X)
            Over($X, $Y)
            Far($Y)
        Far($X)
```

That rule is expressed in a crisp way but again is to be understood in an uncertain way, although we haven't worked out the exact formulae.

EFTA00624419

35.5.2 The Laptop is Safe from the Rain

A laptop is over the desk in the hotel room, the desk is far from the window and we want assess to what extent the laptop is far from the window, therefore same from the rain.

Note that the truth values are ignored but each concept is to be understood as fuzzy, that is having a PLN Fuzzy Truth Value but the numerical calculation are left out.

We want to assess how far the Laptop is from the window
`Far(Window, Laptop)`
Assuming the following

1. The laptop is small
   `Small(Laptop)`
2. The laptop is over the desk
   `Over(Laptop, Desk)`
3. The desk is far from the window
   `Far(Desk, Window)`

Now we can show an inference trail that lead to the conclusion, the numeric calculation are let for later.

1. using axioms 1, 2, 3 and PLN AND rule
   ```
   ANDLink
       Small(Laptop)
       Over(Laptop, Desk)
       Far(Desk, Window)
   ```
2. using spatiotemporal rule 2, instantiated with `$X` = `Laptop`, `$Y` = `Desk` and `$Z` = `Window`
   ```
   ImplicationLink_HOJ
       ANDLink
           Small(Laptop)
           Over(Laptop, Desk)
           Far(Desk, Window)
       Far(Laptop, Window)
   ```
3. using the result of previous step as premise with PLN implication rule
   `Far(Laptop, Window)`

35.5.3 Fetching the Toy Inside the Upper Cupboard

Suppose we know that there is a toy in an upper cupboard and near a bag, and want to assess to which extent climbing on the pillow is going to bring us near the toy.

Here are the following assumptions

1. The toy is near the bag and inside the cupboard. The pillow is near and below the cupboard
   `Near(toy, bag)` $\langle \mathtt{tv}_1 \rangle$
   `Inside(toy, cupboard)` $\langle \mathtt{tv}_2 \rangle$
   `Below(pillow, cupboard)` $\langle \mathtt{tv}_3 \rangle$
   `Near(pillow, cupboard)` $\langle \mathtt{tv}_4 \rangle$

EFTA00624420

2. The toy is near the bag inside the cupboard, how near is the toy to the edge of the cupboard?
   `ImplicationLink_HOJ`
   `ANDLink`
   `Near(toy, bag)` $\langle \mathtt{tv}_1 \rangle$
   `Inside(toy, cupboard)` $\langle \mathtt{tv}_2 \rangle$
   `ANDLink`
   $\mathtt{tv}_3 = \mu_{F_1}(\mathtt{tv}_1, \mathtt{tv}_2)$
   `Near(toy, cupboard_edge)` $\langle \mathtt{tv}_3 \rangle$
3. If I climb on the pillow, then shortly after I'll be on the pillow
   `PredictiveImplicationLink`
   `Climb_on(pillow)`
   `Over(self, pillow)`
4. If I am on the pillow near the edge of the cupboard how near am I to the toy?
   `ImplicationLink_HOJ`
   `ANDLink`
   `Below(pillow, cupboard)` $\langle \mathtt{tv}_1 \rangle$
   `Near(pillow, cupboard)` $\langle \mathtt{tv}_2 \rangle$
   `Over(self, pillow)` $\langle \mathtt{tv}_3 \rangle$
   `Near(toy, cupboard_edge)` $\langle \mathtt{tv}_4 \rangle$
   `ANDLink`
   $\mathtt{tv}_5 = \mu_{F_2}(\mathtt{tv}_1, \mathtt{tv}_2, \mathtt{tv}_3, \mathtt{tv}_4)$
   `Near(self, toy)` $\langle \mathtt{tv}_5 \rangle$

The target theorem is "How near I am to the toy if I climb on the pillow."
`PredictiveImplicationLink`
`Climb_on(pillow)`
`Near(self, toy)` $\langle ? \rangle$
And the inference chain as follows

1. Axiom 2 with axiom 1
   `Near(toy, cupboard_edge)` $\langle \mathtt{tv}_6 \rangle$
2. Step 1 with axiom 1 and 3
   `PredictiveImplicationLink`
   `Climb_on(pillow)`
   `ANDLink`
   `Below(pillow, cupboard)` $\langle \mathtt{tv}_3 \rangle$
   `Near(pillow, cupboard)` $\langle \mathtt{tv}_4 \rangle$
   `Over(self, pillow)` $\langle \mathtt{tv}_7 \rangle$
   `Near(toy, cupboard_edge)` $\langle \mathtt{tv}_6 \rangle$
3. Step 2 with axiom 4, target theorem: How near am I to the toy if I climb on the pillow
   `PredictiveImplicationLink`
   `Climb_on(pillow)`
   `Near(self, toy)` $\langle \mathtt{tv}_9 \rangle$

EFTA00624421

35.6 An Integrative Approach to Planning

Planning is a major research area in the mainstream AI community, and planning algorithms have advanced dramatically in the last decade. However, the best of breed planning algorithms are still not able to deal with planning in complex environments in the face of a high level of uncertainty, which is the sort of situation routinely faced by humans in everyday life. Really powerful planning, we suggest, requires an approach different than any of the dedicated planning algorithms, involving spatiotemporal logic combined with a sophisticated search mechanism (such as MOSES).

It may be valuable (or even necessary) for an intelligent system involved in planning-intensive goals to maintain a specialized planning-focused data structure to guide general learning mechanisms toward more efficient learning in a planning context. But even if so, we believe planning must ultimately be done as a case of more general learning, rather than via a specialized algorithm.

The basic approach we suggest here is to

- use MOSES for the core plan learning algorithm. That is, MOSES would maintain a population of "candidate partial plans", and evolve this population in an effort to find effective complete plans.
- use PLN to help in the fitness evaluation of candidate partial plans. That is, PLN would be used to estimate the probability that a partial plan can be extended into a high-quality complete plan. This requires PLN to make heavy use of spatiotemporal logic, as described in the previous sections of this chapter.
- use a GraphPlan-style [BF97] planning graph, to record information about candidate plans, and to propagate information about mutual exclusion between actions. The planning graph maybe be used to help guide both MOSES and PLN.

In essence, the planning graph simply records different states of the world that may be achievable, with a high-strength PredictiveImplicationLink pointing between state X and Y if X can sensibly serve as a predecessor to X; and a low-strength (but potentially high-confidence) PredictiveImplicationLink between X and Y if the former excludes the latter. This may be a subgraph of the Atomspace or it may be separately cached; but in each case it must be frequently accessed via PLN in order for the latter to avoid making a massive number of unproductive inferences in the course of assisting with planning.

One can think of this as being a bit like PGraphPlan [BL99], except that

- MOSES is being used in place of forward or backward chaining search, enabling a more global search of the plan space (mixing forward and backward learning freely)
- PLN is being used to estimate the value of partial plans, replacing heuristic methods of value propagation

Regarding PLN, one possibility would be to (explicitly, or in effect) create a special API function looking something like

```
EstimateSuccessProbability(PartialPlan PP, Goal G)
```

(assuming the goal statement contains information about the time allotted to achieve the goal). The PartialPlan is simply a predicate composed of predicates linked together via temporal

EFTA00624422

links such as PredictiveImplication and SimultaneousAND. Of course, such a function could be used within many non-MOSES approaches to planning also.

Put simply, the estimation of the success probability is "just" a matter of asking the PLN backward-chainer to figure out the truth value of a certain ImplicationLink, i.e.

```
PredictiveImplicationLink [time-lag T]
   EvaluationLink do PP
   G
```

But of course, this may be a very difficult inference without some special guidance to help the backward chainer. The GraphPlan-style planning graph could be used by PLN to guide it in doing the inference, via telling it what variables to look at, in doing its inferences. This sort of reasoning also requires PLN to have a fairly robust capability to reason about time intervals and events occurring therein (i.e., basic temporal inference).

Regarding MOSES, given a candidate plan, it could look into the planning graph to aid with program tree expansion. That is, given a population of partial plans, MOSES would progressively add new nodes to each plan, representing predecessors or successors to the actions already described in the plans. In choosing which nodes to add, it could be probabilistically biased toward adding nodes suggested by the planning graph.

So, overall what we have is an approach to doing planning via MOSES, with PLN for fitness estimation – but using a GraphPlan-style planning graph to guide MOSES's exploration of the neighborhood of partial plans, and to guide PLN's inferences regarding the success likelihood of partial plans.

EFTA00624423

Chapter 36
Adaptive, Integrative Inference Control

36.1 Introduction

The subtlest and most difficult aspect of logical inference is not the logical rule-set nor the management of uncertainty, but the *control* of inference: the choice of which inference steps to take, in what order, in which contexts. Without effective inference control methods, logical inference is an unscalable and infeasible approach to learning declarative knowledge. One of the key ideas underlying the CogPrime design is that inference control cannot effectively be handled by looking at logic alone. Instead, effective inference control must arise from the intersection between logical methods and other cognitive processes. In this chapter we describe some of the general principles used for inference control in the CogPrime design.

Logic itself is quite abstract and relatively (though not entirely) independent of the specific environment and goals with respect to which a system's intelligence is oriented. Inference control, however, is (among other things) a way of adapting a logic system to operate effectively with respect to a specific environment and goal-set. So, the reliance of CogPrime's inference control methods on the integration between multiple cognitive processes, is a reflection of the foundation of CogPrime on the assumption (articulated in Chapter 9) that the relevant environment and goals embody interactions between world-structures and interaction-structures best addressed by these various processes.

36.2 High-Level Control Mechanisms

The PLN implementation in CogPrime is complex and lends itself to utilization via many different methods. However, a convenient way to think about it is in terms of three basic backward-focused query operations:

- **findtv**, which takes in an expression and tries to find its truth value.
- **findExamples**, which takes an expression containing variables and tries to find concrete terms to fill in for the variables.
- **createExamples**, which takes an expression containing variables and tries to create new Atoms to fill in for the variables, using *concept creation* heuristics as discussed in a Chapter 38, coupled with inference for evaluating the products of concept creation.

EFTA00624424

and one forward-chaining operation:

- **findConclusions**, which takes a set of Atoms and seeks to draw the most interesting possible set of conclusions via combining them with each other and with other knowledge in the AtomTable.

These inference operations may of course call themselves and each other recursively, thus creating lengthy chains of diverse inference.

Findtv is quite straightforward, at the high level of discussion adopted here. Various inference rules may match the Atom; in our current PLN implementation, loosely described below, these inference rules are executed by objects called Rules. In the course of executing findtv, a decision must be made regarding how much attention to allocate to each one of these Rule objects, and some choices must be made by the objects themselves - issues that involve processes beyond pure inference, and will be discussed later in this chapter. Depending on the inference rules chosen, findtv may lead to the construction of inferences involving variable expressions, which may then be evaluated via findExamples or createExamples queries.

The findExamples operation sometimes reduces to a simple search through the AtomSpace. On the other hand, it can also be done in a subtler way. If the findExamples Rule wants to find examples of $X so that F($X), but can't find any, then it can perform some sort of heuristic search, or else it can run another findExamples query, looking for $G so that

```
Implication $G F
```

and then running findExamples on G rather than F. But what if this findExamples query doesn't come up with anything? Then it needs to run a createExamples query on the same implication, trying to build a $G satisfying the implication.

Finally, forward-chaining inference (findConclusions) may be conceived of as a special heuristic for handling special kinds of findExample problems. Suppose we have K Atoms and want to find out what consequences logically ensue from these K Atoms, taken together. We can form the conjunction of the K Atoms (let's call it C), and then look for $D so that

```
Implication C $D
```

Conceptually, this can be approached via findExamples, which defaults to createExamples in cases where nothing is found. However, this sort of findExamples problem is special, involving appropriate heuristics for combining the conjuncts contained in the expression C, which embody the basic logic of forward-chaining rather than backward-chaining inference.

36.2.1 The Need for Adaptive Inference Control

It is clear that in humans, inference control is all about context. We use different inference strategies in different contexts, and learning these strategies is most of what *learning to think* is all about. One might think to approach this aspect of cognition, in the CogPrime design, by introducing a variety of different inference control heuristics, each one giving a certain algorithm for choosing which inferences to carry out in which order in a certain context. (This is similar to what has been done within Cyc, for example http:\cyc.com.) However, in keeping with the *integrated intelligence* theme that pervades CogPrime, we have chosen an alternate strategy for PLN. We have one inference control scheme, which is quite simple, but which relies partially on

EFTA00624425

structures coming from outside PLN proper. The requisite variety of inference control strategies is provided by variety in the non-PLN structures such as

- HebbianLinks existing in the AtomTable.
- Patterns recognized via pattern-mining in the corpus of prior inference trails

36.3 Inference Control in PLN

We will now describe the basic "inference control" loop of PLN in CogPrime. Pre-2013 OpenCog versions used a somewhat different scheme, more similar to a traditional logic engine. The approach presented here is more cognitive synergy oriented, achieving PLN control via a combination of logic engine style methods and integration with attention allocation.

36.3.1 Representing PLN Rules as GroundedSchemaNodes

PLN inference rules may be represented as GroundedSchemaNodes. So for instance the PLN Deduction Rule, becomes a GroundedSchemaNode with the properties:

- Input: a pair of links (L1, L2), where L1 and L2 are the same type, and must be one of InheritanceLink, ImplicationLink, SubsetLink or ExtensionalImplicationLink
- Output: a single link, of the same type as the input

The actual PLN Rules and Formulas are then packed into the internal execution methods of GroundedSchemaNodes.

In the current PLN code, each inference rule has a Rule class and a separate Formula class. So then, e.g. the PLNDeductionRule GroundedSchemaNode, invokes a function of the general form

```
Link PLNDeductionRule(Link L1, Link L2)
```

which calculates the deductive consequence of two links. This function then invokes a function of the form

```
TruthValue PLNDeductionFormula(TruthValue tAB, TruthValue tBC, TruthValue tA, TruthValue tB, Truth
```

which in turn invokes functions such as

```
SimpleTruthValue SimplePLNDeductionFormula(SimpleTruthValue tAB, SimpleTruthValue tBC, SimpleTruth

IndefiniteTruthValue IndefinitePLNDeductionFormula(IndefiniteTruthValue tAB, IndefiniteTruthValue
```

36.3.2 Recording Executed PLN Inferences in the Atomspace

Once an inference has been carried out, it can be represented in the Atomspace, e.g. as

EFTA00624426

```
ExecutionLink
        GroundedSchemaNode: PLNDeductionRule
        ListLink
                HypotheticalLink
                      InheritanceLink people animal <tv1>
                HypotheticalLink
                       InheritanceLink animal breathe <tv2>
        HypotheticalLink
                 InheritanceLink people breathe <tv3>
```

Note that a link such as

```
InheritanceLink people breathe <.8,.2>
```

will have its truth value stored as a truth value version within a CompositeTruthValue object.)

In the above, e.g.

```
InheritanceLink people animal
```

is used as shorthand for

```
InheritanceLink C1 C2
```

where C1 and C2 are ConceptNodes representing "people" and "animal" respectively.

We can also have records of inferences involving variables, such as

```
ExecutionLink
        GroundedSchemaNode: PLNDeductionRule
        ListLink
                HypotheticalLink
                       InheritanceLink $V1 animal <tv1>
                HypotheticalLink
                       InheritanceLink animal breathe <tv2>
        HypotheticalLink
                       InheritanceLink $V1 breathe <tv3>
```

where $V1 is a specific VariableNode.

36.3.3 Anatomy of a Single Inference Step

A single inference step, then, may be viewed as follows:

1. Choose an inference rule R and a tuple of Atoms that collectively match the input conditions of the rule
2. Apply the chosen rule R to the chosen input Atoms
3. Create an ExecutionLink recording the output found
4. In addition to retaining this ExecutionLink in the Atomspace, also save a copy of it to the InferenceRepository [this is not needed for the very first implementation, but will be very useful once PLN is in regular use.

The InferenceRepository, referred to here, is a special Atomspace that exists just to save a record of PLN inferences. It can be mined, after the fact, to learn inference patterns, which can be used to guide future inferences.

EFTA00624427

36.3.4 Basic Forward and Backward Inference Steps

The choice of an inference step, at the microscopic level, may be done in a number of ways, of which perhaps the simplest are:

- "'Basic forward step"'. Choose an Atom A1, then choose a rule R. If R only takes one input, then apply R to A1. If R applies to two Atoms, then find another Atom A2 so that (A1, A2) may be taken as the inputs of R.
- "'Basic backward step."' Choose an Atom A1, then choose a rule R. If R takes only one input, then find an Atom A2 so that applying R to A2, yields A1 as output. If R takes two inputs, then find two Atoms (A2, A3) so that applying R to (A2, A3) yields A1 as output.

Given a target Atom such as

```
A1 = Inheritance $V1 breathe
```

the VariableAbstractionRule will do inferences such as

```
ExecutionLink
        VariableAbstractionRule
        HypotheticalLink
                Inheritance people breathe
        HypotheticalLink
                Inheritance $v1 breathe
```

This allows the basic backward step to carry out variable fulfillment queries as well as truth value queries. We may encapsulate these processes in the Atomspace as

```
GroundedSchemaNode: BasicForwardInferenceStep
GroundedSchemaNode: BasicBackwardInferenceStep
```

which take as input some Atom A1.
and also as

```
GroundedSchemaNode: AttentionalForwardInferenceStep
GroundedSchemaNode: AttentionalBackwardInferenceStep
```

which automatically choose the Atom A1 they start with, via choosing some Atom within the AttentionalFocus, with probability proportional to STI.

Forward chaining, in its simplest form, then becomes: The process of repeatedly executing the AttentionalForwardInferenceStep SchemaNode.

Backward chaining, in the simplest case (we will discuss more complex cases below), becomes the process of

1. Repeatedly executing the BasicBackwardInferenceStep SchemaNode, starting from a given target Atom
2. Concurrently, repeatedly executing the AttentionalBackwardInferenceStep SchemaNode, to ensure that backward inference keeps occurring, regarding Atoms that were created via Step 1

Inside the BasicForwardStep or BasicBackwardStep schema, there are two choices to be made: choosing a rule R, and then choosing additional Atoms A2 and possibly A3.

The choice of the rule R should be made probabilistically, choosing each rule with probability proportional to a certain weight associated with each rule. Initially we can assign these weights

EFTA00624428

generically, by hand, separately for each application domain. Later on they should be chosen adaptively, based on information mined from the InferenceRepository, regarding which rules have been better in which contexts.

The choice of the additional Atoms A2 and A3 is subtler, and should be done using STI values as a guide:

- First the AttentionalFocus is searched, to find all the Atoms there that fit the input criteria of the rule R. Among all the Atoms found, an Atom is chosen with probability proportional to STI.
- If the AttentionalFocus doesn't contain anything suitable, then an effort may be made to search the rest of the Atomspace to find something suitable. If multiple candidates are found within the amount of effort allotted, then one should be chosen with probability proportional to STI

If an Atom A is produced as output of a forward inference step, or is chosen as the input of a backward inference step, then the STI of this Atom A should be incremented. This will increase the probability of A being chosen for ongoing inference. In this way, attention allocation is used to guide the course of ongoing inference.

36.3.5 Interaction of Forward and Backward Inference

Starting from a target, a series of backward inferences can figure out ways to estimate the truth value of that target, or fill in the variables within that target.

However, once the backward-going chain of inferences is done (to some reasonable degree of satisfaction), there is still the remaining task of using all the conclusions drawn during the series of backward inferences, to actually update the target.

Elegantly, this can be done via forward inference. So if forward and backward inference are both operating concurrently on the same pool of Atoms, it is forward inference that will propagate the information learned during backward chaining inference, up to the target of the backward chain.

36.3.6 Coordinating Variable Bindings

Probably the thorniest subtlety that comes up in a PLN implementation is the coordination of the values assigned to variables, across different micro-level inferences that are supposed to be coordinated together as part of the same macro-level inference.

For a very simple example, suppose we have a truth-value query with target

```
A1 = InheritanceLink Bob rich
```

Suppose the deduction rule R is chosen.
Then if we can find (A2, A3) that look like, say,

```
A2 = InheritanceLink Bob owns_mansion
A3 = InheritanceLink owns_mansion rich
```

EFTA00624429

, our problem is solved.

But what if there is no such simple solution in the Atomspace available? Then we have to build something like

```
A2 = InheritanceLink Bob $v1
A3 = InheritanceLink $v1 rich
```

and try to find something that works to fill in the variable $v1.

But this is tricky, because $v1 now has two constraints (A2 and A3). So, suppose A2 and A3 are both created as a result of applying BasicBackwardInferenceStep to A1, and thus A2 and A3 both get high STI values. Then both A2 and A3 are going to be acted on by AttentionalBackwardInferenceStep. But as A2 and A3 are produced via other inputs using backward inference, it is necessary that the values assigned to $v1 in the context of A2 and A3 remain consistent with each other.

Note that, according to the operation of the Atomspace, the same VariableAtom will be used to represent $v1 no matter where it occurs.

For instance, it will be problematic if one inference rule schema tries to instantiate $v1 with "owns_mansion", but another tries to instantiate $v1 with "lives_in_Manhattan".

That is, we don't want to find

```
InheritanceLink Bob lives_in_mansion
InheritanceLink lives_in_mansion owns_mansion
|-
InheritanceLink Bob owns_mansion
```

which binds $v1 to owns_mansion, and

```
InheritanceLink lives_in_Manhattan lives_in_top_city
InheritanceLink lives_in_top_city rich
|-
InheritanceLink lives_in_Manhattan rich
```

which binds $v1 to lives_in_Manhattan

We want *A2* and *A3* to be derived in ways that bind $v1 to the same thing.

The most straightforward way to avoid confusion in this sort of context, is to introduce an addition kind of inference step,

- "'Variable-guided backward step"'. Choose a set V of VariableNodes (which may just be a single VariableNode $v1), and identify the set S_V of all Atoms involving any of the variables in V.
  - Firstly: If V divides into two sets V1 and V2, so that no Atom contains variables in both V1 and V2, then launch separate variable-guided backwards steps for V1 and V2. [This step is "Problem Decomposition"]
  - Carry out the basic backward step for all the Atoms in S_V, but restricting the search for Atoms A2, A3 in such a way that each of the variables in V is consistently instantiated. This is a non-trivial optimization, and more will be said about this below.
- "'Variable-guided backward step, Atom-triggered."' Choose an Atom A1. Identify the set V of VariableNodes targeted by A1, and then do a variable-guided backward step starting from V.

This variable guidance may, of course, be incorporated into the AttentionalBackwardInferenceStep as well. In this case, backward chaining becomes the process of

EFTA00624430

- Repeatedly executing the VariableGuidedBackwardInferenceStep SchemaNode, starting from a given target Atom
- Concurrently, repeatedly executing the AttentionalVariableGuidedBackwardInferenceStep SchemaNode, to ensure that backward inference keeps occurring, regarding Atoms that were created via Step 1

The hard work here is then done in step 2 of the Variable Guided Backward Step, which has to search for multiple Atoms, to fulfill the requirements of multiple inference rules, in a way that keeps consistent variable instantiations. But this same difficulty exists in a conventional backward chaining framework, it's just arranged differently, and not as neatly encapsulated.

36.3.7 An Example of Problem Decomposition

Illustrating a point raised above, we now give an example of a case where, given a problem of finding values to assign a set of variables to make a set of expressions hold simultaneously, the appropriate course is to divide the set of expressions into two separate parts.

Suppose we have the six expressions

```
E1 = Inheritance ( $v1,  Animal)

E2 = Evaluation( $v1, ($v2, Bacon) )

E3 = Inheritance( $v2, $v3)

E4 = Evaluation( Eat, ($v3, $v1) )

E5 = Evaluation (Eat, ($v7, $v9) )

E6 = Inheritance $v9 $v6
```

Since the set {E1, E2, E3, E4} doesn't share any variables with {E5, E6}, there is no reason to consider them all as one problem. Rather we will do better to decompose it into two problems, one involving {E1, E2, E3, E4} and one involving {E5, E6}.

In general, given a set of expressions, one can divide it into subsets, where each subset S has the property that: for every variable v contained in S, all occurrences of v in the Atomspace, are in expressions contained in S.

36.3.8 Example of Casting a Variable Assignment Problem as an Optimization Problem

Suppose we have the four expressions

```
E1 = Inheritance ( $v1,  Animal)
E2 = Evaluation( $v2, ($v1, Bacon) )
E3 = Inheritance( $v2, $v3)
E4 = Evaluation( Enjoy, ($v1, $v3) )
```

EFTA00624431

where Animal, Bacon and Enjoy are specific Atoms.

Suppose the task at hand is to find values for ($v1, $v2, $v3) that will make all of these expressions confidently true.

If there is some assignment

```
($v1, $v2, $v3) = (A1,A2, A3)
```

ready to hand in the Atomspace, that fulfills the equations E1, E2, E3, E4, then the Atomspace API's pattern matcher will find it. For instance,

```
($v1, $v2, $v3) = (Cat, Eat, Chew)
```

would work here, since

```
E1 = Inheritance ( Cat,  Animal)
E2 = Evaluation( Eat, (Cat, Bacon) )
E3 = Inheritance( Eat, Chew)
E4 = Evaluation( Enjoy, (Cat, Chew) )
```

are all reasonably true.

If there is no such assignment ready to hand, then one is faced with a search problem. This can can be approached as an optimization problem, e.g. one of maximizing a function

```
f($v1, $v2, $v3) = sc(E1) * sc(E2) * sc(E3)
```

where

```
sc(A) = A.strength * A.confidence
```

The function f is then a function with signature

```
f: Atom^4 ==> float
```

f can then be optimized by a host of optimization algorithms. For instance a genetic algorithm approach might work, but a BOA (Bayesian Optimization Algorithm) approach would probably be better.

In a GA approach, mutation would work as follows. Suppose one had a candidate

```
($v1, $v2, $v3) = (A1,A2, A3)
```

Then one could mutate this candidate by (for instance) replacing A1 with some other Atom that is similar to A1, e.g. connected to A1 with a high-weight SimilarityLink in the Atomspace.

36.3.9 Backward Chaining via Nested Optimization

Given this framework that does inference involving variables via using optimization to solve simultaneous equations of logical expressions with overlapping variables, "backward chaining" becomes the iterative launch of repeated optimization problems, each one defined in terms of the previous ones. We will now illustrate this point via continuing with the {E1, E2, E3, E4)} example from above. Suppose one found an assignment

```
($v1, $v2, $v3) = (A1,A2, A3)
```

that worked for every equation except E3. Then there is the problem of finding some way to make

EFTA00624432

```
E3 = Inheritance( A2, A3)
```

work.

For instance, what if we have found the assignment

```
($v1, $v2, $v3) = (Cat, Eat, Chase)
```

In this case, we have

```
E1 = Inheritance ( Cat,  Animal)  -- YES
E2 = Evaluation( Eat, (Cat, Bacon) ) -- YES
E3 = Inheritance( Eat, Chase) --  NO
E4 = Evaluation( Enjoy, (Cat, Chase) ) -- YES
```

so the assignment works for every equation except E3. Then there is the problem of finding some way to make

```
E3 = Inheritance( Eat, Chase)
```

work. But if the truth value of

```
Inheritance( Eat, Chase)
```

has a low strength and high confidence, this may seem hopeless, so this assignment may not get followed up on.

On the other hand, we might have the assignment

```
($v1, $v2, $v3) = (Cat, Eat, SocialActivity)
```

In this case, for a particular CogPrime instance, we might have

```
E1 = Inheritance ( Cat,  Animal)  -- YES
E2 = Evaluation( Eat, (Cat, Bacon) ) -- YES
E3 = Inheritance( Eat, SocialActivity) --  UNKNOWN
E4 = Evaluation( Enjoy, (Cat, SocialActivity) ) -- YES
```

The above would hold if the reasoning system knew that cats enjoy social activities, but did not know whether eating is a social activity. In this case, the reasoning system would have reason to launch a new inference process aimed at assessing the truth value of

```
E3 = Inheritance( Eat, SocialActivity) --
```

This is backward chaining: Launching a new inference process to figure out a question raised by another inference process.

For instance, in this case the inference engine might: Choose an inference Rule (let's say it's Deduction, for simplicity), and then look for $v4 so that

```
Inheritance Eat $v4
Inheritance $v4 SocialActivity
```

are both true. In this case one has spawned a new Variable-Guided Backward Inference problem, which must be solved in order to make {A1, A2, A3} an OK solution for the problem of {E1, E2, E3, E4}.

Or it might choose the Induction rule, and look for $v4 so that

```
Inheritance $v4 Eat
Inheritance $v4 SocialActivity
```

Maybe then it would find that $v4=Dinner works, because it knows that

EFTA00624433

```
Inheritance Dinner Eat
Inheritance Dinner SocialActivity
```

But maybe $v4=Dinner doesn't boost the truth value of

```
E3 = Inheritance( Eat, SocialActivity)
```

high enough. In that case it may keep searching for more information about E4 in the context of this particular variable assignment. It might choose Induction again, and discover e.g. that

```
Inheritance Lunch Eat
Inheritance Lunch SocialActivity
```

In this example, weve assumed that some non-backward-chaining heuristic search mechanism found a solution that almost works, so that backward chaining is only needed on E3. But of course, one could backward chain on all of E1, E2, E3, E4 simultaneously – or various subsets thereof.

For a simple example, suppose one backward chains on

```
E1 = Inheritance ( $v1,  Animal)
E3 = Inheritance( $v2, SocialActivity)
```

simultaneously. Then one is seeking, say, ($v4, $v5) so that

```
Inheritance $v1 $v5
Inheritance $v5 Animal
Inheritance $v2 $v4
Inheritance $v4 SocialActivity\
```

This adds no complexity, as the four relations partition into two disjoint sets of two. Separate chaining processes may be carried out for E1 and E3.

On the other hand, for a slightly more complex example, what if we backward chain on

```
E2 = Evaluation( $v2, ($v1, Bacon) )
E3 = Inheritance( $v2, SocialActivity)
```

simultaneously? (Assuming that a decision has already been made to explore the possibility $v3 = SocialActivity.) Then we have a somewhat more complex situation. We are trying to find $v2 that is a SocialActivity, so that $v1 likes to do $v2 in conjunction with Bacon.

If the Member2Evaluation rule is chosen for E2 and the Deduction rule is chosen for E3, then we have

```
E5 = Inheritance $v2 $v6
E6 = Inheritance $v6 SocialActivity
E7 = Member ($v1, Bacon) (SatisfyingSet $v2)
```

and if the Inheritance2Member rule is then chosen for E7, we have

```
E5 = Inheritance $v2 $v6
E6 = Inheritance $v6 SocialActivity
E8 = Inheritance ($v1, Bacon) (SatisfyingSet $v2)
```

and if Deduction is then chosen for E8 then we have

```
E5 = Inheritance $v2 $v6
E6 = Inheritance $v6 SocialActivity
E9 = Inheritance ($v1 , Bacon) $v8
E10 = Inheritance $v8 (SatisfyingSet $v2)
```

EFTA00624434

Following these steps expands the search to involve more variables and means the inference engine now gets to deal with

```
E1 = Inheritance ( $v1,  Animal)
E4 = Evaluation( Enjoy, ($v1, SocialActivity) )
E5 = Inheritance $v2 $v6
E6 = Inheritance $v6 SocialActivity
E9 = Inheritance ($v1 , Bacon) $v8
E10 = Inheritance $v8 (SatisfyingSet $v2)
```

or some such i.e. we have expanded our problem to include more and more simultaneous logical equations in more and more variables! Which is not necessarily a terrible thing, but it does get complicated.

We might find, for example, that $v=1 Pig, $v6=Dance, $v2=Waltz, $v8 = PiggyWaltz

```
E1 = Inheritance ( Pig,  Animal)
E4 = Evaluation( Enjoy, (Pig, SocialActivity) )
E5 = Inheritance Waltz Dance
E6 = Inheritance Dance SocialActivity
E9 = Inheritance (Pig , Bacon) PiggyWaltz
E10 = Inheritance PiggyWaltz (SatisfyingSet Waltz)
```

Here PiggyWaltz is a special dance that pigs do with their Bacon, as a SocialActivity!

Of course, this example is extremely contrived. Real inference examples will rarely be this simple, and will not generally involve Nodes that have simple English names. This example is just for illustration of the concepts involved.

36.4 Combining Backward and Forward Inference Steps with Attention Allocation to Achieve the Same Effect as Backward Chaining (and Even Smarter Inference Dynamics)

Backward chaining is a powerful heuristic, one can achieve the same effect – and even smarter inference dynamics – via a combination of

- heuristic search to satisfy simultaneous expressions
- boosting the STI of expressions being searched
- importance spreading (of STI)
- ongoing background forward inference

can combine to yield the same basic effect as backward chaining, but without explicitly doing backward chaining.

The basic idea is: When system of expressions involving variables is explored using a GA or whatever other optimization process is deployed, these expressions also get their STI boosted.

Then, the atoms with high STI, are explored by the forward inference process, which is always acting in the background on the atoms in the Atomspace. Other atoms related to these also get STI via importance spreading. And these other related Atoms are then acted on by forward inference as well.

This forward chaining will then lead to the formation of new Atoms, which may make the solution of the system of expressions easier the next time it is visited by the backward inference process

EFTA00624435

In the above example, this means:

- E1, E2, E3, E4 will all get their STI boosted
- Other Atoms related to these (Animal, Bacon and Enjoy) will also get their STI boosted
- These other Atoms will get forward inference done on them
- This forward inference will then yield new Atoms that can be drawn on when the solution of the expression-system E1, E2, E3, E4 is pursued the next time

So, for example, if the system did not know that eating is a social activity, it might learn this during forward inference on SocialActivity. The fact that SocialActivity has high STI would cause forward inferences such as

```
Inheritance Dinner Eat
Inheritance Dinner SocialActivity
|-
Inheritance Eat SocialActivity
```

to get done. These forward inferences would then produce links that could simply be found by the pattern matcher when trying to find variable assignments to satisfy {E1, E2, E3, E4}.

36.4.1 Breakdown into MindAgents

To make this sort of PLN dynamic work, we require a number of MindAgents to be operating "ambiently" in the background whenever inference is occurring; to wit:

- attentional forward chaining (i.e. each time this MindAgent is invoked, it chooses high-STI Atoms and does basic forward chaining on them)
- attention allocation (importance updating is critical, Hebbian learning is also useful)
- attentional (variable guided) backward chaining

On top of this ambient inference, we may then have query-driven backward chaining inferences submitted by other processes (via these launching backward inference steps and giving the associated Atoms lots of STI). The ambient inference processes will help the query-driven inference processes to get fulfilled.

36.5 Hebbian Inference Control

A key aspect of the PLN control mechanism described here is the use of attention allocation to guide inference. A key aspect here is the use of attention allocation to guide Atom choice in the course of forward and backward inference. Figure 36.1 gives a simple illustrative example of the use of attention allocation, via HebbianLinks, for PLN backward chaining.

The semantics of a HebbianLink between A and B is, intuitively: In the past, when A was important, B was also important. HebbianLinks are created via two basic mechanisms: pattern-mining of associations between importances in the system's history, and PLN inference based on HebbianLinks created via pattern mining (and inference). Thus, saying that PLN inference control relies largely on HebbianLinks is in part saying that PLN inference control relies on

EFTA00624436

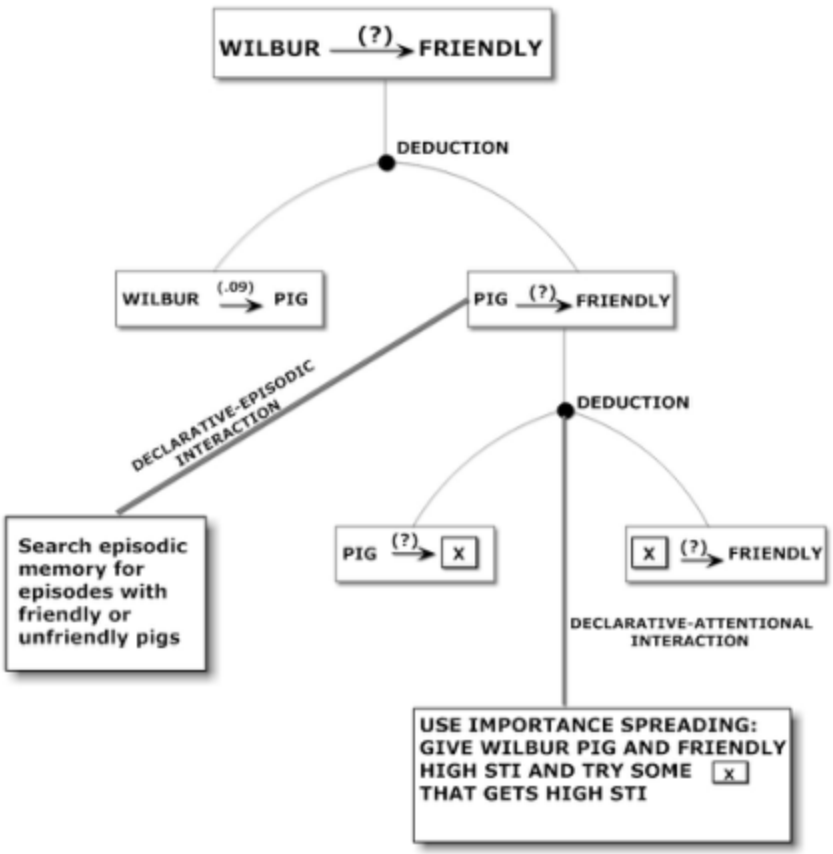


Fig. 36.1: **The Use of Attention Allocation for Guiding Backward Chaining Inference**.

PLN. There is a bit of a recursion here, but it's not a bottomless recursion because it bottoms out with HebbianLinks learned via pattern mining.

As an example of the Atom-choices to be made by a forward or backward inference agent in the course of doing inference, consider that to evaluate (Inheritance A C) via the deduction Rule, some collection of intermediate nodes for the deduction must be chosen. In the case of higher-order deduction, each deduction may involve a number of complicated subsidiary steps, so perhaps only a single intermediate node will be chosen. This choice of intermediate nodes must be made via context-dependent prior probabilities. In the case of other Rules besides deduction, other similar choices must be made.

The basic means of using HebbianLinks in inferential Atom-choice is simple: If there are Atoms linked via HebbianLinks with the other Atoms in the inference tree, then these Atoms should be given preference in the selection process.

Along the same lines but more subtly, another valuable heuristic for guiding inference control is "on-the-fly associatedness assessment." If there is a chance to apply the chosen Rule via working with Atoms that are:

- strongly associated with the Atoms in the Atom being evaluated (via HebbianLinks)

EFTA00624437

- strongly associated with each other via HebbianLinks (hence forming a *cohesive set*)

then this should be ranked as a good thing.

For instance, it may be the case that, when doing deduction regarding relationships between humans, using relationships involving other humans as intermediate nodes in the deduction is often useful. Formally this means that, when doing inference of the form:

```
AND
  Inheritance A human
  Inheritance A B
  Inheritance C human
  Inheritance C B
|-
Inheritance A C
```

then it is often valuable to choose B so that:

```
HebbianLink B human
```

has high strength. This would follow from the above-mentioned heuristic.

Next, suppose one has noticed a more particular heuristic - that in trying to reason about humans, it is particularly useful to think about their wants. This suggests that in abductions of the above form it is often useful to choose B of the form:

```
B = SatisfyingSet [ wants(human, $X) ]
```

This is too fine-grained of a cognitive-control intuition to come from simple association-following. Instead, it requires fairly specific data-mining of the system's inference history. It requires the recognition of "Hebbian predicates" of the form:

```
HebbianImplication
  AND
    Inheritance $A human
    Inheritance $C human
    Similarity
      $B
      SatisfyingSet
         Evaluation wants (human, $X)
  AND
    Inheritance $A $B
    Inheritance $C $B
```

The semantics of:

```
HebbianImplication X Y
```

is that *when X is being thought about, it is often valuable to think about Y shortly thereafter.*

So what is required to do inference control according to heuristics like *think about humans according to their wants* is a kind of backward-chaining inference that combines Hebbian implications with PLN inference rules. PLN inference says that to assess the relationship between two people, one approach is abduction. But Hebbian learning says that when setting up an abduction between two people, one useful precondition is if the intermediate term in the abduction regards wants. Then a check can be made whether there are any relevant intermediate terms regarding wants in the system's memory.

What we see here is that the overall inference control strategy can be quite simple. For each Rule that can be applied, a check can be made for whether there is any relevant Hebbian knowledge regarding the general constructs involved in the Atoms this Rule would be manipulating.

EFTA00624438

If so, then the prior probability of this Rule is increased, for the purposes of the Rule-choice bandit problem. Then, if the Rule is chosen, the specific Atoms this Rule would involve in the inference can be summoned up, and the relevant Hebbian knowledge regarding these Atoms can be utilized.

To take another similar example, suppose we want to evaluate:

```
Inheritance pig dog
```

via the deduction Rule (which also carries out induction and abduction). There are a lot of possible intermediate terms, but a reasonable heuristic is to ask a few basic questions about them: How do they move around? What do they eat? How do they reproduce? How intelligent are they? Some of these standard questions correspond to particular intermediate terms, e.g. the intelligence question partly boils down to computing:

```
Inheritance pig intelligent
```

and:

```
Inheritance dog intelligent
```

So a link:

```
HebbianImplication animal intelligent
```

may be all that's needed to guide inference to asking this question. This HebbianLink says that when thinking about animals, it's often interesting to think about intelligence. This should bias the system to choose "intelligent" as an intermediate node for inference.

On the other hand, the *what do they eat* question is subtler and boils down to asking; Find $X so that when:

```
  R($X) = SatisfyingSet[$Y] eats ($Y,$X)
```

holds (R($X) is a concept representing what eat $X), then we have:

```
Inheritance pig R($X)
```

and:

```
Inheritance dog R($X)
```

In this case, a HebbianLink from animal to eat would not really be fine-grained enough. Instead we want a link of the form:

```
HebbianImplication
    Inheritance $X animal
    SatisfyingSet[$Y] eats ($X, $Y)
```

This says that when thinking about an animal, it's interesting to think about what that animal eats.

The deduction Rule, when choosing which intermediate nodes to use, needs to look at the scope of available HebbianLinks and HebbianPredicates and use them to guide its choice. And if there are no good intermediate nodes available, it may report that it doesn't have enough experience to assess with any confidence whether it can come up with a good conclusion. As a consequence of the bandit-problem dynamics, it may be allocated reduced resources, or another Rule is chosen altogether.

EFTA00624439

36.6 Inference Pattern Mining

Along with general-purpose attention spreading, it it very useful for PLN processes to receive specific guidance based on patterns mined from previously performed and storedife

This information is stored in CogPrime in a data repository called the InferencePattern-Repository - which is, quite simply, a special "data table" containing inference trees extracted from the system's inference history, and patterns recognized therein. An "inference tree" refers to a tree whose nodes, called InferenceTreeNodes, are Atoms (or generally Atom-versions, Atoms with truth value relative to a certain context), and whose links are inference steps (so each link is labeled with a certain inference rule).

In a large CogPrime system it may not be feasible to store all inference trees; but then a wide variety of trees should still be retained, including mainly successful ones as well as a sampling of unsuccessful ones for purpose of comparison.

The InferencePatternRepository may then be used in two ways:

- An inference tree being actively expanded (i.e. utilized within the PLN inference system) may be compared to inference trees in the repository, in real time, for guidance. That is, if a node N in an inference tree is being expanded, then the repository can be searched for nodes similar to N, whose contexts (within their inference trees) are similar to the context of N within its inference tree. A study can then be made regarding which Rules and Atoms were most useful in these prior, similar inferences, and the results of this can be used to guide ongoing inference.
- Patterns can be extracted from the store of inference trees in the InferencePatternRepository, and stored separately from the actual inference trees (in essence, these patterns are inference subtrees with variables in place of some of their concrete nodes or links). An inference tree being expanded can then be compared to these patterns instead of, or in addition to, the actual trees in the repository. This provides greater efficiency in the case of common patterns among inference trees.

A reasonable approach may be to first check for inference patterns and see if there are any close matches; and if there are not, to then search for individual inference trees that are close matches.

Mining patterns from the repository of inference trees is a potentially highly computationally expensive operation, but this doesn't particularly matter since it can be run periodically in the background while inference proceeds at its own pace in the foreground, using the mined patterns. Algorithmically, it may be done either by exhaustive frequent-itemset-mining (as in deterministic greedy datamining algorithms), or by stochastic greedy mining. These operations should be carried out by an InferencePatternMiner MindAgent.

36.7 Evolution As an Inference Control Scheme

It is possible to use PEPL (Probabilistic Evolutionary Program Learning, such as MOSES) as, in essence, an InferenceControl scheme. Suppose we are using an evolutionary learning mechanism such as MOSES or PLEASURE [Goe08a] to evolve populations of predicates or schemata. Recall that there are two ways to evaluate procedures in CogPrime : by inference

EFTA00624440

or by direct evaluation. Consider the case where inference is needed in order to provide high-confidence estimates of the evaluation or execution relationships involved. Then, there is the question of how much effort to spend on inference, for each procedure being evaluated as part of the fitness evaluation process. Spending a small amount of effort on inference means that one doesn't discover much beyond what's immediately apparent in the AtomSpace. Spending a large amount of effort on inference means that one is trying very hard to use indirect evidence to support conjectures regarding the evaluation or execution Links involved.

When one is evolving a large population of procedures, one can't afford to do too much inference on each candidate procedure being evaluated. Yet, of course, doing more inference may yield more accurate fitness evaluations, hence decreasing the number of fitness evaluations required.

Often, a good heuristic is to gradually increase the amount of inference effort spent on procedure evaluation, during the course of evolution. Specifically, one may make the amount of inference effort roughly proportional to the overall population fitness. This way, initially, evolution is doing a cursory search, not thinking too much about each possibility. But once it has some fairly decent guesses in its population, then it starts thinking hard, applying more inference to each conjecture.

Since the procedures in the population are likely to be interrelated to each other, inferences done on one procedure are likely to produce intermediate knowledge that's useful for doing inference on other procedures. Therefore, what one has in this scheme is evolution as a control mechanism for higher-order inference.

Combined with the use of evolutionary learning to achieve memory across optimization runs, this is a very subtle approach to inference control, quite different from anything in the domain of logic-based AI. Rather than guiding individual inference steps on a detailed basis, this type of control mechanism uses evolutionary logic to guide the general direction of inference, pushing the vast mass of exploratory inferences in the direction of solving the problem at hand, based on a flexible usage of prior knowledge.

36.8 Incorporating Other Cognitive Processes into Inference

Hebbian inference control and inference pattern mining are valuable and powerful processes, but they are not always going to be enough. The solution of some problems that CogPrime chooses to address via inference will ultimately require the use of other methods, too. In these cases, one workaround is for inference to call on other cognitive processes to help it out.

This is done via the forward or backward chaining agents identifying specific Atoms deserving of attention by other cognitive processes, and then spawning Tasks executing these other cognitive processes on the appropriate Atoms.

Firstly, which Atoms should be selected for this kind of attention? What we want are InferenceTreeNodes that:

- have high STI.
- have the impact to significantly change the overall truth value of the inference tree they are embedded in (something that can be calculated by hypothetically varying the truth value of the InferenceTreeNode and seeing how the truth value of the overall conclusion is affected).
- have truth values that are known with low confidence.

EFTA00624441

Truth values meeting these criteria should be taken as strong candidates for attention by other cognitive processes.

The next question is which other cognitive processes do we apply in which cases?

MOSES in supervised categorization mode can be applied to a candidate InferenceTreeNode representing a CogPrime Node if it has a sufficient number of members (Atoms linked to it by MemberLinks); and, a sufficient number of new members have been added to it (or have had their membership degree significantly changed) since MOSES in supervised categorization mode was used on it last.

Next, pattern mining can be applied to look for connectivity patterns elsewhere in the AtomTable, similar to the connectivity patterns of the candidate Atom, if the candidate Atom has changed significantly since pattern mining last visited it.

More subtly, what if, we try to find whether "cross breed" implies "Ugliness", and we know that "bad genes" implies Ugliness, but can't find a way, by backward chaining, to prove that "cross breed" implies "bad genes". Then we could launch a non-backward-chaining algorithm to measure the overlap of SatisfyingSet(cross breed) and SatisfyingSet(bad genes). Specifically, we could use MOSES in supervised categorization mode to find relationships characterizing "cross breed" and other relationships characterizing "bad genes", and then do some forward chaining inference on these relationships. This would be a general heuristic for what to do when there's a link with low confidence but high potential importance to the inference tree.

SpeculativeConceptFormation (see Chapter 38) may also be used to create new concepts and attempt to link them to the Atoms involved in an inference (via subsidiary inference processes, or HebbianLink formation based on usage in learned procedures, etc.), so that they may be used in inference.

36.9 PLN and Bayes Nets

Finally, we give some comments on the relationship between PLN and Bayes Nets [PJ88a]. We have not yet implemented such an approach, but it may well be that Bayes Nets methods can serve as a useful augmentation to PLN for certain sorts of inference (specifically, for inference on networks of knowledge that are relatively static in nature).

We can't use standard Bayes Nets as the primary way of structuring reasoning in CogPrime because CogPrime's knowledge network is loopy. The peculiarities that allow standard Bayes net belief propagation to work in standard loopy Bayes nets, don't hold up in CogPrime, because of the way you have to update probabilities when you're managing a very large network in interaction with a changing world, so that different parts of which get different amounts of focus. So in PLN we use different mechanisms (the "inference trail" mechanism) to avoid "repeated evidence counting" whereas in loopy Bayes nets they rely on the fact that in the standard loopy Bayes net configuration, extra evidence counting occurs in a fairly constant way across the network.

However, when you have within the AtomTable a set of interrelated knowledge items that you know are going to be static for a while, and you want to be able to query them probabilistically, then building a Bayes Net of some sort (i.e. "freezing" part of CogPrime's knowledge network and mapping it into a Bayes Net) may be useful. I.e., one way to accelerate some PLN inference would be:

EFTA00624442

1. Freeze a subnetwork of the AtomTable which is expected not to change a lot in the near future
2. Interpret this subnetwork as a loopy Bayes net, and use standard Bayesian belief propagation to calculate probabilities based on it

This would be a highly efficient form of "background inference" in certain contexts. (Note that this ideally requires an "indefinite Bayes net" implementation that propagates indefinite probabilities through the standard Bayes-net local belief propagation algorithms, but this is not mathematically problematic.)

On the other hand, if you have a very important subset of the Atomspace, then it may be worthwhile to maintain a Bayes net modeling the conditional probabilities between these Atoms, but with a dynamically updated structure.

EFTA00624443

Chapter 37
Pattern Mining

Co-authored with Jade O'Neill

37.1 Introduction

Having discussed inference in depth we now turn to other, simpler but equally important approaches to creating declarative knowledge. This chapters deals with pattern mining – the creation of declarative knowledge representing patterns among other knowledge (which may be declarative, sensory, episodic, procedural, etc.) – and the following chapter deals with speculative concept creation.

Within the scope of pattern mining, we will discuss two basic approaches:

- supervised learning: given a predicate, finding a pattern among the entities that satisfy that predicate.
- unsupervised learning: undirected search for "interesting patterns".

The supervised learning case is easier and we have done a number of experiments using MOSES for supervised pattern mining, on biological (microarray gene expression and SNP) and textual data. In the CogPrime case, the "positive examples" are the elements of the SatisfyingSet of the predicate P, and the "negative examples" are everything else. This can be a relatively straightforward problem if there are enough positive examples and they actually share common aspects ... but some trickiness emerges, of course, when the common aspects are, in each example, complexly intertwined with other aspects.

The unsupervised learning case is considerably trickier. The main problem issue here regards the definition of an appropriate fitness function. We are searching for "interesting patterns." So the question is, what constitutes an interesting pattern?

We will also discuss two basic algorithmic approaches:

- program learning, via MOSES or hillclimbing
- frequent subgraph mining, using greedy algorithms

The value of these various approaches is contingent on the environment and goal set being such that algorithms of this nature can actually recognize relevant patterns in the world and mind. Fortunately, the everyday human world does appear to have the property of possessing multiple relevant patterns that are recognizable using varying levels of sophistication and effort. It has patterns that can be recognized via simple frequent pattern mining, and other patterns that are too subtle for this, and are better addressed by a search-based approach. In order for

EFTA00624444

an environment and goal set to be appropriate for the learning and teaching of a human-level AI, it should have the same property of possessing multiple relevant patterns recognizable using varying levels of subtlety.

37.2 Finding Interesting Patterns via Program Learning

As one important case of pattern mining, we now discuss the use of program learning to find "interesting" patterns in sets of Atoms.

Clearly, "interestingness" is a multidimensional concept. One approach to defining it is empirical, based on observation of which predicates have and have not proved interesting to the system in the past (based on their long-term importance values, i.e. LTI).

In this approach, one has a supervised categorization problem: learn a rule predicting whether a predicate will fall into the *interesting* category or the *uninteresting* category. Once one has learned this rule, which has expressed this rule as a predicate itself, one can then use this rule as the fitness function for evolutionary learning evolution.

There is also a simpler approach, which defines an *objective* notion of interestingness. This objective notion is a weighted sum of two factors:

- Compactness.
- Surprisingness of truth value.

Compactness is easy to understand: all else equal, a predicate embodied in a small Combo tree is better than a predicate embodied in a big one. There is some work hidden here in Combo tree reduction; ideally, one would like to find the smallest representation of a given Combo tree, but this is a computationally formidable problem, so one necessarily approaches it via heuristic algorithms.

Surprisingness of truth value is a slightly subtler concept. Given a Boolean predicate, one can envision two extreme ways of evaluating its truth value (represented by two different types of ProcedureEvaluator). One can use an IndependenceAssumingProcedureEvaluator, which deals with all AND and OR operators by assuming probabilistic independence. Or, one can use an ordinary EffortBasedProcedureEvaluator, which uses dependency information wherever feasible to evaluate the truth values of AND and OR operators. These two approaches will normally give different truth values but, how different? The more different, the more *surprising* is the truth value of the predicate, and the more *interesting* may the predicate be.

In order to explore the power of this kind of approach in a simple context, we have tested pattern mining using MOSES on Boolean predicates as a data mining algorithm on a number of different datasets, including some interesting and successful work in the analysis of gene expression data, and some more experimental work analyzing sociological data from the National Longitudinal Survey of Youth (NLSY) (http://stats.bls.gov/nls/).

A very simple illustrative result from the analysis of the NLSY data is the pattern:

```
OR
    (NOT(MothersAge(X)) AND NOT(FirstSexAge(X)))
    (Wealth(X) AND PIAT(X))
```

where the domain of X are individuals, meaning that:

- being the child of a young mother correlates with having sex at a younger age;

EFTA00624445

- being in a wealthier family correlates with better Math (PIAT) scores;
- the two sets previously described tend to be disjoint.

Of course, many data patterns are several times more complex than the simple illustrative pattern shown above. However, one of the strengths of the evolutionary learning approach to pattern mining is its ability to find simple patterns when they do exist, yet without (like some other mining methods) imposing any specific restrictions on the pattern format.

37.3 Pattern Mining via Frequent/Surprising Subgraph Mining

Probabilistic evolutionary learning is an extremely powerful approach to pattern mining, but, may not always be realistic due to its high computational cost. A cheaper, though also weaker, alternative, is to use frequent subgraph mining algorithms such as [HWP03, KK01], which may straightforwardly be adapted to hypergraphs such as the Atomspace.

Frequent subgraph mining is a port to the graph domain of the older, simpler idea of *frequent itemset mining*, which we now briefly review. There are a number of algorithms in the latter category, the classic is Apriori [AS94], and an alternative is Relim [Bor05] which is conceptually similar but seems to give better performance.

The basic goal of frequent itemset mining is to discover frequent subsets ("itemsets") in a group of sets, whose members are all drawn from some base set of items. One knows that for a set of N items, there are $2^N - 1$ possible subgroups. The algorithm operates in several rounds. Round i heuristically computes frequent i-itemsets (i.e. frequent sets containing i items). A round has two steps: candidate generation and candidate counting. In the candidate generation step, the algorithm generates a set of candidate i-itemsets whose support – the percentage of events in which the item must appear – has not been yet been computed. In the candidate-counting step, the algorithm scans its memory database, counting the support of the candidate itemsets. After the scan, the algorithm discards candidates with support lower than the specified minimum (an algorithm parameter) and retains only the sufficiently frequent i-itemsets. The algorithm reduces the number of tested subsets by pruning apriori those candidate itemsets that cannot be frequent, based on the knowledge about infrequent itemsets obtained from previous rounds. So for instance if $\{A, B\}$ is a frequent 2-itemset then $\{A, B, C\}$ will be considered as a potential 3-itemset, on the contrary if $\{A, B\}$ is not a frequent itemset then $\{A, B, C\}$, as well as any superset of $\{A, B\}$, will be discarded. Although the worst case of this sort of algorithm is exponential, practical executions are generally fast, depending essentially on the support limit.

Frequent subgraph mining follows the same pattern, but instead of a set of items it deals with a group of graphs. There are many frequent subgraph mining algorithms in the literature, but the basic concept underlying nearly all of them is the same: first find small frequent subgraphs. Then seek to find slightly larger frequent patterns encompassing these small ones. Then seek to find slightly larger frequent patterns encompassing *these*, etc. This approach is much faster than something like MOSES, although management of the large number of subgraphs to be searched through can require subtle design and implementation of data structures.

If, instead of an ensemble of small graphs, one has a single large graph like the AtomSpace, one can follow the same approach, via randomly subsampling from the large graph to find the graphs forming the ensemble to be mined from; see [ZH10] for a detailed treatment of this sort of

EFTA00624446

approach. The fact that the AtomSpace is a hypergraph rather than a graph doesn't fundamentally affect the matter since a hypergraph may always be considered a graph via introduction of an additional node for each hyperedge (at the cost of a potentially great multiplication of the number of links).

Frequent subgraph mining algorithms appropriately deployed can find subgraphs which occur repeatedly in the Atomspace, including subgraphs containing Atom-valued variables . Each such subgraph may be represented as a PredicateNode, and frequent subgraph mining will find such PredicateNodes that have surprisingly high truth values when evaluated across the Atomspace. But unlike MOSES when applied as described above, such an algorithm will generally find such predicates only in a "greedy" way.

For instance, a greedy subgraph mining algorithm would be unlikely to find

```
OR
    (NOT(MothersAge(X)) AND NOT(FirstSexAge(X)))
    (Wealth(X) AND PIAT(X))
```

as a surprising pattern in an AtomSpace, unless at least one (and preferably both) of

```
Wealth(X) AND PIAT(X)
```

and

```
NOT(MothersAge(X)) AND NOT(FirstSexAge(X))
```

were surprising patterns in that Atomspace on their own.

37.4 Fishgram

Fishgram is a simple example of an algorithm for finding patterns in an Atomspace, instantiating the general concepts presented in the previous section. It represents patterns as conjunctions (AndLink) of Links, which usually contain variables. It does a greedy search, so it can quickly find many patterns. In contrast, algorithms like MOSES are designed to find a small number of the best patterns. Fishgram works by finding a set of objects that have links in common, so it will be most effective if the AtomSpace has a lot of raw data, with simple patterns. For example, it can be used on the perceptions from the virtual world. There are predicates for basic perceptions (e.g. what kind of object something is, objects being near each other, types of blocks, and actions being performed by the user or the AI).

The details of the Fishgram code and design are not sufficiently general or scalable to serve as a robust, omnipurpose pattern mining solution for CogPrime. However, Fishgram is nevertheless interesting, as an existent, implemented and tested prototype of a greedy frequent/interesting subhypergraph mining system. A more scalable analogous system, with a similar principle of operation, has been outlined and is in the process of being designed at time of writing, but will not be presented here.

37.4.1 Example Patterns

Here is some example output from Fishgram, when run on the virtual agent's memories.

EFTA00624447

```
(AndLink
(EvaluationLink is_edible:PredicateNode (ListLink $1000041))
(InheritanceLink $1000041 Battery:ConceptNode)
)
```

This means a battery which can be "eaten" by the virtual robot. The variable $1000041 refers to the object (battery).

Fishgram can also find patterns containing a sequence of events. In this case, there is a list of EvaluationLinks or InheritanceLinks which describe the objects involved, followed by the sequence of events.

```
(AndLink
   (InheritanceLink $1007703 Battery:ConceptNode)
      (SequentialAndLink
         (EvaluationLink isHolding:PredicateNode (ListLink $1008725 $1007703)))
      )
   )
)
```

This means the agent was holding a battery. $1007703 is the battery, and there is also a variable for the agent itself. Many interesting patterns involve more than one object. This pattern would also include the user (or another AI) holding a battery, because the pattern does not refer to the AI character specifically.

It can find patterns where it performs an action and achieves a goal. There is code to create implications based on these conjunctions. After finding many conjunctions, it can produce ImplicationLinks based on some of them. Here is an example where the AI-controlled virtual robot discovers how to get energy.

```
(ImplicationLink
   (AndLink
      (EvaluationLink is_edible:PredicateNode (ListLink $1011619))
      (InheritanceLink $1011619 Battery:ConceptNode)
   )
   (PredictiveImplicationLink
      (EvaluationLink actionDone:PredicateNode (ListLink
                                    (ExecutionLink eat:GroundedSchemaNode
                                     (ListLink $1011619))))
      (EvaluationLink increased:PredicateNode (ListLink
                                   (EvaluationLink
                                    EnergyDemandGoal:PredicateNode)))
)
)
```

37.4.2 The Fishgram Algorithm

The core Fishgram algorithm, in pseudocode, is as follows:

EFTA00624448

```
initial layer = every pair (relation, binding)

while previous layer is not empty:
   foreach (conjunction, binding) in previous layer:
      let incoming = all (relation, binding) pairs containing an object
in the conjunction
         let possible_next_events = all (event, binding) pairs where
the event happens during or shortly
after the last event in conjunction
            foreach (relation, relation_binding) in incoming
             and possible_next_events:
                  (new_relation , new_conjunction_binding) =
                                    map_to_existing_variables(conjunction ,
                                    binding , relation , relation_binding)
                  if new_relation is already in conjunction, skip it
                  new_conjunction = conjunction + new_relation
                  if new_conjunction has been found already, skip it
                  otherwise, add new_conjunction to the current layer

map_to_existing_variables(conjunction, conjunction_binding,
relation, relation_binding)
   r', s' = a copy of the relation and binding using new variables
      foreach variable v, object o in relation_binding:
         foreach variable v2, object o2 in conjunction_binding:
            if o == o2:
               change r' and s' to use v2 instead of v
```

37.4.3 Preprocessing

There are several preprocessing steps to make it easier for the main Fishgram search to find patterns. There is a list of things that have to be variables. For example, any predicate that refers to object (including agents) will be given a variable so it can refer to any object. Other predicates or InheritanceLinks can be added to a pattern, to restrict it to specific kinds of objects, as shown above. So there is a step which goes through all of the links in the AtomSpace, and records a list of predicates with variables. Such as "X is red" or "X eats Y". This makes the search part simpler, because it never has to decide whether something should be a variable or a specific object.

There is also a filter system, so that things which seem irrelevant can be excluded from the search. There is a combinatorial explosion as patterns become larger. Some predicates may be redundant with each other, or known not to be very useful. It can also try to find only patterns in the AI's "attentional focus", which is much smaller than the whole AtomSpace.

The Fishgram algorithm cannot currently handle patterns involving numbers, although it could be extended to do so. The two options would be to either have a separate discretization step, creating predicates for different ranges of a value. Or alternatively, have predicates for mathematical operators. It would be possible to search for a "splitpoint" like in decision trees.

EFTA00624449

So a number would be chosen, and only things above that value (or only things below that value) would count for a pattern. It would also be possible to have multiple numbers in a pattern, and compare them in various ways. It is uncertain how practical this would be in Fishgram. MOSES is good for finding numeric patterns, so it may be better to simply use those patterns inside Fishgram.

The "increased" predicate is added by a preprocessing step. The goals have a fuzzy TruthValue representing how well the goal is achieved at any point in time, so e.g. the EnergyDemandGoal represents how much energy the virtual robot has at some point in time. The predicate records times that a goal's TruthValue increased. This only happens immediately after doing something to increase it, which helps avoid finding spurious patterns.

37.4.4 Search Process

Fishgram search is breadth-first. It starts with all predicates (or InheritanceLinks) found by the preprocessing step. Then it finds pairs of predicates involving the same variable. Then they are extended to conjunctions of three predicates, and so on. Many relations apply at a specific time, for example the agent being near an object, or an action being performed. These are included in a sequence, and are added in the order they occurred.

Fishgram remembers the examples for each pattern. If there is only one variable in the pattern, an example is a single object; otherwise each example is a vector of objects for each variable in the pattern. Each time a relation is added to a pattern, if it has no new variables, some of the examples may be removed, because they don't satisfy the new predicate. It needs to have at least one variable in common with the previous relations. Otherwise the patterns would combine many unrelated things.

In frequent itemset mining (for example the APRIORI algorithm), there is effectively one variable, and adding a new predicate will often decrease the number of items that match. It can never increase it. The number of possible conjunctions increases with the length, up to some point, after which it decreases. But when mining for patterns with multiple objects there is a much larger combinatorial explosion of patterns. Various criteria can be used to prune the search.

The most basic criterion is the frequency. Only patterns with at least N examples will be included, where N is an arbitrary constant. You can also set a maximum number of patterns allowed for each length (number of relations), and only include the best ones. The next level of the breadth-first search will only search for extensions of those patterns.

One can also use a measure of statistical interestingness, to make sure the relations in a pattern are correlated with each other. There are many spurious frequent patterns, because anything which is frequent will occur together with other things, whether they are relevant or not. For example "breathing while typing" is a frequent pattern, because people breathe at all times. But "moving your hands while typing" is a much more interesting pattern. As people only move their hands some of the time, a measure of correlation would prefer the second pattern. The best measure may be interaction information, which is a generalisation of mutual information that applies to patterns with more than two predicates. An early-stage AI would not have much knowledge of cause and effect, so it would rely on statistical measures to find useful patterns.

EFTA00624450

37.4.5 Comparison to other algorithms

Fishgram is more suitable for OpenCogPrime's purposes than existing graph mining algorithms, most of which were designed with molecular datasets in mind. The OpenCog AtomSpace is a different graph in various ways. For one, there are many possible relations between nodes (much like in a semantic network). Many relations involve more than two objects, and there are also properties predicates about a single object. So the relations are effectively directed links of varying arity. It also has events, and many states can change over time (e.g. an egg changes state while it's cooking). Fishgram is designed for general knowledge in an embodied agent.

There are other major differences. Fishgram uses a breadth-first search, rather than depth-first search like most graph mining algorithms. And it does an "embedding-based" search, searching for patterns that can be embedded multiple times in a large graph. Molecular datasets have many separate graphs for separate molecules, but the embodied perceptions are closer to a single, fairly well-connected graph. Depth-first search would be very slow on such a graph, as there are many very long paths through the graph, and the search would mostly find those. Whereas the useful patterns tend to be compact and repeated many times.

Lastly the design of Fishgram makes it easy to experiment with multiple different scoring functions, from simple ones like frequency to much more sophisticated functions such as interaction information.

As mentioned above, the current implementation of Fishgram is not sufficiently scalable to be utilized for general-purpose Atomspaces. The underlying data structure within Fishgram, used to store recognized patterns, would need to be replaced, which would lead to various other modifications within the algorithm. But, the general principle and approach illustrated by Fishgram will be persisted in any more scalable reimplementation.

EFTA00624451

Chapter 38
Speculative Concept Formation

38.1 Introduction

One of the hallmarks of general intelligence is its capability to deal with novelty in its environment and/or goal-set. And dealing with novelty intrinsically requires creating novelty. It's impossible to efficiently handle new situations without creating new ideas appropriately. Thus, in any environment complex and dynamic enough to support human-like general intelligence (or any other kind of highly powerful general intelligence), the creation of novel ideas will be paramount. New idea creation takes place in OpenCog via a variety of methods – e.g. inside MOSES which creates new program trees, PLN which creates new logical relationships, ECAN which creates new associative relationships, etc. But there is also a role for explicit, purposeful creation of new Atoms representing new concepts, outside the scope of these other learning mechanisms.

The human brain gets by, in adulthood, without creating *that many* new neurons – although neurogenesis does occur on an ongoing basis. But this is achieved only via great redundancy, because for the brain it's cheaper to maintain a large number of neurons in memory at the same time, than to create and delete neurons. Things are different in a digital computer: memory is more expensive but creation and deletion of object is cheaper. Thus in CogPrime, forgetting and creation of Atoms is a regularly occurring phenomenon. In this chapter we discuss a key class of mechanisms for Atom creation, "speculative concept formation." Further methods will be discussed in following chapters.

The philosophy underlying CogPrime's speculative concept formation is that new things should be created from pieces of good old things (a form of "evolution", broadly construed), and that probabilistic extrapolation from experience should be used to guide the creation of new things (inference). It's clear that these principles are necessary for the creation of new mental forms but it's not obvious that they're sufficient: this is a nontrivial hypothesis, which may also be considered a family of hypotheses since there are many different ways to do extrapolation and intercombination. In the context of mind-world correspondence, the implicit assumption underlying this sort of mechanism is that the relevant patterns in the world can often be combined to form other relevant patterns. The everyday human world does quite markedly display this kind of combinatory structure, and such a property seems basic enough that it's appropriate for use as an assumption underlying the design of cognitive mechanisms.

In CogPrime we have introduced a variety of heuristics for creating new Atoms - especially ConceptNodes - which may then be reasoned on and subjected to implicit (via attention allo-

EFTA00624452

cation) and explicit (via the application of evolutionary learning to predicates obtained from concepts via "concept predicatization") evolution. Among these are the node logical operators described in the book Probabilistic Logic Networks, which allow the creation of new concepts via AND, OR, XOR and so forth. However, logical heuristics alone are not sufficient. In this chapter we will review some of the nonlogical heuristics that are used for speculative concept formation. These operations play an important role in creativity - to use cognitive-psychology language, they are one of the ways that CogPrime implements the process of blending, which Falconnier and Turner (2003) have argued is key to human creativity on many different levels. Each of these operations may be considered as implicitly associated with a hypothesis that, in fact, the everyday human world tends to assign utility to patterns that are combinations of other patterns produced via said operation.

An evolutionary perspective may also be useful here, on a technical level as well as philosophically. As noted in *The Hidden Pattern* and hinted at in Chapter 3 of Part 1, one way to think about an AGI system like CogPrime is as a huge evolving ecology. The AtomSpace is a biosphere of sorts, and the mapping from Atom types into species has some validity to it (though not complete accuracy: Atom types do not compete with each other; but they do reproduce with each other, and according to most of the reproduction methods in use, Atoms of differing type cannot cross-reproduce). Fitness is defined by importance. Reproduction is defined by various operators that produce new Atoms from old, including the ones discussed in this chapter, as well as other operators such as inference and explicit evolutionary operators.

New ConceptNode creation may be triggered by a variety of circumstances. If two ConceptNodes are created for different purposes, but later the system finds that most of their meanings overlap, then it may be more efficient to merge the two into one. On the other hand, a node may become overloaded with different usages, and it is more useful to split it into multiple nodes, each with a more consistent content. Finally, there may be patterns across large numbers of nodes that merit encapsulation in individual nodes. For instance, if there are 1000 fairly similar ConceptNodes, it may be better not to merge them all together, but rather to create a single node to which they all link, reifying the category that they collectively embody.

In the following sections, we will begin by describing operations that create new ConceptNodes from existing ones on a local basis: by mutating individual ConceptNodes or combining pairs of ConceptNodes. Some of these operations are inspired by evolutionary operators used in the GA, others are based on the cognitive psychology concept of "blending." Then we will turn to the use of clustering and formal concept analysis algorithms inside CogPrime to refine the system's knowledge about existing concepts, and create new concepts.

38.2 Evolutionary Concept Formation

A simple and useful way to combine ConceptNodes is to use GA-inspired evolutionary operators: crossover and mutation. In mutation, one replaces some number of a Node's links with other links in the system. In crossover, one takes two nodes and creates a new node containing some links from one and some links from another.

More concretely, to cross over two ConceptNodes X and Y, one may proceed as follows (in short clustering the union of X and Y):

- Create a series of empty nodes $Z_1, Z_2, \ldots, Z_k$

EFTA00624453

- Form a "link pool" consisting of all X's links and all Y's links, and then divide this pool into clusters (clustering algorithms will be described below).
- For each cluster with significant cohesion, allocate the links in that cluster to one of the new nodes Z_i

On the other hand, to mutate a ConceptNode, a number of different mutation processes are reasonable. For instance, one can

- Cluster the links of a Node, and remove one or more of the clusters, creating a node with less links
- Cluster the links, remove one or more clusters, and then add new links that are similar to the links in the remaining clusters

The EvolutionaryConceptFormation MindAgent selects pairs of nodes from the system, where the probability of selecting a pair is determined by

- the average importance of the pair
- the degree of similarity of the pair
- the degree of association of the pair

(Of course, other heuristics are possible too). It then crosses over the pair, and mutates the result.

Note that, unlike in some GA implementations, the parent node(s) are retained within the system; they are not replaced by the children. Regardless of how many offspring they generate by what methods, and regardless of their age, all Nodes compete and cooperate freely forever according to the fitness criterion defined by the importance updating function. The entire AtomSpace may be interpreted as a large evolutionary, ecological system, and the action of CogPrime dynamics, as a whole, is to create fit nodes.

A more advanced variant of the EvolutionaryConceptFormation MindAgent would adapt its mutation rate in a context-dependent way. But our intuition is that it is best to leave this kind of refinement for learned cognitive schemata, rather than to hard-wire it into a MindAgent. To encourage the formation of such schemata, one may introduce elementary schema functions that embody the basic node-level evolutionary operators:

```
ConceptNode ConceptCrossover(ConceptNode A, ConceptNode B)

ConceptNode mutate(ConceptNode A, mutationAmount m)
```

There will also be a role for more abstract schemata that utilize these. An example cognitive schema of this sort would be one that said: "When all my schema in a certain context seem unable to achieve their goals, then maybe I need new concepts in this context, so I should increase the rate of concept mutation and crossover, hoping to trigger some useful concept formation."

As noted above, this component of CogPrime views the whole AtomSpace as a kind of genetic algorithm - but the fitness function is "ecological" rather than fixed, and of course the crossover and mutation operators are highly specialized. Most of the concepts produced through evolutionary operations are going to be useless nonsense, but will be recognized by the importance updating process and subsequently forgotten from the system. The useful ones will link into other concepts and become ongoing aspects of the system's mind. The importance updating process amounts to fitness evaluation, and it depends implicitly on the sum total of the cognitive processes going on in CogPrime.

EFTA00624454

To ensure that importance updating properly functions as fitness evaluation, it is critical that evolutionarily-created concepts (and other speculatively created Atoms) always comprise a small percentage of the total concepts in the system. This guarantees that importance will serve as a meaningful "fitness function" for newly created ConceptNodes. The reason for this is that the importance measures how useful the newly created node is, in the context of the previously existing Atoms. If there are too many speculative, possibly useless new ConceptNodes in the system at once, the importance becomes an extremely noisy fitness measure, as it's largely measuring the degree to which instances of new nonsense fit in with other instances of new nonsense. One may find interesting self-organizing phenomena in this way, but in an AGI context we are not interested in undirected spontaneous pattern-formation, but rather in harnessing self-organizing phenomena toward system goals. And the latter is achieved by having a modest but not overwhelming amount of speculative new nodes entering into the system.

Finally, as discussed earlier, evolutionary operations on maps may occur naturally and automatically as a consequence of other cognitive operations. Maps are continually mutated due to fluctuations in system dynamics; and maps may combine with other maps with which they overlap, as a consequence of the nonlinear properties of activation spreading and importance updating. Map-level evolutionary operations are not closely tied to their Atom-level counterparts (a difference from e.g. the close correspondence between map-level logical operations and underlying Atom-level logical operations).

38.3 Conceptual Blending

The notion of Conceptual Blending (aka Conceptual Integration) was proposed by Gilles Fauconnier and Mark Turner [FT02] as general theory of cognition. According to this theory, the basic operation of creative thought is the "blend" in which elements and relationships from diverse scenarios are merged together in a judicious way. As a very simple example, we may consider the blend of "tower" and "snake" to form a new concept of "snake tower" (a tower that looks somewhat like a snake). However, most examples of blends will not be nearly so obvious. For instance, the complex numbers could be considered a blend between 2D points and real numbers. Figure 38.1 gives a conceptual illustration of the blending process.

The production of a blend is generally considered to have three key stages (elucidated via the example of building a snake-tower out of blocks):

- *composition*: combining judiciously chosen elements from two or more concept inputs
  - *Example*: Taking the "buildingness" and "verticalness" of a tower, and the "head" and "mouth" and "tail" of a snake
- *completion*: adding new elements from implicit background knowledge about the concept inputs
  - *Example*: Perhaps a mongoose-building will be built out of blocks, poised in a position indicating it is chasing the snake-tower (incorporating the background knowledge that mongeese often chase snakes)
- *elaboration*: fine-tuning, which shapes the elements into a new concept, guided by the desire to optimize certain criteria

EFTA00624455

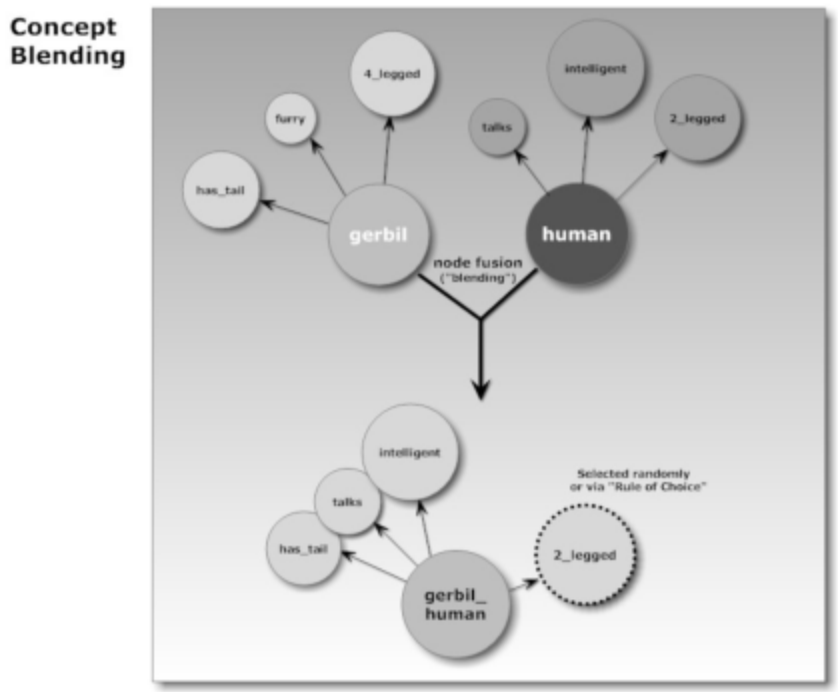


Fig. 38.1: Conceptual Illustration of Conceptual Blending

– *Example*: The tail of the snake-tower is a part of the building that rests on the ground, and connects to the main tower. The head of the snake-tower is a portion that sits atop the main tower, analogous to the restaurant atop the Space Needle.

The "judiciousness" in the composition phase may be partially captured in CogPrime via PLN inference, via introducing a "consistency criterion" that the elements chosen as part of the blend should not dramatically decrease in confidence after the blend's relationships are submitted to PLN inference. One especially doesn't want to choose mutually contradictory elements from the two inputs. For instance one doesn't want to choose "alive" as an element of "snake", and "non-living" as an element of "building." This kind of contradictory choice can be ruled out by inference, because after very few inference steps, this choice would lead to a drastic confidence reduction for the InheritanceLinks to both "alive" and "non-living."

Aside from consistency, some other criteria considered relevant to evaluating the quality of a blend, are:

- *topology principle* that relations in the blend should match the relations of their counterparts in other concepts related to the concept inputs
- *web principle* that the representation in the blended space should maintain mappings to the concept inputs
- *unpacking principle* that, given a blended concept, the interpreter should be able to infer things about other related concepts
- *good reason principle* that there should be simple explanations for the elements of the blend

EFTA00624456

- *metonymic tightening* that when metonymically related elements are projected into the blended space, there is pressure to compress the "distance" between them.

While vague-sounding in their verbal formulations, these criteria have been computationally implemented in the Sapper system, which uses blending theory to model analogy and metaphor [VK94, VO07]; and in a different form in [Car06]'s framework for computational creativity. In CogPrime terms, these various criteria essentially boil down to: **the new, blended concept should get a lot of interesting links**.

One could implement blending in CogPrime very straightforwardly via an evolutionary approach: search the space of possible blends, evaluating each one according to its consistency but also the STI that it achieves when released into the Atomspace. However, this will be quite computationally expensive, so a wiser approach is to introduce heuristics aimed at increasing the odds of producing important blends.

A simple heuristic is to calculate, for each candidate blend, the amount of STI that the blend would possess N cycles later if, at the current time, it was given a certain amount of STI. A blend that would accumulate more STI in this manner may be considered more promising, because this means that its components are more richly interconnected. Further, this heuristic may be used as a guide for greedy heuristics for creating blends: e.g. if one has chosen a certain element A of the first blend input, then one may seek an element B of the second blend input that has a strong Hebbian link to A (if such a B exists).

However, it may also be interesting to pursue different sorts of heuristics, using information-theoretic or other mathematical criteria to preliminarily filter possible blends before they are evaluated more carefully via integrated cognition and importance dynamics.

38.3.1 Outline of a CogPrime Blending Algorithm

A rough outline of a concept blending algorithm for CogPrime is as follows:

- Choose a pair of concepts C1 and C2, which have a nontrivially-strong HebbianLink between them, but not an extremely high-strength SimilarityLink between them (i.e. the concepts should have something to do with each other, but not be extremely similar; blends of extremely similar things are boring). These parameters may be twiddled.
- Form a new concept C3, which has some of C1's links, and some of C2's links
- If C3 has obvious contradictions, resolve them by pruning links. (For instance, if C1 inherits from alive to degree .9 and C2 inherits from alive to degree .1, then one of these two TruthValue versions for the inheritance link from alive, has got to be pruned...)
- For each of C3's remaining links L, make a vector indicating everything it or its targets are associated with (via HebbianLinks or other links). This is basically a list of "what's related to L". Then, assess whether there are a lot of common associations to the links L that came from C1 and the links L that came from C2
- If the filter in step 4 is passed, then let the PLN forward chainer derive some conclusions about C3, and see if it comes up with anything interesting (e.g. anything with surprising truth value, or anything getting high STI, etc.)

Steps 1 and 2 should be repeated over and over. Step 5 is basically "cognition as usual" – i.e. by the time the blended concept is thrown into the Atomspace and subjected to Step 5, it's being treated the same as any other ConceptNode.

EFTA00624457

The above is more of a meta-algorithm than a precise algorithm. Many avenues for variation exist, including

- Step 1: heuristics for choosing what to try to blend
- Step 3: how far do we go here, at removing contradictions? Do we try simple PLN inference to see if contradictions are unveiled, or do we just limit the contradiction-check to seeing if the same exact link is given different truth-values?
- Step 4: there are many different ways to build this association-vector. There are also many ways to measure whether a set of association-vectors demonstrates "common associations". Interaction information [Bel03] is one fancy way; there are also simpler ones.
- Step 5: there are various ways to measure whether PLN has come up with anything interesting

38.3.2 Another Example of Blending

To illustrate these ideas further, consider the example of the SUV – a blend of "Car" and "Jeep"

Among the relevant properties of Car are:

- appealing to ordinary consumers
- fuel efficient
- fits in most parking spots
- easy to drive
- 2 wheel drive

Among the relevant properties of Jeep are:

- 4 wheel drive
- rugged
- capable of driving off road
- high clearance
- open or soft top

Obviously, if we want to blend Car and Jeep, we need to choose properties of each that don't contradict each other. We can't give the Car/Jeep both 2 wheel drive and 4 wheel drive. 4 wheel drive wins for Car/Jeep because sacrificing it would get rid of "capable of driving off road", which is critical to Jeep-ness; whereas sacrificing 2WD doesn't kill anything that's really critical to car-ness.

On the other hand, having a soft top would really harm "appealing to consumers", which from the view of car-makers is a big part of being a successful car. But getting rid of the hard top doesn't really harm other aspects of jeep-ness in any series way.

However, what really made the SUV successful was that "rugged" and "high clearance" turned out to make SUVs look funky to consumers, thus fulfilling the "appealing to ordinary consumers" feature of Car. In other words, the presence of the links

- looks funky → appealing to ordinary consumers
- rugged & high clearance → looks funky

EFTA00624458

made a big difference. This is the sort of thing that gets figured out once one starts doing PLN inference on the links associated with a candidate blend.

However, if one views each feature of the blend as a probability distribution over concept space – for instance indicating how closely associated each concept is with that feature (e.g. via HebbianLinks) then we see that the mutual information (and more generally interaction information) between the features of the blend, is a quick estimate of how likely it is that inference will lead to interesting conclusions via reasoning about the combination of features that the blend possesses.

38.4 Clustering

Next, a different method for creating new ConceptNodes in CogPrime is using clustering algorithms. There are many different clustering algorithms in the statistics and data mining literature, and no doubt many of them could have value inside CogPrime. We have experimented with several different clustering algorithms in the CogPrime context, and have selected one, which we call Omniclust [GCPM06], based on its generally robust performance on high-volume, *noisy* data. However, other methods such as EM (Expectation-Maximization) clustering [WF05] would likely serve the purpose very well also.

In the above discussion on evolutionary concept creation, we mentioned the use of a clustering algorithm to cluster links. The same algorithm we describe here for clustering ConceptNodes directly and creating new ConceptNodes representing these clusters, can also be used for clustering links in the context of node mutation and crossover.

The application of Omniclust or any other clustering algorithm for ConceptNode creation in CogPrime is simple. The clustering algorithm is run periodically, and the most significant clusters that it finds are embodied as ConceptNodes, with InheritanceLinks to their members. If these significant clusters have subclusters also identified by Omniclust, then these subclusters are also made into ConceptNodes, etc., with InheritanceLinks between clusters and subclusters.

Clustering technology is famously unreliable, but this unreliability may be mitigated somewhat by using clusters as initial guesses at concepts, and using other methods to refine the clusters into more useful concepts. For instance, a cluster may be interpreted as a disjunctive predicate, and a search may be made to determine sub-disjunctions about which interesting PLN conclusions may be drawn.

38.5 Concept Formation via Formal Concept Analysis

Another approach to concept formation is an uncertain version of Formal Concept Analysis [GSW05]. There are many ways to create such a version, here we describe one approach we have found interesting, called Fuzzy Concept Formation (FCF).

The general formulation of FCF begins with n objects $O_1, ..., O_n$, m basic attributes $a_1, ..., a_m$, and information that object O_i possesses attribute a_j to degree $w_{ij} \in [0, 1]$. In CogPrime, the objects and attributes are Atoms, and w_{ij} is the strength of the InheritanceLink pointing from O_i to a_j.

EFTA00624459

In this context, we may define a **concept** as a fuzzy set of objects, and a **derived attribute** as a fuzzy set of attributes.

Fuzzy concept formation (FCF) is, then, a process that produces N "concepts" $C_{n+1}, ..., C_{n+N}$ and M "derived attributes" $d_{m+1}, ..., d_{m+M}$, based on the initial set of objects and attributes. We can extend the weight matrix w_{ij} to include entries involving concepts and derived attributes as well, so that e.g. $w_{n+3,m+5}$ indicates the degree to which concept C_{n+3} possesses derived attribute d_{m+5}.

The learning engine underlying FCF is a clustering algorithm $clust = clust(X_1, ..., X_r; b)$ which takes in r vectors $X_r \in [0,1]^n$ and outputs b or fewer clusters of these vectors. The overall FCF process is independent of the particular clustering algorithm involved, though the interestingness of the concepts and attributes formed will of course vary widely based on the specific clustering algorithm. Some clustering algorithms will work better with large values of b, others with smaller values of b.

We then define the process $form_concepts(b)$ to operate as follows. Given a set $S = S_1, ..., S_k$ containing objects, concepts, or a combination of objects and concepts, and an attribute vector w_i of length h with entries in $[0,1]$ corresponding to each S_i, one applies $clust$ to find b clusters of attribute vectors w_i: $B_1, ..., B_b$. Each of these clusters may be considered as a fuzzy set, for instance by considering the membership of x in cluster B to be $2^{-d(x,centroid(B))}$ for an appropriate metric d. These fuzzy sets are the b concepts produced by $form_concepts(b)$.

38.5.1 Calculating Membership Degrees of New Concepts

The degree to which a concept defined in this way possesses an attribute, may be defined in a number of ways, maybe the simplest is: weighted-summing the degree to which the members of the concept possess the attribute. For instance, to figure out the degree to which beautiful women (a concept) are insane (an attribute), one would calculate

$$\frac{\sum_{w \in beautiful_women} \chi_{beautiful_women}(w) \chi_{insane}(w)}{\sum_{w \in beautiful_women} \chi_{beautiful_women}(w)}$$

where $\chi_X(w)$ denotes the fuzzy membership degree of w in X. One could probably also consider *ExtensionalInheritance beautiful_ women insane.*

38.5.2 Forming New Attributes

One may define an analogous process $form_attributes(b)$ that begins with a set $A = A_1, ..., A_k$ containing (basic and/or derived) attributes, and a column vector

$$\begin{pmatrix} w_{1i} \\ ... \\ w_{hi} \end{pmatrix}$$

EFTA00624460

of length h with entries in $[0, 1]$ corresponding to each A_i (the column vector tells the degrees to which various objects possess the attributes A_i). One applies *clust* to find b clusters of vectors v_i: $B_1, ..., B_b$. These clusters may be interpreted as fuzzy sets, which are derived attributes.

38.5.2.1 Calculating Membership Degrees of New, Derived Attributes

One must then defines the degree to which an object or concept possesses a derived attribute. One way to do this is using a geometric mean. For instance, suppose there is a derived attribute formed by combining the attributes *vain*, *selfish* and *egocentric*. Then, the degree to which the concept *banker* possesses this new derived attribute could be defined by

$$\frac{\sum_{b \in banker} \chi_{banker}(b) \left(\chi_{vain}(b) \chi_{selfish}(b) \chi_{egocentric}(b)\right)^{1/3}}{\sum_{b \in banker} \chi_{banker}(b)}$$

38.5.3 Iterating the Fuzzy Concept Formation Process

Given a set S of concepts and/or objects with a set A of attributes, one may define

- $append_concepts(S', S)$ as the result of adding the concepts in the set S' to S, and evaluating all the attributes in A on these concepts, to get an expanded matrix w
- $append_attributes(A', A)$ as the result of adding the attributes in the set A' to A, and evaluating all the attributes in A' on the concepts and objects in S, to get an expanded matrix w
- $collapse(S, A)$ is the result of taking (S, A) and eliminating any concept or attribute that has distance less than ϵ from some other concept or attribute that comes before it in the lexicographic ordering of concepts or attributes. I.e., *collapse* removes near-duplicate concepts or attributes.

Now, one may begin with a set S of objects and attributes, and iteratively run a process such as

```
b = r^c \\e.g. r=2, or r=1.5
while(b>1) {
   S = append_concepts(S, form_concepts(S,b))
   S = collapse(S)
   S = append_attributes(S, form_attributes(S,b))
   S = collapse(S)
   b = b/r
}
```

with c corresponding to the number of iterations. This will terminate in finite time with a finitely expanded matrix w containing a number of concepts and derived attributes in addition to the original objects and basic attributes.

Or, one may look at

```
while(S is different from old_S) {
```

EFTA00624461

```
    old_S = S
    S = add_concepts(S, form_concepts(S,b))
    S = collapse(S)
    S = add_attributes(S, form_attributes(S,b))
    S = collapse(S)
}
```

This second version raises the mathematical question of the speed with which it will terminate (as a function of ϵ). I.e., when does the concept and attribute formation process converge, and how fast? This will surely depend on the clustering algorithm involved.

EFTA00624462

Section VI
Integrative Learning

Chapter 39
Dimensional Embedding

39.1 Introduction

Among the many key features of the human brain omitted by typical formal neural network models, one of the foremost is the brain's three-dimensionality. The brain is not just a network of neurons arranged as an abstract graph; it's a network of neurons arranged in three-dimensional space, and making use of this three-dimensionality directly and indirectly in various ways and for various purposes. The somatosensory cortex contains a geometric map reflecting, approximatively, the geometric structure of parts of the body. The Visual cortex uses the 2D layout of cortical sheets to reflect the geometric structure of perceived space; motion detection neurons often fire in the actual physical direction of motion, etc. The degree to which the brain uses 2D and 3D geometric structure to reflect conceptual rather than perceptual or motoric knowledge is unclear, but we suspect considerable. One well-known idea in this direction is the "self-organizing map" or Kohonen net [Koh01], a highly effective computer science algorithm that performs automated classification and clustering via projecting higher-dimensional (perceptual, conceptual or motoric) vectors into a simulated 2D sheet of cortex.

It's not clear that the exploitation of low-dimensional geometric structure is something an AGI system necessarily *must* support – there are always many different approaches to any aspect of the AGI problem. However, the brain does make clear that exploitation of this sort of structure is a powerful way to integrate various useful heuristics. In the context of mind-world correspondence theory, there seems clear potential value in having a mind mirror the dimensional structure of the world, at some level of approximation.

It's also worth emphasizing that the brain's 3D structure has minuses as well as plusses – one suspects it complexifies and constrains the brain, along with implicitly suggesting various useful heuristics. Any mathematical graph can be represented in 3 dimensions without links crossing (unlike in 2 dimensions), but that doesn't mean the representation will always be efficient or convenient – sometimes it may result in conceptually related, and/or frequently interacting, entities being positioned far away from each other geometrically. Coupled with noisy signaling methods such as the brain uses, this sometime lack of alignment between conceptual/pragmatic and geometric structure can lead to various sorts of confusion (i.e. when neuron A sends a signal to physical distant neurons B, this may cause various side-effects along the path, some of which wouldn't happen if A and B were close to each other).

In the context of CogPrime, the most extreme way to incorporate a brain-like 3D structure would be to actually embed an Atomspace in a bounded 3D region. Then the Atomspace would

EFTA00624466

be geometrically something like a brain, but with abstract nodes and links (some having explicit symbolic content) rather than purely sub symbolic neurons. This would not be a ridiculous thing to do, and could yield interesting results. However, we are unsure this would be an optimal approach. Instead we have opted for a more moderate approach: couple the non-dimensional Atomspace with a dimensional space, containing points corresponding to Atoms. That is, we perform an embedding of Atoms in the OpenCog AtomSpace into n-dimensional space – a judicious transformation of (hyper)graphs into vectors.

This embedding has applications to PLN inference control, and to the guidance of instance generation in PEPL learning of Combo trees. It is also, in itself, a valuable and interesting heuristic for sculpting the *link topology* of a CogPrime AtomSpace. The basic dimensional embedding algorithm described here is fairly simple and not original to CogPrime, but it has not previously been applied in any similar context.

The intuition underlying this approach is that there are some cases (e.g. PLN control, and PEPL guidance) where dimensional geometry provides a useful heuristic for constraining a huge search space, via providing a compact way of storing a large amount of information. Dimensionally embedding Atoms lets CogPrime be dimensional like the brain when it needs to be, yet with the freedom of nondimensionality the rest of the time. This dual strategy is one that may be of value for AGI generally beyond the CogPrime design, and is somewhat related to (though different in detail from) the way the CLARION cognitive architecture [SZ04] maps declarative knowledge into knowledge appropriate for its neural net layer.

There is an obvious way to project CogPrime Atoms into n-dimensional space, by assigning each Atom a numerical vector based on the weights of its links. But this is not a terribly useful approach, because the vectors obtained in this way will live, potentially, in millions- or billions-dimensional space. The approach we describe here is a bit different. We are defining more specific embeddings, each one based on a particular link type or set of link types. And we are doing the embedding into a space whose dimensionality is *high but not too high*, e.g. n=50. This moderate dimensional space could then be projected down into a lower dimensional space, like a 3D space, if needed.

The philosophy underlying the ideas proposed here is similar to that underlying Principal Components Analysis (PCA) in statistics [Jol10]. The n-dimensional spaces we define here, like those used in PCA or LSI (for Latent Semantic Indexing [LMDK07]), are defined by sets of *orthogonal concepts* extracted from the original space of concepts. The difference is that PCA and LSI work on spaces of entities defined by feature vectors, whereas the methods described here work for entities defined as nodes in weighted graphs. There is no precise notion of *orthogonality* for nodes in a weighted graph, but one can introduce a reasonable proxy.

39.2 Link Based Dimensional Embedding

In this section we define the type of dimensional embedding that we will be talking about. For concreteness we will speak in terms of CogPrime nodes and links, but the discussion applies much more generally than that.

A *link based dimensional embedding* is defined as a mapping that maps a set of CogPrime Atoms into points in an n-dimensional real space, by:

- mapping link strength into coordinate values in an embedding space, and

EFTA00624467

- representing nodes as points in this embedding space, using the coordinate values defined by the strengths of their links.

In the usual case, a dimensional embedding is formed from links of a single type, or from links whose types are very closely related (e.g. from all symmetrical logical links).

Mapping all the link strengths of the links of a given type into coordinate values in a dimensional space is a simple, but not a very effective strategy. The approach described here is based on strategically choosing a subset of particular links and forming coordinate values from them. The choice of links is based on the desire for a correspondence between the metric structure of the embedding space, and the metric structure implicit in the weights of the links of the type being embedded. The basic idea of metric preservation is depicted in Figure 39.1.

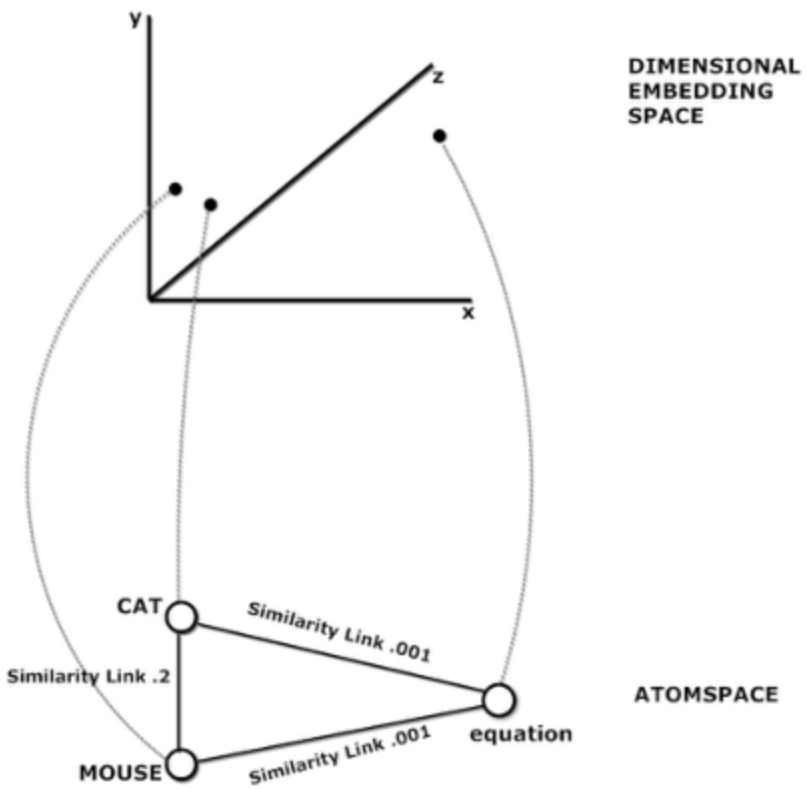


Fig. 39.1: **Metric-Preserving Dimensional Embedding**. The basic idea of the sort of embedding described here is to map Atoms into numerical vectors, in such a way that, on average, distance between Atoms roughly correlates with distance between corresponding vectors. (The picture shows a 3D embedding space for convenience, but in reality the dimension of the embedding space will generally be much higher.)

More formally, let $proj(A)$ denote the point in R^n corresponding to the Atom A. Then if, for example, we are doing an embedding based on SimilarityLinks, we want there to be a strong correlation (or rather anticorrelation) between:

EFTA00624468

```
(SimilarityLink A B).tv.s
```

and

$$d_E(proj(A), proj(B))$$

where d_E denotes the Euclidean distance on the embedding space. This is a simple case because SimilarityLink is symmetric. Dealing with asymmetric links like InheritanceLinks is a little subtler, and will be done below in the context of inference control.

Larger dimensions generally allow greater correlation, but add complexity. If one chooses the dimensionality equal to the number of nodes in the graph, there is really no point in doing the embedding. On the other hand, if one tries to project a huge and complex graph into 1 or 2 dimensions, one is bound to lose a lot of important structure. The optimally useful embedding will be into a space whose dimension is *large but not too large*.

For internal CogPrime inference purposes, we should generally use a moderately high-dimensional embedding space, say n=50 or n=100.

39.3 Harel and Koren's Dimensional Embedding Algorithm

Our technique for embedding CogPrime Atoms into high-dimensional space is based on an algorithm suggested by David Harel and Yehuda Koren [HK02]. Their work is concerned with visualizing large graphs, and they propose a two-phase approach:

1. embed the graph into a high-dimensional real space
2. project the high-dimensional points into 2D or 3D space for visualization

In CogPrime, we don't always require the projection step (step 2); our focus is on the initial embedding step. Harel and Koren's algorithm for dimensional embedding (step 1) is directly applicable to the CogPrime context.

Of course this is not the only embedding algorithm that would be reasonable to use in an CogPrime context; it's just one possibility that seems to make sense.

Their algorithm works as follows.

Suppose one has a graph with symmetric weighted links. Further, assume that between any two nodes in the graph, there is a way to compute the weight that a link between those two nodes would have, even if the graph in fact doesn't contain a link between the two nodes.

In the CogPrime context, for instance, the nodes of the graph may be ConceptNodes, and the links may be SimilarityLinks. We will discuss the extension of the approach to deal with asymmetric links like InheritanceLinks, later on.

Let n denote the dimension of the embedding space (e.g. $n = 50$). We wish to map graph nodes into points in R^n, in such a way that the weight of the graph link between A and B correlates with the distance between $proj(A)$ and $proj(B)$ in R^n.

39.3.1 Step 1: Choosing Pivot Points

Choose n "pivot points" that are roughly uniformly distributed across the graph.

EFTA00624469

To do this, one chooses the first pivot point at random and then iteratively chooses the i'th point to be maximally distant from the previous $(i-1)$ points chosen.

One may also use additional criteria to govern the selection of pivot points. In CogPrime, for instance, we may use *long-term stability* as a secondary criterion for selecting Atoms to serve as pivot points. Greater computational efficiency is achieved if the pivot-point Atoms don't change frequently.

39.3.2 Step 2: Similarity Estimation

Estimate the similarity between each Atom being projected, and the n pivot Atoms.

This is expensive. However, the cost is decreased somewhat in the CogPrime case by caching the similarity values produced in a special table (they may not be important enough otherwise to be preserved in CogPrime). Then, in cases where neither the pivot Atom nor the Atom being compared to it have changed recently, the cached value can be reused.

39.3.3 Step 3: Embedding

Create an n-dimensional space by assigning a coordinate axis to each pivot Atom. Then, for an Atom i, the i'th coordinate value is given by its similarity to the i'th pivot Atom.

After this step, one has transformed one's graph into a collection of n-dimensional vectors. WIKISOURCE:EmbeddingBasedInference

39.4 Embedding Based Inference Control

One important application for dimensional embedding in CogPrime is to help with the control of

- Logical inference
- Direct evaluation of logical links

We describe how it can be used specifically to stop the CogPrime system from continually trying to make the same unproductive inferences.

To understand the problem being addressed, suppose the system tries to evaluate the strength of the relationship

```
SimilarityLink foot toilet
```

Assume that no link exists in the system representing this relationship.

Here "foot" and "toilet" are hypothetical ConceptNodes that represent aspects of the concepts of foot and toilet respectively. In reality these concepts might well be represented by complex maps rather than individual nodes.

EFTA00624470

Suppose the system determines that the strength of this Link is very close to zero. Then (depending on a threshold in the MindAgent), it will probably not create a SimilarityLink between the "foot" and "toilet" nodes.

Now, suppose that a few cycles later, the system again tries to evaluate the strength of the same Link,

```
SimilarityLink foot toilet
```

Again, very likely, it will find a low strength and not create the Link at all.

The same problem may occur with InheritanceLinks, or any other (first or higher order) logical link type.

Why would the system try, over and over again, to evaluate the strength of the same nonexistent relationship? Because the control strategies of the current forward-chaining inference and pattern mining MindAgents are simple by design. These MindAgents work by selecting Atoms from the AtomTable with probability proportional to importance, and trying to build links between them. If the foot and toilet nodes are both important at the same time, then these MindAgents will try to build links between them - regardless of how many times they've tried to build links between these two nodes in the past and failed.

How do we solve this problem using dimensional embedding? Generally:

- one will need a different embedding space for each link type for which one wants to prevent repeated attempted inference of useless relationships. Sometimes, very closely related link types might share the same embedding space; this must be decided on a case-by-case basis.
- in the embedding space for a link type L, one only embeds Atoms of a type that can be related by links of type L

It is too expensive to create a new embedding very often. Fortunately, when a new Atom is created or an old Atom is significantly modified, it's easy to reposition the Atom in the embedding space by computing its relationship to the pivot Atoms. Once enough change has happened, however, new pivot Atoms will need to be recomputed, which is a substantial computational expense. We must update the pivot point set every N cycles, where N is large; or else, whenever the total amount of change in the system has exceeded a certain threshold.

Now, how is this embedding used for inference control? Let's consider the case of similarity first. Quite simply, one selects a pair of Atoms (A,B) for SimilarityMining (or inference of a SimilarityLink) based on some criterion such as, for instance:

```
importance(A) * importance(B) * simproj(A,B)
```

where

```
distproj(A,B) = dE( proj(A) , proj(B) )

simproj = 2-c*distproj
```

and c is an important tunable parameter.

What this means is that, if A and B are far apart in the SimilarityLink embedding space, the system is unlikely to try to assess their similarity.

There is a tremendous space efficiency of this approach, in that, where there are N Atoms and m pivot Atoms, N^2 similarity relationships are being approximately stored in m*N coordinate values. Furthermore, the cost of computation is m*N times the cost of assessing a single SimilarityLink. By accepting crude approximations of actual similarity values, one gets away with linear time and space cost.

EFTA00624471

Because this is just an approximation technique, there are definitely going to be cases where A and B are not similar, even though they're close together in the embedding space. When such a case is found, it may be useful for the AtomSpace to explicitly contain a low-strength SimilarityLink between A and B. This link will prevent the system from making false embedding-based decisions to explore (SimilarityLink A B) in the future. Putting explicit low-strength SimilarityLinks in the system in these cases, is obviously much cheaper than using them for all cases.

We've been talking about SimilarityLinks, but the approach is more broadly applicable. Any symmetric link type can be dealt with about the same way. For instance, it might be useful to keep dimensional embedding maps for

- SimilarityLink
- ExtensionalSimilarityLink
- EquivalenceLink
- ExtensionalEquivalenceLink

On the other hand, dealing with asymmetric links in terms of dimensional embedding requires more subtlety – we turn to this topic below.

39.5 Dimensional Embedding and InheritanceLinks

Next, how can we use dimensional embedding to keep an approximate record of which links do not inherit from each other? Because inheritance is an asymmetric relationship, whereas distance in embedding spaces is a symmetrical relationship, there's no direct and simple way to do so.

However, there is an indirect approach that solves the problem, which involves maintaining two embedding spaces, and combining information about them in an appropriate way. In this subsection, we'll discuss an approach that should work for InheritanceLink, SubsetLink, ImplicationLink, and ExtensionalImplicationLink and other related link types. But we'll explicitly present it only for the InheritanceLink case.

Although the embedding algorithm described above was intended for symmetric weighted graphs, in fact we can use it for asymmetric links in just about the same way. The use of the embedding graph for inference control differs, but not the basic method of defining the embedding.

In the InheritanceLink case, we can define pivot Atoms in the same way, and then we can define two vectors for each Atom A:

```
proj_{parent}(A)_i = (InheritanceLink A A_i).tv.s

proj_{child}(A)_i = (InheritanceLink A_i A).tv.s
```

where A_i is the i'th pivot Atom.

If generally $proj_{child}(A)_i \leq proj_{child}(B)_i$ then qualitatively "children of A are children of B"; and if generally $proj_{parent}(A)_i \geq proj_{paret}(B)_i$ then qualitatively "parents of B are parents of A". The combination of these two conditions means heuristically that $(Inheritance\ A\ B)$ is likely. So, by combining the two embedding vectors assigned to each Atom, one can get heuristic guidance regarding inheritance relations, analogous to the case with similarity relationships. One

EFTA00624472

may produce mathematical formulas estimating the error of this approach under appropriate conditions, but in practice it will depend on the probability distribution of the vectors.

EFTA00624473

Chapter 40
Mental Simulation and Episodic Memory

40.1 Introduction

This brief chapter deals with two important, coupled cognitive components of CogPrime : the component concerned with creating *internal simulations* of situations and episodes in the external physical world, and the one concerned with storing and retrieving memories of situations and episodes.

These are components that are likely significantly different in CogPrime from anything that exists in the human brain, yet, the functions they carry out are obviously essential to human cognition (perhaps more so to human cognition than to CogPrime's cognition, because CogPrime is by design more reliant on formal reasoning than the human brain is).

Much of human thought consists of internal, quasi-sensory "imaging" of the external physical world – and much of human memory consists of remembering autobiographical situations and episodes from daily life, or from stories heard from others or absorbed via media. Often this episodic remembering takes the form of visualization, but not always. Blind people generally think and remember in terms of non-visual imagery, and many sighted people think in terms of sounds, tastes or smells in addition to visual images.

So far, the various mechanisms proposed as part of CogPrime do not have much to do with either internal imagery or episodic remembering, even though both seem to play a large role in human thought. This is OK, of course, since CogPrime is not intended as a simulacrum of human thought, but rather as a different sort of intelligence.

However, we believe it will actually be valuable to CogPrime to incorporate both of these factors. And for that purpose, we propose

- a novel mechanism: the incorporation within the CogPrime system of a 3D physical-world simulation engine.
- an episodic memory store centrally founded on dimensional embedding, and linked to the internal simulation model

EFTA00624474

40.2 Internal Simulations

The current use of virtual worlds for OpenCog is to provide a space in which human-controlled agents and CogPrime -controlled agents can interact, thus allowing flexible instruction of the CogPrime system by humans, and flexible embodied, grounded learning by CogPrime systems. But this very same mechanism may be used internally to CogPrime, i.e. a CogPrime system may be given an *internal simulation world*, which serves as a sort of "mind's eye." Any sufficiently flexible virtual world software may be used for this purpose, for example OpenSim (http:\opensim.org).

Atoms encoding percepts may be drawn from memory and used to generate forms within the internal simulation world. These forms may then interact according to

- the patterns via which they are remembered to act
- the laws of physics, as embodied in the simulation world

This allows a kind of "implicit memory," in that patterns emergent from the world-embedded interaction of a number of entities *need not explicitly be stored in memory*, so long as they will emerge when the entities are re-awakened within the internal simulation world.

The SimulatorMindAgent grabs important perceptual Atoms and uses them to generate forms within the internal simulation world, which then act according to remembered dynamical patterns, with the laws of physics filling in the gaps in memory. This provides a sort of running internal visualization of the world. Just as important, however, are specific schemata that utilize visualization in appropriate contexts. For instance, if reasoning is having trouble solving a problem related to physical entities, it may feed these entities to the internal simulation world to see what can be discovered. Patterns discovered via simulation can then be fed into reasoning for further analysis.

The process of perceiving events and objects in the simulation world is essentially identical to the process of perceiving events and objects in the "actual" world.

And of course, an internal simulation world may be used whether the CogPrime system in question is hooked up to a virtual world like OpenSim, or to a physical robot.

Finally, perhaps the most interesting aspect of internal simulation is the generation of "virtual perceptions" from abstract concepts. Analogical reasoning may be used to generate virtual perceptions that were never actually perceived, and these may then be visualized. The need for "reality discrimination" comes up here, and is easier to enforce in CogPrime than in humans. A PerceptNode that was never actually perceived may be explicitly embedded in a HypotheticalLink, thus avoiding the possibility of confusing virtual percepts with actual ones. How useful the visualization of virtual perceptions will be to CogPrime cognition, remains to be seen. This kind of visualization is key to human imagination but this doesn't mean it will play the same role in CogPrime's quite different cognitive processes. But it is important that CogPrime has the power to carry out this kind of imagination.

40.3 Episodic Memory

Episodic memory refers to the memory of our own "life history" that each of us has. Loss of this kind of memory is the most common type of amnesia in fiction – such amnesia is particularly dramatic because our episodic memories constitute so much of what we consider as our "selves."

EFTA00624475

To a significant extent, we as humans remember, reason and relate in terms of *stories* – and the centerpiece of our understanding of stories is our episodic memory. A CogPrime system need not be as heavily story-focused as a typical human being (though it could be, potentially) – but even so, episodic memory is a critical component of any CogPrime system controlling an agent in a world.

The core idea underlying CogPrime's treatment of episodic memory is a simple one: two dimensional embedding spaces dedicated to episodes. An episode – a coherent collection of happenings, often with causal interrelationships, often (but not always) occurring near the same spatial or temporal locations as each other – may be represented explicitly as an Atom, and implicitly as a map whose key is that Atom. These episode-Atoms may then be mapped into two dedicated embedding spaces:

- one based on a distance metric determined by spatiotemporal proximity
- one based on a distance metric determined by semantic similarity

A *story* is then a series of episodes – ideally one that, if the episodes in the series become important sequentially in the AtomSpace, causes a significant important-goal-related (ergo emotional) response in the system. Stories may also be represented as Atoms, in the simplest case consisting of SequentialAND links joining episode-Atoms. Stories then correspond to paths through the two episodic embedding spaces. Each path through each embedding space implicitly has a sort of "halo" in the space – visualizable as a tube snaking through the space, centered on the path. This tube contains other paths – other stories – that related to the given center story, either spatiotemporally or semantically.

The familiar everyday human experience of episodic memory may then be approximatively emulated via the properties of the dimensional embedding space. For instance, episodic memory is famously associative – when we think of one episode or story, we think of others that are spatiotemporally or semantically associated with it. This emerges naturally from the embedding space approach, due to the natural emergence of distance-based associative memory in an embedding space.

Figures 40.1 and 40.2 roughly illustrates the link between episodic/perceptual and declarative memory.

EFTA00624476

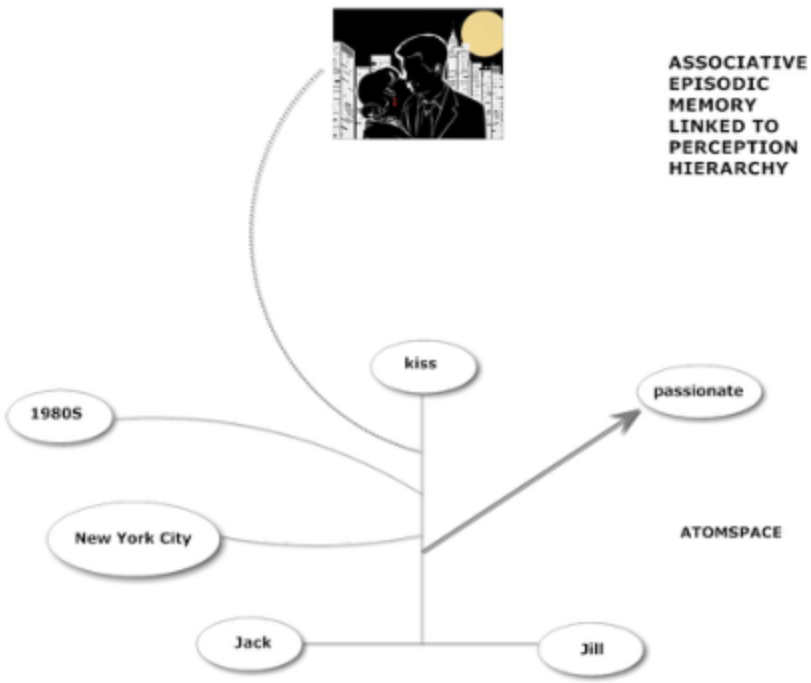


Fig. 40.1: **Relationship Between Episodic, Declarative and Perceptual Memory**. The nodes and links at the bottom depict declarative memory stored in the Atomspace; the picture at the top illustrates an archetypal episode stored in episodic memory, and linked to the perceptual hierarchy enabling imagistic simulation.

EFTA00624477

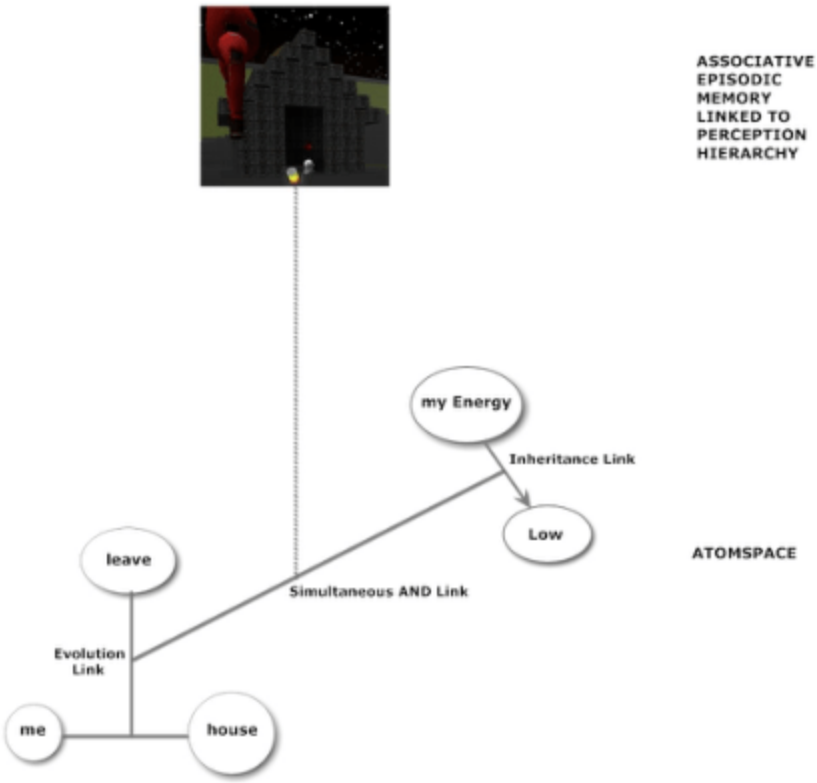


Fig. 40.2: **Relationship Between Episodic, Declarative and Perceptual Memory**. Another example similar to the one in **??**, but referring specifically to events occurring in an OpenCogPrime -controlled agent's virtual world.

EFTA00624478

Chapter 41
Integrative Procedure Learning

41.1 Introduction

"Procedure learning" – learning step-by-step procedures for carrying out internal or external operations – is a highly critical aspect of general intelligence, and is carried out in CogPrime via a complex combination of methods. This somewhat heterogeneous chapter reviews several advanced aspects of procedure learning in CogPrime, mainly having to do with the integration between different cognitive processes.

In terms of general cognitive theory and mind-world correspondence, this is some of the subtlest material in the book. We are not concerned just with how the mind's learning of one sort of knowledge correlated with the way this sort of knowledge is structured in the mind's habitual environments, in the context of its habitual goals. Rather, we are concerned with how various sorts of knowledge intersect and interact with each other. The proposed algorithmic intersections between, for instance, declarative and procedural learning processes, are reflective of implicit assumptions about how declarative and procedural knowledge are presented in the world in the context of the system's goals – but these implicit assumptions are not always easy to tease out and state in a compact way. We will do our best to highlight these assumptions as they arise throughout the chapter.

Key among these assumptions, however, are that a human-like mind

- is presented with various procedure learning problems at various levels of difficulty (so that different algorithms may be appropriate depending on the difficulty level). *This leads for instance to the possibility of using various different algorithms like MOSES or hill climbing, for different procedure learning problems.*
- is presented with some procedure learning problems that may be handled in a relatively isolated way, and others that are extremely heavily dependent on context, often in a way that recurs across multiple learning instances in similar contexts. *This leads to a situations where the value of bringing declarative (PLN) and associative (ECAN) and episodic knowledge into the procedure learning process, has varying value depending on the situation.*
- is presented with a rich variety of procedure learning problems with complex interrelationships, including many problems that are closely related to previously solved problems in various ways. *This highlights the value of using PLN analogical reasoning, and importance spreading along HebbianLinks learned by ECAN, to help guide procedure learning in various ways.*

EFTA00624480

- needs to learn some procedures whose execution may be carried out in a relatively isolated way, and other procedures whose execution requires intensive ongoing interaction with other cognitive processes

The diversity of procedure learning situations reflected in these assumptions, leads naturally to the diversity of technical procedure learning approaches described in this chapter. Potentially one could have a single, unified algorithm covering all the different sorts of procedure learning, but instead we have found it more practical to articulate a small number of algorithms which are then combined in different ways to yield the different kinds of procedure learning.

41.1.1 The Diverse Technicalities of Procedure Learning in CogPrime

On a technical level, this chapter discusses two closely related aspects of CogPrime : schema learning and predicate learning, which we group under the general category of "procedure learning."

Schema learning - the learning of SchemaNodes and schema maps (explained further in the Chapter 42) - is CogPrime lingo for learning how to do things. Learning how to act, how to perceive, and how to think - beyond what's explicitly encoded in the system's MindAgents. As an advanced CogPrime system becomes more profoundly self-modifying, schema learning will drive more and more of its evolution.

Predicate learning, on the other hand, is the most abstract and general manifestation of pattern recognition in the CogPrime system. PredicateNodes, along with predicate maps, are CogPrime's way of representing general patterns (*general* within the constraints imposed by the system parameters, which in turn are governed by hardware constraints). Predicate evolution, predicate mining and higher-order inference - specialized and powerful forms of predicate learning - are the system's most powerful ways of creating general patterns in the world and in the mind. Simpler forms of predicate learning are grist for the mill of these processes.

It may be useful to draw an analogy with another (closely related) very hard problem in CogPrime, discussed in the book Probabilistic Logic Networks: probabilistic logical unification, which in the CogPrime /PLN framework basically comes down to finding the SatisfyingSets of given predicates. Hard logical unification problems can often be avoided by breaking down large predicates into small ones in strategic ways, guided by non-inferential mind processes, and then doing unification only on the smaller predicates. Our limited experimental experience indicates that the same "hierarchical breakdown" strategy also works for schema and predicate learning, to an extent. But still, as with unification, even when one does break down a large schema or predicate learning problem into a set of smaller problems, one is still in most cases left with a set of fairly hard problems.

More concretely, CogPrime procedure learning may be generally decomposed into three aspects:

1. Converting back and forth between maps and ProcedureNodes (*encapsulation and expansion*)
2. Learning the Combo Trees to be embedded in grounded ProcedureNodes
3. Learning procedure maps (networks of grounded ProcedureNodes acting in a coordinated way to carry out procedures)

EFTA00624481

Each of these three aspects of CogPrime procedure learning mentioned above may be dealt with somewhat separately, though relying on largely overlapping methods.

CogPrime approaches these problems using a combination of techniques:

- Evolutionary procedure learning and hillclimbing for dealing with *brand new* procedure learning problems, requiring the origination of innovative, highly approximate solutions out of the blue
- Inferential procedure learning for taking approximate solutions and making them exact, and for dealing with procedure learning problems within domains where closely analogous procedure learning problems have previously been solved
- Heuristic, probabilistic data mining for the creation of encapsulated procedures (which then feed into inferential and evolutionary procedure learning), and the expansion of encapsulated procedures into procedure maps
- PredictiveImplicationLink formation (augmented by PLN inference on such links) as a CogPrime version of goal-directed reinforcement learning

Using these different learning methods together, as a coherently-tuned whole, one arrives at a holistic procedure learning approach that combines speculation, systematic inference, encapsulation and credit assignment in a single adaptive dynamic process.

We are relying on a combination of techniques to do what none of the techniques can accomplish on their own. The combination is far from arbitrary, however. As we will see, each of the techniques involved plays a unique and important role.

41.1.1.1 Comments on an Alternative Representational Approach

We briefly pause to contrast certain technical aspects of the present approach to analogous aspects of the Webmind AI Engine (one of CogPrime's predecessor AI systems, briefly discussed in Chapter 19.1). This predecessor system used a knowledge representation somewhat similar to the Atomspace, but with various differences; for instance the base types were Node and Link rather than Atom, and there was a Node type not used in CogPrime called the SchemaInstanceNode (each one corresponding to a particular instance of a SchemaNode, used within a particular procedure).

In this approach, complex, learned schema were represented as distributed networks of elementary SchemaInstanceNodes, but these networks were not defined purely by function application - they involved explicit passing of variable values through VariableNodes. Special logic-gate-bearing objects were created to deal with the distinction between arguments of a SchemaInstanceNode, and *predecessor* tokens giving a SchemaInstanceNode permission to act.

While this approach is in principle workable, it proved highly complex in practice, and for the Novamente Cognition Engine and CogPrime we chose to store and manipulate procedural knowledge separately from declarative knowledge (via Combo trees).

EFTA00624482

41.2 Preliminary Comments on Procedure Map Encapsulation and Expansion

Like other knowledge in CogPrime, procedures may be stored in either a localized (Combo tree) or globalized (procedure map) manner, with the different approaches being appropriate for different purposes. Activation of a localized procedure may spur activation of a globalized procedure, and vice versa – so on the overall mind-network level the representation of procedures is heavily "glocal."

One issue that looms large in this context is the conversion between localized and globalized procedures – i.e., in CogPrime lingo, the encapsulation and expansion of procedure maps. This matter will be considered in more detail in Chapter 42 but here we briefly review some key ideas.

Converting from grounded ProcedureNodes into maps is a relatively simple learning problem: one enacts the procedure, observes which Atoms are active at what times during the enaction process, and then creating PredictiveImplicationLinks between the Atoms active at a certain time and those active at subsequent times. Generally it will be necessary to enact the procedure multiple times and with different inputs, to build up the appropriate library of PredictiveImplicationLinks.

Converting from maps into ProcedureNodes is significantly trickier. First, it involves carrying out data mining over the network of ProcedureNodes, identifying subnetworks that are coherent schema or predicate maps. Then it involves translating the control structure of the map into explicit logical form, so that the encapsulated version will follow the same order of execution as the map version. This is an important case of the general process of map encapsulation, to be discussed in Chapter 42

Next, the learning of grounded ProcedureNodes is carried out by a synergistic combination of multiple mechanisms, including pure procedure learning methods like hillclimbing and evolutionary learning, and logical inference. These two approaches have quite different characteristics. Evolutionary learning and hillclimbing excel at confronting a problem that the system has no clue about, and arriving at a reasonably good solution in the form of a schema or predicate. Inference excels at deploying the system's existing knowledge to form useful schemata or predicates. The choice of the appropriate mechanism for a given problem instance depends largely on how much relevant knowledge is available.

A relatively simple case of ProcedureNode learning is where one is given a ConceptNode and wants to find a ProcedureNode matching it. For instance, given a ConceptNode C, one may wish to find the simplest possible predicate whose corresponding PredicateNode P satisfies

```
SatisfyingSet(P) = C
```

On the other hand, given a ConceptNode C involved in inferred ExecutionLinks of the form

```
ExecutionLink C Ai Bi
i=1,...,n
```

one may wish to find a SchemaNode so that the corresponding SchemaNode will fulfill this same set of ExecutionLinks. It may seem surprising at first that a ConceptNode might be involved with ExecutionLinks, but remember that a function can be seen as a set of tuples (ListLink in CogPrime) where the first elements, the inputs of the function, are associated with a unique output. These kinds of ProcedureNode learning may be cast as optimization problems, and carried out by hillclimbing or evolutionary programming. Once procedures are learned via evolutionary programming or other techniques, they may be refined via inference.

EFTA00624483

The other case of ProcedureNode learning is goal-driven learning. Here one seeks a SchemaNode whose execution will cause a given goal (represented by a Goal Node) to be satisfied. The details of Goal Nodes have already been reviewed; but all we need to know here is simply that a Goal Node presents an objective function, a function to be maximized; and that it poses the problem of finding schemata whose enaction will cause this function to be maximized in specified contexts.

The learning of procedure maps, on the other hand, is carried out by reinforcement learning, augmented by inference. This is a matter of the system learning HebbianLinks between ProcedureNodes, as will be described below.

41.3 Predicate Schematization

Now we turn to the process called "predicate schematization," by which declarative knowledge about how to carry out actions may be translated into Combo trees embodying specific procedures for carrying out actions. This process is straightforward and automatic in some cases, but in other cases requires significant contextually-savvy inference. This is a critical process because some procedure knowledge – especially that which is heavily dependent on context in either its execution or its utility – will be more easily learned via inferential methods than via pure procedure-learning methods. But, even if a procedure is initially learned via inference (or is learned by inference based on cruder initial guesses produced by pure procedure learning methods), it may still be valuable to have this procedure in compact and rapidly executable form such as Combo provides.

To proceed with the technical description of predicate schematization in CogPrime, we first need the notion of an "executable predicate". Some predicates are executable in the sense that they correspond to executable schemata, others are not. There are executable atomic predicates (represented by individual PredicateNodes), and executable predicates (which are link structures). In general, a predicate may be turned into a schema if it is an atomic executable predicate, or if it is a compound link structure that consists entirely of executable atomic predicates (e.g. pick_up, walk_to, can_do, etc.) and temporal links (e.g. SimultaneousAND, PredictiveImplication, etc.)

Records of predicate execution may then be made using ExecutionLinks, e.g.

```
ExecutionLink pick_up ( me, ball_7)
```

is a record of the fact that the schema corresponding to the pick_up predicate was executed on the arguments (me, ball_7).

It is also useful to introduce some special (executable) predicates related to schema execution:

- can_do, which represents the system's perceived ability to do something
- do, which denotes the system actually doing something; this is used to mark actions as opposed to perceptions
- just_done, which is true of a schema if the schema has very recently been executed.

The general procedure used in figuring out what predicates to schematize, in order to create a procedure achieving a certain goal, is: Start from the goal and work backwards, following PredictiveImplications and EventualPredictiveImplications and treating can_do's as transparent, stopping when you find something that can currently be done, or else when the process dwindles due to lack of links or lack of sufficiently certain links.

EFTA00624484

In this process, an ordered list of perceptions and actions will be created. The Atoms in this perception/action-series (PA-series) are linked together via temporal-logical links.

The subtlety of this process, in general, will occur because there may be many different paths to follow. One has the familiar combinatorial explosion of backward-chaining inference, and it may be hard to find the best PA-series among all the mess. Experience-guided pruning is needed here just as with backward-chaining inference.

Specific rules for translating temporal links into executable schemata, used in this process, are as follows. All these rule-statements assume that B is in the selected PA-series. All node variables not preceded by do or can_do are assumed to be perceptions. The → denotes the transformation from predicates to executable schemata.

```
EventualPredictiveImplicationLink (do A) B

  →

Repeat (do A) Until B

EventualPredictiveImplicationLink (do A) (can_do B)

  →

Repeat
    do A
    do B
Until
    Evaluation just_done B
```

the understanding being that the agent may try to do B and fail, and then try again the next time around the loop

```
PredictiveImplicationLink (do A) (can_do B) <time-lag T>

  →

do A
wait T
do B

SimultaneousImplicationLink A (can_do B)

  →

if A then do B

SimultaneousImplicationLink (do A) (can_do B)

  →

do A
do B

PredictiveImplicationLink A (can_do B)

  →

if A then do B

SequentialAndLink A1 ... An
```

EFTA00624485

```
  →

A1
...
An

SequentialAndLink A1 ... An <time_lag T>

  →

A1
Wait T
A2
Wait T
...
Wait T
An

SimultaneousANDLink A1 ? An

  →

A1
...
An
```

Note how all instances of can_do are stripped out upon conversion from predicate to schema, and replaced with instances of do.

41.3.1 A Concrete Example

For a specific example of this process, consider the knowledge that: "If I walk to the teacher while whistling, and then give the teacher the ball, I'll get rewarded."

This might be represented by the predicates

walk to the teacher while whistling

```
A_1 :=
SimultaneousAND
    do Walk_to
        ExOutLink locate teacher
    EvaluationLink do whistle
```

If I walk to the teacher while whistling, eventually I will be next to the teacher

```
EventualPredictiveImplication
    A_1
    Evaluation next_to teacher
```

While next to the teacher, I can give the teacher the ball

```
SimultaneousImplication
    EvaluationLink next_to teacher
    can_do
        EvaluationLink give (teacher, ball)
```

If I give the teacher the ball, I will get rewarded

EFTA00624486

```
PredictiveImplication
    just_done
        EvaluationLink done give (teacher, ball)
    Evaluation reward
```

Via goal-driven predicate schematization, these predicates would become the schemata *walk toward the teacher while whistling*

```
Repeat:
    do WalkTo
        ExOut locate teacher
    do Whistle
Until:
    next_to(teacher, ball)
```

if next to the teacher, give the teacher the ball

```
If:
    Evaluation next_to teacher
Then
    do give(teacher, ball)
```

Carrying out these two schemata will lead to the desired behavior of walking toward the teacher while whistling, and then giving the teacher the ball when next to the teacher.

Note that, in this example:

- The walk_to, whistle, locate and give used in the example schemata are procedures corresponding to the executable predicates walk_to, whistle, locate and give used in the example predicates
- Next_to is evaluated rather than executed because (unlike the other atomic predicates in the overall predicate being made executable) it has no "do" or "can_do" next to it

41.4 Concept-Driven Schema and Predicate Creation

In this section we will deal with the "conversion" of ConceptNodes into SchemaNodes or PredicateNodes. The two cases involve similar but nonidentical methods; we will begin with the simpler PredicateNode case. Conceptually, the importance of this should be clear: sometimes knowledge may be gained via concept-learning or linguistic means, but yet may be useful to the mind in other forms, e.g. as executable schema or evaluable predicates. For instance, the system may learn conceptually about bicycle-riding, but then may also want to learn executable procedures allowing it to ride a bicycle. Or it may learn conceptually about criminal individuals, but may then want to learn evaluable predicates allowing it to quickly evaluate whether a given individual is a criminal or not.

41.4.1 Concept-Driven Predicate Creation

Suppose we have a ConceptNode C, with a set of links of the form

EFTA00624487

```
MemberLink A_i C, i=1,...,n
```

Our goal is to find a PredicateNode so that firstly,

```
MemberLink X C

is equivalent to

X ''within'' SatisfyingSet(P)
```

and secondly,

```
P is as simple as possible
```

This is related to the "Occam's Razor," Solomonoff induction related heuristic to be presented later in this chapter.

We now have an optimization problem: search the space of predicates for P that maximize the objective function f(P,C), defined as for instance

$$f(P, C) = cp(P) \times r(C, P)$$

where $cp(P)$, the complexity penalty of P, is a positive function that decreases when P gets larger and with $r(C, P) =$

```
GetStrength
    SimilarityLink
        C
        SatisfyingSet(P)
```

This is an optimization problem over predicate space, which can be solved in an approximate way by the evolutionary programming methods described earlier.

The ConceptPredicatization MindAgent selects ConceptNodes based on

- Importance
- Total (truth value based) weight of attached MemberLinks and EvaluationLinks

and launches an evolutionary learning or hillclimbing task focused on learning predicates based on the nodes it selects.

41.4.2 Concept-Driven Schema Creation

In the schema learning case, instead of a ConceptNode with MemberLinks and EvaluationLinks, we begin with a ConceptNode C with ExecutionLinks. These ExecutionLinks were presumably produced by inference (the only CogPrime cognitive process that knows how to create ExecutionLinks for non-ProcedureNodes).

The optimization problem we have here is: search the space of schemata for S that maximize the objective function $f(S, C)$, defined as follows:

$$f(S, C) = cp(S) \times r(S, C)$$

Let $Q(C)$ be the set of pairs (X, Y) so that $ExecutionLink\ C\ X\ Y$, and

EFTA00624488

```
r(S,C) =

GetStrength
    SubsetLink
        Q(C)
        Graph(S)
```

where $Graph(S)$ denotes the set of pairs (X, Y) so that $ExecutionLink\ S\ X\ Y$, where S has been executed over all valid inputs.

Note that we consider a SubsetLink here because in practice C would have been observed on a partial set of inputs.

Operationally, the situation here is very similar to that with concept predicatization. The ConceptSchematization MindAgent must select ConceptNodes based on:

- Importance
- Total (truth value based) weight of ExecutionLinks

and then feed these to evolutionary optimization or hillclimbing.

41.5 Inference-Guided Evolution of Pattern-Embodying Predicates

Now we turn to predicate learning – the learning of PredicateNodes, in particular.

Aside from logical inference and learning predicates to match existing concepts, how does the system create new predicates? Goal-driven schema learning (via evolution or reinforcement learning) provides one alternate approach: create predicates in the context of creating useful schema. Pattern mining, discussed in Chapter 37, provides another. Here we will describe (yet) another complementary dynamic for predicate creation: pattern-oriented, inference-guided PredicateNode evolution.

In most general terms, the notion pursued here is to form predicates that embody patterns in itself and in the world. This brings us straight back to the foundations of the patternist philosophy of mind, in which mind is viewed as a system for recognizing patterns in itself and in the world, and then embodying these patterns in itself. This general concept is manifested in many ways in the CogPrime design, and in this section we will discuss two of them:

- Reward of surprisingly probable Predicates
- Evolutionary learning of pattern-embodying Predicates

These are emphatically not the only way pattern-embodying PredicateNodes get into the system. Inference and concept-based predicate learning also create PredicateNodes embodying patterns. But these two mechanisms complete the picture.

41.5.1 Rewarding Surprising Predicates

The TruthValue of a PredicateNode represents the expected TruthValue obtained by averaging its TruthValue over all its possible legal argument-values. Some Predicates, however, may have high TruthValue without really being *worthwhile*. They may not add any information to their

EFTA00624489

components. We want to identify and reward those Predicates whose TruthValues actually add information beyond what is implicit in the simple fact of combining their components.

For instance, consider the PredicateNode

```
AND
   InheritanceLink X man
   InheritanceLink X ugly
```

If we assume the man and ugly concepts are independent, then this PredicateNode will have the TruthValue

$$man.tv.s \times ugly.tv.s$$

In general, a PredicateNode will be considered interesting if:

1. Its Links are important
2. Its TruthValue differs significantly from what would be expected based on independence assumptions about its components

It is of value to have interesting Predicates allocated more attention than uninteresting ones. Factor 1 is already taken into account, in a sense: if the PredicateNode is involved in many Links this will boost its activation which will boost its importance. On the other hand, Factor 2 is not taken into account by any previously discussed mechanisms.

For instance, we may wish to reward a PredicateNode if it has a surprisingly large or small strength value. One way to do this is to calculate:

$$sdiff = |actual\ strength - strength\ predicted\ via\ independence\ assumptions| \times weight_of_evidence$$

and then increment the value:

$$K \times sdiff$$

onto the PredicateNode's LongTermImportance value, and similarly increment STI using a different constant.

Another factor that might usefully be caused to increment LTI is the simplicity of a PredicateNode. Given two Predicates with equal strength, we want the system to prefer the simpler one over the more complex one. However, the OccamsRazor MindAgent, to be presented below, rewards simpler Predicates directly in their strength values. Hence if the latter is in use, it seems unnecessary to reward them for their simplicity in their LTI values as well. This is an issue that may require some experimentation as the system develops.

Returning to the surprisingness factor, consider the PredicateNode representing

```
AND
    InheritanceLink X cat
    EvaluationLink (eats X) fish
```

If this has a surprisingly high truth value, this means that there are more X known to (or inferred by) the system, that both inherit from *cat* and eat fish, than one would expect given the probabilities of a random X both inheriting from cat and eating fish. Thus, roughly speaking, the conjunction of inheriting from cat and eating fish may be a pattern in the world.

EFTA00624490

We now see one very clear sense in which CogPrime dynamics implicitly leads to predicates representing patterns. Small predicates that have surprising truth values get extra activation, hence are more likely to stick around in the system. Thus the mind fills up with patterns.

41.5.2 A More Formal Treatment

It is worth taking a little time to clarify the sense in which we have a *pattern* in the above example, using the mathematical notion of pattern reviewed in Chapter 3 of Part 1.

Consider the predicate:

```
pred1(T).tv
equals
        >
           GetStrength
               AND
                    Inheritance $X cat
                    Evaluation eats ($X, fish)
           T
```

where T is some threshold value (e.g. .8). Let B = SatisfyingSet(pred1(T)). B is the set of everything that inherits from cat and eats fish.

Now we will make use of the notion of *basic complexity*. If one assumes the entire AtomSpace A constituting a given CogPrime system as given background information, then the basic complexity $c(B||A)$ may be considered as the number of bits required to list the handles of the elements of B, for lookup in A; whereas $c(B)$ is the number of bits required to actually list the elements of B. Now, the formula given above, defining the set B, may be considered as a process P whose output is the set B. The simplicity $c(P||A)$ is the number of bits needed to describe this process, which is a fairly small number. We assume A is given as background information, accessible to the process.

Then the degree to which P is a pattern in B is given by

$$1 - c(P||A)/c(B||A)$$

which, if B is a sizable category, is going to be pretty close to 1.

The key to there being a pattern here is that the relation:

```
(Inheritance X cat) AND (eats X fish)
```

has a high strength and also a high count. The high count means that B is a large set, either by direct observation or by hypothesis (inference). In the case where the count represents actual pieces of evidence observed by the system and retained in memory, then quite literally and directly, the PredicateNode represents a pattern in a subset of the system (relative to the background knowledge consisting of the system as a whole). On the other hand, if the count value has been obtained indirectly by inference, then it is possible that the system does not actually know any examples of the relation. In this case, the PredicateNode is not a pattern in the actual memory store of the system, but it is being hypothesized to be a pattern in the world in which the system is embedded.

EFTA00624491

41.6 PredicateNode Mining

We have seen how the natural dynamics of the CogPrime system, with a little help from special heuristics, can lead to the evolution of Predicates that embody patterns in the system's perceived or inferred world. But it is also valuable to more aggressively and directly create pattern-embodying Predicates. This does not contradict the implicit process, but rather complements it. The explicit process we use is called *PredicateNode Mining* and is carried out by a PredicateNodeMiner MindAgent.

Define an Atom structure template as a schema expression corresponding to a CogPrime Link in which some of the arguments are replaced with variables. For instance,

```
Inheritance X cat
```

```
EvaluationLink (eats X) fish
```

are Atom structure templates. (Recall that Atom structure templates are important in PLN inference control, as reviewed in 36)

What the PredicateNodeMiner does is to look for Atom structure templates and logical combinations thereof which

- Minimize PredicateNode size
- Maximize surprisingness of truth value

This is accomplished by a combination of heuristics.

The first step in PredicateNode mining is to find Atom structure templates with high truth values. This can be done by a fairly simple heuristic search process.

First, note that if one specifies an (Atom, Link type), one is specifying a set of Atom structure templates. For instance, if one specifies

```
(cat, InheritanceLink)
```

then one is specifying the templates

```
InheritanceLink $X cat
```

and

```
InheritanceLink cat $X
```

One can thus find Atom structure templates as follows. Choose an Atom with high truth value, and then, for each Link type, tabulate the total truth value of the Links of this type involving this Atom. When one finds a promising (Atom, Link type) pair, one can then do inference to test the truth value of the Atom structure template one has found.

Next, given high-truth-value Atom structure templates, the PredicateNodeMiner experiments with joining them together using logical connectives. For each potential combination it assesses the fitness in terms of size and surprisingness. This may be carried out in two ways:

1. By incrementally building up larger combinations from smaller ones, at each incremental stage keeping only those combinations found to be valuable
2. For large combinations, by evolution of combinations

Option 1 is basically greedy data mining (which may be carried out via various standard algorithms, as discussed in Chapter 37), which has the advantage of being much more rapid

EFTA00624492

than evolutionary programming, but the disadvantage that it misses large combinations whose subsets are not as surprising as the combinations themselves. It seems there is room for both approaches in CogPrime (and potentially many other approaches as well). The PredicateNodeMiner MindAgent contains a parameter telling it how much time to spend on stochastic pattern mining vs. evolution, as well as parameters guiding the processes it invokes.

So far we have discussed the process of finding single-variable Atom structure templates. But multivariable Atom structure templates may be obtained by combining single-variable ones. For instance, given

```
eats $X fish
```

```
lives_in $X Antarctica
```

one may choose to investigate various combinations such as

```
(eats $X $Y) AND (lives_in $X $Y)
```

(this particular example will have a predictably low truth value). So, the introduction of multiple variables may be done in the same process as the creation of single-variable combinations of Atom structure templates.

When a suitably fit Atom structure template or logical combination thereof is found, then a PredicateNode is created embodying it, and placed into the AtomSpace. WIKISOURCE:SchemaMaps

41.7 Learning Schema Maps

Next we plunge into the issue of procedure maps - schema maps in particular. A schema map is a simple yet subtle thing - a subnetwork of the AtomSpace consisting of SchemaNodes, computing some useful quantity or carrying out some useful process in a cooperative way. The general purpose of schema maps is to allow schema execution to interact with other mental processes in a more flexible way than is allowed by compact Combo trees with internal hooks into the AtomSpace. I.e., to handle cases where procedure execution needs to be *very* highly interactive, mediated by attention allocation and other CogPrime dynamics in a flexible way.

But how can schema maps be learned? The basic idea is simply reinforcement learning. In a goal-directed system consisting of interconnected, cooperative elements, you reinforce those connections and/or those elements that have been helpful for achieving goals, and weaken those connections that haven't. Thus, over time, you obtain a network of elements that achieves goals effectively.

The central difficulty in all reinforcement learning approaches is the 'assignment of credit' problem. If a component of a system has been directly useful for achieving a goal, then rewarding it is easy. But if the relevance of a component to a goal is indirect, then things aren't so simple. Measuring indirect usefulness in a large, richly connected system is difficult - inaccuracies creep into the process easily.

In CogPrime, reinforcement learning is handled via HebbianLinks, acted on by a combination of cognitive processes. Earlier, in Chapter 23, we reviewed the semantics of HebbianLinks, and discussed two methods for forming HebbianLinks:

1. Updating HebbianLink strengths via mining of the System Activity Table

EFTA00624493

2. Logical inference on HebbianLinks, which may also incorporate the use of inference to combine HebbianLinks with other logical links (for instance, in the reinforcement learning context, PredictiveImplicationLinks)

We now describe how HebbianLinks, formed and manipulated in this manner, may play a key role in goal-driven reinforcement learning. In effect, what we will describe is an implicit integration of the bucket brigade with PLN inference. The addition of robust probabilistic inference adds a new kind of depth and precision to the reinforcement learning process.

Goal Nodes have an important ability to stimulate a lot of SchemaNode execution activity. If a goal needs to be fulfilled, it stimulates schemata that are known to make this happen. But how is it known which schemata tend to fulfill a given goal? A link:

```
PredictiveImplicationLink S G
```

means that after schema S has been executed, goal G tends to be fulfilled. If these links between goals and goal-valuable schemata exist, then activation spreading from goals can serve the purpose of causing goal-useful schemata to become active.

The trick, then, is to use HebbianLinks and inference thereon to implicitly guess PredictiveImplicationLinks. A HebbianLink between S1 and S says that when thinking about S1 was useful in the past, thinking about S was also often useful. This suggests that if doing S achieves goal G, maybe doing S1 is also a good idea. The system may then try to find (by direct lookup or reasoning) whether, in the current context, there is a PredictiveImplication joining S1 to S. In this way Hebbian reinforcement learning is being used as an inference control mechanism to aid in the construction of a goal-directed chain of PredictiveImplicationLinks, which may then be schematized into a contextually useful procedure.

Note finally that this process feeds back into itself in an interesting way, via contributing to ongoing HebbianLink formation. Along the way, while leading to the on-the-fly construction of context-appropriate procedures that achieve goals, it also reinforces the HebbianLinks that hold together schema maps, sculpting new schema maps out of the existing field of interlinked SchemaNodes.

41.7.1 Goal-Directed Schema Evolution

Finally, as a complement to goal-driven reinforcement learning, there is also a process of goal-directed SchemaNode learning. This combines features of the goal-driven reinforcement learning and concept-driven schema evolution methods discussed above. Here we use a Goal Node to provide the fitness function for schema evolution.

The basic idea is that the fitness of a schema is defined by the degree to which enactment of that schema causes fulfillment of the goal. This requires the introduction of CausalImplicationLinks, as defined in PLN. In the simplest case, a CausalImplicationLink is simply a PredictiveImplicationLink.

One relatively simple implementation of the idea is as follows. Suppose we have a Goal Node G, whose satisfaction we desire to have achieved by time T1. Suppose we want to find a SchemaNode S whose execution at time T2 will cause G to be achieved. We may define a fitness function for evaluating candidate S by:

$$f(S, G, T1, T2) = cp(S) \times r(S, G, T1, T2)$$

EFTA00624494

```
r(S,G,T1,T2) =
    GetStrength
        CausalImplicationLink
            EvaluationLink
                AtTime
                    T1
                    ExecutionLink S X Y
            EvaluationLink AtTime (T2, G)
```

Another variant specifies only a relative time lag, not two absolute times.

$$f(S,G,T) = cp(S) \times v(S,G,T)$$

```
v(S,G,T) =
      AND
          NonEmpty
              SatisfyingSet r(S,G,T1,T2)
          T1 > T2 - T
```

Using evolutionary learning or hillclimbing to find schemata fulfilling these fitness functions, results in SchemaNodes whose execution is expected to cause the achievement of given goals. This is a complementary approach to reinforcement-learning based schema learning, and to schema learning based on PredicateNode concept creation. The strengths and weaknesses of these different approaches need to be extensively experimentally explored. However, prior experience with the learning algorithms involved gives us some guidance.

We know that when absolutely nothing is known about an objective function, evolutionary programming is often the best way to proceed. Even when there is knowledge about an objective function, the evolution process can take it into account, because the fitness functions involve logical links, and the evaluation of these logical links may involve inference operations.

On the other hand, when there's a lot of relevant knowledge embodied in previously executed procedures, using logical reasoning to guide new procedure creation can be cumbersome, due to the overwhelming potentially useful number of facts to choose when carrying inference. The Hebbian mechanisms used in reinforcement learning may be understood as inferential in their conceptual foundations (since a HebbianLink is equivalent to an ImplicationLink between two propositions about importance levels). But in practice they provide a much-streamlined approach to bringing knowledge implicit in existing procedures to bear on the creation of new procedures. Reinforcement learning, we believe, will excel at combining existing procedures to form new ones, and modifying existing procedures to work well in new contexts. Logical inference can also help here, acting in cooperation with reinforcement learning. But when the system has no clue how a certain goal might be fulfilled, evolutionary schema learning provides a relatively time-efficient way for it to find something minimally workable.

Pragmatically, the GoalDrivenSchemaLearning MindAgent handles this aspect of the system's operations. It selects Goal Nodes with probability proportional to importance, and then spawns problems for the Evolutionary Optimization Unit Group accordingly. For a given Goal Node, PLN control mechanisms are used to study its properties and select between the above objective functions to use, on an heuristic basis.

EFTA00624495

41.8 Occam's Razor

Finally we turn to an important cognitive process that fits only loosely into the category of "CogPrime Procedure learning" – it's not actually a *procedure learning* process, but rather a process that utilizes the fruits of procedure learning.

The well-known "Occam's razor" heuristic says that all else being equal, simpler is better. This notion is embodied mathematically in the Solomonoff-Levin "universal prior," according to which the a priori probability of a computational entity X is defined as a normalized version of:

$$m(X) = \sum_p 2^{-l(p)}$$

where:

- the sum is taken over all programs p that compute X
- l(p) denotes the length of the program p

Normalization is necessary because these values will not automatically sum to 1 over the space of all X.

Without normalization, m is a semimeasure rather than a measure; with normalization it becomes the "Solomonoff-Levin measure" [Lev94].

Roughly speaking, Solomonoff's induction theorem [Sol64a, Sol64b] shows that, if one is trying to learn the computer program underlying a given set of observed data, and one does Bayesian inference over the set of all programs to try and obtain the answer, then if one uses the universal prior distribution one will arrive at the correct answer.

CogPrime is not a Solomonoff induction engine. The computational cost of actually applying Solomonoff induction is unrealistically large. However, as we have seen in this chapter, there are aspects of CogPrime that are reminiscent of Solomonoff induction. In concept-directed schema and predicate learning, in pattern-based predicate learning - and in causal schema learning, we are searching for schemata and predicates that minimize complexity while maximizing some other quality. These processes all implement the Occam's Razor heuristic in a Solomonoffian style.

Now we will introduce one more method of imposing the heuristic of algorithmic simplicity on CogPrime Atoms (and hence, indirectly, on CogPrime maps as well). This is simply to give a higher a priori probability to entities that are more simply computable.

For starters, we may increase the node probability of ProcedureNodes proportionately to their simplicity. A reasonable formula here is simply:

$$2^{-rc(P)}$$

where P is the ProcedureNode and $r > 0$ is a parameter. This means that infinitely complex P have a priori probability zero, whereas an infinitely simple P has an a priori probability 1.

This is not an exact implementation of the Solomonoff-Levin measure, but it's a decent heuristic approximation. It is not pragmatically realistic to sum over the lengths of all programs that do the same thing as a given predicate P. Generally the first term of the Solomonoff-Levin summation is going to dominate the sum anyway, so if the ProcedureNode P is maximally compact, then our simplified formula will be a good approximation of the Solomonoff-Levin

EFTA00624496

summation. These a priori probabilities may be merged with node probability estimates from other sources, using the revision rule.

A similar strategy may be taken with ConceptNodes. We want to reward a ConceptNode C with a higher a priori probability if $C \in SatisfyingSet(P)$ for a simple PredicateNode P. To achieve this formulaically, let $sim(X, Y)$ denote the strength of the SimilarityLink between X and Y, and let:

$$sim'(C, P) = sim(C, SatisfyingSet(P))$$

We may then define the a priori probability of a ConceptNode as:

$$pr(C) = \sum_P sim'(C, P) 2^{-rc(P)}$$

where the sum goes over all P in the system. In practice of course it's only necessary to compute the terms of the sum corresponding to P so that $sim'(C, P)$ is large.

As with the a priori PredicateNode probabilities discussed above, these a priori ConceptNode probabilities may be merged with other node probability information, using the revision rule, and using a default parameter value for the weight of evidence. There is one pragmatic difference here from the PredicateNode case, though. As the system learns new PredicateNodes, its best estimate of $pr(C)$ may change. Thus it makes sense for the system to store the a priori probabilities of ConceptNodes separately from the node probabilities, so that when the a priori probability is changed, a two step operation can be carried out:

- First, remove the old a priori probability from the node probability estimate, using the reverse of the revision rule
- Then, add in the new a priori probability

Finally, we can take a similar approach to any Atom Y produced by a SchemaNode. We can construct:

$$pr(Y) = \sum_{S,X} s(S, X, Y) 2^{-r(c(S)+c(X))}$$

where the sum goes over all pairs (S, X) so that:

```
ExecutionLink S X Y
```

and $s(S, X, Y)$ is the strength of this ExecutionLink. Here, we are rewarding Atoms that are produced by simple schemata based on simple inputs.

The combined result of these heuristics is to cause the system to prefer simpler explanations, analysis, procedures and ideas. But of course this is only an apriori preference, and if more complex entities prove more useful, these will quickly gain greater strength and importance in the system.

Implementationally, these various processes are carried out by the OccamsRazor MindAgent. This dynamic selects ConceptNodes based on a combination of:

- importance
- time since the a priori probability was last updated (a long time is preferred)

It selects ExecutionLinks based on importance and based on the amount of time since they were last visited by the OccamsRazor MindAgent. And it selects PredicateNodes based on importance, filtering out PredicateNodes it has visited before.

EFTA00624497

Chapter 42
Map Formation

Abstract

42.1 Introduction

In Chapter 20 we distinguished the explicit versus implicit aspects of knowledge representation in CogPrime. The explicit level consists of Atoms with clearly comprehensible meanings, whereas the implicit level consists of "maps" – collections of Atoms that become important in a coordinated manner, analogously to cell assemblies in an attractor neural net. The combination of the two is valuable because the world-patterns useful to human-like minds in achieving their goals, involve varying degrees of isolation and interpenetration, and their effective goal-oriented processing involves both symbolic manipulation (for which explicit representation is most valuable) and associative creative manipulation (for which distributed, implicit representation is most valuable).

The chapters since have focused primarily on explicit representation, commenting on the implicit "map" level only occasionally. There are two reasons for this: one theoretical, one pragmatic. The theoretical reason is that the majority of map dynamics and representations are implicit in Atom-level correlates. And the pragmatic reason is that, at this stage, we simply do not know as much about CogPrime maps as we do about CogPrime Atoms. Maps are emergent entities and, lacking a detailed theory of CogPrime dynamics, the only way we have to study them in detail is to run CogPrime systems and mine their System Activity Tables and logs for information. If CogPrime research goes well, then updated versions of this book may include more details on observed map dynamics in various contexts.

In this chapter, however, we finally turn our gaze directly to maps and their relationships to Atoms, and discuss processes that convert Atoms into maps (expansion) and vice versa (encapsulation). These processes represent a bridge between the concretely-implemented and emergent aspects of CogPrime's mind.

Map encapsulation is the process of recognizing Atoms that tend to become important in a coordinated manner, and then creating new Atoms grouping these. As such it is essentially a form of AtomSpace pattern mining. In terms of patternist philosophy, map encapsulation is a direct incarnation of the so-called "cognitive equation"; that is, the process by which the mind recognizes patterns in itself, and then embodies these patterns as new content within

EFTA00624498

itself – an instance of what Hofstadter famously labeled a "strange loop" [Hof79]. In SMEPH terms, the encapsulation process is how CogPrime explicitly studies its own derived hypergraph and then works to implement this derived hypergraph more efficiently by recapitulating it at the concretely-implemented-mind level. This of course may change the derived hypergraph considerably. Among other things, map encapsulation has the possibility of taking the things that were the most abstract, *highest level* patterns in the system and forming new patterns involving them and their interrelationships – thus building the highest level of patterns in the system higher and higher. Figures 42.2 and 42.1 illustrate concrete examples of the process.

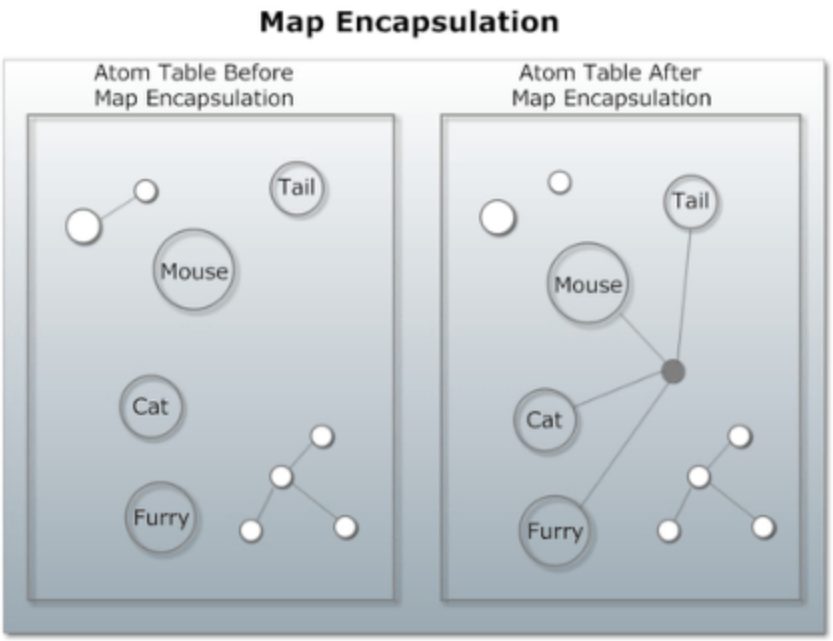


Fig. 42.1: Illustration of the process of creating explicit Atoms corresponding to a pattern previously represented as a distributed "map."

Map expansion, on the other hand, is the process of taking knowledge that is explicitly represented and causing the AtomSpace to represent it *implicitly*, on the map level. In many cases this will happen automatically. For instance, a ConceptNode C may turn into a concept map if the importance updating process iteratively acts in such a way as to create/reinforce a map consisting of C and its relata. Or, an Atom-level InheritanceLink may implicitly spawn a map-level InheritanceEdge (in SMEPH terms). However, there is one important case in which Atom-to-map conversion must occur explicitly: the expansion of compound ProcedureNodes into procedure maps. This must occur explicitly because the process graphs inside ProcedureNodes have no dynamics going on except evaluation; there is no opportunity for them to manifest themselves as maps, unless a MindAgent is introduced that explicitly does so. Of course, just unfolding a Combo tree into a procedure map doesn't intrinsically make it a significant part of the derived hypergraph - but it opens the door for the inter-cognitive-process integration that may make this occur.

EFTA00624499

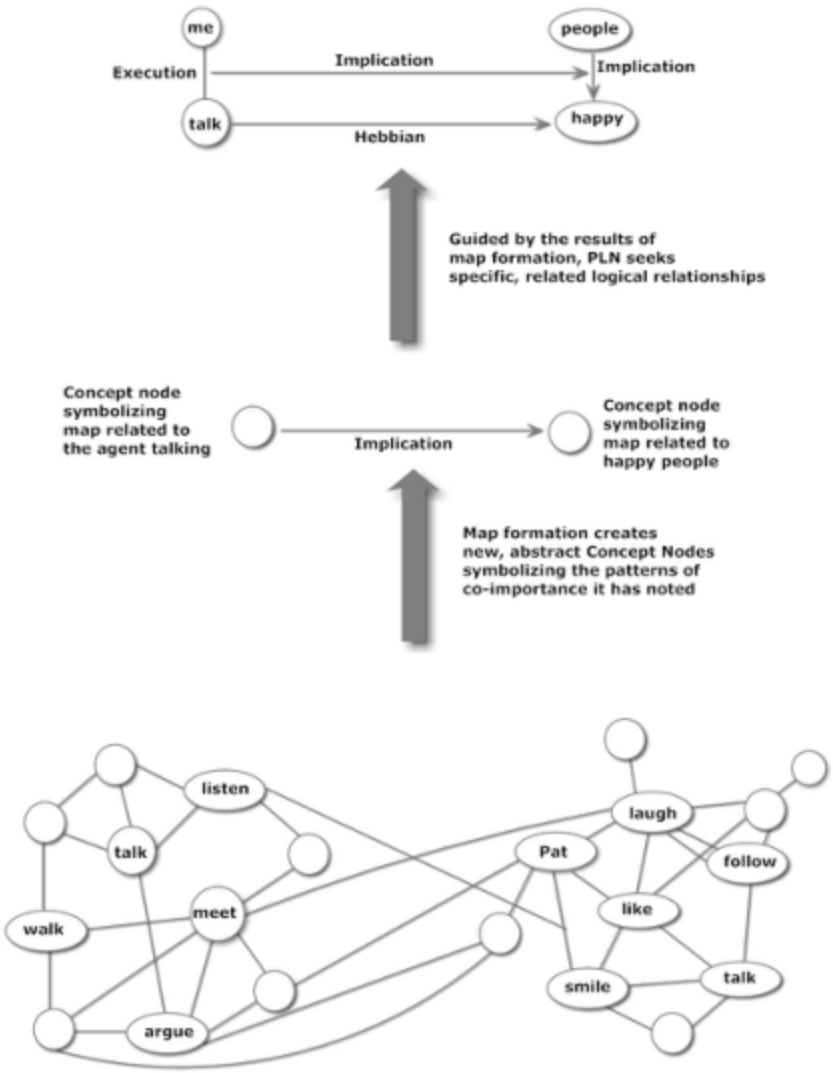


Fig. 42.2: Illustration of the process of creating explicit Atoms corresponding to a pattern previously represented as a distributed "map."

42.2 Map Encapsulation

Returning to encapsulation: it may be viewed as a form of *symbolization*, in which the system creates concrete entities to serve as symbols for its own emergent patterns. It can then study an emergent pattern's interrelationships by studying the interrelationships of the symbol with other symbols.

For instance, suppose a system has three derived-hypergraph ConceptVertices A, B and C, and observes that:

EFTA00624500

```
InheritanceEdge A B
InheritanceEdge B C
```

Then encapsulation may create ConceptNodes A', B' and C' for A, B and C, and InheritanceLinks corresponding to the InheritanceEdges, where e.g. A' is a set containing all the Atoms contained in the static map A. First-order PLN inference will then immediately conclude:

```
InheritanceLink A' C'
```

and it may possibly do so with a higher strength than the strength corresponding to the (perhaps not significant) InheritanceEdge between A and C. But if the encapsulation is done right then the existence of the new InheritanceLink will indirectly cause the formation of the corresponding:

```
InheritanceEdge A C
```

via the further action of inference, which will use (InheritanceLink A' C') to trigger the inference of further inheritance relationships between members of A' and members of C', which will create an emergent inheritance between members of A (the map corresponding to A') and C (the map corresponding to C').

The above example involved the conversion of static maps into ConceptNodes. Another approach to map encapsulation is to represent the fact that a set of Atoms constitutes a map as a predicate; for instance if the nodes A, B and C are habitually used together, then the predicate P may be formed, where:

```
P =
AND
   A is used at time T
   B is used at time T
   C is used at time T
```

The habitualness of A, B and C being used together will be reflected in the fact that P has a surprisingly high truth value. By a simple concept formation heuristic, this may be used to form a link AND(A, B, C), so that:

```
AND(A, B, C) is used at time T
```

This composite link AND(A, B, C) is then an embodiment of the map in single-Atom form.

Similarly, if a set of schemata is commonly used in a certain series, this may be recognized in a predicate, and a composite schema may then be created embodying the component schemata. For instance, suppose it is recognized as a pattern that:

```
AND
   S1 is used at time T on input I1 producing output O1
   S2 is used at time T+s on input O1 producing output O2
```

Then we may explicitly create a schema that consists of S1 taking input and feeding its output to S2. This cannot be done via any standard concept formation heuristic; it requires a special process.

One might wonder why this Atom-to-map conversion process is necessary: Why not just let maps combine to build new maps, hierarchically, rather than artificially transforming some maps into Atoms and letting maps then form from these map-representing Atoms. It is all a matter of precision. Operations on the map level are fuzzier and less reliable than operations on the Atom level. This fuzziness has its positive and its negative aspects. For example, it is

EFTA00624501

good for spontaneous creativity, but bad for constructing lengthy, confident chains of thought. WIKISOURCE:ActivityTables

42.3 Atom and Predicate Activity Tables

A major role in map formation is played by a collection of special tables. Map encapsulation takes place, not by data mining directly on the AtomTable, but by data mining on these special tables constructed from the AtomTable, specifically with efficiency of map mining in mind.

First, there is the Atom Utilization Table, which may be derived from the SystemActivityTable. The Atom Utilization Table, in its most simple possible version, takes the form shown in Table 42.1.

Time		Atom Handle H
?	?	?
T	?	(Effort spent on Atom H at time t, utility derived from atom H at time t)
?	?	?

Table 42.1: Atom Utilization Table

The calculation of "utility" values for this purpose must be done in a "local" way by MindAgents, rather than by a global calculation of the degree to which utilizing a certain Atom has led to the achievement of a certain system goal (this kind of global calculation would be better in principle, but it would require massive computational effort to calculate for every Atom in the system at frequent intervals). Each MindAgent needs to estimate how much utility it has obtained from a given Atom, as well as how much effort it has spent on this Atom, and report these numbers to the Atom Utilization Table.

The normalization of effort values is simple, since effort can be quantified in terms of time and space expended. Normalization of utility values is harder, as it is difficult to define a common scale to span all the different MindAgents, which in some cases carry out very different sorts of operations. One reasonably "objective" approach is to assign each MindAgent an amount of "utility credit", at time T, equal to the amount of currency that the MindAgent has spent since it last disbursed its utility credits. It may then divide up its utility credit among the Atoms it has utilized. Other reasonable approaches may also be defined.

The use of utility and utility credit for Atoms and MindAgents is similar to the stimulus used in the Attention allocation system. There, MindAgents reward Atoms with stimulus to indicate that their short and long term importance should be increased. Merging utility and stimulus is a natural approach to implementing utility in OpenCogPrime.

Note that there are many practical manifestations that the abstract notion of an ActivityTable may take. It could be an ordinary row-and-column style table, but that is not the only nor the most interesting possibility. An ActivityTable may also be effectively stored as a series of graphs corresponding to time intervals - one graph for each interval, consisting of HebbianLinks formed solely based on importance during that interval. In this case it is basically a set of graphs, which may be stored for instance in an AtomTable, perhaps with a special index.

Then there is the Procedure Activity Table, which records the inputs and outputs associated with procedures:

EFTA00624502

Time		ProcedureNode Handle H
?	?	?
T	?	(Inputs to H, Outputs from H)
?	?	?

Table 42.2: Procedure Activity Table for a Particular MindAgent

Data mining on these tables may be carried out by a variety of algorithms (see MapMining) - the more advanced the algorithm, the fuller the transfer from the derived-hypergraph level to the concretely-implemented level. There is a tradeoff here similar to that with attention allocation - if too much time is spent studying the derived hypergraph, then there will not be any interesting cognitive dynamics going on anymore because other cognitive processes get no resources, so the map encapsulation process will fail because there is nothing to study!

These same tables may be used in the attention allocation process, for assigning of MindAgent-specific AttentionValues to Atoms. WIKISOURCE:MapMining

42.4 Mining the AtomSpace for Maps

Searching for general maps in a complex AtomSpace is an unrealistically difficult problem, as the search space is huge. So, the bulk of map-mining activity involves looking for the most simple and obvious sorts of maps. A certain amount of resources may also be allocated to looking for subtler maps using more resource-intensive methods.

The following categories of maps can be searched for at relatively low cost:

- Static maps
- Temporal motif maps

Conceptually, a static map is simply a set of Atoms that all tend to be active at the same time.

Next, by a "temporal motif map" we mean a set of pairs:

$$(A_i, t_i)$$

of the type:

$$(Atom, int)$$

so that for many activation cycle indices T, A_i is highly active at some time very close to index $T + t_i$. The reason both static maps and temporal motif maps are easy to recognize is that they are both simply repeated patterns.

Perceptual context formation involves a special case of static and temporal motif mining. In perceptual context formation, one specifically wishes to mine maps involving perceptual nodes associated with a single interaction channel (see Chapter 26 for interaction channel). These maps then represent real-world contexts, that may be useful in guiding real-world-oriented goal activity (via schema-context-goal triads).

In CogPrime so far we have considered three broad approaches for mining static and temporal motif maps from AtomSpaces:

EFTA00624503

- Frequent subgraph mining, frequent itemset mining, or other sorts of datamining on Activity Tables
- Clustering on the network of HebbianLinks
- Evolutionary Optimization based datamining on Activity Tables

The first two approaches are significantly more time-efficient than the latter, but also significantly more limited in the scope of patterns they can find.

Any of these approaches can be used to look for maps subject to several types of constraints, such as for instance:

- **Unconstrained**: maps may contain any kinds of Atoms
- **Strictly constrained**: maps may only contain Atom types contained on a certain list
- **Probabilistically constrained**: maps must contain Atom types contained on a certain list, as x% of their elements
- **Trigger-constrained**: the map must contain an Atom whose type is on a certain list, as its most active element

Different sorts of constraints will lead to different sorts of maps, of course. We don't know at this stage which sorts of constraints will yield the best results. Some special cases, however, are reasonably well understood. For instance:

- procedure encapsulation, to be discussed below, involves searching for (strictly-constrained) maps consisting solely of ProcedureInstanceNodes.
- to enhance goal achievement, it is likely useful to search for trigger-constrained maps triggered by Goal Nodes.

What the MapEncapsulation CIM-Dynamic (Concretely-Implemented-Mind-Dynamic, see Chapter 19) does once it finds a map, is dependent upon the type of map it's found. In the special case of procedure encapsulation, it creates a compound ProcedureNode (selecting SchemaNode or PredicateNode based on whether the output is a TruthValue or not). For static maps, it creates a ConceptNode, which links to all members of the map with MemberLinks, the weight of which is determined by the degree of map membership. For dynamic maps, it creates PredictiveImplication links depicting the pattern of change.

42.4.1 Frequent Itemset Mining for Map Mining

One class of technique that is useful here is *frequent itemset mining* (FIM), a process that looks to find all frequent combinations of items occurring in a set of data. Another useful class of algorithms is greedy or stochastic itemset mining, which does roughly the same thing as FIM but without being completely exhaustive (the advantage being greater execution speed). Here we will discuss FIM, but the basic concepts are the same if one is doing greedy or stochastic mining instead.

The basic goal of frequent itemset mining is to discover frequent subsets in a group of items. One knows that for a set of N items, there are 2N-1 possible subgroups. To avoid the exponential explosion of subsets, one may compute the frequent *itemsets* in several rounds. Round i computes all frequent i-itemsets.

EFTA00624504

A round has two steps: candidate generation and candidate counting. In the candidate generation step, the algorithm generates a set of candidate i-itemsets whose support - a minimum percentage of events in which the item must appear - has not been yet been computed. In the candidate-counting step, the algorithm scans its memory database, counting the support of the candidate itemsets. After the scan, the algorithm discards candidates with support lower than the specified minimum (an algorithm parameter) and retains only the frequent i-itemsets. The algorithm reduces the number of tested subsets by pruning a priori those candidate itemsets that cannot be frequent, based on the knowledge about infrequent itemsets obtained from previous rounds. So for instance if $\{A, B\}$ is a frequent 2-itemset then $\{A, B, C\}$ may possibly be a 3-itemset, on the contrary if $\{A, B\}$ is not a frequent itemset then $\{A, B, C\}$, as well as any super set of $\{A, B\}$, will be discarded. Although the worst case of this sort of algorithm is exponential, practical executions are generally fast, depending essentially on the support limit.

To apply this kind of approach to search for static maps, one simply creates a large set of sets of Atoms – one set for each time-point. In the set S(t) corresponding to time t, we place all Atoms that were firing activation at time t. The itemset miner then searches for sets of Atoms that are subsets of many different S(t) corresponding to many different times t. These are Atom sets that are frequently co-active.

Table **??** presents a typical example of data prepared for frequent itemset mining, in the context of context formation via static-map recognition. Columns represent important nodes and rows indicate time slices. For simplicity, we have *thresholded* the values and show only activity values; so that a 1 in a cell indicates that the Atom indicated by the column was being utilized at the time indicated by the row.

In the example, if we assume minimum support as 50 percent, the context nodes C1 = {Q, R}, and C2 = {Q, T, U} would be created.

Using frequent itemset mining to find temporal motif maps is a similar, but slightly more complex process. Here, one fixes a time-window W. Then, for each activation cycle index t, one creates a set S(t) consisting of pairs of the form:

```
(A, s)
```

where A is an Atom and $0 \leq s \leq W$ is an integer temporal offset. We have:

```
(A,s) ''within'' S(t)
```

if Atom A is firing activation at time t+s. Itemset mining is then used to search for common subsets among the S(t). These common subsets are common patterns of temporal activation, i.e. repeated temporal motifs.

The strength of this approach is its ability to rapidly search through a huge space of possibly significant subsets. Its weakness is its restriction to finding maps that can be incrementally built up from smaller maps. How significant this weakness is, depends on the particular statistics of map occurrence in CogPrime. Intuitively, we believe frequent itemset mining can perform rather well in this context, and our preliminary experiments have supported this intuition.

Frequent Subgraph Mining for Map Mining

A limitation of FIM techniques, from a CogPrime perspective, is that they are intended for relational databases (RDBs); but the information about co-activity in a CogPrime instance is generally going to be more efficiently stored as graphs rather than RDB's. Indeed an ActivityTable may be effectively stored as a series of graphs corresponding to time intervals - one

EFTA00624505

graph for each interval, consisting of HebbianLinks formed solely based on importance during that interval. From ActivityTable stores like this, the way to find maps is not frequent itemset mining but rather frequent subgraph mining – a variant of FIM that is conceptually similar but algorithmically more subtle, and on which there has arisen a significant literature in recent years. We have already briefly discussed this technology in Chapter 37 on pattern mining the Atomspace – map mining being an important special case of Atomspace pattern mining. As noted there, some of the many approaches to frequent subgraph mining are described in [HWP03, KK01].

42.4.2 Evolutionary Map Detection

Just as general Atomspace pattern mining may be done via evolutionary learning as well as greedy mining, the same holds for the special case of map mining. Complementary to the itemset mining approach, the CogPrime design also uses evolutionary optimization to find maps. Here the data setup is the same as in the itemset mining case, but instead of using an incremental search approach, one sets up a population of subsets of the sets S(t), and seeks to evolve the population to find an optimally fit S(t). Fitness is defined simply as high frequency - relative to the frequency one would expect based on statistical independence assumptions alone.

In principle one could use evolutionary learning to do all map encapsulation, but this isn't computationally feasible – it would limit too severely the amount of map encapsulation that could be done. Instead, evolutionary learning must be supplemented by some more rapid, less expensive technique.

42.5 Map Dynamics

Assume one has a collection of Atoms, with:

- Importance values I(A), assigned via the economic attention allocation mechanism.
- HebbianLink strengths $(HebbianLink\ A\ B).tv.s$, assigned as (loosely speaking) the probability of B's importance assuming A's importance.

Then, one way to search for static maps is to look for collections C of Atoms that are strong clusters according to HebbianLinks. That is, for instance, to find collections C so that:

- The mean strength of $(HebbianLink\ A\ B).tv.s$, where A and B are in the collection C, is large.
- The mean strength of $(HebbianLink\ A\ Z).tv.s$, where A is in the collection C and Z is not, is small.

(this is just a very simple cluster quality measurement; there is a variety of other cluster quality measurements one might use instead.)

Dynamic maps may be more complex, for instance there might be two collections C1 and C2 so that:

- Mean strength of (HebbianLink A B).s, where A is in C1 and B is in C2

EFTA00624506

- Mean strength of (HebbianLink B A).s, where B is in C2 and A is in C1

are both very large.

A static map will tend to be an attractor for CogPrime's attention-allocation-based dynamics, in the sense that when a few elements of the map are acted upon, it is likely that other elements of the map will soon also come to be acted upon. The reason is that, if a few elements of the map are acted upon usefully, then their importance values will increase. Node probability inference based on the HebbianLinks will then cause the importance values of the other nodes in the map to increase, thus increasing the probability that the other nodes in the map are acted upon. Critical here is that the HebbianLinks have a higher weight of evidence than the node importance values. This is because the node importance values are assumed to be ephemeral - they reflect whether a given node is important at a given moment or not - whereas the HebbianLinks are assumed to reflect longer-lasting information.

A dynamic map will also be an attractor, but of a more complex kind. The example given above, with C1 and C2, will be a periodic attractor rather than a fixed-point attractor. WIKISOURCE:ProcedureEncapsulation

42.6 Procedure Encapsulation and Expansion

One of the most important special cases of map encapsulation is procedure encapsulation. This refers to the process of taking a schema/predicate map and embodying it in a single ProcedureNode. This may be done by mining on the Procedure Activity Table, described in Activity Tables, using either:

- a special variant of itemset mining that seeks for procedures whose outputs serve as inputs for other procedures.
- Evolutionary optimization with a fitness function that restricts attention to sets of procedures that form a digraph, where the procedures lie at the vertices and an arrow from vertex A to vertex B indicates that the outputs of A become the inputs of B.

The reverse of this process, procedure expansion, is also interesting, though algorithmically easier - here one takes a compound ProcedureNode and expands its internals into a collection of appropriately interlinked ProcedureNodes. The challenge here is to figure out where to split a complex Combo tree into subtrees. But if the Combo tree has a hierarchical structure then this is very simple; the hierarchical subunits may simply be split into separate ProcedureNodes.

These two processes may be used in sequence to interesting effect: expanding an important compound ProcedureNode so it can be modified via reinforcement learning, then encapsulating its modified version for efficient execution, then perhaps expanding this modified version later on.

To an extent, the existence of these two different representations of procedures is an artifact of CogPrime's particular software design (and ultimately, a reflection of certain properties of the von Neumann computing architecture). But it also represents a more fundamental dichotomy, between:

- Procedures represented in a way that allows them to be dynamically, improvisationally restructured via interaction with other mental processes during the execution process.

EFTA00624507

- Procedures represented in a way that is relatively encapsulated and mechanical, allowing collaboration with other aspects of the mind during execution only in fairly limited ways

Conceptually, we believe that this is a very useful distinction for a mind to make. In nearly any reasonable cognitive architecture, it's going to be more efficient to execute a procedure if that procedure is treated as something with a relatively rigid structure, so it can simply be executed without worrying about interactions except in a few specific regards. This is a strong motivation for an artificial cognitive system to have a dual (at least) representation of procedures, or else a subtle representation that is flexible regarding its degree of flexibility, and automagically translates constraint into efficiency.

42.6.1 Procedure Encapsulation in More Detail

A procedure map is a temporal motif: it is a set of Atoms (ProcedureNodes), which are habitually executed in a particular temporal order, and which implicitly pass arguments amongst each other. For instance, if procedure A acts to create node X, and procedure B then takes node X as input, then we may say that A has implicitly passed an argument to B.

The encapsulation process can recognize some very subtle patterns, but a fair fraction of its activity can be understood in terms of some simple heuristics.

For instance, the map encapsulation process will create a node

$$h = Bfg = f \circ g = f \textit{ composed with } g$$

(B as in combinatory logic) when there are many examples in the system of:

```
ExecutionLink g x y
ExecutionLink f y z
```

The procedure encapsulation process will also recognize larger repeated subgraphs, and their patterns of execution over time. But some of its recognition of larger subgraphs may be done incrementally, by repeated recognition of simple patterns like the ones just described.

42.6.2 Procedure Encapsulation in the Human Brain

Finally, we briefly discuss some conceptual issues regarding the relation between CogPrime procedure encapsulation and the human brain. Current knowledge of the human brain is weak in this regard, but we won't be surprised if, in time, it is revealed that the brain stores procedures in several different ways, that one distinction between these different ways has to do with degree of openness to interactions, and that the less open ways lead to faster execution.

Generally speaking, there is good evidence for a neural distinction between procedural, episodic and declarative memory. But knowledge about distinctions between different kinds of procedural memory is scanter. It is known that procedural knowledge can be "routinized" - so that, e.g., once you get good at serving a tennis ball or solving a quadratic equation, your brain handles the process in a different way than before when you were learning. And it seems plausible that routinized knowledge, as represented in the brain, has fewer connections

EFTA00624508

back to the rest of the brain than the pre-routinized knowledge. But there will be much firmer knowledge about such things in the coming years and decades as brain scanning technology advances.

Overall, there is more knowledge in cognitive and neural science about motor procedures than cognitive procedures (see e.g. [SW05]. In the brain, much of motor procedural memory resides in the pre-motor area of the cortex. The *motor plans* stored here are not static entities and are easily modified through feedback, and through interaction with other brain regions. Generally, a motor plan will be stored in a distributed way across a significant percentage of the premotor cortex; and a complex or multipart actions will tend to involve numerous sub-plans, executed in both parallel and in serial. Often what we think of as separate/distinct motor-plans may in fact be just slightly different combinations of subplans (a phenomenon also occurring with schema maps in CogPrime).

In the case of motor plans, a great deal of the *routinization* process has to do with learning the timing necessary for correct coordination between muscles and motor subplans. This involves integration of several brain regions - for instance, timing is handled by the cerebellum to a degree, and some motor-execution decisions are regulated by the basal ganglia.

One can think of many motor plans as involving abstract and concrete sub-plans. The abstract sub-plans are more likely to involve integration with those parts of the cortex dealing with conceptual thought. The concrete sub-plans have highly optimized timings, based on close integration with cerebellum, basal ganglia and so forth - but are not closely integrated with the conceptualization-focused parts of the brain. So, a rough CogPrime model of human motor procedures might involve schema maps coordinating the abstract aspects of motor procedures, triggering activity of complex SchemaNodes containing precisely optimized procedures that interact carefully with external actuators. WIKISOURCE:MapsAndAttention

42.7 Maps and Focused Attention

The cause of map formation is important to understand. Formation of small maps seems to follow from the logic of focused attention, along with hierarchical maps of a certain nature. But the argument for this is somewhat subtle, involving cognitive synergy between PLN inference and economic attention allocation.

The nature of PLN is that the effectiveness of reasoning is maximized by (among other strategies) minimizing the number of incorrect independence assumptions. If reasoning on N nodes, the way to minimize independence assumptions is to use the full inclusion-exclusion formula to calculate interdependencies between the N nodes. This involves 2^N terms, one for each subset of the N nodes. Very rarely, in practical cases, will one have significant information about all these subsets. However, the nature of focused attention is that the system seeks to find out about as many of these subsets as possible, so as to be able to make the most accurate possible inferences, hence minimizing the use of unjustified independence assumptions. This implies that focused attention cannot hold too many items within it at one time, because if N is too big, then doing a decent sampling of the subsets of the N items is no longer realistic.

So, suppose that N items have been held within focused attention, meaning that a lot of predicates embodying combinations of N items have been constructed and evaluated and reasoned on. Then, during this extensive process of attentional focus, many of the N items will be useful in combination with each other - because of the existence of predicates joining the items.

EFTA00624509

Hence, many HebbianLinks will grow between the N items - causing the set of N items to form a map.

By this reasoning, it seems that focused attention will implicitly be a map formation process - even though its immediate purpose is not map formation, but rather accurate inference (inference that minimizes independence assumptions by computing as many cross terms as is possible based on available direct and indirect evidence). Furthermore, it will encourage the formation of maps with a small number of elements in them (say, N<10). However, these elements may themselves be ConceptNodes grouping other nodes together, perhaps grouping together nodes that are involved in maps. In this way, one may see the formation of hierarchical maps, formed of clusters of clusters of clusters..., where each cluster has N<10 elements in it. These hierarchical maps manifest the abstract *dual network* concept that occurs frequently in CogPrime philosophy.

It is tempting to postulate that any intelligent system must display similar properties - so that focused attention, in general, has a strictly limited scope and causes the formation of maps that have *central cores* of roughly the same size as its scope. If this is indeed a general principle, it is an important one, because it tells you something about the general structure of derived hypergraphs associated with intelligent systems, based on the computational resource constraints of the systems.

The scope of an intelligent system's attentional focus would seem to generally increase logarithmically with the system's computational power. This follows immediately if one assumes that attentional focus involves free intercombination of the items within it. If attentional focus is the major locus of map formation, then - lapsing into SMEPH-speak - it follows that the bulk of the ConceptVertices in the intelligent system's derived hypergraphs may correspond to maps focused on a fairly small number of other ConceptVertices. In other words, derived hypergraphs may tend to have a fairly localized structure, in which each ConceptVertex has very strong InheritanceEdges pointing from a handful of other ConceptVertices (corresponding to the other things that were in the attentional focus when that ConceptVertex was formed). WIKISOURCE:RecognizingAndCreatingSelfReferentialStructures

42.8 Recognizing and Creating Self-Referential Structures

Finally, this brief section covers a large and essential topic: how CogPrime will be able to recognize and create large-scale *self-referential structures*.

Some of the most essential structures underlying human-level intelligence are self-referential in nature. These include:

- the phenomenal self (see Thomas Metzinger's book "Being No One")
- the will
- reflective awareness

These structures are arguably not critical for basic survival functionality in natural environments. However, they are important for adequate functionality within advanced social systems, and for abstract thinking regarding science, humanities, arts and technology.

Recall that in Chapter 3 of Part 1 these entities are formalized in terms of hypersets and, the following recursive definitions are given:

EFTA00624510

- *"S is conscious of X" is defined as: The declarative content that "S is conscious of X" correlates with "X is a pattern in S"*
- *"S wills X" is defined as: The declarative content that "S wills X" causally implies "S does X"*
- *"X is part of S's self" is defined as: The declarative content that "X is a part of S's self" correlates with "X is a persistent pattern in S over time"*

Relatedly, one may posit multiple similar processes that are mutually recursive, e.g.

- S is conscious of T and U
- T is conscious of S and U
- U is conscious of S and T

The cognitive importance of this sort of mutual recursion is further discussed in Appendix **??**.

According to the philosophy underlying CogPrime, none of these are things that should be programmed into an artificial mind. Rather, they must emerge in the course of a mind's self-organization in connection with its environment. However, a mind may be constructed so that, by design, these sorts of important self-referential structures are *encouraged* to emerge.

42.8.1 Encouraging the Recognition of Self-Referential Structures in the AtomSpace

How can we do this - encourage a CogPrime instance to recognize complex self-referential structures that may exist in its AtomTable? This is important, because, according to the same logic as map formation: if these structures are explicitly recognized when they exist, they can then be reasoned on and otherwise further refined, which will then cause them to exist more definitively, and hence to be explicitly recognized as yet more prominent patterns ... etc. The same virtuous cycle via which ongoing map recognition and encapsulation is supposed to lead to concept formation, may be posited on the level of complex self-referential structures, leading to their refinement, development and ongoing complexity.

One really simple way is to encode self-referential operators in the Combo vocabulary, that is used to represent the procedures grounding GroundedPredicateNodes.

That way, one can recognize self-referential patterns in the AtomTable via standard CogPrime methods like MOSES and integrative procedure and predicate learning as discussed in Chapter 41, so long as one uses Combo trees that are allowed to include self-referential operators at their nodes. All that matters is that one is able to take one of these Combo trees, compare it to an AtomTable, and assess the degree to which that Combo tree constitutes a pattern in that AtomTable.

But how can we do this? How can we match a self-referential structure like:

```
EquivalenceLink
  EvaluationLink will (S,X)
  CausalImplicationLink
    EvaluationLink will (S,X)
    EvaluationLink do (S,X)
```

against an AtomTable or portion thereof?

The question is whether there is some "map" of Atoms (some set of PredicateNodes) willMap, so that we may infer the SMEPH (see Chapter 14) relationship:

EFTA00624511

```
EquivalenceEdge
  EvaluationEdge willMap (S,X)
  CausalImplicationEdge
    EvaluationEdge willMap (S,X)
    EvaluationEdge doMap (S,X)
```

as a statistical pattern in the AtomTable's history over the recent past. (Here, doMap is defined to be the map corresponding to the built-in "do" predicate.)

If so, then this map willMap, may be encapsulated in a single new Node (call it willNode), which represents the system's will. This willNode may then be explicitly reasoned upon, used within concept creation, etc. It will lead to the spontaneous formation of a more sophisticated, fully-fleshed-out will map. And so forth.

Now, what is required for this sort of statistical pattern to be recognizable in the AtomTable's history? What is required is that EquivalenceEdges (which, note, must be part of the Combo vocabulary in order for any MOSES-related algorithms to recognize patterns involving them) must be defined according to the logic of hypersets rather than the logic of sets. What is fascinating is that this is no big deal! In fact, the AtomTable software structures support this automatically; it's just not the way most people are used to thinking about things. There is no reason, in terms of the AtomTable, not to create self-referential structures like the one given above.

The next question, though, is how do we calculate the truth values of structures like those above. The truth value of a hyperset structure turns out to be an infinite order probability distribution, which a complex and peculiar entity [Goe10a]. Infinite-order probability distributions are partially-ordered, and so one can compare the extent to which two different self-referential structures apply to a given body of data (e.g. an AtomTable), via comparing the infinite-order distros that constitute their truth values. In this way, one can recognize self-referential patterns in an AtomTable, and carry out encapsulation of self-referential maps. This sounds very abstract and complicated, but the class of infinite-order distributions defined in the above-referenced papers actually have their truth values defined by simple matrix mathematics, so there is really nothing that abstruse involved in practice.

Finally, there is the question of how these hyperset structures are to be logically manipulated within PLN. The answer is that regular PLN inference can be applied perfectly well to hypersets, but some additional hyperset operations may also be introduced; these are currently being researched.

Clearly, with this subtle, currently unimplemented component of the CogPrime design we have veered rather far from anything the human brain could plausibly be doing in detail. But yet, some meaningful connections may be drawn. In Chapter 13 of Part 1 we have discussed how probabilistic logic might emerge from the brain, and also how the brain may embody self-referential structures like the ones considered here, via (perhaps using the hippocampus) encoding whole neural nets as inputs to other neural nets. Regarding infinite-order probabilities, it is certainly the case that the brain is efficient at carrying out operations equivalent to matrix manipulations (e.g. in vision and audition), and [Goe10a] reduced infinite-order probabilities to finite matrix manipulations, so that it's not completely outlandish to posit the brain could be doing something mathematically analogous. Thus, all in all, it seems at least *plausible* that the brain could be doing something roughly analogous to what we've described here, though the details would obviously be very different.

EFTA00624512

Section VII
Communication Between Human and Artificial Minds

Chapter 43
Communication Between Artificial Minds

43.1 Introduction

Language is a key aspect of human intelligence, and seems to be one of two critical factors separating humans from other intelligent animals – the other being the ability to use tools. Steven Mithen [Mit96] argues that the key factor in the emergence of the modern human mind from its predecessors was the coming-together of formerly largely distinct mental modules for linguistic communication and tool making/use. Other animals do appear to have fairly sophisticated forms of linguistic communication, which we don't understand very well at present; but as best we can tell, modern human language has many qualitatively different aspects from these, which enable it to synergize effectively with tool making and use, and which have enabled it to co-evolve with various aspects of tool-dependent culture.

Some AGI theorists have argued that, since the human brain is largely the same as that of apes and other mammals without human-like language, the emulation of human-like language is not the right place to focus if one wants to build human-level AGI. Rather, this argument goes, one should proceed in the same order that evolution did – start with motivated perception and action, and then once these are mastered, human-like language will only be a small additional step. We suspect this would indeed be a viable approach – but may not be well suited for the hardware available today. Robot hardware is quite primitive compared to animal bodies, but the kind of motivated perception and action that non-human animals do is extremely body-centric (even more so than is the case in humans). On the other hand, modern computing technology is quite sophisticated as regards language – we program computers (including AIs) using languages of a sort, for example. This suggests that on a pragmatic basis, it may make sense to start working with language at an earlier stage in AGI development, than the analogue with the evolution of natural organisms would suggest.

The CogPrime architecture is compatible with a variety of different approaches to language learning and capability, and frankly at this stage we are not sure which approach is best. Our intention is to experiment with a variety of approaches and proceed pragmatically and empirically. One option is to follow the more "natural" course and let sophisticated non-linguistic cognition emerge first, before dealing with language in any serious way – and then encourage human-like language facility to emerge via experience. Another option is to integrate some sort of traditional computational linguistics system into CogPrime, and then allow CogPrime's learning algorithms to modify this system based on its experience. Discussion of this latter option occupies most of this section of the book – involves many tricks and compromises, but

EFTA00624516

could potentially constitute a faster route to success. Yet another option is to communicate with young CogPrime systems using an invented language halfway between the human-language and programming-language domains, such as Lojban (this possibility is discussed in Appendix E).

In this initial chapter on communication, we will pursue a direction quite different from the latter chapters, and discuss a kind of communication that we think may be very valuable in the CogPrime domain, although it has no close analogue among human beings. Many aspects of CogPrime closely resemble aspects of the human mind; but in the end CogPrime is not intended as an emulation of human intelligence, and there are some aspects of CogPrime that bear no resemblance to anything in the human mind, but exploit some of the advantages of digital computing infrastructure over neural wetware. One of the latter aspects is *Psynese*, a word we have introduced to refer to direct mind-to-mind information transfer between artificial minds.

Psynese has some relatively simple practical applications: e.g. it could aid with the use of linguistic resources and hand-coded or statistical language parsers within a learning-based language system, to be discussed in following chapters. In this use case, one sets up one CogPrime using the traditional NLP approaches, and another CogPrime using a purer learning-based approach, and lets the two systems share mind-stuff in a controlled way. Psynese may also be useful in the context of intelligent virtual pets, where one may wish to set up a CogPrime representing "collective knowledge" of multiple virtual pets.

But it also has some grander potential implications, such as the ability to fuse multiple AI systems into "mindplexes" as discussed in Chapter 12 of Part 1.

One might wonder why a community of two or more CogPrime s would need a language at all, in order to communicate. After all, unlike humans, CogPrime systems can simply exchange "brain fragments" - subspaces of their Atomspaces. One CogPrime can just send relevant nodes and links to another CogPrime (in binary form, or in an XML representation, etc.), bypassing the linear syntax of language. This is in fact the basis of Psynese: why transmit linear strings of characters when one can directly transit Atoms? But the details are subtler than it might at first seem.

One CogPrime can't simply "transfer a thought" to another CogPrime. The problem is that the meaning of an Atom consists largely of its relationships with other Atoms, and so to pass a node to another CogPrime, it also has to pass the Atoms that it is related to, and so on, and so on. Atomspaces tend to be densely interconnected, and so to transmit one thought fully accurately, a CogPrime system is going to end up having to transmit a copy of its entire Atomspace! Even if privacy were not an issue, this form of communication (each utterance coming packaged with a whole mind-copy) would present rather severe processing load on the communicators involved.

The idea of Psynese is to work around this interconnectedness problem by defining a mechanism for CogPrime instances to query each others' minds directly, and explicitly represent each others' concepts internally. This doesn't involve any unique cognitive operations besides those required for ordinary individual thought, but it requires some unique ways of wrapping up these operations and keeping track of their products.

Another idea this leads to is the notion of a PsyneseVocabulary: a collection of Atoms, associated with a community of CogPrime s, approximating the most important Atoms inside that community. The combinatorial explosion of direct-Atomspace communication is then halted by an appeal to standardized Psynese Atoms. Pragmatically, a PsyneseVocabulary might be contained in a PsyneseVocabulary server, a special CogPrime instance that exists to mediate communications between other CogPrime s, and provide CogPrime s with information. Psynese makes sense both as a mechanism for peer-to-peer communication between CogPrime s, and as

EFTA00624517

a mechanism allowing standardized communication between a community of CogPrime s using a PsyneseVocabulary server.

43.2 A Simple Example Using a PsyneseVocabulary Server

Suppose CogPrime 1 wanted to tell CogPrime 2 that "Russians are crazy" (with the latter word meaning something inbetween "insane" and "impractical"); and suppose that both CogPrime s are connected to the same Psynese CogPrime with PsyneseVocabulary PV. Then, for instance, it must find the Atom in PV corresponding to its concept "crazy." To do this it must create an AtomStructureTemplate such as

```
Pred1(C1)
equals
ThereExists
   W1, C2, C3, W2, W3
   AND
      ConceptNode: C1
      ReferenceLink C1 W1
      WordNode: W1 #crazy
      ConceptNode: C2
      HebbianLink C1 C2
      ReferenceLink C2 W2
      WordNode: W2 #insane
      ConceptNode: C3
      HebbianLink C1 C3
      ReferenceLink C3 W3
      WordNode: W3 #impractical
```

encapsulating relevant properties of the Atom it wants to grab from PV. In this example the properties specified are:

- ConceptNode, linked via a ReferenceLink to the WordNode for "crazy"
- HebbianLinks with ConceptNodes linked via ReferenceLinks to the WordNodes for "insane" and "impractical"

So, what CogPrime 1 can do is fish in PV for "some concept that is denoted by the word 'crazy' and is associated with 'insane' and 'impractical'." The association with "insane" provides more insurance of getting the correct sense of the word "crazy" as opposed to e.g. the one in the phrase "He was crazy about her" or in "That's crazy, man, crazy" (in the latter slang usage "crazy" basically means "excellent"). The association with "impractical" biases away from the interpretation that all Russians are literally psychiatric patients. *

So, suppose that CogPrime 1 has fished the appropriate Atoms for "crazy" and "Russian" from PV. Then it may represent in its Atomspace something we may denote crudely (a better notation will be introduced later) as

```
InheritanceLink PV:477335:1256953732 PV:744444:1256953735 <.8.,6>
```

* A similar but perhaps more compelling example would be the interpretation of the phrase "the accountant cooked the books." In this case both "cooked" and "books" are used in atypical senses, but specifying a HebbianLink to "accounting" would cause the right Nodes to get retrieved from PV.

EFTA00624518

where e.g. "PV:744444" means "the Atom with Handle 744444 in CogPrime PV at time 1256953735," and may also wish to store additional information such as

```
PsyneseEvaluationLink <.9>
        PV
        Pred1
        PV:744444:1256953735
```

meaning that $Pred1(PV : 744444 : 1256953735)$ holds true with truth value $< .9 >$ *if all the Atoms referred to within Pred1 are interpreted as existing in PV rather than CogPrime 1.*

The InheritanceLink then means: "In the opinion of CogPrime 1, 'Russian' as defined by PV:477335:1256953732 inherits from 'crazy' as defined by PV:744444:1256953735 with truth value <.8,.6>."

Suppose CogPrime 1 then sends the InheritanceLink to CogPrime 2. It is going to be meaningfully interpretable by CogPrime 2 to the extent that CogPrime 2 can interpret the relevant PV Atoms, for instance by finding Atoms of its own that correspond to them. To interpret these Atoms, CogPrime 2 must carry out the reverse process that CogPrime 1 did to find the Atoms in the first place. For instance, to figure out what PV:744444:1256953735 means to it, CogPrime 2 may find some of the important links associated with the Node in PV, and make a predicate accordingly, e.g.:

```
Pred2(C1)
equals
ThereExists
   W1, C2, C3, W2, W3
   AND
      ConceptNode: C1
      ReferenceLink C1 W1
      WordNode: W1 #crazy
      ConceptNode: C2
      HebbianLink C1 C2
      ReferenceLink C2 W2
      WordNode: W2 #lunatic
      ConceptNode: C3
      HebbianLink C1 C3
      ReferenceLink C3 W3
      WordNode: W3 #unrealistic
```

On the other hand, if there is no PsyneseVocabulary involved, then CogPrime 1 can submit the same query directly to CogPrime 2. There is no problem with this, but if there is a reasonably large community of CogPrime s it becomes more efficient for them all to agree on a standard vocabulary of Atoms to be used for communication – just as, at a certain point in human history, it was recognized as more efficient for people to use dictionaries rather than to rely on peer-to-peer methods for resolution of linguistic disagreements.

The above examples involve human natural language terms, but this does not have to be the case. PsyneseVocabularies can contain Atoms representing quantitative or other types of data, and can also contain purely abstract concepts. The basic idea is the same. A CogPrime has some Atoms it wants to convey to another CogPrime, and it looks in a PsyneseVocabulary to see how easily it can approximate these Atoms in terms of "socially understood" Atoms. This is particularly effective if the CogPrime receiving the communication is familiar with the PsyneseVocabulary in question. Then the recipient may already know the PsyneseVocabulary Atoms it is being pointed to; it may have already thought about the difference between these consensus concepts and its own related concepts. Also, if the sender CogPrime is encapsulating

EFTA00624519

maps for easy communication, it may specifically seek approximate encapsulations involving PsyneseVocabulary terms, rather than first encapsulating in its own terms and then translating into PsyneseVocabulary terms.

43.2.1 The Psynese Match Schema

One way to streamline the above operations is to introduce a *Psynese Match Schema*, with the property that

```
ExOut
        PsyneseMatch PV A
```

within CogPrime instance CP_1, denotes the Atom within CogPrime instance PV that most closely matches the Atom A in CP_1. Note that the PsyneseMatch schema implicitly relies on various parameters, because it must encapsulate the kind of process described explicitly in the above example. PsyneseMatch must, internally, decide how many and which Atoms related to A should be used to formulate a query to PV, and also how to rank the responses to the query (e.g. by *strength* $\times$ *confidence*).

Using PsyneseMatch, the example written above as

```
Inheritance PV:477335:1256953732 PV:744444:1256953735 <.8.,6>
```

could be rewritten as

```
Inheritance <.8.,6>
        ExOut
                PsyneseMatch PV C1
        ExOut
                PsyneseMatch PV C2
```

where $C1$ and $C2$ are the ConceptNodes in CP_1 corresponding to the intended senses of "crazy" and "Russian."

43.3 Psynese as a Language

The general definition of a psynese expression for CogPrime is a Set of Atoms that contains only:

- Nodes from PsyneseVocabularies
- Perceptual nodes (numbers, words, etc.)
- Relationships relating no nodes other than the ones in the above two categories, and relating no relationships except ones in this category
- Predicates or Schemata involving no relationships or nodes other than the ones in the above three categories, or in this category

The PsyneseEvaluationLink type indicated earlier forces interpretation of a predicate as a Psynese expression.

In what sense is the use of Psynese expressions to communicate a language? Clearly it is a formal language in the mathematical sense. It is not quite a "human language" as we normally

EFTA00624520

conceive it, but it is ideally suited to serve the same functions for CogPrime s as human language serves for humans. The biggest differences from human language are:

- Psynese uses weighted, typed hypergraphs (i.e. Atomspaces) instead of linear strings of symbols. This eliminates the "parsing" aspect of language (syntax being mainly a way of projecting graph structures into linear expressions).
- Psynese lacks subtle and ambiguous referential constructions like "this", "it" and so forth. These are tools allowing complex thoughts to be compactly expressed in a linear way, but CogPrime s don't need them. Atoms can be named and pointed to directly without complex, poorly-specified mechanisms mediating the process.
- Psynese has far less ambiguity. There may be Atoms with more than one aspect to their meanings, but the cost of clarifying such ambiguities is much lower for CogPrime s than for humans using language, and so habitually there will not be the rampant ambiguity that we see in human expressions.

On the other hand, mapping Psynese into *Lojban* – a syntactically formal, semantically highly precise language created for communication between humans – rather than a true natural language would be much more straightforward. Indeed, one could create a PsyneseVocabulary based on Lojban, which might be ideally suited to serve as an intermediary between different CogPrime s. And Lojban may be used to create a linearized version of Psynese that looks more like a natural language. We return to this point in Appendix **??**.

43.4 Psynese Mindplexes

We now recall from Chapter 12 of Part 1 the notion of a *mindplex*: that is, an intelligent system that:

1. Is composed of a collection of intelligent systems, each of which has its own "theater of consciousness" and autonomous control system, but which interact tightly, exchanging large quantities of information frequently
2. Has a powerful control system on the collective level, and an active "theater of consciousness" on the collective level as well

In informal discussions, we have found that some people, on being introduced to the mindplex concept, react by contending that either human minds or human social groups are mindplexes. However, I believe that, while there are significant similarities between mindplexes and minds, and between mindplexes and social groups, there are also major qualitative differences. It's true that an individual human mind may be viewed as a collective, both from a theory-of-cognition perspective (e.g. Minsky's "society of mind" theory [Min88]) and from a personality-psychology perspective (e.g. the theory of subpersonalities [Row90]). And it's true that social groups display some autonomous control and some emergent-level awareness. However, in a healthy human mind, the collective level rather than the cognitive-agent or subpersonality level is dominant, the latter existing in service of the former; and in a human social group, the individual-human level is dominant, the group-mind clearly "cognizing" much more crudely than its individual-human components, and exerting most of its intelligence via its impact on individual human minds. A mindplex is a hypothetical intelligent system in which neither level is dominant, and both levels are extremely powerful. A mindplex is like a human mind in

EFTA00624521

which the subpersonalities are fully-developed human personalities, with full independence of thought, and yet the combination of subpersonalities is also an effective personality. Or, from the other direction, a mindplex is like a human society that has become so integrated and so cohesive that it displays the kind of consciousness and self-control that we normally associate with individuals.

There are two mechanisms via which mindplexes may possibly arise in the medium-term future:

1. Humans becoming more tightly coupled via the advance of communication technologies, and a communication-centric AI system coming to embody the "emergent conscious theater" of a human-incorporating mindplex
2. A society of AI systems communicating amongst each other with a richness not possible for human beings, and coming to form a mindplex rather than merely a society of distinct AI's

The former sort of mindplex relates to the concept of a "global brain" discussed in Chapter 12 of Part 1. Of course, these two sorts of mindplexes are not mutually contradictory, and may coexist or fuse. The possibility also exists for higher-order mindplexes, meaning mindplexes whose component minds are themselves mindplexes. This would occur, for example, if one had a mindplex composed of a family of closely-interacting AI systems, which acted within a mindplex associated with the global communication network.

Psynese, however, is more directly relevant to the latter form of mindplex. It gives a concrete mechanism via which such a mindplex might be sculpted.

43.4.1 AGI Mindplexes

How does one get from CogPrime s communicating via Psynese to CogPrime mindplexes?

Clearly, with the Psynese mode of communication, the potential is there for much richer communication than exists between humans. There are limitations, posed by the private nature of many concepts – but these limitations are much less onerous than for human language, and can be overcome to some extent by the learning of complex cognitive schemata for translation between the "private languages" of individual Atomspaces and the "public languages" of Psynese servers.

But rich communication does not in itself imply the evolution of mindplexes. It is possible that a community of Psynese-communicating CogPrime s might spontaneously evolve a mindplex structure - at this point, we don't know enough about CogPrime individual or collective dynamics to say. But it is not necessary to rely on spontaneous evolution. In fact it is possible, and even architecturally simple, to design a community of CogPrime s in such a way as to encourage and almost force the emergence of a mindplex structure.

The solution is simple: simply beef up PsyneseVocabulary servers. Rather than relatively passive receptacles of knowledge from the CogPrime s they serve, allow them to be active, creative entities, with their own feelings, goals and motivations.

The PsyneseVocabulary servers serving a community of CogPrime s are absolutely critical to these CogPrime s. Without them, high-level inter-CogPrime communication is effectively impossible. And without the concepts the PsyneseVocabularies supply, high-level individual CogPrime thought will be difficult, because CogPrime s will come to think in Psynese to at least the same extent to which humans think in language.

EFTA00624522

Suppose each PsyneseVocabulary server has its own full CogPrime mind, its own "conscious theater". These minds are in a sense "emergent minds" of the CogPrime community they serve - because their contents are a kind of "nonlinear weighted average" of the mind-contents of the community. Furthermore, the actions these minds take will feed back and affect the community in direct and indirect ways - by affecting the language by which the minds communicate. Clearly, the definition of a mindplex is fulfilled.

But what will the dynamics of such a CogPrime mindplex be like? What will be the properties of its cognitive and personality psychology? We could speculate on this here, but would have very little faith in the possible accuracy of our speculations. The psychology of mindplexes will reveal itself to us experimentally as our work on AGI engineering, education and socialization proceeds.

One major issue that arises, however, is that of *personality filtering*. Put simply: each intelligent agent in a mindplex must somehow decide for itself which knowledge to grab from available PsyneseVocabulary servers and other minds, and which knowledge to *avoid* grabbing from others in the name of individuality. Different minds may make different choices in this regard. For instance, one choice could be to, as a matter of routine, take only extremely confident knowledge from the PsyneseVocabulary server. This corresponds roughly to ingesting "facts" from the collective knowledge pool, but not opinions or speculations. Less confident knowledge would then be ingested from the collective knowledge pool on a carefully calculated and as-needed basis. Another choice could be to accept only small networks of Atoms from the collective knowledge pool, on the principle that these can be reflectively understood as they are ingested, whereas large networks of Atoms are difficult to deliberate and reflect about. But any policies like this are merely heuristic ones.

43.5 Psynese and Natural Language Processing

Next we review a more near-term, practical application of the Psynese mechanism: the fusion of two different approaches to natural language processing in CogPrime, the experiential learning approach and the "engineered NLP subsystem" approach.

In the former approach, language is not given any extremely special role, and CogPrime is expected to learn language much as it would learn any other complex sort of knowledge. There may of course be *learning biases* programmed into the system, to enable it to learn language based on its experience more rapidly. But there is no *concrete linguistic knowledge* programmed in.

In the latter approach, one may use knowledge from statistical corpus analysis, one may use electronic resources like WordNet and FrameNet, and one may use sophisticated, specialized tools like natural language parsers with hand-coded grammars. Rather than trying to emulate the way a human child learns language, one is trying to emulate the way a human adult comprehends and generates language.

Of course, there is not really a rigid dichotomy between these two approaches. Many linguistic theorists who focus on experiential learning also believe in some form of universal grammar, and would advocate for an approach where learning is foundational but is biased by in-built abstract structures representing universal grammar. There is of course very little knowledge (and few detailed hypotheses) about how universal grammar might be encoded in the human brain, though there is reason to think it may be at a very abstract level, due to the significant

EFTA00624523

overlaps between grammatical structure, social role structure [CB00], and physical reasoning [Cas04].

The engineered approach to NLP provides better functionality right "out of the box," and enables the exploitation of the vast knowledge accumulated by computational linguists in the past decades. However, we suspect that computational linguistics may have hit a ceiling in some regards, in terms of the quality of the language comprehension and generation that it can deliver. It runs up against problems related to the disambiguation of complex syntactic constructs, which don't seem to be resolvable using either a tractable number of hand-coded rules, or supervised or unsupervised learning based on a tractably large set of examples. This conclusion may be disputed, and some researchers believe that statistical computational linguistics can eventually provide human-level functionality, once the Web becomes a bit larger and the computers used to analyze it become a bit more powerful. But in our view it is interesting to explore hybridization between the engineered and experiential approaches, with the motivation that the experiential approach may provide a level of flexibility and insight at dealing with ambiguity that the engineered approach apparently lacks.

After all, the way a human child deals with the tricky disambiguation problems that stump current computational linguistics systems is not via analysis of trillion-word corpuses, bur rather via correlating language with non-linguistic experience. One may argue that the genome implicitly contains a massive corpus of speech, but there it's also to be noted that this is *experientially contextualized speech*. And it seems clear from the psycholinguistic evidence [Tom03] that for young human children, language is part and parcel of social and physical experience, learned in a manner that's intricately tied up with the learning of many other sorts of skills.

One interesting approach to this sort of hybridization, using Psynese, is to create multiple CogPrime instances taking different approaches to language learning, and let them communicate. Most simply one may create

- A CogPrime instance that learns language mainly based on experience, with perhaps some basic in-built structure and some judicious biasing to its learning (let's call this CP_{exp})
- A CogPrime instance using an engineered NLP system (let's call this CP_{eng})

In this case, CP_{exp} can use CP_{eng} as a cheap way to test its ideas. For instance suppose, CP_{exp} thinks it has correctly interpreted a certain sentence S into Atom-set A. Then it can send its interpretation A to CP_{eng} and see whether CP_{eng} thinks A is a good interpretation of S, by consulting $CP_{eng}$the truth value of

```
ReferenceLink
        ExOut
                PsyneseMatch CP_eng S
        ExOut
                PsyneseMatch CP_eng A
```

Similarly, if CP_{exp} believes it has found a good way (S) to linguistically express a collection S of Atoms A, it can check whether these two match reasonably well in CP_{eng}.

Of course, this approach could be abused in an inefficient and foolish way, for instance if CP_{exp} did nothing but randomly generate sentences and then test them against CP_{eng}. In this case we would have a much less efficient approach than simply using CP_{eng} directly. However, effectively making use of CP_{eng} as a resource requires a different strategy: throwing CP_{eng} only a relatively small selection of things that seem to make sense, and using CP_{eng} as a filter to avoid trying out rough-draft guesses in actual human conversation.

EFTA00624524

This hybrid approach, we suggest, may provide a way of getting the best of both worlds: the flexibility of a experiential-learning-based language approach, together with the exploitation of existing linguistic tools and resources. With this in mind, in the following chapters we will describe both engineering and experiential-learning based approaches to NLP.

43.5.1 Collective Language Learning

Finally we bring the language-learning and mindplex themes together, in the notion of *collective language learning*. One of the most interesting uses for a mindplex architecture is to allow multiple CogPrime agents to share the linguistic knowledge they gain. One may envision a PsyneseVocabulary server into which a population of CogPrime agents input their *linguistic* knowledge specifically, and which these agents then consult when they wish to comprehend or express something in language, and their individual NLP systems are not up to the task.

This could be a very powerful approach to language learning, because it would allow a potentially very large number of AI systems to effectively act as a *single* language learning system. It is an especially appealing approach in the context of CogPrime systems used to control animated agents in online virtual worlds or multiplayer games. The amount of linguistic experience undergone by, say, 100,000 virtually embodied CogPrime agents communicating with human virtual world avatars and game players, would be far more than any single human child or any single agent could undergo. Thus, to the extent that language learning can be accelerated by additional experience, this approach could enable language to be learned quite rapidly.

EFTA00624525

Chapter 44
Natural Language Comprehension

Co-authored with Michael Ross and Linas Vepstas and Ruiting Lian

44.1 Introduction

Two key approaches to endowing AGI systems with linguistic facility exist, as noted above:

- "Experiential" – shorthand here for "gaining most of its linguistic knowledge from interactive experience, in such a way that language learning is not easily separable from generic learning how to survive and flourish"
- "Engineered" – shorthand here for "gaining most of its linguistic knowledge from sources other than the system's own experience in the world" (including learning language from resources like corpora)

This dichotomy is somewhat fuzzy, since getting experiential language learning to work well may involve some specialized engineering, and engineered NLP systems may also involve some learning from experience. However, in spite of the fuzziness, the dichotomy is still real and important; there are concrete choices to be made in designing an NLP system and this dichotomy compactly symbolizes some of them. Much of this chapter and the next few will be focused on the engineering approach, but we will also devote some space to discussing the experience-based approach. Our overall perspective on the dichotomy is that

- the engineering-based approach, on its own, is unlikely to take us to human-level NLP ... but this isn't wholly impossible, if the engineering is done in a manner that integrates linguistic functionality richly with other kinds of experiential learning
- using a combination of experience-based and engineering-based approaches may be the most practical option
- the engineering approach is useful for guiding the experiential approach, because it tells us a lot about what kinds of general structures and dynamics may be adequate for intelligent language processing. To simplify a bit, one can prepare an AGI system for experiential learning by supplying it with structures and dynamics capable of supporting the key components of an engineered NLP system – and biased toward learning things similar to known engineered NLP systems – but requiring all, or the bulk of, the actual linguistic content to be learned via experience. This approach may be preferable to requiring a system to learn language based on more abstract structures and dynamics, and may indeed be more comparable to what human brains do, given the large amount of linguistic biasing that is probably built into the human genome.

EFTA00624526

Further distinctions, overlapping with this one, may also be useful. One may distinguish (at least) five modes of instructing NLP systems, the first three of which are valid only for engineered NLP systems, but the latter two of which are valid both for engineered and experiential NLP systems:

- hand-coded rules
- supervised learning on hand-tagged corpuses, or via other mechanisms of explicit human training
- unsupervised learning from static bodies of data
- unsupervised learning via interactive experience
- supervised learning via interactive experience

Note that, in principle, any of these modes may be used in a pure-language or a socially/physically embodied language context. Of course, there is also semi-supervised learning which may be used in place of supervised learning in the above list [CSZ06].

Another key dichotomy related to linguistic facility is language comprehension versus language generation (each of which is typically divided into a number of different subprocesses). In language comprehension, we have processes like stemming, part-of-speech tagging, grammar-based parsing, semantic analysis, reference resolution and discourse analysis. In language generation, we have semantic analysis, syntactic sentence generation, pragmatic discourse generation, reference-insertion, and so forth. In this chapter and the next two we will briefly review all these different topics and explain how they may be embodied in CogPrime. Then, in Chapter **??** we present a complementary approach to linguistic interaction with AGI systems based on the invented language Lojban; and in Chapter 48 we discuss the use of CogPrime cognition to regulate the dialogue process.

A typical, engineered computational NLP system involves hand-coded algorithms carrying out each of the specific tasks mentioned in the previous paragraph, sometimes with parameters, rules or number tables that are tuned or learned statistically based on corpuses of data. In fact, most NLP systems handle only understanding or only generation; systems that cover both aspects in a unified way are quite rare. The human mind, on the other hand, carries out these tasks in a much more interconnected way – using separate procedures for the separate tasks, to some extent, but allowing each of these procedures to be deeply informed by the information generated by the other procedures. This interconnectedness is what allows the human mind to really understand language - specifically because human language syntax is complex and ambiguous enough that the only way to master it is to infuse one's syntactic analyses with semantic (and to a lesser extent pragmatic) knowledge. In our treatment of NLP we will pay attention to connections between linguistic functionalities, as well as to linguistic functionalities in isolation.

It's worth emphasizing that what we mean by a "experience based" language system is quite different from corpus-based language systems as are commonplace in computational linguistics today [MS99] (and from the corpus-based learning algorithm to be discussed in Chapter **??** below). In fact, we feel the distinction between corpus-based and rule-based language processing systems is often overblown. Whether one hand-codes a set of rules, or carefully marks up a corpus so that rules can be induced from it, doesn't ultimately make that much difference. For instance, OpenCogPrime's RelEx system (to be described below) uses hand-coded rules to do much the same thing that the Stanford parser does using rules induced from a tagged corpus. But both systems do roughly the same thing. RelEx is currently faster due to using fewer rules, and it handles some complex cases like comparatives better (presumably because they were not

EFTA00624527

well covered in the Stanford parser's training corpus); but the Stanford parser may be preferable in other respects, for instance it's more easily generalizable to languages beyond English (for a language with structure fairly similar to English, one just has to supply a new marked-up training corpus; whereas porting RelEx rules to other languages requires more effort).

An unsupervised corpus-based learning system like the one to be described in Chapter?? is a little more distinct from rule-based systems, in that it is based on inducing patterns from natural rather than specially prepared data. But still, it is learning language as a phenomenon unto itself, rather than learning language as part and parcel of a system's overall experience in the world.

The key distinction to be made, in our view, is between language systems that learn language in a social and physical context, versus those that deal with language in isolation. Dealing with language in context immediately changes the way the linguistics problem appears (to the AI system, and also to the researcher), and makes hand-coded rules and hand-tagged corpuses less viable, shifting attention toward experiential learning based approaches.

Ultimately we believe that the "right" way to teach an AGI system language is via semi-supervised learning in a socially and physically embodied context. That is: talk to the system, and have it learn both from your reinforcement signals and from unsupervised analysis of the dialogue. However, we believe that other modes of teaching NLP systems can also contribute, especially if used in support of a system that also does semi-supervised learning based on embodied interactive dialogue.

Finally, a note on one aspect of language comprehension that we don't deal with here. We deal only with text processing, not speech understanding or generation. A CogPrime approach to speech would be quite feasible to develop, for instance using neural-symbolic hybridization with DeSTIN or a similar perceptual-motor hierarchy. However, this potential aspect of CogPrime has not been pursued in detail yet, and we won't devote space to it here.

44.2 Linguistic Atom Types

Explicit representation of linguistic knowledge in terms of Atoms is not a deep issue, more of a "plumbing" type of issue, but it must be dealt with before moving on to subtler aspects.

In principle, for dealing with linguistic information coming in through ASCII, all we need besides the generic CogPrime structures and dynamics are two node types and one relationship type:

- CharacterNode
- CharacterInstanceNode
- a unary relationship concat denoting an externally-observed list of items

Sequences of characters may then be represented in terms of lists and the concat schema. For instance the word "pig" is represented by the list $concat(\#p, \#i, \#g)$

The concat operator can be used to help define special NL atom types, such as:

- MorphemeNode/ MorphemeInstanceNode
- WordNode/WordInstanceNode
- PhraseNode/PhraseInstanceNode
- SentenceNode/ SentenceInstanceNode
- UtteranceNode/ UtteranceInstanceNode

EFTA00624528

44.3 The Comprehension and Generation Pipelines

Exactly how the "comprehension pipeline" is broken down into component transformations, depends on one's linguistic theory of choice. The approach taken in OpenCogPrimes engineered NLP framework, in use from 2008-2012, looked like:

```
Text --> Tokenizer --> Link Parser -->
Syntactico-Semantic Relationship Extractor (RelEx) -->
Semantic RelationshipExtractor  (RelEx2Frame) -->
SemanticNodes  & Links
```

In 2012-13, a new approach has been undertaken, which simplifies things a little and looks like

```
Text -->  Tokenizer -->   Link Parser -->
Syntactico-Semantic Relationship Extractor  (Syn2Sem) -->
Semantic Nodes & Links
```

Note that many other variants of the NL pipeline include a "tagging" stage, which assigns part of speech tags to words based on the words occurring around them. In our current approach, tagging is essentially subsumed within parsing; the choice of a POS (part-of-speech) tag for a word instance is carried out within the link parser. However, it may still be valuable to derive information about likely POS tags for word instances from other techniques, and use this information within a link parsing framework by allowing it to bias the probabilities used in the parsing process.

None of the processes in this pipeline are terribly difficult to carry out, if one is willing to use hand-coded rules within each step, or derive rules via supervised learning, to govern their operation. The truly tricky aspects of NL comprehension are:

- arriving at the rules used by the various subprocesses, in a way that naturally supports generalization and modification of the rules based on ongoing experience
- allowing semantic understanding to bias the choice of rules in particular contexts
- knowing when to break the rules and be guided by semantic intuition instead

Importing rules straight from linguistic databases results in a system that (like the current RelEx system) is reasonably linguistically savvy on the surface, but lacks the ability to adapt its knowledge effectively based on experience, and has trouble comprehending complex language. Supervised learning based on hand-created corpuses tends to result in rule-bases with similar problems. This doesn't necessarily mean that hand-coding or supervised learning of linguistic rules has no place in an AGI system, but it means that if one uses these methods, one must take extra care to make one's rules modifiable and generalizable based on ongoing experience, because the initial version of one's rules is not going to be good enough.

Generation is the subject of the following chapter, but for comparison we give here a high-level overview of the generation pipeline, which may be conceived as:

1. *Content determination*: figuring out what needs to be said in a given context
2. *Discourse planning*: overall organization of the information to be communicated
3. *Lexicalization*: assigning words to concepts
4. *Reference generation*: linking words in the generated sentences using pronouns and other kinds of reference

EFTA00624529

5. *Syntactic and morphological realization*: the generation of sentences via a process inverse to parsing, representing the information gathered in the above phases
6. *Phonological or orthographic realization*: turning the above into spoken or written words, complete with timing (in the spoken case), punctuation (in the written case), etc.

In Chapter 46 we explain how this pipeline is realized in OpenCogPrimes current engineered NL generation system.

44.4 Parsing with Link Grammar

Now we proceed to explain some of the details of OpenCogPrime's engineered NL comprehension system. This section gives an overview of link grammar, a key part of the current OpenCog NLP framework, and explains what makes it different from other linguistic formalisms.

We emphasize that this particular grammatical formalism is not, in itself, a critical part of the CogPrime design. In fact, it should be quite possible to create and teach a CogPrime AGI system without using any particular grammatical formalism – having it acquire linguistic knowledge in a purely experiential way. However, a great deal of insight into CogPrime -based language processing may be obtained by considering the relevant issues in the concrete detail that the assumption of a specific grammatical formalism provides. This insight is of course useful if one is building a CogPrime that makes use of that particular grammatical formalism, but it's also useful to some degree even if one is building a CogPrime that deals with human language entirely experientially.

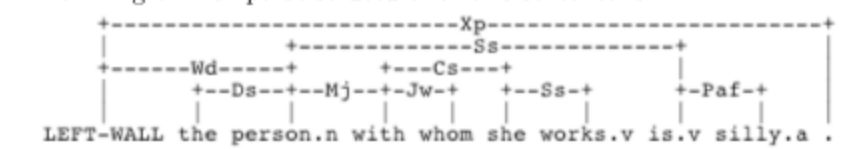
This material will be more comprehensible to the reader who has some familiarity with computational linguistics, e.g. with notions such as parts of speech, feature structures, lexicons, dependency grammars, and so forth. Excellent references are [MS99, Jac03]. We will try to keep the discussion relatively elementary, but have opted not to insert a computational linguistics tutorial.

The essential idea of link grammar is that each word comes with a feature structure consisting of a set of typed connectors. Parsing consists of matching up connectors from one word with connectors from another

To understand this in detail, the best course is to consider an example sentence. We will use the following example, drawn from the classic paper "Parsing with a Link Grammar" by Sleator and Temperley [ST93]:

The cat chased a snake

The link grammar parse structure for this sentence is:



In phrase structure grammar terms, this corresponds loosely to

```
(S (NP The cat)
        (VP chased
                (NP a snake))
        .)
```

EFTA00624530

but the OpenCog linguistic pipeline makes scant use of this kind of phrase structure rendition (which is fine in this simple example; but in the case of complex sentences, construction of analogous mappings from link parse structures to phrase structure grammar parse trees can be complex and problematic). Currently the hierarchical view is used in OpenCog only within some reference resolution heuristics.

There is a database called the "link grammar dictionary" which contains connectors associated with all common English words. The notation used to describe feature structures in this dictionary is quite simple. Different kinds of connectors are denoted by letters or pairs of letters like S or SX. Then if a word W1 has the connector S+, this means that the word can have an S link coming out to the right side. If a word W2 has the connector S-, this means that the word can have an S link coming out to the left side. In this case, if W1 occurs to the left of W2 in a sentence, then the two words can be joined together with an S link.

The features of the words in our example sentence, as given in the S&T paper, are:

Words	**Formula**
a, the	D+
snake, cat	D- & (O- or S+)
Chased	S- & O+

To illustrate the role of syntactic sense disambiguation, we will introduce alternate formulas for one of the words in the example: the verb sense of "snake." We then have

Words	**Formula**
A, the	D+
snake_N, cat, ran_N	D- & (O- or S+)
Chased	S- & O+
snake_V	S-

The variables to be used in parsing this sentence are, for each word:

1. the features in the Agreement structure of the word (for any of its senses)
2. the words matching each of the connectors of the word

For example,

1. For "snake," there are features for "word that links to D-", "word that links to O-" and "word that links to S+". There are also features for "tense" and "person".
2. For "the", the only feature is "word that links to D+". No features for Agreement are needed.

The nature of linkage imposes constraints on the variable assignments; for instance, if "the" is assigned as the value of the "word that links to D-" feature of "snake", then "snake" must be assigned as the value of the "word that links to D+" feature of "the."

The rules of link grammar impose additional constraints – i.e. the planarity, connectivity, ordering and exclusion metarules described in Sleator and Temperley's papers. Planarity means that links don't cross – a rule that S&T's parser enforces with absoluteness, whereas we have found it is probably better to impose it as a probabilistic constraint, since sometimes it's really nice to let links cross (the representation of conjunctions is one example). Connectivity means that the links and words of a sentence must form a connected graph – all the words must be linked into the other words in the sentence via some path. Again connectivity is a valuable constraint but in some cases one wants to relax it – if one just can't understand the whole sentence, one may wish to understand at least some parts of it, meaning that one has a disconnected graph whose components are the phrases of the sentence that have been

EFTA00624531

successfully comprehended. Finally, linguistic transformations may potentially be applied while checking if these constraints are fulfilled (that is, instead of just checking if the constraints are fulfilled, one may check if the constraints are fulfilled after one or more transformations are performed.)

We will use the term "Agreement" to refer to "person" values or ordered pairs (tense, person), and NAGR to refer to the number of agreement values (12-40, perhaps, in most realistic linguistic theories). Agreement may be dealt with alongside the connector constraints. For instance, "chased" has the Agreement values (past, third person), and it has the constraint that its S- argument must match the person component of its Agreement structure.

Semantic restrictions may be imposed in the same framework. For instance, it may be known that the subject of "chased" is generally animate. In that case, we'd say

Words	**Formula**
A, the	D+
snake_N, cat	D- & (O- or S+)
Chased	(S-, C Inheritance animate <.8>) & O+
Snake_V	S-

where we've added the modifier (C Inheritance animate) to the S- connector of the verb "chased," to indicate that with strength .8, the word connecting to this S- connector should denote something inheriting from "animate." In this example, "snake" and "cat" inherit from "animate", so the probabilistic restriction doesn't help the parser any. If the sentence were instead

The snake in the hat chased the car

then the "animate" constraint would tell the parsing process not to start out by trying to connect "hat" to "chased", because the connection is semantically unlikely.

44.4.1 Link Grammar vs. Phrase Structure Grammar

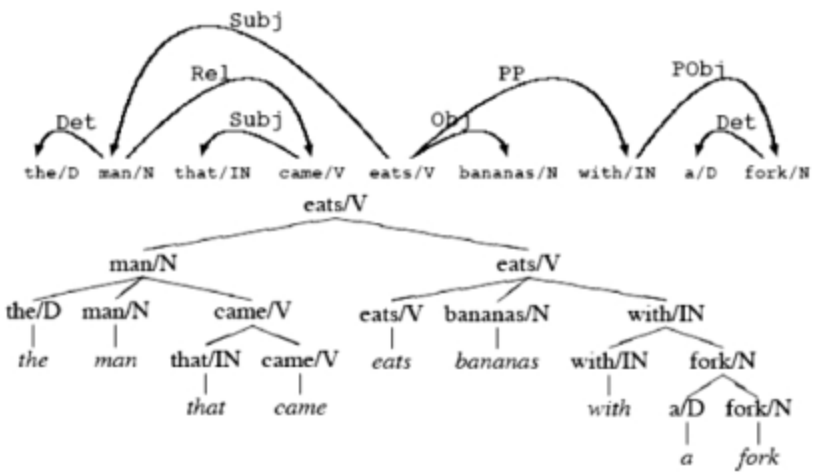
Before proceeding further, it's worth making a couple observations about the relationship between link grammars and typical phrase structure grammars. These could also be formulated as observations about the relationship between dependency grammars and phrase structure grammars, but that gets a little more complicated as there are many kinds of dependency grammars with different properties; for simplicity we will restrict our discussion here to the link grammar that we actually use in OpenCog. Two useful observations may be:

1. Link grammar formulas correspond to grammatical categories. For example, the link structure for "chased" is "S- & O+." In categorical grammar, this would seem to mean that " 'chased' belongs to the category of words with link structure 'S- & O+'." In other words, each "formula" in link grammar corresponds to a category of words attached to that formula.
2. Links to words might as well be interpreted as links to phrases headed by those words. For example, in the sentence "the cat chased a snake", there's an O-link from "chased" to "snake." This might as well be interpreted as "there's an O-link from the phrase headed by 'chased' to the phrase headed by 'snake'." Link grammar simplifies things by implicitly identifying each phrase by its head.

EFTA00624532

Based on these observations, one could look at phrase structure as implicit in a link parse; and this does make sense, but also leads to some linguistic complexities that we won't enter into here.

Fig. 44.1: Dependency and Phrase-Structure Parses



A comparison of dependency (above) and phrase-structure (below) parses. In general, one can be converted to the other (algorithmically); dependency grammars tend to be easier understand.
(Image taken from G. Schneider, "Learning to Disambiguate Syntactic Relations" Linguistik online 17, 5/03)

44.5 The RelEx Framework for Natural Language Comprehension

Now we move forward in the pipeline from syntax toward semantics. The NL comprehension framework provided with OpenCog at its inception in 2008 is RelEx, an English-language semantic relationship extractor, which consists of two main components: the dependency extractor and the relationship extractor. It can identify subject, object, indirect object and many other dependency relationships between words in a sentence; it generates dependency trees, resembling those of dependency grammars. In 2012 we are in the process of replacing RelEx with a different approach that we believe will be more amenable to generalization based on experience. Here we will describe both approaches.

The overall processing scheme of RelEx is shown in Figure 44.2.

The dependency extractor component carries out dependency grammar parsing via a customized version of the open-source Sleator and Temperley's link parser, as reviewed above. The link parser outputs several parses, and the dependencies of the best one are taken. The relationship extractor component is composed of a number of template matching algorithms that act upon the link parser's output to produce a semantic interpretation of the parse. It contains three steps:

EFTA00624533

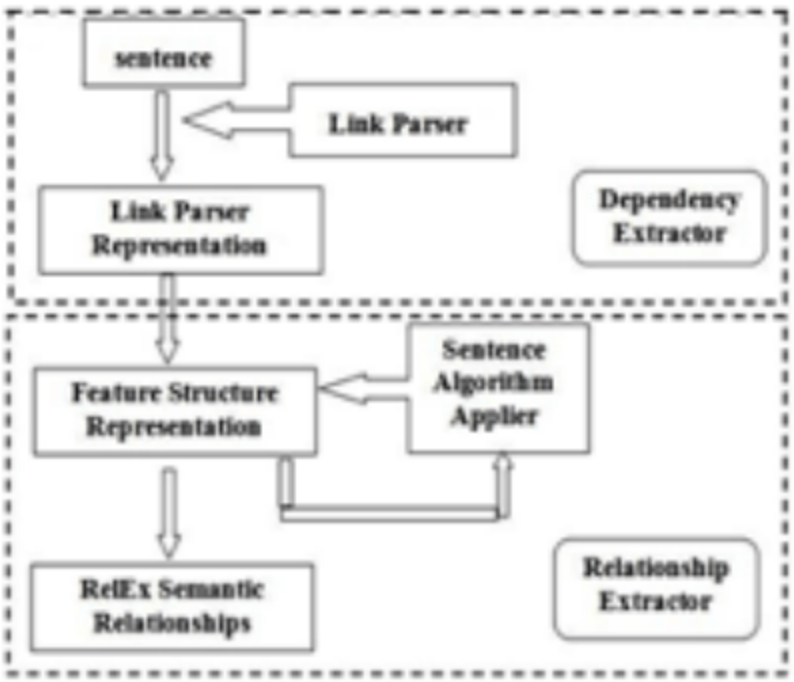


Fig. 44.2: A Overview of the RelEx Architecture for Language Comprehension

1. Convert the Link Parser output to a feature structure representation
2. Execute the Sentence Algorithm Applier, which contains a series of *Sentence Algorithms*, to modify the feature structure.
3. Extract the final output representation by traversing the feature structure.

A feature structure, in the RelEx context, is a directed graph in which each node contains either a value, or an unordered list of features. A feature is just a labeled link to another node. Sentence Algorithm Applier loads a list of SentenceAlgorithms from the algorithm definition file, and the SentenceAlgorithms are executed in the order they are listed in the file. RelEx iterates through every single feature node in the feature structure, and attempts to apply the algorithm to each node. Then the modified feature structures are used to generate the final RelEx semantic relationships.

44.5.1 RelEx2Frame: Mapping Syntactico-Semantic Relationships into FrameNet Based Logical Relationships

Next in the current OpenCog NL comprehension pipeline, the RelEx2Frame component uses hand-coded rules to map RelEx output into sets of relationships utilizing FrameNet and other similar semantic resources. This is definitively viewed as a "stopgap" without a role in a human-level AGI system, but it's described here because it's part of the current OpenCog system and

EFTA00624534

is now being used together with other OpenCog components in practical projects, including those with proto-AGI intentions.

The syntax currently used for describing semantic relationships drawn from FrameNet and other sources is exemplified by the example

```
^1_Benefit:Benefitor(give,$var1)
```

The [1] indicates the data source, where 1 is a number indicating that the resource is FrameNet. The "give" indicates the word in the original sentence from which the relationship is drawn, that embodies the given semantic relationship. So far the resources we've utilized are:

1. FrameNet
2. Custom relationship names

but using other resources in future is quite possible.

An example using a custom relationship would be:

```
^2_inheritance($var1,$var2)
```

which defines an inheritance relationship: something that is part of CogPrime's ontology but not part of FrameNet.

The "Benefit" part of the first example indicates the frame indicated, and the "Benefitor" indicates the frame element indicated. This distinction (frame vs. frame element) is particular to FrameNet; other knowledge resources might use a different sort of identifier. In general, whatever lies between the underscore and the initial parenthese should be considered as particular to the knowledge-resource in question, and may have different format and semantics depending on the knowledge resource (but shouldn't contain parentheses or underscores unless those are preceded by an escape character).

As an example, consider:

```
Put the ball on the table
```

Here the RelEx output is:

```
imperative(Put) [1]
_obj(Put, ball) [1]
on(Put, table) [1]
singular(ball) [1]
singular(table) [1]
```

, The relevant FrameNet Mapping Rules are:

```
$var0 = ball
$var1 = table
# IF imperative(put) THEN ^1_Placing:Agent(put,you)
# IF _obj(put,$var0) THEN ^1_Placing:Theme(put,$var0)
# IF on(put,$var1) & _obj(put,$var0) THEN ^1_Placing:Goal(put,$var1) \
 ^1_Locative_relation:Figure($var0) ^1_Locative_relation:Ground($var1)
```

Finally, the output FrameNet Mapping is:

EFTA00624535

```
^1_Placing:Agent(put,you)
^1_Placing:Theme(put,ball)
^1_Placing:Goal(put,table)
^1_Locative_relation:Figure(put,ball)
^1_Locative_relation:Ground(put,table)
```

The textual syntax used for the hand-coded rules mapping RelEx to FrameNet, at the moment, looks like:

```
# IF imperative(put) THEN ^1_Placing:Agent(put,you)
# IF _obj(put,$var0) THEN ^1_Placing:Theme(put,$var0)
# IF on(put,$var1) & _obj(put,$var0) THEN ^1_Placing:Goal(put,$var1) \
 ^1_Locative_relation:Figure($var0) ^1_Locative_relation:Ground($var1)
```

Basically, this means each rule looks like

```
# IF condition THEN action
```

where the condition is a series of RelEx relationships, and the action is a series of FrameNet relationships. The arguments of the relationships may be words or may be variables in which case their names must start with $ *. The only variables appearing in the action should be ones that appeared in the condition.

44.5.2 A Priori Probabilities For Rules

It can be useful to attach a priori, heuristic probabilities to RelEx2Frame rules, say

```
# IF _obj(put,$var0) THEN ^1_Placing:Theme(put,$var0) <.5>
```

to denote that the a priori probability for the rule is 0.5

This is a crude mechanism because the probability of a rule being useful, in reality, depends so much on context; but it still has some nonzero value.

44.5.3 Exclusions Between Rules

It may be also useful to specify that two rules can't semantically-consistently be applied to the same RelEx relationship. To do this, we need to associate rules with labels, and then specify exclusion relationships such as

```
# IF on(put,$var1) & _obj(put,$var0) THEN ^1_Placing:Goal(put,$var1) \
  ^1_Locative_relation:Figure($var0) ^1_Locative_relation:Ground($var1) [1]
# IF on(put,$var1) & _subj(put,$var0) THEN \
  ^1_Performing_arts:Performance(put,$var1) \
  ^1_Performing_arts:Performer(put,$var0) [2]
# EXCLUSION 1 2
```

* An escape character "\" must be used to handle cases where the character "$" starts a word

EFTA00624536

In this example, Rule 1 would apply to "He put the ball on the table", whereas Rule 2 would apply to "He put on a show". The exclusion says that generally these two rules shouldn't be applied to the same situation. Of course some jokes, poetic expressions, etc., may involve applying excluded rules in parallel.

44.5.4 Handling Multiple Prepositional Relationships

Finally, one complexity arising in such rules is exemplified by the sentence:

"Bob says killing for the Mafia beats killing for the government"

whose RelEx mapping looks like

```
uncountable(Bob) [6]
present(says) [6]
_subj(says, Bob) [6]
_that(says, beats) [3]
uncountable(killing) [6]
for(killing, Mafia) [3]
singular(Mafia) [6]
definite(Mafia) [6]
hyp(beats) [3]
present(beats) [5]
_subj(beats, killing) [3]
_obj(beats, killing_1) [5]
uncountable(killing_1) [5]
for(killing_1, government) [2]
definite(government) [6]
```

In this case there are two instances of "for". The output of RelEx2Frame must thus take care to distinguish the two different for's (or we might want to modify RelEx to make this distinction). The mechanism currently used for this is to subscript the for's, as in

```
uncountable(Bob) [6]
present(says) [6]
_subj(says, Bob) [6]
_that(says, beats) [3]
uncountable(killing) [6]
for(killing, Mafia) [3]
singular(Mafia) [6]
definite(Mafia) [6]
hyp(beats) [3]
present(beats) [5]
_subj(beats, killing) [3]
_obj(beats, killing_1) [5]
uncountable(killing_1) [5]
for_1(killing_1, government) [2]
definite(government) [6]
```

EFTA00624537

so that upon applying the rule:

```
# IF for($var0,$var1) ^ {present($var0) OR past($var0) OR future($var0)} \
 THEN  ^2_Benefit:Benefitor(for,$var1) ^2_Benefit:Act(for,$var0)
```

we obtain

```
^2_Benefit:Benefitor(for,Mafia)
^2_Benefit:Act(for,killing)

^2_Benefit:Benefitor(for_1,government)
^2_Benefit:Act(for_1,killing_1)
```

Here the first argument of the output relationships allows us to correctly associate the different acts of killing with the different benefitors

44.5.5 Comparatives and Phantom Nodes

Next, a bit of subtlety is needed to deal with sentences like

Mike eats more cookies than Ben.

which RelEx handles via

```
_subj(eat, Mike)
_obj(eat, cookie)
more(cookie, $cVar0)
$cVar0(Ben)
```

Then a RelEx2FrameNet mapping rule such as:

```
IF
_subj(eat,$var0)
_obj(eat,$var1)
more($var1,$cVar0)
$cVar0($var2)
THEN
^2_AsymmetricEvaluativeComparison:ProfiledItem(more, $var1)
^2_AsymmetricEvaluativeComparison:StandardItem(more, $var1_1)
^2_AsymmetricEvaluativeComparison:Valence(more, more)
^1_Ingestion:Ingestor(eat,$var0)
^1_Ingestion:Ingested(eat,$var1)
^1_Ingestion:Ingestor(eat_1,$var2)
^1_Ingestion:Ingested(eat_1,$var1_1)
```

applies, which embodies the commonsense intuition about comparisons regarding eating. (Note that we have introduced a new frame AsymmetricEvaluativeComparison here, by analogy to the standard FrameNet frame Evaluative_comparison.)

Note also that the above rule may be too specialized, though it's not incorrect. One could also try more general rules like

EFTA00624538

```
IF
%Agent($var0)
%Agent($var1)
_subj($var3,$var0)
_obj($var3,$var1)
more($var1,$cVar0)
$cVar0($var2)
THEN
^2_AsymmetricEvaluativeComparison:ProfiledItem(more, $var1)
^2_AsymmetricEvaluativeComparison:StandardItem(more, $var1_1)
^2_AsymmetricEvaluativeComparison:Valence(more, more)
_subj($var3,$var0)
_obj($var3,$var1)
_subj($var3_1,$var2)
_obj($var3_1,$var1_1)
```

However, this rule is a little different than most RelEx2Frame rules, in that it produces output that then needs to be processed by the RelEx2Frame rule-base a second time. There's nothing wrong with this, it's just an added layer of complexity.

44.6 Frame2Atom

The next step in the current OpenCog NLP comprehension pipeline is to translate the output of RelEx2Frame into Atoms. This may be done in a variety of ways; the current Frame2Atom script embodies one approach that has proved workable, but is certainly not the only useful one.

The Node types currently used in Frame2Atom are:

- WordNode
- ConceptNode
  - DefinedFrameNode
  - DefinedLinguisticConceptNode
- PredicateNode
  - DefinedFrameElementNode
  - DefinedLinguisticRelationshipNode
- SpecificEntityNode

The special node types

- DefinedFrameNode
- DefinedFrameElementNode

have been created to correspond to FrameNet frames and elements respectively (or frames and elements drawn from similar resources to FrameNet, such as our own frame dictionary).

Similarly, the special node types

EFTA00624539

- DefinedLinguisticConceptNode
- DefinedLinguisticRelationshipNode

have been created to correspond to RelEx unary and binary relationships respectively.

The "defined" is in the names because once we have a more advanced CogPrime system, it will be able to learn its own frames, frame elements, linguistic concepts and relationships. But what distinguishes these "defined" Atoms is that they have names which correspond to specific external resources.

The Link types we need for Frame2Atom are:

- InheritanceLink
- ReferenceLink (current using WRLink aka "word reference link")
- FrameElementLink

ReferenceLink is a special link type for connecting concepts to the words that they refer to. (This could be eliminated via using more complex constructs, but it's a very common case so for practical purposes it makes sense to define it as a link type.)

FrameElementLink is a special link type connecting a frame to its element. Its semantics (and how it could be eliminated at cost of increased memory and complexity) will be explained below.

44.6.1 Examples of Frame2Atom

Below follow some examples to illustrate the nature of the mapping intended. The examples include a lot of explanatory discussion as well.

Note that, in these examples, $[n]$ denotes an Atom with AtomHandle n. All Atoms have Handles, but Handles are only denoted in cases where this seems useful. (In the XML representation used in the current OpenCogPrime impelmentation, these are replaced by UUID's)

The notation

$$WordNode\#pig$$

denotes a WordNode with name pig, and a similar convention is used for other AtomTypes whose names are useful to know.

These examples pertain to fragments of the parse

```
Ben slowly ate the fat chickens.

A:_advmod:V(slowly:A, eat:V)
N:_nn:N(fat:N, chicken:N)
N:definite(Ben:N)
N:definite(chicken:N)
N:masculine(Ben:N)
N:person(Ben:N)
N:plural(chicken:N)
N:singular(Ben:N)
V:_obj:N(eat:V, chicken:N)
V:_subj:N(eat:V, Ben:N)
V:past(eat:V)
```

EFTA00624540

```
^1_Ingestion:Ingestor(eat,Ben)
^1_Temporal_colocation:Event(past,eat)
^1_Ingestion:Ingestibles(eat,chicken)
^1_Activity:Agent(subject,Ben)
^1_Activity:Activity(verb,eat)
^1_Transitive_action:Event(verb,eat)
^1_Transitive_action:Patient(object,chicken)
```

44.6.1.1 Example 1

$$_obj(eat, chicken)$$

would map into

```
EvaluationLink
   DefinedLinguisticRelationshipNode #_obj
   ListLink
        ConceptNode [2]
        ConceptNode [3]

InheritanceLink
   [2]
   ConceptNode [4]

InheritanceLink
   [3]
   ConceptNode [5]

ReferenceLink [6]
   WordNode #eat  [8]
   [4]

ReferenceLink [7]
   WordNode #chicken [9]
   [5]
```

Please note that the Atoms labeled 4,5,6,7,8,9 would not normally have to be created when entering the relationship

$$_obj(eat, chicken)$$

into the AtomTable. They should already be there, assuming the system already knows about the concepts of eating and chickens. These would need to be newly created only if the system had never seen these words before.

For instance, the Atom [2] represents the specific instance of "eat" involved in the relationship being entered into the system. The Atom [4] represents the general concept of "eat", which is what is linked to the word "eat."

Note that a very simple step of inference, from these Atoms, would lead to the conclusion

```
EvaluationLink
```

EFTA00624541

```
DefinedLinguisticRelationshipNode #_obj
ListLink
     ConceptNode [4]
     ConceptNode [5]
```

which represents the general statement that chickens are eaten. This is such an obvious and important step, that perhaps as soon as the relationship $_obj(eat, chicken)$ is entered into the system, it should immediately be carried out (i.e. that link if not present should be created, and if present should have its truth value updated). This is a choice to be implemented in the specific scripts or schema that deal with ingestion of natural language text.

44.6.1.2 Example 2

$$masculine(Ben)$$

would map into

```
InheritanceLink
    SpecificEntityNode [40]
    DefinedLinguisticConceptNode #masculine

InheritanceLink
   [40]
   [10]

ReferenceLink
   WordNode #Ben
   [10]
```

44.6.1.3 Example 3

The mapping of the RelExToFrame output

$$Ingestion : Ingestor(eat, Ben)$$

would use the existing Atoms

```
DefinedFrameNode #Ingestion [11]
DefinedFrameElementNode #Ingestion:Ingestor [12]
```

which would be related via

```
FrameElementLink [11] [12]
```

(Note that FrameElementLink may in principle be reduced to more elementary PLN link types.)

Note that each FrameNet frame contains some core elements and some optional elements. This may be handled by giving core elements links such as

```
FrameElementLink F E <1>
```

EFTA00624542

and optional ones links such as

```
FrameElementLink F E <.7>
```

Getting back to the example at hand, we would then have

```
InheritanceLink [2] [11]
```

(recall, [2] is the instance of eating involved in Example 1; and, [11] is the Ingestion frame), which says that this instance of eating is an instance of ingestion. (In principle, some instances of eating might not be instances of ingestion – or more generally, we can't assume that all instances of a given concept will always associate with the same FrameNodes. This could be assumed only if we assumed all word-associated concepts were disambiguated to a single known FrameNet frame, but this can't be assumed, especially if later on we want to use cognitive processes to do sense disambiguation.)

We would then also have links denoting the role of Ben as an Ingestor in the frame-instance [2], i.e.

```
EvaluationLink
   DefinedFrameElementNode #Ingestion:Ingestor [12]
   ListLink
        [2]
        [40]
```

This says that the specific instance of Ben observed in that sentence ([4]) served the role of Ingestion:Ingestor in regard to the frame-instance [2] (which is an instance of eating, which is known to be an instance of the frame of Ingestion).

44.6.2 Issues Involving Disambiguation

Right now, OpenCogPrime's RelEx2Frame rulebase is far from adequately large (there are currently around 5000 rules) and the link parser and RelEx are also imperfect. The current OpenCog NLP system does work, but for complex sentences it tends to generate too many interpretations of each sentence – "parse selection" or more generally "interpretation selection" is not yet adequately addressed. This is a tricky issue that can be addressed to some extent via statistical linguistics methods, but we believe that to solve it convincingly and thoroughly will require more cognitively sophisticated methods.

The most straightforward way to approach it statistically is to process a large number of sentences, and then tabulate co-occurrence probabilities of different relationships across all the sentences. This allows one to calculate the probability of a given interpretation conditional on the corpus, via looking at the probabilities of the combinations of relationships in the interpretation. This may be done using a Bayes Net or using PLN – in any case the problem is one of calculating the probability of a conjunction of terms based on knowledge regarding the probabilities of various sub-conjunctions. As this method doesn't require marked-up training data, but is rather purely unsupervised, it's feasible to apply it to a very large corpus of text – the only cost is computer time.

What the statistical approach won't handle, though, are the more conceptually original linguistic constructs, containing combinations that didn't occur frequently in the system's training

EFTA00624543

corpus. It will rate innovative semantic constructs as unlikely, which will lead it to errors sometimes – errors of choosing an interpretation that seems odd in terms of the sentence's real-world interpretation, but matches well with things the system has seen before. The only way to solve this is with genuine understanding – with the system reasoning on each of the interpretations and seeing which one makes more sense. And this kind of reasoning generally requires some relevant commonsense background knowledge – which must be gained via experience, reading and conversing, or from a hand-coded knowledge base, or via some combination of the above.

Related issues also involving disambiguation include word sense disambiguation (words with multiple meanings) and anaphor resolution (recognizing the referents of pronouns, and of nouns that refer to other nouns, etc.).

The current RelEx system contains a simple statistical parse ranker (which rates a parse higher if the links it includes occur more frequently in a large parsed corpus), statistical methods for word sense disambiguation [Mih07] inspired by those in Rada Mihalcea's work [SM09], and an anaphor resolution algorithm based on the classic Hobbs Algorithm (customized to work with the link parser) [Hob78]. While reasonably effective in many cases, from an AGI perspective these must all be considered "stopgaps" to be replaced with code that handles these tasks using probabilistic inference. It is conceptually straightforward to replace statistical linguistic algorithms with comparable PLN-based methods, however significant attention must be paid to code optimization as using a more general algorithm is rarely as efficient as using a specialized one. But once one is handling things in PLN and the Atomspace rather than in specialized computational linguistics code, there is the opportunity to use a variety of inference rules for generalization, analogy and so forth, which enables a radically more robust form of linguistic intelligence.

44.7 Syn2Sem: A Semi-Supervised Alternative to RelEx and RelEx2Frame

This section describes an alternative approach to the RelEx / RelEx2Frame approach described above, which is in the midst of implementation at time of writing. This alternative represents a sort of midway point between the rule-based RelEx / RelEx2Frame approach, and a conceptually ideal fully experiential learning based approach.

The motivations underlying this alternative approach have been to create an OpenCog NLP system with the capability to:

- support simple dialogue in a video game like world, and a robot system
- leverage primarily semi-supervised experiential learning
- replace the RelEx2Frame rules, which are currently problematic, with a different way of mapping syntactic relationships into Atoms, that is still reasoning and learning friendly
- require only relatively modest effort for implementation (not multiple human-years)

The latter requirement ruled out a pure "learn language from experience with no aid from computational linguistics tools" approach, which may well happen within OpenCog at some point.

EFTA00624544

44.8 Mapping Link Parses into Atom Structures

The core idea of the new approach is to learn "Syn2Sem" rules that map *link parses* into *Atom structures*. These rules may then be automatically reversed to form Sem2Syn rules, which may be used in language generation.

Note that this is different from the RelEx approach as currently pursued (the "old approach"), which contains

- one set of rules (the RelEx rules) mapping link parses into semantic relation-sets ("RelEx relation-sets" or rel-sets)
- another set of rules (the RelEx2Frame rules) mapping rel-sets into FrameNet-based relation-sets
- another set of rules (the Frame2Atom rules) mapping FrameNet-based relation-sets into Atom-sets

In the old approach, all the rules were hand-coded. In the new approach

- nothing *needs* to be hand-coded (except the existing link parser dictionary); the rules can be learned from a corpus of (link-parse, Atom-set) pairs. This corpus may be human-created; or may be derived via a system's experience in some domain where sentences are heard or read, and can be correlated with observed nonlinguistic structures that can be described by Atoms.
- in practice, some hand-coded rules are being created to map RelEx rel-sets into Atom-sets directly (bypassing RelEx2Frame) in a simple way. These rules will be used, together with RelEx, to create a large corpus of (link parse, Atom-set) pairs, which will be used as a training corpus. This training corpus will have more errors than a hand-created corpus, but will have the compensating advantage of being significantly larger than any hand-created corpus would feasibly be.

In the old approach, NL generation was done by using a pattern-matching approach, applied to a corpus of (link parse, rel-set) pairs, to mine rules mapping rel-sets to sets of link parser links. This worked to an extent, but the process of piecing together the generated sets of link parser links to form coherent "sentence parses" (that could then be turned into sentences) turned out to be subtler than expected, and appeared to require an escalatingly complex set of hand-coded rules, to be extended beyond simple cases.

In the new approach, NL generation is done by explicitly reversing the mapping rules learned for mapping link parses into Atom sets. This is possible because the rules are explicitly given in a form enabling easy reversal; whereas in the old approach, RelEx transformed link parses into rel-sets using a process of successively applying many rules to an ornamented tree, each rule acting on variables ("ornaments") deposited by previous rules. Put simply, RelEx transformed link parses into rel-sets via imperative programming, whereas in the new approach, link parses are transformed into Atom-sets using learned rules that are *logical* in nature. The movement from imperative to logical style dramatically eases automated rule reversal.

EFTA00624545

44.8.1 Example Training Pair

For concreteness, an example (link parse, Atom-set) pair would be as follows. For the sentence "Trains move quickly", the link parse looks like

```
Sp(trains, move)
MVa(move, quickly)
```

whereas the Atom-set looks like

```
Inheritance
   move_1
   move

Evaluation
   move_1
   train

 Inheritance
   move_1
   quick
```

Rule learning proceeds, in the new approach, from a corpus consisting of such pairs.

44.9 Making a Training Corpus

44.9.1 Leveraging RelEx to Create a Training Corpus

To create a substantial training corpus for the new approach, we are leveraging the existence of RelEx. We have a large corpus of sentences parsed by the link parser and then processed by RelEx. A new collection of rules is being created, RelEx2Atom, that directly translates RelEx parses into Atoms, in a simple way, embodying the minimal necessary degree of disambiguation (in a sense to be described just below). Using these RelEx2Atom rules, one can transform a corpus of (link parse, RelEx rel-set) triples into a corpus of (link parse, Atom-set) pairs – which can then be used as training data for learning Syn2Sem rules.

44.9.2 Making an Experience Based Training Corpus

An alternate approach to making a training corpus would be to utilize a virtual world such as the Unity3D world now being used for OpenCog game AI research and development.

A human game-player could create a training corpus by repeated:

- typing in a sentence
- indicating, via the graphic user interface, which entities or events in the virtual world were referred to by the sentence

EFTA00624546

Since OpenCog possesses code for transforming entities and events in the virtual world into Atom-sets, this would implicitly produce a training corpus of (sentence, Atom-set) pairs, which using the link parser could then be transformed into (link parse, Atom-set) pairs.

44.9.3 Unsupervised, Experience Based Corpus Creation

One could also dispense with the explicit reference-indication GUI, and just have a user type sentences to the AI agent as the latter proceeds through the virtual world. The AI agent would then have to figure out what specifically the sentences were referring to – maybe the human-controlled avatar is pointing at something; maybe one thing recently changed in the game world and nothing else did; etc. This mode of corpus creation would be reasonably similar to human first language learning in format (though of course there are many differences from human first language learning in the overall approach, for instance we are assuming the link parser, whereas a human language learner has to learn grammar for themselves, based on complex and ill-understood genetically encoded prior probabilistic knowledge regarding the likely aspects of the grammar to be learned).

This seems a very interesting direction to explore later on, but at time of writing we are proceeding with the RelEx-based training corpus, for sake of simplicity and speed of development.

44.10 Limiting the Degree of Disambiguation Attempted

The old approach is in a sense more ambitious than the new approach, because the RelEx2Frame rules attempt to perform a deeper and more thorough level of semantic disambiguation than the new rules. However, the RelEx2Frame rule-set in its current state is too "noisy" to be really useful; it would need dramatic improvement to be helpful in practice. The key difference is that,

- In the new approach, the syntax-to-semantics mapping rules attempt *only* the disambiguation that needs to be done to get the structure of the resultant Atom-set correct. Any further disambiguation is left to be done later, by MindAgents acting on the Atom-sets after they've already been placed in the AtomSpace.
- In the old approach, the RelEx2Frame rules attempted, in many cases, to disambiguate between different meanings beyond the level needed to disambiguate the structure of the Atom-set

To illustrate the difference, consider the sentences

- Love moves quickly.
- Trains move quickly.

These sentences involve different senses of "move" – change in physical location, versus a more general notion of progress. However, both sentences map to the same basic conceptual structure, e.g.

```
Inheritance
   move_1
```

EFTA00624547

```
      move

  Evaluation
     move_1
     train

   Inheritance
     move_1
     quick
```

versus

```
  Inheritance
     move_2
     move

  Evaluation
     move_2
     love

   Inheritance
     move_2
     quick
```

The RelEx2Frame rules try to distinguish between these cases via, in effect, associating the two instances move_1 and move_2 with different frames, using hand-coded rules that map RelEx rel-sets into appropriate Atom-sets defined in terms of FrameNet relations. This is not a useless thing to do; however, doing it well requires a very large and well-honed rule-base. Cyc's natural language engine attempts to do something similar, though using a different parser than the link parser and a different ontology than FrameNet; it does a much better job than the current version of RelEx2Frame, but still does a surprisingly incomplete job given the massive amount of effort put into sculpting the relevant rule-sets.

The new approach does not try to perform this kind of disambiguation prior to mapping things into Atom-sets. Rather, this kind of disambiguation is left for inference to do, after the relevant Atoms have already been placed in the AtomSpace. The rule of thumb is: Do precisely the disambiguation needed to map the parse into a compact, simple Atom-set, whose component nodes correspond to English words. Let the disambiguation of the meaning of the English words be done by some other process acting on the AtomSpace.

44.11 Rule Format

To represent Syn2Sem rules, it is convenient to represent link parses as Atom-sets. Each element of the training corpus will then be of the form (Atom set representing link parse, Atom-set representing semantic interpretation). Syn2Sem rules are then rules mapping Atom-sets to Atom-sets.

Broadly speaking, the format of a Syn2Sem rule is then

EFTA00624548

```
Implication
    Atom-set representing portion of link parse
    Atom-set representing portion of semantic interpretation
```

44.11.1 Example Rule

A simple example rule would be

```
Implication
    Evaluation
         Predicate: Sp
         \$V1
         \$V2
    Evaluation
         \$V2
         \$V1
```

This rule, in essence, maps verbs into predicates that take their subjects as arguments.

On the other hand, an Sem2Syn rule would look like the reverse:

```
Implication
    Atom-set representing portion of link parse
    Atom-set representing portion of semantic interpretation
```

Our currentapproach is to begin with Syn2Sem rules, because, due to the nature of natural language, these rules will tend to be more certain. That is: it is more strongly the case in natural languages that each syntactic construct maps into a small set of semantic structures, than that each semantic structure is realizable only via a small set of syntactic constructs. There are usually more ways structurally different, reasonably sensible ways to say an arbitrary thought, than there are structurally different, reasonably sensible ways to interpret an arbitrary sentence. Because of this fact about language, the design of the Atom-sets in the corpus is based on the principle of finding an Atom structure that most simply represents the meaning of the sentence corresponding to each given link parse. Thus, there will be many Syn2Sem rules with a high degree of certitude attached to them. On the other hand, the Sem2Syn rules will tend to have less certitude, because there may be many different syntactic ways to realize a given semantic expression.

44.12 Rule Learning

Learning of Syn2Sem rules may be done via any algorithm that is able to search rule space for rules of the proper format with high truth value as evaluated across the training set. Currently we are experimenting with using OpenCogPrime's frequent subgraph mining algorithm in this context. MOSES could also potentially be used to learn Syn2Sem rules. One suspects that MOSES might be better than frequent subgraph mining for learning complex rules, but based

EFTA00624549

on preliminary experimentation, frequent subgraph mining seems fine for learning the simple rules involved in simple sentences.

PLN inference may also be used to generate new rules by combining previous ones, and to generalize rules into more abstract forms.

44.13 Creating a Cyc-Like Database via Text Mining

The discussion of these NL comprehension mechanisms leads naturally to one interesting potential application of the OpenCog NL comprehension pipeline – which is only indirectly related to CogPrime, but would create a valuable resource for use by CogPrime if implemented. The possibility exists to use the OpenCog NL comprehension system to create a vaguely *Cyc-like database of common-sense rules.*

The approach would be as follows:

1. Get a corpus of text
2. Parse the text using OpenCog (RelEx or Syn2Sem)
3. Mine logical relationships among Atomrelationships from the data thus produced, using greedy data-mining, MOSES, or other methods

These mined logical relationships will then be loosely analogous to the rules the Cyc team have programmed in. For instance, there will be many rules like:

```
# IF _subj(understand,$var0) THEN ^1_Grasp:Cognizer(understand,$var0)
# IF _subj(know,$var0) THEN ^1_Grasp:Cognizer(understand,$var0)
```

So statistical mining would learn rules like

```
IF ^1_Mental_property(stupid) & ^1_Mental_property:Protagonist($var0)
  THEN ^1_Grasp:Cognizer(understand,$var0) <.3>
IF ^1_Mental_property(smart) & ^1_Mental_property:Protagonist($var0)
  THEN ^1_Grasp:Cognizer(understand,$var0) <.8>
```

which means that stupid people mentally grasp less than smart people do.

Note that these commonsense rules would come out automatically probabilistically quantified.

Note also that to make such rules come out well, one needs to do some (probabilistic) synonym-matching on nouns, adverbs and adjectives, e.g. so that mentions of "smart", "intelligent", "clever", etc. will count as instances of

```
^1_Mental_property(smart)
```

By combining probabilistic synonym matching on words, with mapping RelEx output into FrameNet input, and doing statistical mining, it should be possible to build a database like Cyc but far more complete and with coherent probabilistic weightings.

Although this way of building a commonsense knowledge base requires a lot of human engineering, it requires far less than something like Cyc. One "just" needs to build the RelEx2FrameNet mapping rules, not all the commonsense knowledge relationships directly –

EFTA00624550

those come from text. We do not advocate this as a solution to the AGI problem, but merely suggest that it could produce a large amount of useful knowledge to feed into an AGI's brain.

And of course, the better an AI one has, the better one can do the step labeled "Rank the parses and FrameNet interpretations using inference or heuristics or both." So there is a potential virtuous cycle here: more commonsense knowledge mined helps create a better AI mind, which helps mine better commonsense knowledge, etc.

44.14 PROWL Grammar

We have described the crux of the NL comprehension pipeline that is currently in place in the OpenCog codebase, plus some ideas for fairly moderate modifications or extensions. This section is a little more speculative, and describes an alternative approach that fits better with the overall CogPrime design, which however has not yet been implemented. The ideas given here lead more naturally to a design for experience-based language learning and processing, a connection that will be pointed out in a later section.

What we describe here is a partially-new theory of language formed via combining ideas from three sources: Hudson's Word Grammar [Hud90, Hud07a], Sleator and Temperley's link grammar, and Probabilistic Logic Networks. Reflecting its origin in these three sources, we have named the new theory PROWL grammar, meaning PRObabilistic Word Link Grammar. We believe PROWL has value purely as a conceptual approach to understanding language; however, it has been developed largely from the standpoint of computational linguistics – as part of an attempt to create a framework for computational language understanding and generation that both

1. yields broadly adequate behavior based on hand-coding of "expert rules" such as grammatical rules, combined with statistical corpus analysis
2. integrates naturally with a broader AI framework that combines language with embodied social, experiential learning, that ultimately will allow linguistic rules derived via expert encoding and statistical corpus analysis to be replaced with comparable, more refined rules resulting from the system's own experience

PROWL has been developed as part of the larger CogPrime project; but, it is described in this section mostly in a CogPrime -independent way, and is intended to be independently evaluable (and, hopefully, valuable).

As an integration of three existing frameworks, PROWL could be presented in various different ways. One could choose any one of the three components as an initial foundation, and then present the combined theory as an expansion/modification of this component. Here we choose to present it as an expansion/modification of Word Grammar, as this is the way it originated, and it is also the most natural approach for readers with a linguistics background. From this perspective, to simplify a fair bit, one may describe PROWL as consisting of Word Grammar with three major changes:

1. Word Grammar's network knowledge representation is replaced with a richer PLN-based network knowledge representation.

EFTA00624551

   a. This includes, for instance, the replacement of Word Grammar's single "isa" relationship type with a more nuanced collection of logically distinct probabilistic inheritance relationship types

2. Going along with the above, Word Grammar's "default inheritance" mechanism is replaced by an appropriate PLN control mechanism that guides the use of standard PLN inference rules

   a. This allows the same default-inheritance based inferences that Word Grammar relies upon, but embeds these inferences in a richer probabilistic framework that allows them to be integrated with a wide variety of other inferences

3. Word Grammar's small set of syntactic link types is replaced with a richer set of syntactic link types as used in Link Grammar

   a. The precise optimal set of link types is not clear; it may be that the link grammar's syntactic link type vocabulary is larger than necessary, but we also find it clear that the current version of Word Grammar's syntactic link type vocabulary is smaller than feasible (at least, without the addition of large, new, and as yet unspecified ideas to Word Grammar)

In the following subsections we will review these changes in a little more detail. Basic familiarity with Word Grammar, Link Grammar and PLN is assumed.

Note that in this section we will focus mainly on those issues that are somehow nonobvious. This means that a host of very important topics that come along with the Word Grammar / PLN integration are not even mentioned. The way Word Grammar deals with morphology, semantics and pragmatics, for instance, seems to us quite sensible and workable – and doesn't really change at all when you integrate Word Grammar with PLN, except that Word Grammar's crisp isa links become PLN-style probabilistic Inheritance links.

44.14.1 Brief Review of Word Grammar

Word Grammar is a theory of language structure which Richard Hudson began developing in the early 1980's [Hud90]. While partly descended from Systemic Functional Grammar, there are also significant differences. The main ideas of Word Grammar are as follows [†]:

- It presents language as a network of knowledge, linking concepts about words, their meanings, etc. - e.g. the word "dog" is linked to the meaning 'dog', to the form /dog/, to the word-class 'noun', etc.
- If language is a network, then it is possible to decide what kind of network it is (e.g. it seems to be a scale-free small-world network)
- It is monostratal - only one structure per sentence, no transformations.
- It uses word-word dependencies - e.g. a noun is the subject of a verb.
- It does not use phrase structure - e.g. it does not recognise a noun phrase as the subject of a clause, though these phrases are implicit in the dependency structure.

[†] the following list is paraphrased with edits from http://www.phon.ucl.ac.uk/home/dick/wg.htm downloaded on June 27 2010

EFTA00624552

- It shows grammatical relations/functions by explicit labels - e.g. 'subject' and 'object'.
- It uses features only for inflectional contrasts that are mentioned in agreement rules - e.g. number but not tense or transitivity.
- It uses default inheritance, as a very general way of capturing the contrast between 'basic' or 'underlying' patterns and 'exceptions' or 'transformations' - e.g. by default, English words follow the word they depend on, but exceptionally subjects precede it; particular cases 'inherit' the default pattern unless it is explicitly overridden by a contradictory rule.
- It views concepts as prototypes rather than 'classical' categories that can be defined by necessary and sufficient conditions. All characteristics (i.e. all links in the network) have equal status, though some may for pragmatic reasons be harder to override than others.
- In this network there are no clear boundaries between different areas of knowledge - e.g. between 'lexicon' and 'grammar', or between 'linguistic meaning' and 'encyclopedic knowledge'; language is not a separate module of cognition.
- In particular, there is no clear boundary between 'internal' and 'external' facts about words, so a grammar should be able to incorporate sociolinguistic facts - e.g. the speaker of "sidewalk" is an American.

44.14.2 Word Grammar's Logical Network Model

Word Grammar presents an elegant framework in which all the different aspects of language are encompassed within a single knowledge network. Representationally, this network combines two key aspects:

1. Inheritance (called is-a) is explicitly represented
2. General relationships between n-ary predicates and their arguments, including syntactic relationships, are explicitly represented

Dynamically, the network contains two key aspects:

1. An inference rule called "default inheritance"
2. Activation-spreading, similar to that in a neural network or standard semantic network

The similarity between Word Grammar and CogPrime is fairly strong. In the latter, inheritance and generic predicate-argument relationships are explicitly represented; and, a close analogue of activation spreading is present in the "attention allocation" subsystem. As in Word Grammar, important cognitive phenomena are grounded in the symbiotic combination of logical-inference and activation-spreading dynamics.

At the most general level, the reaction of the Word Grammar network to any situation is proposed to involve three stages:

1. Node creation and identification: of nodes representing the situation as understood, in its most relevant aspects
2. Where choices need to be made (e.g. where an identified predicate needs to choose which other nodes to bind to as arguments), activation spreading is used, and the most active eligible argument is utilized (this is called "best fit binding")
3. Default inheritance is used to supply new links to the relevant nodes as necessary

EFTA00624553

Default inheritance is a process that relies on the placement of each node in a directed acyclic graph hierarchy (dag) of isa links. The basic idea is as follows. Suppose one has a node N, and a predicate f(N,L), where L is another argument or list of arguments. Then, if the truth value of f(N,L) is not explicitly stored in the network, N inherits the value from any ancestor A in the dag so that: f(A,L) is explicitly stored in the network; and there is not any node P inbetween N and A for which f(P,L) is explicitly stored in the network. Note that multiple inheritance is explicitly supported, and in cases where this leads to multiple assignments of truth values to a predicate, confusion in the linguistic mind may ensue. In many cases the option coming from the ancestor with the highest level of activity may be selected.

Our suggestion is that Word Grammar's network representation may be replaced with PLN's logical network representation without any loss, and with significant gain. Word Grammar's network representation has not been fleshed out as thoroughly as that of PLN, it does not handle uncertainty, and it is not associated with general mechanisms for inference. The one nontrivial issue that must be addressed in porting Word Grammar to the PLN representation is the role of default inheritance in Word Grammar. This is covered in the following subsection.

The integration of activation spreading and default inheritance proposed in Word Grammar, should be easily achievable within CogPrime assuming a functional attention allocation subsystem.

44.14.3 Link Grammar Parsing vs Word Grammar Parsing

From a CogPrime /PLN point of view, perhaps the most striking original contribution of Word Grammar is in the area of syntax parsing. Word Grammar's treatment of morphology and semantics is, basically, exactly what one would expect from representing such things in a richly structured semantic network. PLN adds much additional richness to Word Grammar via allowing nuanced representation of uncertainty, which is critical on every level of the linguistic hierarchy – but this doesn't change the fundamental linguistic approach of Word Grammar. Regarding syntax processing, however, Word Grammar makes some quite specific and unique hypotheses, which if correct are very valuable contributions.

The conceptual assumption we make here is that syntax processing, while carried out using generic cognitive processes for uncertain inference and activation spreading, also involves some highly specific constraints on these processes. The extent to which these constraints are learned versus inherited is yet unknown, and for the subtleties of this issue the reader is referred to [EBJ+97]. Word Grammar and Link Grammar are then understood as embodying different hypotheses regarding what these constraints actually are.

It is interesting to consider the contributions of Word Grammar to syntax parsing via comparing it to Link Grammar.

Note that Link Grammar, while a less comprehensive conceptual theory than Word Grammar, has been used to produce a state-of-the-art syntax parser, which has been incorporated into a number of other software systems including OpenCog. So it is clear that the Link Grammar approach has a great deal of pragmatic value. On the other hand, it also seems clear that Link Grammar has certain theoretical shortcomings. It deals with many linguistic phenomena very elegantly, but there are other phenomena for which its approach can only be described as "hacky."

EFTA00624554

Word Grammar contains fewer hacks than Link Grammar, but has not yet been put to the test of large-scale computational implementation, so it's not yet clear how many hacks would need to be added to give it the relatively broad coverage that Link Grammar currently has. Our own impression is that to make Word Grammar actually work as the foundation for a broad-coverage grammar parser (whether standalone, or integrated into a broader artificial cognition framework), one would need to move it somewhat in the direction of link grammar, via adding a greater number of specialized syntactic link types (more on this shortly). There are in fact concrete indications of this in [Hud07a].

The Link Grammar framework may be decomposed into three aspects:

1. The link grammar dictionary, which for each word in English, contains a number of links of different types. Some links point left, some point right, and each link is labeled. Furthermore, some links are required and others are optional.
2. The "no-links-cross" constraint, which states that the correct parse of a sentence will involve drawing links between words, in such a way that all the required links of each word are fulfilled, and no two links cross when the links are depicted in two dimensions
3. A processing algorithm, which involves first searching the space of all possible linkages among the words in a sentence to find all complete linkages that obey the no-links-cross constraint; and then applying various postprocessing rules to handle cases (such as conjunctions) that aren't handled properly by this algorithm

In PROWL, what we suggest is that

1. The link grammar dictionary is highly valuable and provides a level of linguistic detail that is not present in Word Grammar; and, we suggest that in order to turn Word Grammar into a computationally tractable system, one will need something at least halfway between the currently minimal collection of syntactic link types used in Word Grammar and the much richer collection used in Link Grammar
2. The no-links-cross constraint is an approximation of a deeper syntactic constraint ("landmark transitivity") that has been articulated in the most recent formulations of Word Grammar. Specifically: when a no-links-crossing parse is found, it is correct according to Word Grammar; but Word Grammar correctly recognizes some parses that violate this constraint
3. The Link Grammar parsing algorithm is not cognitively natural, but is effective in a standalone-parsing framework. The Word Grammar approach to parsing is cognitively natural, but as formulated could only be computationally implemented in the context of an already-very-powerful general intelligence system. Fortunately, various intermediary approaches to parsing seem possible.

44.14.3.1 Using Landmark Transitivity with the Link Grammar Dictionary

An earlier version of Word Grammar utilized a constraint called "no tangled links" which is equivalent to the link parser's "no links cross" constraint. In the new version of Word Grammar this is replaced with a subtler and more permissive constraint called "landmark transitivity." While in Word Grammar, landmark transitivity is used with a small set of syntactic link types, there is no reason why it can't be used with the richer set of link types that Link Grammar provides. In fact, this seems to us a probably effective method of eliminating most or all of the "postprocessing rules" that exist in the link parser, and that constitute the least elegant aspect of the Link Grammar framework.

EFTA00624555

The first foundational concept, on the path to the notion of landmark transitivity, is the notion of a syntactic parent. In Word Grammar each syntactic link has a parent end and a child end. In a dependency grammar context, the notion is that the child depends upon the parent. For instance, in Word Grammar, in the link between a noun and an adjective, the noun is the parent.

To apply landmark transitivity in the context of the Link Grammar, one needs to provide some additional information regarding each link in the Link Grammar dictionary. One needs to specify which end of each of the link grammar links is the "parent" and which is the "child." Examples of this kind of markup are as follows (with P shown by the parent):

```
S link: subject-noun ------- finite verb (P)

O link: transitive verb (P) ----- direct or indirect object

D link: determiner ----- noun (P)

MV link: verb (P) ----- verb modifier

J link: preposition ----- object (P)

ON link: on  ----- time-expression [P]

M link: noun [P]----- modifiers
```

In some cases a word may have more than one parent. In this case, the rule is that the landmark is the one that is superordinate to all the other parents. In the rare case that two words are each others' parents, then either may serve as the landmark.

The concept of a parent leads naturally into that of a landmark. The first rule regarding landmarks is that a parent is a landmark for its child. Next, two kinds of landmarks are introduced: Before landmarks (in which the child is before the parent) and After landmarks (in which the child is after the parent). The Before/After distinction should be obvious in the Link Grammar examples given above.

The landmark transitivity rule, then, has two parts. If A is a landmark for B, of subtype L (where L is either Before or After), then

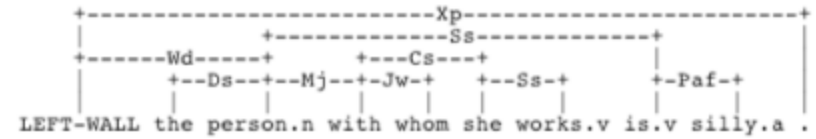
1. **Subordinate transitivity** says that if B is a landmark for C, then A is also a type-L landmark for C
2. **Sister transitivity** says that if A is a landmark for C, then B is also a landmark for C

Finally, there are some special link types that cause a word to depend on its grandparents or higher ancestors as well as its parents. I note that these are not treated thoroughly in (Hudson, 2007); one needs to look to the earlier, longer and rarer work [Hud90]. Some questions are dealt with this way. Another example is what in Word Grammar is called a "proxy link", as occurs between "with" and "whom" in

The person with whom she works

EFTA00624556

The link parser deals with this particular example via a Jw link



so to apply landmark transitivity in the context of the Link Grammar, in this case, it seems one would need to implement the rule that in the case of two words connected by a Jw-link, the child of one of the words is also the child of the other. Handling other special cases like this in the context of Link Grammar seems conceptually unproblematic, though naturally some hidden rocks may appear. Basically a list needs to be made of which kinds of link parser links embody proxy relationships for which other kinds of link parser links.

According to the landmark transitivity approach, then, the criterion for syntactic correctness of a parse is that, if one takes the links in the parse and applies the landmark transitivity rule (along with the other special-case "raising" rules we've discussed), one does not arrive at any contradictions (i.e. no situations where A is a Before landmark of B, and

The main problem with the landmark-transitivity constraint seems to be computational tractability. The problem exists for both comprehension and generation, but we'll focus on comprehension here.

To find all possible parses of a sentence using Hudson's landmark-transitivity-based approach, one needs to find all linkages that don't lead to contradictions when used as premises for reasoning based on the landmark-transitivity axioms. This appears to be extremely computationally intensive! So, it seems that Word Grammar style parsing is only computationally feasible for a system that has extremely strong semantic understanding, so as to be able to filter out the vast majority of possible parses on semantic rather than purely syntactic grounds.

On the other hand, it seems possible to apply landmark-transitivity together with no-links-cross, to provide parsing that is both efficient and general. If applying the no-links-cross constraint finds a parse in which no links cross, without using postprocessing rules, then this will always be a legal parse according to the landmark-transitivity rule.

However, landmark-transitivity also allows a lot of other parses that link grammar either needs postprocessing rules to handle, or can't find even with postprocessing rules. So, it would make sense to apply no-links-cross parsing first, but then if this fails, apply landmark-transitivity parsing starting from the partial parses that the former stage produced. This is the approach suggested in PROWL, and a similar approach may be suggested for language generation.

44.14.3.2 Overcoming the Current Limitations of Word Grammar

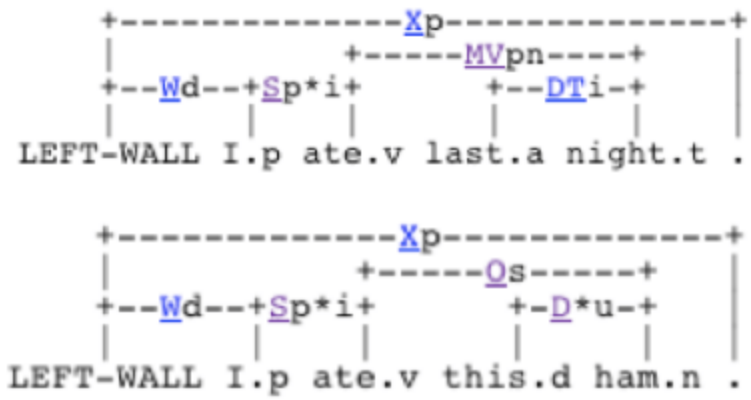
Finally, it is worth noting that expanding the Word Grammar parsing framework to include the link grammar dictionary, will likely allow us to solve some unsolved problems in Word Grammar. For instance, [Hud07a] notes that the current formulation of Word Grammar has no way to distinguish the behavior of last vs. this in

I ate last night
I ate this ham

The issue he sees is that in the first case, *night* should be considered the parent of *last*; whereas in the second case, *this* should be considered the parent of *ham*.

EFTA00624557

The current link parser also fails to handle this issue according to Hudson's intuition:



However, the link grammar framework gives us a clear possibility for allowing the kind of interpretation Hudson wants: just allow *this* to take a left-going O-link, and (in PROWL) let it optionally assume the parent role when involved in a D-link relationship. There are no funky link-crossing or semantic issues here; just a straightforward link-grammar dictionary edit.

This illustrates the syntactic flexibility of the link parsing framework, and also the inelegance – adding new links to the dictionary generally solves syntactic problems, but at the cost of creating more complexity to be dealt with further down the pipeline, when the various link types need to be compressed into a smaller number of semantic relationship types for purposes of actual comprehension (as is done in RelEx, for example). However, as far as we can tell, this seems to be a necessary cost for adequately handling the full complexity of natural language syntax. Word Grammar holds out the hope of possibly avoiding this kind of complexity, but without filling in enough details to allow a clear estimate of whether this hope can ever be fulfilled.

44.14.4 Contextually Guided Greedy Parsing and Generation Using Word Link Grammar

Another difference between Link Grammar and currently utilized, and Word Grammar as described, is the nature of the parsing algorithm. Link Grammar operates in a manner that is fairly traditional among contemporary parsing algorithms: given a sentence, it produces a large set of possible parses, and then it is left to other methods/algorithms to select the right parse, and to form a semantic interpretation of the selected parse. Parse selection may of course involve semantic interpretation: one way to choose the right parse is to choose the one that has the most contextually sensible semantic interpretation. We may call this approach *whole-sentence purely-syntactic parsing*, or WSPS parsing.

One of the nice things about Link Grammar, as compared to many other computational parsing frameworks, is that it produces a relatively small number of parses, compared for instance to typical head-driven phrase-structure grammar parsers. For simple sentences the link parser generally produces only handful of parses. But for complex sentences the link parser can produce hundreds of parses, which can be computationally costly to sift through.

EFTA00624558

Word Grammar, on the other hand, presents far fewer constraints regarding which words may link to other words. Therefore, to apply parsing in the style of the current link parser, in the context of Word Grammar, would be completely infeasible. The number of possible parses would be tremendous. The idea of Word Grammar is to pare down parses via semantic/pragmatic sensibleness, during the course of the syntax parsing process, rather than breaking things down into two phases (parsing followed by semantic/pragmatic interpretation). Parsing is suggested to proceed forward through a sentence: when a word is encountered, it is linked to the words coming before it in the sentence, in a way that makes sense. If this seems impossible, consistently with the links that have already been drawn in the course of the parsing process, then some backtracking is done and prior choices may be revisited. This approach is more like what humans do when parsing a sentence, and does not have the effect of producing a large number of syntactically possible, semantically/pragmatically absurd parses, and then sorting through them afterwards. It is what we call a *contextually-guided greedy parsing* (CGGP) approach.

For language generation, the link parser and Word Grammar approaches also suggest different strategies. Link Grammar suggests taking a semantic network, then searching holistically for a linear sequence of words that, when link-parsed, would give rise to that semantic network as the interpretation. On the other hand, Word Grammar suggests taking that same semantic network and iterating through it progressively, verbalizing each node of the network as one walks through it, and backtracking if one reaches a point where there is no way to verbalize the current node consistently with how one has already verbalized the previous nodes.

The main observation we want to make here is that, while Word Grammar by its nature (due to the relative paucity of explicit constraints on which syntactic links may be formed), can operate with CGGP but not WSPS parsing. On the other hand, while Link Grammar is currently utilized with WPSP parsing, there is no reason one can't use it with CGGP parsing just as well. There is no objection to using CGGP parsing together with the link-parser dictionary, nor with the no-links cross constraint rather than the landmark-transitivity constraint (in fact, as noted above, earlier versions of Word Grammar made use of the no-links-cross constraint).

What we propose in PROWL is to use the link grammar dictionary together with the CGGP parsing approach. The WSPS parsing approach may perhaps be useful as a fallback for handling extremely complex and perverted sentences where CGGP takes too long to come to an answer – it corresponds to sentences that are so obscure one has to do really hard, analytical thinking to figure out what they mean.

Regarding constraints on link structure, the suggestion in PROWL is to use the no-links-cross constraint as a first approximation. In comprehension, if no sufficiently high-probability interpretation obeying the no-links-cross constraint is found, then the scope of investigation should expand to include link-structures obeying landmark-transitivity but violating no-links-cross. In generation, things are a little subtler: a list should be kept of link-type combinations that often correctly violate no-links-cross, and when these combinations are encountered in the generation process, then constructs that satisfy landmark-transitivity but not no-links-cross should be considered.

Arguably, the PROWL approach is less elegant than either Link Grammar or Word Grammar considered on its own. However, we are dubious of the proposition that human syntax processing, with all its surface messiness and complexity, is really generated by a simple, unified, mathematically elegant underlying framework. Our goal is not to find a maximally elegant theoretical framework, but rather one that works both as a standalone computational-linguistics system, and as an integrated component of an adaptively-learning AGI system.

EFTA00624559

44.15 Aspects of Language Learning

Now we finally turn to language learning – a topic that spans the engineered and experiential approaches to NLP. In the experiential approach, learning is required to gain even simple linguistic functionality. In the engineered approach, even if a great deal of linguistic functionality is built in, learning may be used for adding new functionality and modifying the initially given functionality. In this section we will focus on a few aspects of language learning that would be required even if the current engineered OpenCog comprehension pipeline were completed to a high level of functionality. The more thoroughgoing language learning required for the experiential approach will then be discussed in the following section. Further, Chapter 45 will dig in depth into an aspect of language learning that to some extent cuts across the engineered/-experiential dichotomy – unsupervised learning of linguistic structures from large corpora of text.

44.15.1 Word Sense Creation

In our examples above, we've frequently referred to ReferenceLinks between WordNodes and ConceptNodes. But, how do these links get built? One aspect of this is the process of *word sense creation*.

Suppose we have a WordNode W that has ReferenceLinks to a number of different ConceptNodes. A common case is that these ConceptNodes fall into clusters, each one denoting a "sense" of the word. The clusters are defined by the following relationships:

1. ConceptNodes within a cluster have high-strength SimilarityLinks to each other
2. ConceptNodes in different clusters have low-strength (i.e. dissimilarity-denoting) SimilarityLinks to each other

When a word is first learned, it will normally be linked only to mutually agreeable ConceptNodes, i.e. there will only be one sense of the word. As more and more instances of the word are seen, however, eventually the WordNode will gather more than one sense. Sometimes different senses are different syntactically, other times they are different only semantically, but are involved in the same syntactic relationships. In the case of a word with multiple senses, most of the relevant feature structure information will be attached to word-sense-representing ConceptNodes, not to WordNodes themselves.

The formation of sense-representing ConceptNodes may be done by the standard clustering and predicate mining processes, which will create such ConceptNodes when there are adequately many Atoms in the system satisfying the criteria represent. It may also be valuable to create a particular SenseMining CIM-Dynamic, which uses the same criteria for node formation as the clustering and predicate mining CIM-Dynamics, but focuses specifically on creating predicates related to WordNodes and their nearby ConceptNodes.

EFTA00624560

44.15.2 Feature Structure Learning

We've mentioned above the obvious fact that, to intelligently use a feature-structure based grammar, the system needs to be capable of learning new linguistic feature structures. Probing into this in more detail, we see that there are two distinct but related kinds of feature structure learning:

1. learning the values that features have for particular word senses.
2. learning new features altogether.

Learning the values that features have for particular word senses must be done when new senses are created; and even for features imported from resources like the link grammar, the possibility of corrections must obviously be accepted. This kind of learning can be done by straightforward inference – inference from examples of word usage, and by analogy from features for similar words. A simple example to think about, e.g., is learning the verb sense of "fax" when only the noun sense is known.

Next, the learning of new features can be viewed as a reasoning problem, in that inference can learn new relations applied to nodes representing syntactic senses of words. In principle, these "features" may be very general or very specialized, depending on the case. New feature learning, in practice, requires a lot of examples, and is a more fundamental but less common kind of learning than learning feature values for known word senses. A good example would be the learning of "third person" by an agent that knows only first and second person.

In this example, it's clear that information from embodied experience would be extremely helpful. In principle, it could be learned from corpus analysis alone – but the presence of knowledge that certain words ("him", "her", "they", etc.) tend to occur in association with observed agents different from the speaker or the hearer, would certainly help a lot with identifying "third person" as a separate construct. It seems that either a very large number of un-embodied examples or a relatively small number of embodied examples would be needed to support the inference of the "third person" feature. And we suspect this example is typical – i.e. that the most effective route to new feature structure learning involves both embodied social experience and rather deep commonsense knowledge about the world.

44.15.3 Transformation and Semantic Mapping Rule Learning

Word sense learning and feature structure learning are important parts of language learning, but they're far from the whole story. An equally important role is played by *linguistic transformations*, such as the rules used in RelEx and RelEx2Frame. At least some of these must be learned based on experience, for human-level intelligent language processing to proceed.

Each of these transformations can be straightforwardly cast as an ImplicationLink between PredicateNodes, and hence formalistically can be learned by PLN inference, combined with one or another heuristic methods for compound predicate creation. The question is what knowledge exists for PLN to draw on in assessing the strengths of these links, and more critically, to guide the heuristic predicate formation methods. This is a case that likely requires the full complexity of "integrative predicate learning" as discussed in Chapter 41. And, as with feature structure learning, it's a case that will be much more effectively handled using knowledge from social embodied experience alongside purely linguistic knowledge.

EFTA00624561

44.16 Experiential Language Learning

We have talked a great deal about "engineered" approaches to NL comprehension and only peripherally about experiential approaches. But there has been a not-so-secret plan underlying this approach. There are many approaches to experiential language learning, ranging from a "tabula rasa" approach in which language is just treated as raw data, to an approach where the whole structure of a language comprehension system is programmed in, and "merely" the *content* remains to be learned. There isn't much to say about the tabula rasa approach – we have already discussed CogPrime's approach to learning, and in principle it is just as applicable to language learning as to any other kind of learning. The more structured approach has more unique aspects to it, so we will turn attention to it here. Of course, various intermediate approaches may be constructed by leaving out various structures.

The approach to experiential language learning we consider most promising is based on the PROWL approach, discussed above. In this approach one programs in a certain amount of "universal grammar," and then allows the system to learn content via experience that obeys this universal grammar. In a PROWL approach, the basic linguistic representational infrastructure is given by the Atomspace that already exists in OpenCog, so the content of "universal grammar" is basically

- the propensity to identify words
- the propensity to create a small set of asymmetric (i.e. parent/child) labeled relationship types, to use to label relationships between semantically related word-instances. These are "syntactic link types."
- the set of *constraints on syntactic links* implicit in word grammar, e.g. landmark transitivity or no-links-cross

Building in the above items, without building in any particular syntactic links, seems enough to motivate a system to learn a *grammar resembling that of human languages*.

Of course, experiential language learning of this nature is very, very different from "tabula rasa" experiential language learning. But we note that, while PROWL style experiential language learning seems like a *difficult* problem given existing AI technologies, tabula rasa language learning seems like a nearly unapproachable problem. One could infer from this that current AI technologies are simply inadequate to approach the problem that the young human child mind solves. However, there seems to be some solid evidence that the young human child mind does contain some form of universal grammar guiding its learning. Though we don't yet know what form this universal prior linguistic knowledge takes in the human mind or brain, the evidence regarding common structures arising spontaneously in various unrelated Creole languages is extremely compelling [Bic08], supporting ideas presented previously based on different lines of evidence. So we suggest that PROWL based experiential language learning is actually conceptually closer to human child language learning than a tabula rasa approach – although we certainly don't claim that the PROWL based approach builds in the exact same things as the human genome does.

What we need to make experiential language learning work, then, is a language-focused inference-control mechanism that includes, e.g.

- a propensity to look for syntactic link types, as outlined just above
- a propensity to form new word senses, as outlined earlier

EFTA00624562

- a propensity to search for implications of the general form of RelEx and RelEx2Frame or Syn2Sem rules

Given these propensities, it seems reasonable to expect a PLN inference system to be able to "fill in the linguistic content" based on its experience, using links between linguistic and other experiential content as its guide. This is a very difficult learning problem, to be sure, but it seems in principle a tractable one, since we have broken it down into a number of interrelated component learning problems in a manner guided by the structure of language.

Other aspects of language comprehension, such as word sense disambiguation and anaphor resolution, seem to plausibly follow from applying inference to linguistic data in the context of embodied experiential data, without requiring especial attention to inference control or supplying prior knowledge.

Chapter **??** presents an elaboration of this sort of perspective, in a limited case which enables greater clarity: the learning of linguistic content from an unsupervised corpus, based on the assumption of linguistic infrastructure s just summarized above.

44.17 Which Path(s) Forward?

We have discussed a variety of approaches to achieving human-level NL comprehension in the CogPrime framework. Which approach do we think is best? All things considered, we suspect that a tabula rasa experiential approach is impractical, whereas a traditional computational linguistics approach (whether based on hand-coded rules, corpus analysis, or a combination thereof) will reach an intelligence ceiling well short of human capability. On the other hand we believe that all of these options

1. the creation of an engineered NL comprehension system (as we have already done), and the adaptation and enhancement of this system using learning that incorporates knowledge from embodied experience
2. the creation of an engineered NL comprehension system via unsupervised learning from a large corpus, as described in Chapter **??** below
3. the creation of an experiential learning based NL comprehension system using in-built structures, such as the PROWL based approach described above
4. the creation of an experiential learning based system as described above, using an engineered system (like the current one) as a "fitness estimation" resource in the manner described at the end of Chapter 43

have significant promise and are worthy of pursuit. Which of these approaches we focus on in our ongoing OpenCogPrime implementation work will depend on logistical issues as much as on theoretical preference.

EFTA00624563

Chapter 45
Language Learning via Unsupervised Corpus Analysis

Co-authored with Linas Vepstas *

45.1 Introduction

The approach taken to NLP in the OpenCog project up through 2013, in practice, has involved engineering and integrating rule-based NLP systems as "scaffolding", with a view toward later replacing the rule content with alternative content learned via an OpenCog system's experience.

In this chapter we present a variant on this approach, in which the rule content of the existing rule-based NLP system is replaced with new content learned via unsupervised corpus analysis. This content can then be modified and improved via an OpenCog system's experience, embodied and otherwise, as needed.

This unsupervised corpus analysis based approach deviates fairly far from human cognitive science. However, as discussed above, language processing is one of those areas where the pragmatic differences between young humans and early-stage AGI systems may be critical to consider. The automated learning of language from embodied, social experience is a key part of the path to AGI, and is one way that CogPrimes and other AGI systems should learn language. On the other hand, unsupervised corpus based language learning, may perhaps also have a significant role to play in the path to linguistically savvy AGI, leveraging some advantages that AGIs have that humans do not, such as direct access to massive amounts of online text (without the need to filter the text through slow-paced sense-perception systems like eyes).

The learning of language from unannotated text corpora is not a major pursuit within the computational linguistics community currently. *Supervised* learning of linguistic structures from expert-annotated corpora plays a large role, but this is a wholly different sort of pursuit, more analogous to rule-based NLP, in that it involves humans explicitly specifying formal linguistic structures (e.g. parse trees for sentences in a corpus). However, we hypothesize that unsupervised corpus-based language learning can be carried out by properly orchestrating the use of some fairly standard machine learning algorithms (already included in OpenCog / CogPrime), within an appropriate structured framework (such as OpenCog's current NLP framework).

The review of [KM04] provides a summary of the state of the art in automatic grammar induction (the third alternative listed above), as it stood a decade ago: it addresses a num-

* Dr. Vepstas would properly be listed as the first author of this chapter; this material was developed in a collaboration between Vepstas and Goertzel. However, as with all the co-authored chapters in this book, final responsibility for any flaws in the presentation of the material lies with Ben Goertzel, the chief author of the bok.

EFTA00624564

ber of linguistic issues and difficulties that arise in actual implementations of algorithms. It is also notable in that it builds a bridge between phrase-structure grammars and dependency grammars, essentially pointing out that these are more or less equivalent, and that, in fact, significant progress can be achieved by taking on both points of view at once. Grammar induction has progressed somewhat since this review was written, and we will mention some of the more recent work below; but yet, it is fair to say that there has been no truly dramatic progress in this direction.

In this chapter we describe a novel approach to achieving automated grammar induction, i.e. to machine learning of linguistic content from a large, unannotated text corpus. The methods described may also be useful for language learning based on embodied experience; and may make use of content created using hand-coded rules or machine learning from annotated corpora. But our focus in this chapter will be on learning linguistic content from a large, unannotated text corpus.

The algorithmic approach given in this chapter is wholly in the spirit of the "PROWL" approach reviewed above in Chapter 44. However, PROWL is a quite general idea. Here we present a highly specific PROWL-like algorithm, which is focused on learning from a large unannotated corpus rather than from embodied experience. Because of the corpus-oriented focus, it is possible to tie the algorithm of this chapter in with the statistical language learning literature, more tightly than is possible with PROWL language learning in general. Yet, the specifics presented here could largely be generalized to a broader PROWL context.

We consider the approach described here as "deep learning" oriented because it is based on hierarchical pattern recognition in linguistic data: identifying patterns, then patterns among these patterns, etc., in a hierarchy that allows "higher level" (more abstract) patterns to feed back down the hierarchy and affect the recognition of lower level patterns. Our approach does not use conventional deep learning architectures like Deep Boltzmann machines or recurrent neural networks. Conceptually, our approach is based on a similar intuition to these algorithms, in that it relies on the presence of hierarchical structure in its input data, and utilizes a hierarchical pattern recognition structure with copious feedback to adaptively identify this hierarchical structure. But the specific pattern recognition algorithms we use, and the specific nature of the hierarchy we construct, are guided by existing knowledge about what works and what doesn't in (both statistical and rule-based) computational linguistics.

While the overall approach presented here is novel, most of the detailed ideas are extensions and generalizations of the prior work of multiple authors, which will be referenced and in some cases discussed below. In our view, the body of ideas needed to enable unsupervised learning of language from large corpora has been gradually emerging during the last decade. The approach given here has unique aspects, but also many aspects already validated by the work of others.

For sake of simplicity, we will deal here only with learning from written text here. We believe that conceptually very similar methods can be applied to spoken language as well, but this brings extra complexities that we will avoid for the purpose of the present document. (In short: Below we represent syntactic and semantic learning as separate but similarly structured and closely coupled learning processes. To handle speech input thoroughly, we would suggest phonological learning as another separate, similarly structured and closely coupled learning process.)

Finally, we stress that the algorithms presented here are intended to be used in conjunction with a large corpus, and a large amount of processing power. Without a very large corpus, some of the feedbacks required for the learning process described would be unlikely to happen (e.g. the ability of syntactic and semantic learning to guide each other). We have not yet sought

EFTA00624565

to estimate exactly *how* large a corpus would be required, but our informal estimate is that Wikipedia might or might not be large enough, and the Web is certainly more than enough.

We don't pretend to know just how far this sort of unsupervised, corpus based learning can be pushed. To what extent can the content of a natural language like English be learned this way. How much, if any, ambiguity will be left over once this kind of learning has been thoroughly done – only pragmatically disambiguable via embodied social learning? Strong opinions on these sorts of issues abound in the cognitive science, linguistics and AI communities; but the only apparent way to resolve these questions is empirically.

45.2 Assumed Linguistic Infrastructure

While the approach outlined in this chapter aims to learn the linguistic content of a language from textual data, it does not aim to learn the *idea* of language. Implicitly, we assume a model in which a learning system begins with a basic "linguistic infrastructure" indicating the various parts of a natural language and how they generally interrelate; and it then learns the linguistic content characterizing a particular language. In principle, it would also be possible to have an AI system to learn the very concept of a language and build its own linguistic infrastructure. However, that is not the problem we address here; and we suspect such an approach would require drastically more computational resources.

The basic linguistic infrastructure assumed here includes:

- A formalism for expressing grammatical (dependency) rules is assumed.
  - The ideas given here are not tied to any specific grammatical formalism, but as in Chapter **??** we find it convenient to make use of a formalism in the style of dependency grammar[Tes59]. Taking a mathematical perspective, different grammar formalisms can be translated into one-another, using relatively simple rules and algorithms[KM04]. The primary difference between them is more a matter of taste, perceived linguistic 'naturalness', adaptability, and choice of parser algorithm. In particular, categorial grammars can be converted into link grammars in a straight-forward way, and vice versa, but link grammars provide a more compact dictionary. Link grammars[ST91, ST93] are a type of dependency grammar; these, in turn, can be converted to and from phrase-structure grammars. We believe that dependency grammars provide a more simple and natural description of linguistic phenomena. We also believe that dependency grammars have a more natural fit with maximum-entropy ideas, where a dependency relationship can be literally interpreted as the mutual information between word-pairs[Yur98]. Dependency grammars also work well with Markov models; dependency parsers can be implemented as Viterbi decoders. Figure 44.1 illustrates two different formalisms.
  - The discussion below assumes the use of a formalism similar that of Link Grammar, as described above. In this theory, each word is associated with a set of 'connector disjuncts', each connector disjunct controlling the possible linkages that the word may take part in. A disjunct can be thought of as a jig-saw puzzle-piece; valid syntactic word orders are those for which the puzzle-pieces can be validly connected. A single connector can be thought of as a single tab on a puzzle-piece (shown in figure **??**). Connectors are thus 'types' X with a + or - sign indicating that they connect to the left or right. For

EFTA00624566

example, a typical verb disjunct might be $S- \& O+$ indicating that a subject (a noun) is expected on the left, and an object (also a noun) is expected on the right.
  - Some of the discussion below assumes select aspects of (Dick Hudson's) Word Grammar[Hud84, Hud07b]. As reviewed above, Word Grammar theory (implicitly) uses connectors similar to those of Link Grammar, but allows each connector to be marked as the head of a link or not. A link then becomes an arrow from a head word to the dependent word. (Somewhat confusingly, the head of the arrow points at the dependent word; this means the tail of the arrow is attached to the head word).
  - Each word is associated with a "lexical entry"; in Link Grammar, this is the set of connector disjuncts for that word. It is usually the case that many words share a common lexical entry; for example, most common nouns are syntactically similar enough that they can all be grouped under a single lexical entry. Conversely, a single word is allowed to have multiple lexical entries; so, for example, "saw", the noun, will have a different lexical entry from "saw", the past tense of the verb "to see". That is, lexical entries can loosely correspond to traditional dictionary entries. Whether or not a word has multiple lexical entries is a matter of convenience, rather than a fundamental aspect. Curiously, a single Link Grammar connector disjunct can be viewed as a very fine-grained part-of-speech. In this way, it is a stepping stone to the semantic meaning of a word.

- A parser, for extracting syntactic structure from sentences, is assumed. What's more, it is assumed that the parser is capable of using semantic relationships to guide parsing.
  - A paradigmatic example of such a parser is the "Viterbi Link Parser", currently under development for use with the Link Grammar. This parser is currently operational in a simple form. The name refers to its use of a the general ideas of the Viterbi algorithm. This algorithm seems biologically plausible, in that it applies only a local analysis of sentence structure, of limited scope, as opposed to a global optimization, thus roughly emulating the process of human listening. The current set of legal parses of a sentence is pruned incrementally and probabilistically, based on flexible criteria. These potentially include the semantic relationships extractable from the partial parse obtained at a given point in time. It also allows for parsing to be guided by inter-sentence relationships, such as pronoun resolution, to disambiguate otherwise ambiguous sentences.

- A formalism for expressing semantic relationships is assumed.
  - A semantic relationship generalizes the notion of a lexical entry to allow for changes of word order, paraphrasing, tense, number, the presence or absence of modifiers, *etc.* An example of such a relationship would be *eat(X, Y)* – indicating the eating of some entity Y by some entity X. This abstracts into common form several different syntactic expressions: "*Ben ate a cookie*", "*A cookie will be eaten by Ben*", "*Ben sat, eating cookies*".
  - Nothing particularly special is assumed here regarding semantic relationships, beyond a basic predicate-argument structure. It *is* assumed that predicates can have arguments that are other predicates, and not just atomic terms; this has an explicit impact on how predicates and arguments are represented. A "semantic representation" of a sentence is a network of arrows (defining predicates and arguments), each arrow or a small subset of arrows defining a "semantic relationship". However, the beginning or end of an arrow is not necessarily a single node, but may land on a subgraph.

EFTA00624567

- Type constraints seem reasonable, but its not clear if these must be made explicit, or if they are the implicit result of learning. Thus, *eat(X, Y)* requires that X and Y both be entities, and not, for example, actions or prepositions.
- We have not yet thought through exactly how rich the semantic formalism should be for handling the full variety of quantifier constructs in complex natural language. But we suspect that it's OK to just use basic predicate-argument relationships and not build explicit quantification into the formalism, allowing quantifiers to be treated like other predicates.
- Obviously, CogPrime's formalism for expressing linguistic structures in terms of Atoms, presented in Chapter 44, fulfills the requirements of the learning scheme presented in this chapter. However, we wish to stress that the learning scheme presented her does not depend on the particulars of CogPrime's representation scheme, though it is very compatible with them.

45.3 Linguistic Content To Be Learned

Given the above linguistic infrastructure, what remains for a language learning system to learn is the *linguistic content* that characterizes a particular language. Everything included in OpenCog's existing "scaffolding" rule-based NLP system would, in this approach, be learned to first approximation via unsupervised corpus analysis.

Specifically, given the assumed framework, key things to be learned include:

- A list of 'link types' that will be used to form 'disjuncts' must be learned.
  - An example of a link type is the 'subject' link S. This link typically connects the subject of a sentence to the head verb. Given the normal English subject-verb word order, nouns will typically have an S+connector, indicating that an S link may be formed only when the noun appears to the left of a word bearing an $S-$ connector. Likewise, verbs will typically be associated with $S-$ connectors. The current Link Grammar contains roughly one hundred different link-types, with additional optional subtypes that are used to further constrain syntactic structure. This number of different link types seems required simply because there are many relationships between words: there is not just a subject-verb or verb-object relationship, but also rather fine distinctions, such as those needed to form grammatical time, date, money, and measurement expressions, punctuation use, including street-addresses, cardinal and ordinal relationships, proper (given) names, titles and suffixes, and other highly constrained grammatical constructions. This is in addition to the usual linguistic territory of needing to indicate dependent clauses, comparatives, subject-verb inversion, and so on. It is expected that a comparable number of link types will need to be learned.
  - Some link types are rather strict, such as those connect verb subjects and objects, while other types are considerably more ambiguous, such as those involving prepositions. This reflects the structure of English, where subject-verb-object order is fairly rigorously enforced, but the ordering and use of prepositions is considerably looser. When considering the looser cases, it becomes clear that there is no single, inherent 'right answer' for the creation and assignment of link types, and that several different, yet linguistically plausible linkage assignments may be made.

EFTA00624568

  - The definition of a good link-type is one that leads the parser – applied across the whole corpus – to allow parsing to be successful for almost all sentences, and yet not to be so broad as to enable parsing of word-salads. Significant pressure must be applied to prevent excess proliferation of link types, yet no so much as to over-simplify things, and provide valid parses for unobserved, ungrammatical sentences.

- Lexical entries for different words must be learned.

  - Typically, multiple connectors are needed to define how a word can link syntactically to others. Thus, for example, many verbs have the disjunct $S- \& O+$ indicating that they need a subject noun to the left, and an object to the right. All words have at least a handful of valid disjuncts that they can be used with, and sometimes hundreds or even more. Thus, a "lexical entry" must be learned for each word, the lexical entry being a set of disjuncts that can be used with that word.
  - Many words are syntactically similar; most common nouns can share a single lexical entry. Yet, there are many exceptions. Thus, during learning, there is a back-and forth process of grouping and ungrouping words; clustering them so that they share lexical entries, but also splitting apart clusters when its realized that some words behave differently. Thus for example, the words "sing" and "apologize" are both verbs, and thus share some linguistic structure, but one cannot say "I apologized a song to Vicky"; if these two verbs were initially grouped together into a common lexical entry, they must later be split apart.
  - The definition of a good lexical entry is much the same as that for a good link type: observed sentences must be parsable; random sentences mostly must not be, and excessive proliferation and complexity must be prevented.

- Semantic relationships must be learned.

  - The semantic relationship *eat(X,Y)* is prototypical. Foundationally, such a semantic relationship may be represented as a set whose elements consist of syntactico-semantic subgraphs. For the relation *eat(X,Y)*, a subgraph may be as simple as a single (syntactic) disjunct $S- \& O+$ for the normal word order "*Ben ate a cookie*", but it may also be a more complex set needed to represent the inverted word order in *"a cookie was eaten by Ben"*. The set of all of these different subgraphs defines the semantic relationship. The subgraphs themselves may be syntactic (as in the example above), or they may be other semantic relationships, or a mixture thereof.
  - Not all re-phrasings are semantically equivalent. "*Mr. Smith is late*" has a rather different meaning from "*The late Mr. Smith.*"
  - In general, place-holders like X and Y may be words or category labels. In early stages of learning, it is expected that X and Y are each just sets of words. At some point, though, it should become clear that these sets are not specific to this one relationship, but can appropriately take part in many relationships. In the above example, X and Y must be entities (physical objects), and, as such, can participate in (most) any other relationships where entities are called for. More narrowly, X is presumably a person or animal, while Y is a foodstuff. Furthermore, as entities, it might be inferred when these refer to the same physical object (see the section 'reference resolution' below).
  - Categories can be understood as sets of synonyms, including hyponyms (thus, "*grub*" is a synonym for "*food*", while "*cookie*" is a hyponym.

EFTA00624569

- Idioms and set phrases must be learned.
  - English has a large number of idiomatic expressions whose meanings cannot be inferred from the constituent words (such as "*to pull one's leg*"). In this way, idioms present a challenge: their sometimes complex syntactic constructions belie their often simpler semantic content. On the other hand, idioms have a very rigid word-choice and word order, and are highly invariant. Set phrases take a middle ground: word-choice is not quite as fixed as for idioms, but, none-the-less, there is a conventional word order that is usually employed. Not that the manually-constructed Link Grammar dictionaries contain thousands of lexical entries for idiomatic constructions. In essence, these are multi-word constructions that are treated as if they were a single word.

Each of the above tasks have already been accomplished and described in the literature; for example, automated learning of synonymous words and phrases has been described by Lin[LP01] and Poon & Domingos[PD09]. The authors are not aware of any attempts to learn all of these, together, in one go, rather than presuming the pre-existance of dependent layers.

45.3.1 Deeper Aspects of Comprehension

While the learning of the above aspects of language is the focus of our discussion here, the search for semantic structure does not end there; more is possible. In particular, natural language generation has a vital need for lexical functions, so that appropriate word-choice can be made when vocalizing ideas. In order to truly understand text, one also needs, as a minimum, to discern referential structure, and sophisticated understanding requires discerning topics. We believe automated, unsupervised learning of these aspects is attainable, but is best addressed after the 'simpler' language learning described above. We are not aware of any prior work aimed at automatically learning these, aside from relatively simple, unsophisticated (bag-of-words style) efforts at topic categorization.

45.4 A Methodology for Unsupervised Language Learning from a Large Corpus

The language learning approach presented here is novel in its overall nature. Each part of it, however, draws on prior experimental and theoretical research by others on particular aspects of language learning, as well as on our own previous work building computational linguistic systems. The goal is to assemble a system out of parts that are already known to work well in isolation.

Prior published research, from a multitude of authors over the last few decades, has already demonstrated how many of the items listed above can be learnt in an unsupervised setting (see e.g. [Yur98, KM04, LP01, CS10, PD09, Mih07, KSPC13] for relevant background). All of the previously demonstrated results, however, were obtained in isolation, via research that assumed the pre-existence of surrounding infrastructure far beyond what we assume above. The approach proposed here may be understood as a combination, generalization and refinement

EFTA00624570

these techniques, to create a system that can learn, more or less *ab initio* from a large corpus, with a final result of a working, usable natural language comprehension system.

However, we must caution that the proposed approach is in no way a haphazard mash-up of techniques. There is a deep algorithmic commonality to the different prior methods we combine, which has not always been apparent in the prior literature due to the different emphases and technical vocabularies used in the research papers in question. In parallel with implementing the ideas presented here, we intend to workin fully formalizing the underlying mathematics of the undertaking, so that it becomes clear what approximations are being taken, and what avenues remain unexplored. Some fairly specific directions in this regard suggest themselves.

All of the prior research alluded to above invokes some or another variation of maximum entropy principles, sometimes explicitly, but usually implicitly. In general, entropy maximization principles provide the foundation for learning systems such as (hidden) Markov models, Markov networks and Hopfield neural networks, and they connect indirectly with Bayesian probability based analyses. However, the actual task of maximizing the entropy is an NP-hard problem; forward progress depends on short-cuts, approximations and clever algorithms, some of which are of general nature, and some domain-dependent. Part of the task of refining the details of the language learning methodology presented here, is to explore various short-cuts and approximations to entropy maximization, and discover new, clever algorithms of this nature that are relevant to the language learning domain. As has been the case in physics and other domains, we suspect that progress here will be best achieved via a coupled exploration of experimental and mathematical aspects of the subject matter.

45.4.1 A High Level Perspective on Language Learning

On an abstract conceptual level, the approach proposed here depicts language learning as an instance of a general learning loop such as:

1. Group together linguistic entities (*i.e.* words or linguistic relationships, such as those described in the previous section) that display similar usage patterns (where one is looking at usage patterns that are compactly describable given one's meta-language). Many but not necessarily all usage patterns for a given linguistic entity, will involve its use in conjunction with other linguistic entities.
2. For each such grouping make a category label.
3. Add these category labels to one's meta-language
4. Return to Step 1

It stands to reason that the result of this sort of learning loop, if successful, will be a hierarchically composed collection linguistic relationships possessing the following

Linguistic Coherence Property: Linguistic entities are reasonably well characterizable in terms of the compactly describable patterns observable in their relationship with with other linguistic entities.

Note that there is nothing intrinsically "deep" or hierarchical in this sort of linguistic coherence. However, the ability to learn the patterns relating linguistic entities with others, via a recursive hierarchical learning loop such as described above, is contingent on the presence of a fairly marked hierarchical structure in the linguistic data being studied. There is much evidence that such hierarchical structure does indeed exist in natural languages. The "deep learning" in

EFTA00624571

our approach is embedded in the repeated cycles through the loop given above – each time one goes through the loop, the learning gets one level deeper.

This sort of property has observed to hold for many linguistic entities, an observation dating back at least to Saussure [dS77] and the start of structuralist linguistics. It is basically a fancier way of saying that the meanings of words and other linguistic constructs, may be found via their *relationships* to other words and linguistic constructs. We are not committed to structuralism as a theoretical paradigm, and we have considerable respect for the aid that non-linguistic information –such as the sensorimotor data that comes from embodiment – can add to language, as should be apparent from the overall discussion in this book. However, the potential dramatic utility of non-linguistic information for language learning does not imply the impossibility or infeasibility of learning language from corpus data alone. It is inarguable that non-linguistic relationships comprise a significant portion of the everyday meaning of linguistic entities; but yet, redundancy is prevalent in natural systems, and we believe that purely linguistic relationships may well provide sufficient data for learning of natural languages. If there are some aspects of natural language that cannot be learned via corpus analysis, it seems difficult to identify what these aspects are via armchair theorizing, and likely that they will only be accurately identified via pushing corpus linguistics as far as it can go.

This generic learning process is a special case of the general process of symbolization, described in *Chaotic Logic* [Goe94] and elsewhere as a key aspect of general intelligence. In this process, a system finds patterns in itself and its environment, and then symbolizes these patterns via simple tokens or symbols that become part of the system's native knowledge representation scheme (and hence parts of its "metalanguage" for describing things to itself). Having represented a complex pattern as a simple symbolic token, it can then easily look at other patterns involving this patterns as a component.

Note that in its generic format as stated above, the "language learning loop" is not restricted to corpus based analysis, but may also include extralinguistic aspects of usage patterns, such as gestures, tones of voice, and the physical and social context of linguistic communication. Linguistic and extra-linguistic factors may come together to comprise "usage patterns." However, the restriction to corpus data does not necessarily denude the language learning loop of its power; it merely restricts one to particular classes of usage patterns, whose informativeness must be empirically determined.

In principle, one might be able to create a functional language learning system based only on a very generic implementation of the above learning loops. In practice, however, biases toward particular sorts of usage patterns can be very valuable in guiding language learning. In a computational language learning context, it may be worthwhile to break down the language learning process into multiple instances of the basic language learning loops, each focused on different sorts of usage patterns, and coupled with each other in specific ways. This is in fact what we will propose here.

Specifically, the language learning process proposed here involves:

- One language learning loop for learning purely syntactic linguistic relationships (such as link types and lexical entries, described above), which are then used to provide input to a syntax parser.
- One language learning loop for learning higher-level "syntactico-semantic" linguistic relationships (such as semantic relationships, idioms, and lexical functions, described above), which are extracted from the output of the syntax parser.

EFTA00624572

These two loops are not independent of one-another; the second loop can provide feedback to the first, regarding the correctness of the extracted structures; then as the first loop produces more correct, confident results, the second loop can in turn become more confident in it's output. In this sense, the two loops attack the same sort of slow-convergence issues that 'deep learning' tackles in neural-net training.

The syntax parser itself, in this context, is used to extract *directed acyclic graphs* (dags), usually trees, from the graph of syntactic relationships associated with a sentence. These dags represent parses of the sentence. So the overall scope of the learning process proposed here is to learn **a system of syntactic relationships that displays appropriate coherence and that, when fed into an appropriate parser, will yield parse trees that give rise to a system of syntactico-semantic relationships that displays appropriate coherence**.

45.4.2 Learning Syntax

The process of learning syntax from a corpus may be understood fairly directly in terms of entropy maximization. As a simple example, consider the measurement of the entropy of the arrangement of words in a sentence. To a fair degree, this can be approximated by the sum of the mutual entropy between pairs of words. Yuret showed that by searching for and maximizing this sum of entropies, one obtains a tree structure that closely resembles that of a dependency parse[Yur98]. That is, the word pairs with the highest mutual entropy are more or less the same as the arrows in a dependency parse, such as that shown in figure 44.1. Thus, an initial task is to create a catalog of word-pairs with a large mutual entropy (mutual information, or MI) between them. This catalog can then be used to approximate the most-likely dependency parse of a sentence, although, at this stage, the link-types are as yet unknown.

Finding dependency links using mutual information is just the first step to building a practical parser. The generation of high-MI word-pairs works well for isolating which words should be linked, but it does have several major drawbacks. First and foremost, the word-pairs do not come with any sort of classification; there is no **link type** describing the dependency relationship between two words. Secondly, most words fall into classes (*e.g.* nouns, verbs, *etc.*), but the high-MI links do not tell us what these are. A compact, efficient parser appears to require this sort of type information.

To discover syntactic link types, it is necessary to start grouping together words that appear in similar contexts. This can be done with clustering and similarity techniques, which appears to be sufficient to discover not only basic parts of speech (verbs, nouns, modifiers, determiners), but also link types. So, for example, the computation of word-pair MI is likely to reveal the following high-MI word pairs: "*big car*", "*fast car*", "*expensive car*", "*red car*". It is reasonable to group together the words *big*, *expensive*, *fast* and *red* into a single category, interpreted as modifiers to *car*. The grouping can be further refined if these same modifiers are observed with other words (*e.g.* "*big bicycle*", "*fast bicycle*", *etc.*) This has two effects: it not only reinforces the correctness of the original grouping of modifiers, but also suggests that perhaps cars and bicycles should be grouped together. Thus, one has discovered two classes of words: modifiers and nouns. In essence, one has crudely discovered parts of speech.

The link between these two classes carries a type; the type of that link is *defined* by these two classes. The use of a pair of word classes to define a link type is a basic premise of categorial grammar[KSPC13]. In this example, a link between a modifier and a noun would be a type

EFTA00624573

denoted as M\N in categorial grammar, M denoting the class of modifiers, and N the class of nouns. In the system of Link Grammar, this is replaced by a simple name, but its really one and the same thing. (In this case, the existing dictionaries use the A link for this relation, with A conjuring up 'adjective' as a mnemonic.) The simple-name is a boon for readability, as categorial grammars usually have very complex-looking link-type names: *e.g.* (NP\S)/NP for the simplest transitive verbs. Typing seems to be an inherent part of language; types must be extracted durng the learning process.

The introduction of types here has mathematical underpinnings provided by type theory. An introduction to type theory can be found in [Pro13], and an application of type theory to linguistics can be found in[KSPC13]. This is a rather abstract work, but it sheds light on the nature of link types, word-classes, parts-of-speech and the like as formal types of type theory. This is useful in dispelling the seeming taint of *ad hoc* arbitrariness of clustering: in a linguistic context, it is not so much *ad hoc* as it is a way of guaranteeing that only certain words can appear in certain positions in grammatically correct sentences, a sort of constraint that seems to be an inherent part of language, and seems to be effectively formalizable via type theory.

Word-clustering, as worked in the above example, can be viewed as another entropy-maximization technique. It is essentially a kind of factorization of dependent probabilities into most likely factors. By classifying a large number of words as 'modifiers of nouns', one is essentially admitting that they are equi-probable in that role, in the Markovian sense[Ash65] (equivalently, treating them as equally-weighted priors, in the Bayesian probability sense). That is, given the word "*car*", we should treat *big, fast, expensive* and *red* as being equi-probable (in the absence of other information). Equi-probability is an axiom in Bayesian probability (the axiom of priors), but it derives from the principle of maximum entropy (as any other probability assignment would have a lower entropy).

We have described how link types may be learned in an unsupervised setting. Connector types are then trivially assigned to the left and right words of a word-pair. The dependency graph, as obtained by linking only those word pairs with a high MI, then allows disjuncts to be easily extracted, on a sentence-by-sentence basis. At this point, another stage of pattern recognition may be applied: Given a single word, appearing in many different sentences, one should presumably find that this word only makes use of a relatively small, limited set of disjuncts. It is then a counting exercise to determine which disjuncts are occurring the most often for this word: these then form this word's lexical entry. (This "counting exercise" may also be thought of as an instance of frequent subgraph mining, as will be elaborated below.)

A second clustering step may then be applied: it's presumably noticeable that many words use more-or-less the same disjuncts in syntactic constructions. These can then be grouped into the same lexical entry. However, we previously generated a different set of word groupings (into parts of speech), and one may ask: how does that grouping compare to this grouping? Is it close, or can the groupings be refined? If the groupings cannot be harmonized, then perhaps there is a certain level of detail that was previously missed: perhaps one of the groups should be split into several parts. Conversely, perhaps one of the groupings was incomplete, and should be expanded to include more words. Thus, there is a certain back-and-forth feedback between these different learning steps, with later steps reinforcing or refining earlier steps, forcing a new revision of the later steps.

EFTA00624574

45.4.2.1 Loose language

A recognized difficulty with the direct application of Yuret's observation (that the high-MI word-pair tree is essentially identical to the dependency parse tree) is the flexibility of the preposition in the English language[KM04]. The preposition is so widely used, in such a large variety of situations and contexts, that the mutual information between it, and any other word or word-set, is rather low (is unlikely and thus carries little information). The two-point, pair-wise mutual entropy provides a poor approximation to what the English language is doing in this particular case. It appears that the situation can be rescued with the use of a three-point mutual information (a special case of interaction information [Bel03]).

The discovery and use of such constructs is described in [PD09]. A similar, related issue can be termed "the richness of the MV link type in Link Grammar". This one link type, describing verb modifiers (which includes prepositions) can be applied in a very large class of situations; as a result, discovering this link type, while at the same time limiting its deployment to only grammatical sentences, may prove to be a bit of a challenge. Even in the manually maintained Link Grammar dictionaries, it can present a parsing challenge because so many narrower cases can often be treated with an MV link. In summary, some constructions in English are so flexible that it can be difficult to discern a uniform set of rules for describing them; certainly, pair-wise mutual information seems insufficient to elucidate these cases.

Curiously, these more challenging situations occur primarily with more complex sentence constructions. Perhaps the flexibility is associated with the difficulty that humans have with composing complex sentences; short sentences are almost 'set phrases', while longer sentences can be a semi-grammatical jumble. In any case, some of the trouble might be avoided by limiting the corpus to smaller, easier sentences at first, perhaps by working with children's literature at first.

45.4.2.2 Elaboration of the Syntactic Learning Loop

We now reiterate the syntactic learning process described above in a more systematic way. By getting more concrete, we also make certain assumptions, and restrictions, some of which may end up getting changed or lifted in the course of implementation and detailed exploration of the overall approach. What is discussed in this section is merely one simple, initial approach to concretizing the core language learning loop we envision in a syntactic context.

Syntax, as we consider it here, involves the following basic entities:

- words
- categories of words
- "co-occurrence links", each one defined as (in the simplest case) an ordered pair or triple of words, labeled with an uncertain truth value
- "syntactic link types", each one defined as a certain set of ordered pairs of words
- "disjuncts", each one associated with a particular word w, and consisting of an ordered set of link types involving the word w. That is, each of these links contains at least one word-pair containing w as first or second argument. (This nomenclature here comes from Link Grammar; each disjunct is a conjunction of link types. A word is associated with a set of disjuncts. In the course of parsing, one must choose between the multiple disjuncts associated with a word, to fulfill the constraints required of an appropriate parse structure.)

EFTA00624575

An elementary version of the basic syntactic language learning loop described above would take the form.

1. Search for high-MI word pairs. Define one's usage links as the given co-occurrence links
2. Cluster words into categories based on the similarity of their associated usage links
   - Note that this will likely be a tricky instance of clustering, and classical clustering algorithms may not perform well. One interesting, less standard approach would be to use OpenCog's MOSES algorithm [Loo06, Loo07c] to learn an array of program trees, each one serving as a recognizer for a single cluster, in the same general manner done with Genetic Programming in [BE07].
3. Define initial syntactic link types from categories that are joined by large bundles of usage links
   - That is, if the words in category C_1 have a lot of usage links to the words in category C_2, then create a syntactic link type whose elements are (w_1, w_2), for all $w_1 \in C_1, w_2 \in C_2$.
4. Associate each word with an extended set of usage links, consisting of: its existing usage links, plus the syntactic links that one can infer for it based on the categories the word belongs to. One may also look at chains of (*e.g.*) 2 syntactic links originating at the word.
   - For example, suppose $cat \in C_1$ and C_1 has syntactic link L_1. Suppose (cat, eat) and (dog, run) are both in L_1. Then if there is a sentence "The cat likes to run", the link L_1 lets one infer the syntactic link cat $\stackrel{L_1}{\rightarrow}$ run. The frequency of this syntactic link in a relevant corpus may be used to assign it an uncertain truth value.
   - Given the sentence "The cat likes to run in the park," a chain of syntactic links such as cat $\stackrel{L_1}{\rightarrow}$ run $\stackrel{L_2}{\rightarrow}$ park may be constructed.
5. Return to Step 2, but using the extended set of usage links produced in Step 4, with the goal of refining both clusters and the set of link types for accuracy. Initially, all categories contain one word each, and there is a unique link type for each pair of categories. This is an inefficeint representation of language, and so the goal of clustering is to have a relatively small set of clusters and link types, with many words/word-pairs assigned to each. This can be done by maximizing the sum of the logarithms of the sizes of the clusters and link types; that is, by maximing entropy. Since the category assignments depend on the link types, and vice versa, a very large number of iterations of the loop are likely to be required. Based on the current Link Grammar English dictionaries, one expects to discover hundreds of link types (or more, depending on how subtypes are counted), and perhaps a thousand word clusters (most of these corresponding to irregular verbs and idiomatic phrases).

Many variants of this same sort of process are conceivable, and it's currently unclear what sort of variant will work best. But this *kind* of process is what one obtains when one implements the basic language learning loop described above on a purely syntactic level.

How might one integrate semantic understanding into this syntactic learning loop? Once one has semantic relationships associated with a word, one uses them to generate new "usage links" for the word, and includes these usage links in the algorithm from Step 1 onwards. This may be done in a variety of different ways, and one may give different weightings to syntactic versus semantic usage links, resulting in the learning of different links.

EFTA00624576

The above process would produce a large set of syntactic links between words. We then find a further series of steps. These may be carried out concurrently with the above steps, as soon as Step 4 has been reached for the first time.

1. This syntactic graph (with nodes as words and syntactic links joining them) may then be mined, using a variety of graph mining tools, to find common combinations of links. This gives the "disjuncts" mentioned above.
2. Given the set of disjuncts, one carries out parsing using a process such as link parsing or word grammar parsing, thus arriving at a set of parses for the sentences in one's reference corpus. Depending on the nature of one's parser, these parses may be ranked according to semantic plausibility. Each parse may be viewed as a directed acyclic graph (dag), usually a tree, with words at the nodes and syntactic-link type labels on the links.
3. One can now define new usage links for each word: namely, the syntactic links occurring in sentence parses, containing the word in question. These links may be weighted based on the weights of the parses they occur in.
4. One can now return to Step 2 using the new usage links, alongside the previous ones. Weighting these usage links relative to the others may be done in various ways.

Several subtleties have been ignored in the above, such as the proper discovery and treatment of idomatic phrases, the discovery of sentence boundaries, the handling of embedded data (price quotes, lists, chapter titles, *etc.*) as well as the potential speed bump that are prepositions. Fleshing out the details of this loop into a workable, efficient design is the primary engineering challenge. This will take significant time and effort.

45.4.3 Learning Semantics

Syntactic relationships provide only the shallowest interpretation of language; semantics comes next. One may view semantic relationships (including semantic relationships close to the syntax level, which we may call "syntactico-semantic" relationships) as ensuing from syntactic relationships, via a similar but separate learning process to the one proposed above. Just as our approach to syntax learning is heavily influenced by our work with Link Grammar, our approach to semantics is heavily influenced by our work on the RelEx system [RVG05, LGE10, GPPG06, LGK+12], which maps the output of the Link Grammar parser into a more abstract, semantic form. Prototype systems [Goe10b, LGK+12] have also been written mapping the output of RelEx into even more abstract semantic form, consistent with the semantics of the Probabilistic Logic Networks [GIGH08] formalism as implemented in CogPrime. These systems are largely based on hand-coded rules, and thus not in the spirit of language learning pursued in this proposal. However, they display the same *structure* that we assume here; the difference being that here we specify a mechanism for learning the linguistic content that fills in the structure via unsupervised corpus learning, obviating the need for hand-coding.

Specifically, we suggest that discovery of semantic relations requires the implementation of something similar to [LP01], except that this work needs to be generalized from 2-point relations to 3-point and N-point relations, roughly as described in [PD09]. This allows the automatic, unsupervised recognition of synonymous phrases, such as "Texas borders on Mexico" and "Texas is next to Mexico", to extract the general semantic relation *next_ to(X,Y)*, and the fact that this relation can be expressed in one of several different ways.

EFTA00624577

At the simplest level, in this approach, semantic learning proceeds by scanning the corpus for sentences that use similar or the same words, yet employ them in a different order, or have point substitutions of single words, or of small phrases. Sentences which are very similar, or identical, save for one word, offer up candidates for synonyms, or sometimes antonyms. Sentences which use the same words, but in seemingly different syntactic constructions, are candidates for synonymous sentences. These may be used to extract semantic relations: the recognition of sets of different syntactic constructions that carry the same meaning.

In essence, similar contexts must be recognized, and then word and word-order differences between these other-wise similar contexts must be compared. There are two primary challenges: how to recognize similar contexts, and how to assign probabilities.

The work of [PD09] articulates solutions to both challenges. For the first, it describes a general framework in which relations such as *next_to(X,Y)* can be understood as lambda-expressions $\lambda x \lambda y.\text{next_to}(x, y)$, so that one can employ first-order logic constructions in place of graphical representations. This is partly a notational trick; it just shows how to split up input syntactic constructions into atoms and terms, for which probabilities can be assigned. For the second challenge, they show how probabilities can be assigned to these expressions, by making explicit use of the notions of conditional random fields (or rather, a certain special case, termed Markov Logic Networks). Conditional random fields, or Markov networks, are a certain mathematical formalism that provides the most general framework in which entropy maximization problems can be solved: roughly speaking, it can be understood as a means of properly distributing probabilities across networks. Unfortunately, this work is quite abstract and rather dense. Thus, a much easier understanding to the general idea can be obtained from [LP01]; unfortunately, the later fails to provide the general N-point case needed for semantic relations in general, and also fails to consider the use of maximum entropy principles to obtain similarity measures.

The above can be used to extract synonymous constructions, and, in this way, semantic relations. However, neither of the above references deal with distinguishing different meanings for a given word. That is, while *eats(X,Y)* might be a learnable semantic relation, the sentence "*He ate it*" does not necessarily justify its use. Of course: "*He ate it*" is an idiomatic expression meaning "*he crashed*", which should be associated with the semantic relation *crash*(X), not *eat(X,Y)*. There are global textual clues that this may be the case: trouble resolving the reference "*it*", and a lack of mention of foodstuffs in neighboring sentences. A viable yet simple algorithm for the disambiguation of meaning is offered by the Mihalcea algorithm[MTF04, Mih05, SM07]. This is an application of the (google) PageRank algorithm to word senses, taken across words appearing in multiple sentences. The premise is that the correct word-sense is the one that is most strongly supported by senses of nearby words; a graph between word senses is drawn, and then solved as a Markov chain. In the original formulation, word senses are defined by appealing to WordNet, and affinity between word-senses is obtained via one of several similarity measures. Neither of these can be applied in learning a language *de novo*. Instead, these must both be deduced by clustering and splitting, again. So, for example, it is known that word senses correlate fairly strongly with disjuncts (based on authors unpublished experiments), and thus, a reasonable first cut is to presume that every different disjunct in a lexical entry conveys a different meaning, until proved otherwise. The above-described discovery of synonymous phrases can then be used to group different disjuncts into a single "word sense". Disjuncts that remain ungrouped after this process are already considered to have distinct senses, and so can be used as distinct senses in the Mihalcea network.

EFTA00624578

Sense similarity measures can then be developed by using the above-discovered senses, and measuring how well they correlate across different texts. That is, if the word "*bell*" occurs multiple times in a sequence of paragraphs, it is reasonable to assume that each of these occurrences are associated with the same meaning. Thus, each distinct disjunct for the word "*bell*" can then be presumed to still convey the same sense. One now asks, what words co-occur with the word "*bell*"? The frequent appearance of "*chime*" and "*ring*" can and should be noted. In essence, one is once-again computing word-pair mutual information, except that now, instead of limiting word-pairs to be words that are near each other, they can instead involve far-away words, several sentences apart. One can then expand the word sense of "*bell*" to include a list of co-occurring words (and indeed, this is the slippery slope leading to set phrases and eventually idioms).

Failures of co-occurrences can also further strengthen distinct meanings. Consider "*he chimed in*" and "*the bell chimed*". In both cases, *chime* is a verb. In the first sentence, *chime* carries the disjunct `S- & K+` (here, `K+` is the standard Link Grammar connector to particles) while the second has only the simpler disjunct `S-`. Thus, based on disjunct usage alone, one already suspects that these two have a different meaning. This is strengthened by the lack of occurrence of words such as "*bell*" or "*ring*" in the first case, with a frequent observation of words pertaining to talking.

There is one final trick that must be applied in order to get reasonably rapid learning; this can be loosely thought of as "the sigmoid function trick of neural networks", though it may also be manifested in other ways not utilizing specific neural net mathematics. The key point is that semantics intrinsically involves a variety of uncertain, probabilistic and fuzzy relationships; but in order to learn a robust hierarchy of semantic structures, one needs to iteratively crispen these fuzzy relationships into strict ones.

In much of the above, there is a recurring need to categorize, classify and discover similarity. The naivest means of doing so is by counting, and applying basic probability (Bayesian, Markovian) to the resulting counts to deduce likelihoods. Unfortunately, such formulas distribute probabilities in essentially linear ways (*i.e.* form a linear algebra), and thus have a rather poor ability to discriminate or distinguish (in the sense of receiver operating characteristics, of discriminating signal from noise). Consider the last example: the list of words co-occurring with *chime*, over the space of a few paragraphs, is likely to be tremendous. Most of this is surely noise. There is a trick to over-coming this that is deeply embedded in the theory of neural networks, and yet completely ignored in probabilistic (Bayesian, Markovian) networks: the sigmoid function. The sigmoid function serves to focus in on a single stimulus, and elevate its importance, and, at the same time, strongly suppress all other stimuli. In essence, the sigmoid function looks at two probabilities, say 0.55 and 0.45, and says "let's pretend the first one is 0.9 and the second one is 0.1, and move forward from there". It builds in a strong discrimination to all inputs. In the language of standard, text-book probability theory, such discrimination is utterly unwarranted; and indeed, it is. However, applying strong discrimination to learning can help speed learning by converting certain vague impressions into certainties. These certainties can then be built upon to obtain additional certainties, or to be torn apart, as needed.

Thus, in all of the above efforts to gauge the similarity between different things, it is useful to have a sharp yes/no answer, rather than a vague muddling with likelihoods. In some of the above-described algorithms, this sharpness is already built in: so, Yuret approximates the mutual information of an entire sentence as the sum of mutual information between word pairs: the smaller, unlikely corrections are discarded. Clearly, they must also be revived in order to handle prepositions. Something similar must also be done in the extraction of synonymous

EFTA00624579

phrases, semantic relations, and meaning; the domain is that much likelier to be noisy, and thus, the need to discriminate signal from noise that much more important.

45.4.3.1 Elaboration of the Semantic Learning Loop

We now provide a more detailed elaboration of a simple version of the general semantic learning process described above. The same caveat applies here as in our elaborated description of syntactic learning above: the specific algorithmic approach outlined here is a simple instantiation of the general approach we have in mind, which may well require refinement based on lessons learned during experimentation and further theoretical analysis.

One way to do semantic learning, according to the approach outlined above, is as follows:

1. An initial semantic corpus is posited, whose elements are parse graphs produced by the syntactic process described earlier
2. A semantic relationship set (or **rel-set**) is computed from the semantic corpus, via calculating the frequent (or otherwise statistically informative) subgraphs occurring in the elements of the corpus. Each node of such a subgraph may contain a word, a category or a variable; the links of the subgraph are labeled with (syntactic, or semantic) link types. Each parse graph is annotated with the semantic graphs associated with the words it contains (explicitly: each word in a parse graph may be linked via a ReferenceLink to each variable or literal with a semantic graph that corresponds to that word in the context of the sentence underlying the parse graph.)
   - For instance, the link combination $v_1 \stackrel{S}{\rightarrow} v_2 \stackrel{O}{\rightarrow} v_3$ may commonly occur (representing the standard Subject-Verb-Object (SVO) structure)
   - In this case, for the sentence "The rock broke the window," we would have links such as rock $\stackrel{ReferenceLink}{\rightarrow}$ v_1 connecting the nodes (such as the "rock" node) in the parse structure with nodes (such as v_1) in this associated semantic subgraph.
3. Rel-sets are divided into categories based on the similarities of their associated semantic graphs.
   - This division into categories manifests the sigmoid-function-style crispening mentioned above. Each rel-set will have similarities to other rel-sets, to varying fuzzy degrees. Defining specific categories turns a fuzzy web of similarities into crisp categorial boundaries; which involves some loss of information, but also creates a simpler platform for further steps of learning.
   - Two semantic graphs may be called "associated" if they have a nonempty intersection. The intersection determines the type of association involved. Similarity assessment between graphs G and H may involve estimation of which graphs G and H are associated with in which ways.
   - For instance, "The cat ate the dog" and "The frog was eaten by the walrus" represent the semantic structure *eat(cat,dog)* in two different ways. In link parser terminology, they do so respectively via the subgraphs $\mathcal{G}_1 = v_1 \stackrel{S}{\rightarrow} v_2 \stackrel{O}{\rightarrow} v_3$ and $\mathcal{G}_2 = v_1 \stackrel{S}{\rightarrow} v_2 \stackrel{P}{\rightarrow} v_3 \stackrel{MV}{\rightarrow} v_4 \stackrel{J}{\rightarrow} v_5$. These two semantic graphs will have a lot of the same associations. For instance, in our corpus we may have "The big cat ate the dog in the morning" (including big $\stackrel{A}{\rightarrow}$ cat) and also "The big frog was eaten by the walrus in the morning"

EFTA00624580

(including big $\stackrel{A}{\rightarrow}$ frog), meaning that big $\stackrel{A}{\rightarrow}$ v_5 is a graph commonly associated with both $\mathcal{G}_1$ and $\mathcal{G}_2$. Due to having many commonly associated graphs like this, $\mathcal{G}_1$ and $\mathcal{G}_2$ are likely to be associated to a common cluster.

4. Nodes referring to these categories are added to the parse graphs in the semantic corpus. Most simply, a category node C is assigned a link of type L pointing to another node x, if any element of C has a link of type L pointing to x. (More sophisticated methods of assigning links to category nodes may also be worth exploring.)
   - If $\mathcal{G}_1$ and $\mathcal{G}_2$ have been assigned to a common category C, then "I believe the pig ate the horse" and "I believe the law was invalidated by the revolution" will both appear as instantiations of the graph $\mathcal{G}_3 = v_1 \stackrel{S}{\rightarrow}$ believe $\stackrel{CV}{\rightarrow}$ C. This $\mathcal{G}_3$ is compact because of the recognition of C as a cluster, leading to its representation as a single symbol. The recognition of $\mathcal{G}_3$ will occur in Step 2 the next time around the learning loop.
5. Return to Step 2, with the newly enriched semantic corpus. As before, one wants to discover not too many and not too few categories; again, the appropriate solution to this problem appears to be entropy maximization. That is, during the frequent subgraph mining stages, one maintains counts of how often these occur in the corpus; from these, one constructs the equivalent of the mutual information associated with the subgraphs; categorization requires maximizing the sum of the log of the sizes of the categories.

As noted earlier, these semantic relationships may be used in the syntactic phase of language understanding in two ways:

- Semantic graphs associated with words may be considered as "usage links" and thus included as part of the data used for syntactic category formation.
- During the parsing process, full or partial parses leading to higher-probability semantic graphs may be favored.

45.5 The Importance of Incremental Learning

The learning process described here builds up complex syntactic and semantic structures from simpler ones. To start it, all one needs are basic before and after relationships derived from a corpus. Everything else is built up from there, given the assumption of appropriate syntactic and semantic formalisms and a semantics-guided syntax parser.

As we have noted, the series of learning steps we propose falls into the broad category of "deep learning", or of hierarchical modeling. That is, learning must occur at several levels at once, each reinforcing, and making use of results from another. Link types cannot be identified until word clusters are found, and word clusters cannot be found until word-pair relationships are discovered. However, once link-types are known, these can be then used to refine clusters and the selected word-pair relations. Further, the process of finding word clusters – both pre and post parsing – relies on a hierarchical build-up of clusters, each phase of clustering utilizing results of the previous "lower level" phrase.

However, for this bootstrapping learning to work well, one will likely need to begin with simple language, so that the semantic relationships embodied in the text are not that far removed from the simple before/after relationships. The complexity of the texts may then be

EFTA00624581

ramped up gradually. For instance, the needed effect might be achieved via sorting a very large corpus in order of increasing reading level.

45.6 Integrating Language Learned via Corpus Analysis into CogPrime's Experiential Learning

Supposing everything in this chapter were implemented and tested and worked reasonably well as envisioned. What would this get us in terms of progress toward AGI?

Arguably, with a relatively modest additional effort, it could get us a natural language question answering system, answering a variety of questions based on the text corpus available to it. One would have to use the learned rules for language generation, but the methods of Chapter 46 would likely suffice for that.

Such a dialogue system would be a valuable achievement in its own right, of scientific, commercial and humanistic interest – but of course, it wouldn't be AGI. To get something approaching AGI from this sort of effort, one would have to utilize additional reasoning and concept creation algorithms to enable the answering of questions based on knowledge not stored explicitly in the provided corpus. The dialogue system would have to be able to piece together new answers from various fragmentary, perhaps contradictory pieces of information contain in the corpus. Ultimately, we suspect, one would need something like the CogPrime architecture, or something else with a comparable level of sophistication, to appropriately leverage the information extracted from texts via the learned language rules.

An open question, as indicated above, is how much of *language* itself would a corpus based language learning system like the one outlined here miss, assuming a massive but realistic corpus (say, a significant fraction of the Web). This is unresolved and ultimately will only be determined via experiment. Our suspicion is that a very large percentage of language can be understood via these corpus-based methods. But there may be exceptions that would require an unrealistically large corpus size.

As a simple example, consider the ability to interpret vaguely given spatial directions like "Go right out the door, past a few curves in the road, then when you get to a hill with a big red house on it (well not that big, but bigger than most of the others you'll see on the walk), start heading down toward the water, till the brush gets thick, then start heading left.... Follow the ground as it rises and eventually you'll see the lake." Of course, it is theoretically possible for an AGI system to learn to interpret directions like this purely via corpus analysis. But it seems the task would be a lot easier for an AGI endowed with a body so that it could actually experience routes like the one being described. And space and time are not the only source of relevant examples; social and emotional reasoning have a similar property. Learning to interpret language about these from reading is certainly possible, but one will have an easier time and do a better job if one is out in the world experiencing social and emotional life oneself.

Even if there turn out to be significant limitations regarding what can be learned in practice about language via corpus analysis, though, it may still prove a valuable contributor to the mind of a CogPrime system. As compared to hand-coded rules, comparably abstract linguistic knowledge achieved via statistical corpus analysis should be much easier to integrate with the results of probabilistic inference and embodied learning, due to its probabilistic weighting and its connection with the specific examples that gave rise to it.

EFTA00624582

Chapter 46
Natural Language Generation

Co-authored with Ruiting Lian and Rui Liu

46.1 Introduction

Language generation, unsurprisingly, shares most of the key features of language comprehension discussed in chapter 44 – after all, the division between generation and comprehension is to some extent an artificial convention, and the two functions are intimately bound up both in the human mind and in the CogPrime architecture.

In this chapter we discuss language generation, in a manner similar to the previous chapter's treatment of language comprehension. First we discuss our currently implemented, "engineered" language generation system, and then we discuss some alternative approaches:

- how a more experiential-learning based system might be made by retaining the basic structure of the engineered system but removing the "pre-wired" contents.
- how a "Sem2Syn" system might be made, via reversing the Syn2Sem system described in Chapter 44. This is the subject of implementation effort, at time of writing.

At the start of Chapter 44 we gave a high-level overview of a typical NL generation pipeline. Here we will focus largely but not entirely on the "syntactic and morphological realization" stage, which we refer to for simplicity as "sentence generation" (taking a slight terminological liberty, as "sentence fragment generation" is also included here). All of the stages of language generation are important, and there is a nontrivial amount of feedback among them. However, there is also a significant amount of autonomy, such that it often makes sense to analyze each one separately and then tease out its interactions with the other stages.

46.2 SegSim for Sentence Generation

The sentence generation approach currently taken in OpenCog (from 2009 - early 2012), which we call SegSim, is relatively simple and is depicted in Figure 46.1 and described as follows:

1. The NL generation system stores a large set of pairs of the form (semantic structure, syntactic/morphological realization)
2. When it is given a new semantic structure to express, it first breaks this semantic structure into natural parts, using a set of simple syntactic-semantic rules

EFTA00624584

3. For each of these parts, it then matches the parts against its memory to find relevant pairs (which may be full or partial matches), and uses these pairs to generate a set of syntactic realizations (which may be sentences or sentence fragments)
4. If the matching has failed, then (a) it returns to Step 2 and carries out the breakdown into parts again. But if this has happened too many times, then (b) it recourses to a different algorithm (most likely a search or optimization based approach, which is more computationally costly) to determine the syntactic realization of the part in question.
5. If the above step generated multiple fragments, they are pieced together, and a certain rating function is used to judge if this has been done adequately (using criteria of grammaticality and expected comprehensibility, among others). If this fails, then Step 3 is tried again on one or more of the parts; or Step 2 is tried again. (Note that one option for piecing the fragments together is to string together a number of different sentences; but this may not be judged optimal by the rating function.)
6. Finally, a "cleanup" phase is conducted, in which correct morphological forms are inserted, and articles and certain other "function words" are inserted.

The specific OpenCog software implementing the SegSim algorithm is called "NLGen"; this is an implementation of the SegSim concept that focuses on sentence generation from RelEx semantic relationship. In the current (early 2012) NLGen version, Step 1 is handled in a very simple way using a relational database; but this will be modified in future so as to properly use the AtomSpace. Work is currently underway to replace NLGen with a different "Sem2Syn" approach, that will be described at the end of this chapter. But discussion of NLGen is still instructive regarding the intersection of language generation concepts with OpenCog concepts.

The substructure currently used in Step 2 is defined by the predicates of the sentence, i.e. we define one substructure for each predicate, which can be described as follows:

$$Predicate(Argument_i(Modify_j))$$

where

- $1 \leq i \leq m$ and $0 \leq j \leq n$ and m, n are integers
- "Predicate" stands for the predicate of the sentence, corresponding to the variable $0 of the RelEx relationship _subj($0, $1) or _obj($0, $1)
- $Argument_i$ is the i-th semantic parameter related with the predicate
- $Modify_j$ is the j-th modifier of the $Argument_i$

If there is more than one predicate, then multiple subnets are extracted analogously.

For instance, given the sentence "I happily study beautiful mathematics in beautiful China with beautiful people." The substructure can be defined as Figure 46.2.

For each of these substructures, Step 3 is supposed to match the substructures of a sentence against its global memory (which contains a large body of previously encountered [semantic structure, syntactic/morphological realization] pairs) to find the most similar or same substructures and the relevant syntactic relations to generate a set of syntactic realizations, which may be sentences or sentence fragments. In our current implementation, a customized subgraph matching algorithm has been used to match the subnets from the parsed corpus at this step.

If Step 3 generated multiple fragments, they must be pieced together. In Step 4, the Link Parser's dictionary has been used for detecting the dangling syntactic links corresponding to the fragments, which can be used to integrate the multiple fragments. For instance, in the example of Figure 46.3, according to the last 3 steps, SegSim would generate two fragments: "the parser

EFTA00624585

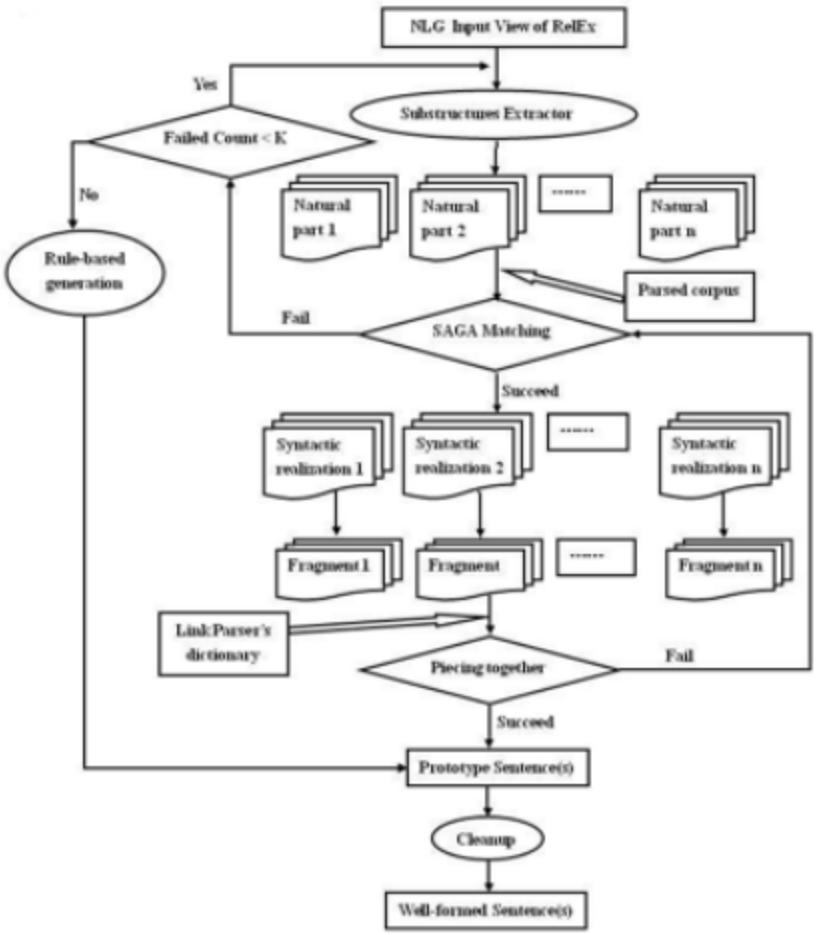


Fig. 46.1: A Overview of the SegSim Architecture for Language Generation

will ignore the sentence" and "whose length is too long". Then it consults the Link Parser's dictionary, and finds that "whose" has a connector "Mr-", which is used for relative clauses involving "whose", to connect to the previous noun "sentence". Analogously, we can integrate the other fragments into a whole sentence.

For instance, in the example of Figure 46.3, according to the last 3 steps, SegSim would generate two fragments: "the parser will ignore the sentence" and "whose length is too long". Then it consults the Link Parser's dictionary, and finds that "whose" has a connector "Mr-", which is used for relative clauses involving "whose", to connect to the previous noun "sentence". Analogously, we can integrate the other fragments into a whole sentence.

Finally, a "cleanup" or "post-processing" phase is conducted, applying the correct inflections to each word depending on the word properties provided by the input RelEx relations. For example, we can use the RelEx relation "DEFINITE-FLAG(cover, T)" to insert the article "the" in front of the word "cover". We have considered five factors in this version of NLGen: article,

EFTA00624586

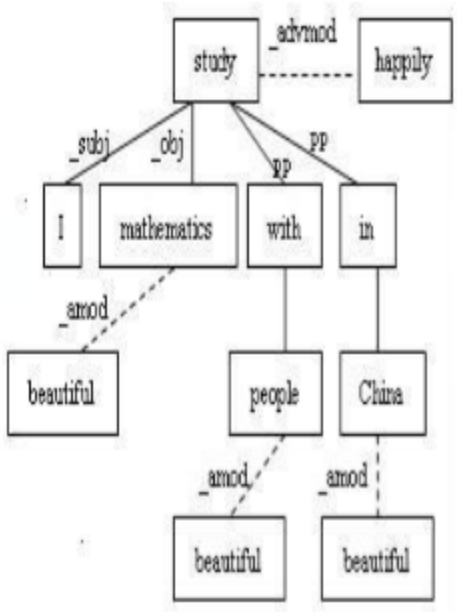


Fig. 46.2: Example of a substructure

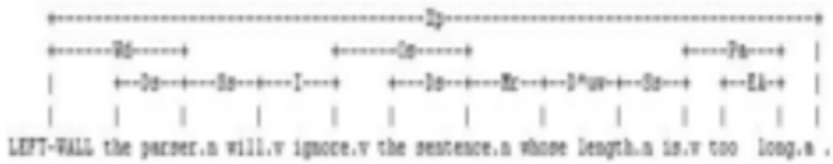


Fig. 46.3: Linkage of an example

noun plural, verb intense, possessive and query type (the latter which is only for interrogative sentences).

In the "cleanup" step, we also use the chunk parser tool from OpenNLP* for adjusting the position of an article being inserted. For instance, consider the proto-sentence "I have big red apple." If we use the RelEx relation "noun_number(apple, singular)" to inflect the word "apple" directly, the final sentence will be "I have big red an apple", which is not well-formed. So we use the chunk parser to detect the phrase "big red apple" first, then apply the article rule in front of the noun phrase. This is a pragmatic approach which may be replaced with something more elegant and principled in later revisions of the NLGen system.

* http://opennlp.sourceforge.net/

EFTA00624587

46.2.1 NLGen: Example Results

NLGen is currently in a relatively early stage of development, and does not handle the full range of linguistic and semantic phenomena that it will when it's completed. However, it can already express a variety of sentences encapsulating a variety of syntactic and semantic phenomena; in this section we will give some specific examples of what it can do.

The SegSim approach performs sentence generation by matching portions of propositional input to a large corpus of parsed sentences, therefore, when the successful matches can be found in the corpus, it can generate some similar and well-formed sentences via the relevant syntactic structures.

There currently do not exist any authoritative evaluation criteria for Natural Language Generation systems. Among many complicating factors here is the fact that different language generation systems have different kinds of inputs, depending on many things including their application area. So it's complicated to compare the results of NLGen with those obtained by other systems.

It is easier however to test whether NLGen is implementing SegSim successfully. One approach is to take a sentence, run it through RelEx to generate a set of relationships, and see if NLGen can regenerate the sentence from the relationship. We show here the results of some simple tests of this nature, performed with the current version of the system using a very small test corpus for similarity matching. Note: In each of these example results, the input of NLGen is the RelEx relationships produced by the sentence before "==>"; and the sentence to the right side of "==>" was one of the sentences generated by NLGen.

Example 1:

```
I study mathematics at Xiamen University.
==> I study mathematics at Xiamen\_University.
```

Example 2:

```
The red block is next to the blue cup.
==> The red block is next\_to the blue cup.
```

Example 3:

```
Ruiting walked from the table to the door.
==> Ruiting walked to the door from the table.
```

Example 4:

```
I intend to walk very slowly from Ben to Ruiting,
with Hugo's pencil in my hand.
==> With Hugo's pencil in my hand,
I intend to walk very slowly from Ben to Ruiting.
```

Example 5:

```
I'd like you to tell me where the red ball is.
==> I tell like you to 'd me where the red ball is.
```

Example 6:

EFTA00624588

```
I study at Xiamen University, which is located in China.
==> I study at Xiamen\_University.
```

Example 7:

```
What is next to the tree?
 ==> What is next to the tree?
```

Example 8:

```
Where  is the red ball?
==> Where is the red ball?
```

Example 9:

```
Pass me the ball.
 ==> Pass me the ball.
```

Example 10:

```
Tell Ben where the red ball is.
 ==> Tell Ben where the red ball is.
```

To make the process clearer, for Example 1 we also show the RelEx relationships produced from the sentence before the "==>":

For instance, in the example of Figure 46.3, according to the last 3 steps, SegSim would generate two fragments: "the parser will ignore the sentence" and "whose length is too long". Then it consults the Link Parser's dictionary, and finds that "whose" has a connector "Mr-", which is used for relative clauses involving "whose", to connect to the previous noun "sentence". Analogously, we can integrate the other fragments into a whole sentence.

Figure 46.4 shows the relationships of Example 1 fed to NLGen as input. The types of the semantic relationships are documented in the RelEx's wiki pages[†].

These examples illustrate some key points about the current version of NLGen. It works well on simple, commonplace sentences (Example 1, 2), though it may reorder the sentence fragments sometimes (Example 3, 4). On the other hand, because of its reliance on matching against a corpus, NLGen is incapable of forming good sentences with syntactic structures not found in the corpus (Example 5, 6). On a larger corpus these examples would have given successful results. In Example 5, the odd error is due to the presence of too many "_subj" RelEx relationships in the relationship-set corresponding to the sentence, which distracts the matching process when it attempts to find similar substructures in the small test corpus. Then from Example 7 to 10, we can see NLGen still works well for question sentences and imperative sentence if the substructures we extract can be matched, but the substructures may be similar with the assertive sentence, so we need to refine it in the "cleanup" step. For example: the substructures we extracted for the sentence "are you a student?" are the same as the ones for "you are a student?", since the two sentences both have the same binary RelEx relationships:

```
_subj(be, you)
_obj(be, student)
```

[†] http://opencog.org/wiki/RelEx#Relations_and_Features

EFTA00624589

```
pos(., punctuation)
noun_number(Xiamen_University, singular)
pos(Xiamen_University, noun)
DEFINITE-FLAG(Xiamen_University, T)
pos(at, prep)
_subj(study, I)
at(study, Xiamen_University)
tense(study, present)
pos(study, verb)
gender(I, person)
noun_number(I, singular)
pos(I, noun)
PRONOUN-FLAG(I, T)
DEFINITE-FLAG(I, T)
```

Fig. 46.4: RelEx relationships for Example 1

which are used to guide the extraction of the substructures. So we need to refine the sentence via some grammatical rules in the post-processing phase, using the word properties from RelEx,

EFTA00624590

like "TRUTH-QUERY-FLAG(be, T)" which means if that the referent "be" is a verb/event and the event is involved is a question.

The particular shortcomings demonstrated in these examples are simple to remedy within the current NLGen framework, via simply expanding the corpus. However, to get truly general behavior from NLGen it will be necessary to insert some other generation method to cover those cases where similarity matching fails, as discussed above. The NLGen2 system created by Blake Lemoine [Lem10] is one possibility in this regard: based on RelEx and the link parser, it carries out rule-based generation using an implementation of Chomsky's Merge operator. Integration of NLGen with NLGen2 is currently being considered. We note that the Merge operator is computationally inefficient by nature, so that it will likely never be suitable for the primary sentence generation method in a language generation system. However, pairing NLGen for generation of familiar and routine utterances with a Merge-based approach for generation of complex or unfamiliar utterances, may prove a robust approach.

46.3 Experiential Learning of Language Generation

As in the case of language comprehension, there are multiple ways to create an experiential learning based language generation system, involving various levels of "wired in" knowledge. Our best guess is that for generation as for comprehension, a "tabula rasa" approach will prove computationally intractable for quite some time to come, and an approach in which some basic structures and processes are provided, and then filled out with content learned via experience, will provide the greatest odds of success.

A highly abstracted version of SegSim may be formulated as follows:

1. The AI system stores semantic and syntactic structures, and its control mechanism is biased to search for, and remember, linkages between them
2. When it is given a new semantic structure to express, it first breaks this semantic structure into natural parts, using inference based on whatever implications it has in its memory that will serve this purpose
3. Its inference control mechanism is biased to carry out inferences with the following implication: For each of these parts, match it against its memory to find relevant pairs (which may be full or partial matches), and use these pairs to generate a set of syntactic realizations (which may be sentences or sentence fragments)
4. If the matching has failed to yield results with sufficient confidence, then (a) it returns to Step 2 and carries out the breakdown into parts again. But if this has happened too many times, then (b) it uses its ordinary inference control routine to try to determine the syntactic realization of the part in question.
5. If the above step generated multiple fragments, they are pieced together, and an attempt is made to infer, based on experience, whether the result will be effectively communicative. If this fails, then Step 3 is tried again on one or more of the parts; or Step 2 is tried again.
6. Other inference-driven transformations may occur at any step of the process, but are particularly likely to occur at the end. In some languages these transformations may result in the insertion of correct morphological forms or other "function words."

What we suggest is that it may be interesting to supply a CogPrime system with this overall process, and let it fill in the rest by experiential adaptation. In the case that the system is

EFTA00624591

learning to comprehend at the same time as it's learning to generate, this means that its early-stage generations will be based on its rough, early-stage comprehension of syntax – but that's OK. Comprehension and generation will then "grow up" together.

46.4 Sem2Syn

A subject of current research is the extension of the Syn2Sem approach mentioned above into a reverse-order, Sem2Syn system for language generation.

Given that the Syn2Sem rules are expressed as ImplicationLinks, they can be reversed automatically and immediately – although, the reversed versions will not necessarily have the same truth values. So if a collection of Syn2Sem rules are learned from a corpus, then they can be used to automatically generate a set of Sem2Syn rules, each tagged with a probabilistic truth value. Application of the whole set of Sem2Syn rules to a given Atom-set in need of articulation, will result in a collection of link-parse links.

To produce a sentence from such a collection of link-parse links, another process is also needed, which will select a subset of the collection that corresponds to a complete sentence, legally parsable via the link parser. The overall collection might naturally break down into more than one sentence.

In terms of the abstracted version of SegSim given above, the primary difference between NLGen and SegSim lies in Step 3. Syn2Sem replaces the SegSim "data-store matching" algorithm with inference based on implications obtained from reversing the implications used for language comprehension.

46.5 Conclusion

There are many different ways to do language generation within OpenCog, ranging from pure experiential learning to a database-driven approach like NLGen. Each of these different ways may have value for certain applications, and it's unclear which ones may be viable in a human-level AGI context. Conceptually we would favor a pure experiential learning approach, but, we are currently exploring a "compromise" approach based on Sem2Syn. This is an area where experimentation is going to tell us more than abstract theory.

EFTA00624592

Chapter 47
Embodied Language Processing

Co-authored with Samir Araujo and Welter Silva

47.1 Introduction

"Language" is an important abstraction – but one should never forget that it's an abstraction. Language evolved in the context of embodied action, and even the most abstract language is full of words and phrases referring to embodied experience. Even our mathematics is heavily based on our embodied experience – geometry is about space; calculus is about space and time; algebra is a sort of linguistic manipulation generalized from experience-oriented language, etc. (see [LN00] for detailed arguments in this regard). To consider language in the context of human-like general intelligence, one needs to consider it in the context of embodied experience.

There is a large literature on the importance of embodiment for child language learning, but perhaps the most eloquent case has been made by Michael Tomasello, in his excellent book *Constructing a Language* **??**. Citing a host of relevant research by himself and others, Tomasello gives a very clear summary of the value of social interaction and embodiment for language learning in human children. And while he doesn't phrase it in these terms, the picture he portrays includes central roles for reinforcement, imitative and corrective learning. Imitative learning is obvious: so much of embodied language learning has to do with the learner copying what it has heard other say in similar contexts. Corrective learning occurs every time a parent or peer rephrases something for a child.

In this chapter, after some theoretical discussion of the nature of symbolism and the role of gesture and sound in language, we describe some computational experiments run with OpenCog controlling virtual pets in a virtual world, regarding the use of embodied experience for anaphor resolution and question-answering. These comprise an extremely simplistic example of the interplay between language and embodiment, but have the advantage of concreteness, since they were actually implemented and experimented with. Some of the specific OpenCog tools used in these experiments are no longer current (e.g. the use of RelEx2Frame, which is now deprecated in favor of alternative approaches to mapping parses into more abstract semantic relationships); but the basic principles and flow illustrated here are still relevant to current and future work.

EFTA00624594

47.2 Semiosis

The foundation of communication is semiosis – the representation between the signifier and the signified. Often the signified has to do with the external world or the communicating agent's body; hence the critical role of embodiment in language.

Thus, before turning to the topic of embodied *language* use and learning per se, we will briefly treat the related topic of how an AGI system may learn *semiosis* itself via its embodied experience. This is a large and rich topic, but we will restrict ourselves to giving a few relatively simple examples intended to make the principles clear. We will structure our discussion of semiotic learning according to Charles Sanders Peirce's theory of semiosis [Pei34], in which there are three basic types of signs: icons, indices and symbols.

In Peirce's ontology of semiosis, an icon is a sign that physically resembles what it stands for. Representational pictures, for example, are icons because they look like the thing they represent. Onomatopoeic words are icons, as they sound like the object or fact they signify. The iconicity of an icon need not be immediate to appreciate. The fact that "kirikiriki" is iconic for a rooster's crow is not obvious to English-speakers yet it is to many Spanish-speakers; and the the converse is true for "cock-a-doodle-doo."

Next, an index is a sign whose occurrence probabilistically implies the occurrence of some other event or object (for reasons other than the habitual usage of the sign in connection with the event or object among some community of communicating agents). The index can be the cause of the signified thing, or its consequence, or merely be correlated to it. For example, a smile on your face is an index of your happy state of mind. Loud music and the sound of many people moving and talking in a room is an index for a party in the room. On the whole, more contextual background knowledge is required to appreciate an index than an icon.

Finally, any sign that is not an icon or index is a symbol. More explicitly, one may say that a symbol is a sign whose relation to the signified thing is conventional or arbitrary. For instance, the stop sign is a symbol for the imperative to stop; the word "dog" is a symbol for the concept it refers to.

The distinction between the various types of signs is not always obvious, and some signs may have multiple aspects. For instance, the thumbs-up gesture is a symbol for positive emotion or encouragement. It is not an index – unlike a smile which is an index for happiness because smiling is intrinsically biologically tied to happiness, there is no intrinsic connection between the thumbs-up signal and positive emotion or encouragement. On the other hand, one might argue that the thumbs-up signal is very weakly iconic, in that its up-ness resembles the subjective up-ness of a positive emotion (note that in English an idiom for happiness is "feeling up").

Teaching an embodied virtual agent to recognize simple icons is a relatively straightforward learning task. For instance, suppose one wanted to teach an agent that in order to get the teacher to give it a certain type of object, it should go to a box full of pictures and select a picture of an object of that type, and bring it to the teacher. One way this may occur in an OpenCog-controlled agent is for the agent to learn a rule of the following form:

EFTA00624595

```
ImplicationLink
    ANDLink
        ContextLink
             Visual
            SimilarityLink $X $Y
    PredictiveImplicationLink
        SequentialANDLink
            ExecutionLink goto box
            ExecutionLink grab $X
            ExecutionLink goto teacher
        EvaluationLink give me teacher $Y
```

While not a trivial learning problem, this is straightforward to a CogPrime -controlled agent that is primed to consider visual similarities as significant (i.e. is primed to consider the visual-appearance context within its search for patterns in its experience).

Next, proceeding from icons to indices: Suppose one wanted to teach an agent that in order to get the teacher to give it a certain type of object, it should go to a box full of pictures and select a picture of an object that has commonly been used together with objects of that type, and bring it to the teacher. This is a combination of iconic and indexical semiosis, and would be achieved via the agent learning a rule of the form

```
Implication
   AND
      Context
          Visual
          Similarity $X $Z
       Context
           Experience
           SpatioTemporalAssociation $Z $Y
       PredictiveImplication
           SequentialAnd
              Execution goto box
              Execution grab $X
              Execution goto teacher
           Evaluation give me teacher $Y
```

Symbolism, finally, may be seen to emerge as a fairly straightforward extension of indexing. After all, how does an agent come to learn that a certain symbol refers to a certain entity? An advanced linguistic agent can learn this via explicit verbal instruction, e.g. one may tell it "The word 'hideous' means 'very ugly'." But in the early stages of language learning, this sort of instructional device is not available, and so the way an agent learns that a word is associated with an object or an action is through spatiotemporal association. For instance, suppose the teacher wants to teach the agent to dance every time the teacher says the word "dance" – a very simple example of symbolism. Assuming the agent already knows how to dance, this merely requires the agent learn the implication

```
PredictiveImplication
   SequentialAND
      Evaluation say teacher me "dance"
      Execution dance
```

EFTA00624596

```
give teacher me Reward
```

And, once this has been learned, then simultaneously the relationship

```
SpatioTemporalAssociation dance "dance"
```

will be learned. What's interesting is what happens after a number of associations of this nature have been learned. Then, the system may infer a general rule of the form

```
Implication
   AND
      SpatioTemporalAssociation \$X \$Z
      HasType \$X GroundedSchema
   PredictiveImplication
      SequentialAND
         Evaluation say teacher me \$Z
         Execution \$X
      Evaluation give teacher me Reward
```

This implication represents the general rule that if the teacher says a word corresponding to an action the agent knows how to do, and the agent does it, then the agent may get a reward from the teacher. Abstracting this from a number of pertinent examples is a relatively straightforward feat of probabilistic inference for the PLN inference engine.

Of course, the above implication is overly simplistic, and would lead an agent to stupidly start walking every time its teacher used the word "walk" in conversation and the agent overheard it. To be useful in a realistic social context, the implication must be made more complex so as to include some of the pragmatic surround in which the teacher utters the word or phrase $Z.

47.3 Teaching Gestural Communication

Based on the ideas described above, it is relatively straightforward to teach virtually embodied agents the elements of gestural comunication. This is important for two reasons: gestural communication is extremely useful unto itself, as one sees from its role in communication among young children and primates [22]; and, gestural communication forms a foundation for verbal communication, during the typical course of human language learning [23]. Note for instance the study described in [22], which "reports empirical longitudinal data on the early stages of language development," concluding that

...the output systems of speech and gesture may draw on underlying brain mechanisms common to both language and motor functions. We analyze the spontaneous interaction with their parents of three typically-developing children (2 M, 1 F) videotaped monthly at home between 10 and 23 months of age. Data analyses focused on the production of actions, representational and deictic gestures and words, and gesture-word combinations. Results indicate that there is a continuity between the production of the first action schemes, the first gestures and the first words produced by children. The relationship between gestures and words changes over time. The onset of two-word speech was preceded by the emergence of gesture-word combinations.

If young children learn language as a continuous outgrowth of gestural communication, perhaps the same approach may be effective for (virtually or physically) embodied AI's.

EFTA00624597

An example of an iconic gesture occurs when one smiles explicitly to illustrate to some other agent that one is happy. Smiling is a natural expression of happiness, but of course one doesn't always smile when one's happy. The reason that explicit smiling is iconic is that the explicit smile actually resembles the unintentional smile, which is what it "stands for."

This kind of iconic gesture may emerge in a socially-embedded learning agent through a very simple logic. Suppose that when the agent is happy, it benefits from its nearby friends being happy as well, so that they may then do happy things together. And suppose that the agent has noticed that when it smiles, this has a statistical tendency to make its friends happy. Then, when it is happy and near its friends, it will have a good reason to smile. So through very simple probabilistic reasoning, the use of explicit smiling as a communicative tool may result. But what if the agent is not actually happy, but still wants some other agent to be happy? Using the reasoning from the prior paragraph, it will likely figure out to smile to make the other agent happy – even though it isn't actually happy.

Another simple example of an iconic gesture would be moving one's hands towards one's mouth, mimicking the movements of feeding oneself, when one wants to eat. Many analogous iconic gestures exist, such as doing a small solo part of a two-person dance to indicate that one wants to do the whole dance together with another person. The general rule an agent needs to learn in order to generate iconic gestures of this nature is that, in the context of shared activity, mimicking part of a process will sometimes serve the function of evoking that whole process.

This sort of iconic gesture may be learned in essentially the same way as an indexical gesture such as a dog repeatedly drawing the owner's attention to the owner's backpack, when the dog wants to go outside. The dog doesn't actually care about going outside with the backpack – he would just as soon go outside without it – but he knows the backpack is correlated with going outside, which is his actual interest.

The general rule here is

```
R :=
Implication
   SimultaneousImplication
      Execution $X $Y
      PredictiveImplication $X $Y
```

I.e., if doing $X often correlates with $Y, then maybe doing $X will bring about $Y. This sort of rule can bring about a lot of silly "superstitious" behavior but also can be particularly effective in social contexts, meaning in formal terms that

```
Context
   near_teacher
   R
```

holds with a higher truth value than R itself. This is a very small conglomeration of semantic nodes and links yet it encapsulates a very important communicational pattern: that if you want something to happen, and act out part of it – or something historically associated with it – around your teacher, then the thing may happen.

Many other cases of iconic gesture are more complex and mix iconic with symbolic aspects. For instance, one waves one hand away from oneself, to try to get someone else to go away. The hand is moving, roughly speaking, in the direction one wants the other to move in. However, understanding the meaning of this gesture requires a bit of savvy or experience. One one does grasp it, however, then one can understand its nuances: For instance, if I wave my hand in an

EFTA00624598

arc leading from your direction toward the direction of the door, maybe that means I want you to go out the door.

Purely symbolic (or nearly so) gestures include the thumbs-up symbol mentioned above, and many others including valence-indicating symbols like a nodded head for YES, a shaken-side-to-side head for NO, and shrugged shoulders for "I don't know." Each of these valence-indicating symbols actually indicates a fairly complex concept, which is learned from experience partly via attention to the symbol itself. So, an agent may learn that the nodded head corresponds with situations where the teacher gives it a reward, and also with situations where the agent makes a request and the teacher complies. The cluster of situations corresponding to the nodded-head then forms the agent's initial concept of "positive valence," which encompasses, loosely speaking, both the good and the true.

Summarizing our discussion of gestural communication: An awful lot of language exists between intelligent agents even if no word is ever spoken. And, our belief is that these sorts of non-verbal semiosis form the best possible context for the learning of verbal language, and that to attack verbal language learning outside this sort of context is to make an intrinsically-difficult problem even harder than it has to be. And this leads us to the final part of the chapter, which is a bit more speculative and adventuresome. The material in this section and the prior ones describes experiments of the sort we are currently carrying out with our virtual agent control software. We have not yet demonstrated all the forms of semiosis and non-linguistic communication described in the last section using our virtual agent control system, but we have demonstrated some of them and are actively working on extending our system's capabilities. In the following section, we venture a bit further into the realm of hypothesis and describe some functionalities that are beyond the scope of our current virtual agent control software, but that we hope to put into place gradually during the next 1-2 years. The basic goal of this work is to move from non-verbal to verbal communication.

It is interesting to enumerate the aspects in which each of the above components appears to be capable of tractable adaptation via experiential, embodied learning:

- Words and phrases that are found to be systematically associated with particular objects in the world, may be added to the "gazeteer list" used by the entity extractor
- The link parser dictionary may be automatically extended. In cases where the agent hears a sentence that is supposed to describe a certain situation, and realizes that in order for the sentence to be mapped into a set of logical relationships accurately describing the situation, it would be necessary for a certain word to have a certain syntactic link that it doesn't have, then the link parser dictionary may be modified to add the link to the word. (On the other hand, creating new link parser link types seems like a very difficult sort of learning – not to say it is unaddressable, but it will not be our focus in the near term.)
- Similar to with the link parser dictionary, if it is apparent that to interpret an utterance in accordance with reality a RelEx rule must be added or modified, this may be automatically done. The RelEx rules are expressed in the format of relatively simple logical implications between Boolean combinations of syntactic and semantic relationships, so that learning and modifying them is within the scope of a probabilistic logic system such as OpenCogPrime's PLN inference engine.
- The rules used by RelEx2Frame may be experientially modified quite analogously to those used by RelEx
- Our current statistical parse ranker ranks an interpretation of a sentence based on the frequency of occurrence of its component links across a parsed corpus. A deeper approach, however, would be to rank an interpretation based on its commonsensical plausibility, as

EFTA00624599

inferred from experienced-world-knowledge as well as corpus-derived knowledge. Again, this is within the scope of what an inference engine such as PLN should be able to do.

- Our word sense disambiguation and reference resolution algorithms involve probabilistic estimations that could be extended to refer to the experienced world as well as to a parsed corpus. For example, in assessing which sense of the noun "run" is intended in a certain context, the system could check whether stockings, or sports-events or series-of-events, are more prominent in the currently-observed situation. In assessing the sentence "The children kicked the dogs, and then they laughed," the system could map "they" into "children" via experientially-acquired knowledge that children laugh much more often than dogs.
- NLGen uses the link parser dictionary, treated above, and also uses rules analogous to (but inverse to) RelEx rules, mapping semantic relations into brief word-sequences. The "gold standard" for NLGen is whether, when it produces a sentence S from a set R of semantic relationships, the feeding of S into the language comprehension subsystem produces R (or a close approximation) as output. Thus, as the semantic mapping rules in RelEx and RelEx2Frame adapt to experience, the rules used in NLGen must adapt accordingly, which poses an inference problem unto itself.

All in all, when one delves in detail into the components that make up our hybrid statistical/rule-based NLP system, one sees there is a strong opportunity for experiential adaptive learning to substantially modify nearly every aspect of the NLP system, while leaving the basic framework intact.

This approach, we suggest, may provide means of dealing with a number of problems that have systematically vexed existing linguistic approaches. One example is parse ranking for complex sentences: this seems almost entirely a matter of the ability to assess the semantic plausibility of different parses, and doing this based on statistical corpus analysis seems unreasonable. One needs knowledge about a world to ground reasoning about plausiblity.

Another example is preposition disambiguation, a topic that is barely dealt with at all in the computational linguistics literature (see e.g. [33] for an indication of the state of the art). Consider the problem of assessing which meaning of "with" is intended in sentences like "I ate dinner with a fork", "I ate dinner with my sister", "I ate dinner with dessert." In performing this sort of judgment, an embodied system may use knowledge about which interpretations have matched observed reality in the case of similar utterances it has processed in the past, and for which it has directly seen the situations referred to by the utterances. If it has seen in the past, through direct embodied experience, that when someone said "I ate cereal with a spoon," they meant that the spoon was their tool not part of their food or their eating-partner; then when it hears "I ate dinner with a fork," it may match "cereal" to "dinner" and "spoon" to "fork" (based on probabilistic similarity measurement) and infer that the interpretation of "with" in the latter sentence should also be to denote a tool. How does this approach to computational language understanding tie in with gestural and general semiotic learning as we discussed earlier? The study of child language has shown that early language use is not purely verbal by any means, but is in fact a complex combination of verbal and gestural communication [23]. With the exception of first bullet point (entity extraction) above, every one of our instances of experiential modification of our language framework listed above involves the use of an understanding of what situation actually exists in the world, to help the system identify what the logical relationships output by the NLP system are supposed to be in a certain context. But a large amount of early-stage linguistic communication is social in nature, and a large amount of the remainder has to do with the body's relationship to physical objects. And, in understanding "what actually exists in the world" regarding social and physical relationships, a

EFTA00624600

full understanding of gestural communication is important. So, the overall pathway we propose for achieving robust, ultimately human-level NLP functionality is as follows:

- The capability for learning diverse instances of semiosis is established
- Gestural communication is mastered, via nonverbal imitative/reinforcement/corrective learning mechanisms such as we utilized for our embodied virtual agents
- Gestural communication, combined with observation of and action in the world and verbal interaction with teachers, allows the system to adapt numerous aspects of its initial NLP engine to allow it to more effectively interpret simple sentences pertaining to social and physical relationships
- Finally, given the ability to effectively interpret and produce these simple and practical sentences, probabilistic logical inference allows the system to gradually extend this ability to more and more complex and abstract senses, incrementally adapting aspects of the NLP engine as its scope broadens.

In this brief section we will mention another potentially important factor that we have intentionally omitted in the above analysis – but that may wind up being very important, and that can certainly be taken into account in our framework if this proves necessary. We have argued that gesture is an important predecessor to language in human children, and that incorporating it in AI language learning may be valuable. But there is another aspect of early language use that plays a similar role to gesture, which we have left out in the above discussion: this is the acoustic aspects of speech.

Clearly, pre-linguistic children make ample use of communicative sounds of various sorts. These sounds may be iconic, indexical or symbolic; and they may have a great deal of subtlety. Steven Mithen [Mit96] has argued that non-verbal utterances constitute a kind of proto-language, and that both music and language evolved out of this. Their role in language learning is well-known. We are uncertain as to whether an exclusive focus on text rather than speech would critically impair the language learning process of an AI system. We are fairly strongly convinced of the importance of gesture because it seems bound up with the importance of semiosis – gesture, it seems, is how young children learn flexible semiotic communication skills, and then these skills are gradually ported from the gestural to the verbal domain. Semiotically, on the other hand, phonology doesn't seem to give anything special beyond what gesture gives. What it does give is an added subtlety of emotional expressiveness – something that is largely missing from virtual agents as implemented today, due to the lack of really fine-grained facial expressions. Also, it provides valuable clues to parsing, in that groups of words that are syntactically bound together are often phrased together acoustically.

If one wished to incorporate acoustics into the framework described above, it would not be objectionably difficult on a technical level. Speech-to-text and text-to-speech software both exist, but neither have been developed with a view specifically toward conveyance of emotional information. One could approach the problem of assessing the emotional state of an utterance based on its sound as a supervised categorization problem, to be solved via supplying a machine learning algorithm with training data consisting of human-created pairs of the form (utterance, emotional valence). Similarly, one could tune the dependence of text-to-speech software for appropriate emotional expressiveness based on the same training corpus.

EFTA00624601

47.4 Simple Experiments with Embodiment and Anaphor Resolution

Now we turn to some fairly simple practical work that was done in 2008 with the OpenCog-based PetBrain software, involving the use of virtually embodied experience to help with interpretation of linguistic utterances. This work has been superseded somewhat by more recent work using OpenCog to control virtual agents; but the PetBrain work was especially clear and simple, so suitable in an expository sense for in-depth discussion here.

One of the two ways the PetBrain related language processing to embodied experience was via using the latter to resolve anaphoric references in text produced by human-controlled avatars.

The PetBrain controlled agent lived in a world with many objects, each one with their own characteristics. For example, we could have multiple balls, with varying colors and sizes. We represent this in the OpenCog Atomspace via using multiple nodes: a single ConceptNode to represent the concept "ball", a WordNode associated with the word "ball", and numerous SemeNodes representing particular balls. There may of course also be ConceptNodes representing ball-related ideas not summarized in any natural language word, e.g. "big fat squishy balls," "balls that can usefully be hit with a bat", etc.

As the agent interacts with the world, it acquires information about the objects it finds, through perceptions. The perceptions associated with a given object are stored as other nodes linked to the node representing the specific object instance. All this information is represented in the Atomspace using FrameNet-style relationships (exemplified in the next section).

When the user says, e.g., "Grab the red ball", the agent needs to figure out which specific ball the user is referring to – i.e. it needs to invoke the Reference Resolution (RR) process. RR uses the information in the sentence to select instances and also a few heuristic rules. Broadly speaking, Reference Resolution maps nouns in the user's sentences to actual objects in the virtual world, based on world-knowledge obtained by the agent through perceptions.

In this example, first the brain selects the ConceptNodes related to the word "ball". Then it examines all individual instances associated with these concepts, using the determiners in the sentence along with other appropriate restrictions (in this example the determiner is the adjective "red"; and since the verb is "grab" it also looks for objects that can be fetched). If it finds more than one "fetchable red ball", a heuristic is used to select one (in this case, it chooses the nearest instance).

The agent also needs to map pronouns in the sentences to actual objects in the virtual world. For example, if the user says "I like the red ball. Grab it.", the agent must map the pronoun "it" to a specific red ball. This process is done in two stages: first using anaphor resolution to associate the pronoun "it" with the previously heard noun "ball"; then using reference resolution to associate the noun "ball" with the actual object.

The subtlety of anaphor resolution is that there may be more than one plausible "candidate" noun corresponding to a given pronouns. As noted above, at time writing RelEx's anaphor resolution system is somewhat simplistic and is based on the classical Hobbs algorithm[Hob78]. Basically, when a pronoun (it, he, she, they and so on) is identified in a sentence, the Hobbs algorithm searches through recent sentences to find the nouns that fit this pronoun according to number, gender and other characteristics. The Hobbs algorithm is used to create a ranking of candidate nouns, ordered by time (most recently mentioned nouns come first).

We improved the Hobbs algorithm results by using the agent's world-knowledge to help choose the best candidate noun. Suppose the agent heard the sentences:

```
"The ball is red."
```

EFTA00624602

```
"The stick is brown."
```

and then it receives a third sentence

```
"Grab it.".
```

The anaphor resolver will build a list containing two options for the pronoun "it" of the third sentence: ball and stick. Given that the stick corresponds to the most recently mentioned noun, the agent will grab it instead of (as Hobbs would suggest) the ball.

Similarly, if the agent's history contains

```
"From here I can see as tree and a ball."
"Grab it."
```

Hobbs algorithm returns as candidate nouns "tree" and "ball", in this order. But using our integrative Reference Resolution process, the agent will conclude that a tree cannot be grabbed, so this candidate is discarded and "ball" is chosen.

47.5 Simple Experiments with Embodiment and Question Answering

The PetBrain was also capable of answering simple questions about its feelings/emotions (happiness, fear, etc.) and about the environment in which it lives. After a question is asked to the agent, it is parsed by RelEx and classified as either a truth question or a discursive one. After that, RelEx rewrites the given question as a list of Frames (based on FrameNet* with some customizations), which represent its semantic content. The Frames version of the question is then processed by the agent and the answer is also written in Frames. The answer Frames are then sent to a module that converts it back to the RelEx format. Finally the answer, in RelEx format, is processed by the NLGen module, that generates the text of the answer in English. We will discuss this process here in the context of the simple question "What is next to the tree?", which in an appropriate environment receives the answer "The red ball is next to the tree."

Question answering (QA) of course has a long history in AI [May04], and our approach fits squarely into the tradition of "deep semantic QA systems"; however it is innovative in its combination of dependency parsing with FrameNet and most importantly in the manner of its integration of QA with an overall cognitive architecture for agent control.

47.5.1 Preparing/Matching Frames

In order to answer an incoming question, the agent tries to match the Frames list, created by RelEx, against the Frames stored in its own memory. In general these Frames could come from a variety of sources, including inference, concept creation and perception; but in the current PetBrain they primarily come from perception, and simple transformations of perceptions.

However, the agent cannot use the incoming perceptual Frames in their original format because they lack grounding information (information that connects the mentioned elements to

* http://framenet.icsi.berkeley.edu

EFTA00624603

the real elements of the environment). So, two steps are then executed before trying to match the Frames: Reference Resolution (described above) and Frames Rewriting. Frames Rewriting is a process that changes the values of the incoming Frames elements into grounded values. Here is an example:

Incoming Frame (Generated by RelEx)

```
EvaluationLink
  DefinedFrameElementNode Color:Color
  WordInstanceNode ``red@aaa''
EvaluationLink
  DefinedFrameElementNode Color:Entity
  WordInstanceNode ``ball@bbb''
ReferenceLink
  WordInstanceNode ``red@aaa''
  WordNode ``red''
```

After Reference Resolution

```
ReferenceLink
  WordInstanceNode ``ball@bbb''
  SemeNode ``ball_99''
```

Grounded Frame (After Rewriting)

```
EvaluationLink
  DefinedFrameElementNode Color:Color
  ConceptNode ``red''
EvaluationLink
  DefinedFrameElementNode Color:Entity
  SemeNode ``ball_99''
```

Frame Rewriting serves to convert the incoming Frames to the same structure used by the Frames stored into the agent's memory. After Rewriting, the new Frames are then matched against the agent's memory and if all Frames were found in it, the answer is known by the agent, otherwise it is unknown.

In the PetBrain system, if a truth question was posed and all Frames were matched successfully, the answer would be be "yes"; otherwise the answer is "no". Mapping of ambiguous matching results into ambiguous responses were not handled in the PetBrain.

If the question requires a discursive answer the process is slightly different. For known answers the matched Frames are converted into RelEx format by Frames2RelEx and then sent to NLGen, which prepares the final English text to be answered. There are two types of unknown answers. The first one is when at least one Frame cannot be matched against the agent's memory and the answer is "I don't know". And the second type of unknown answer occurs when all Frames were matched successfully but they cannot be correctly converted into RelEx format or NLGen cannot identify the incoming relations. In this case the answer is "I know the answer, but I don't know how to say it".

EFTA00624604

47.5.2 Frames2RelEx

As mentioned above, this module is responsible for receiving a list of grounded Frames and returning another list containing the relations, in RelEx format, which represents the grammatical form of the sentence described by the given Frames. That is, the Frames list represents a sentence that the agent wants to say to another agent. NLGen needs an input in RelEx Format in order to generate an English version of the sentence; Frames2RelEx does this conversion.

Currently, Frames2RelEx is implemented as a rule-based system in which the preconditions are the required frames and the output is one or more RelEx relations e.g.

```
#Color(Entity,Color) =>
  present($2) .a($2) adj($2) _predadj($1, $2)
    definite($1) .n($1) noun($1) singular($1)
      .v(be) verb(be) punctuation(.) det(the)
```

where the precondition comes before the symbol => and *Color* is a frame which has two elements: Entity and Color. Each element is interpreted as a variable *Entity* = $1 and *Color* = $2. The effect, or output of the rule, is a list of RelEx relations. As in the case of RelEx2Frame, the use of hand-coded rules is considered a stopgap, and for a powerful AGI system based on this framework such rules will need to be learned via experience.

47.5.3 Example of the Question Answering Pipeline

Turning to the example "What is next to the tree?", Figure ?? illustrates the processes involved:

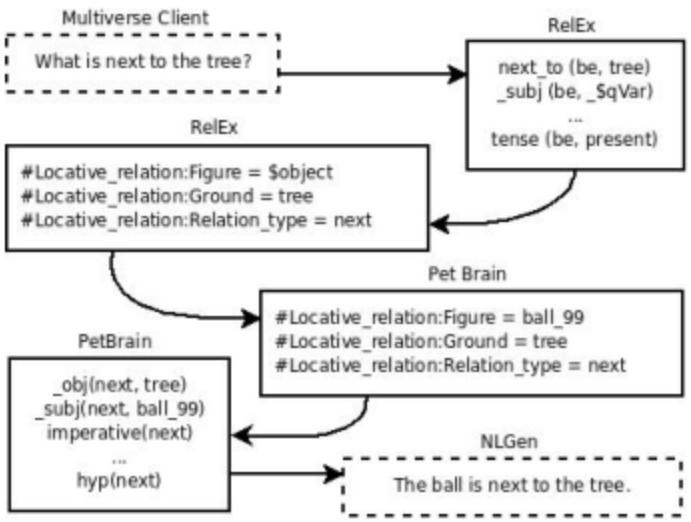


Fig. 47.1: Overview of current PetBrain language comprehension process

EFTA00624605

The question is parsed by RelEx, which creates the frames indicating that the sentence is a question regarding a location reference (next) relative to an object (tree). The frame that represents questions is called Questioning and it contains the elements Manner that indicates the kind of question (truth-question, what, where, and so on), Message that indicates the main term of the question and Addressee that indicates the target of the question. To indicate that the question is related to a location, the *Locative_relation* frame is also created with a variable inserted in its element Figure, which represents the expected answer (in this specific case, the object that is next to the tree).

The question-answer module tries to match the question frames in the Atomspace to fit the variable element. Suppose that the object that is next to the tree is the red ball. In this way, the module will match all the frames requested and realize that the answer is the value of the element Figure of the frame *Locative_relation* stored in the Atom Table. Then, it creates location frames indicating the red ball as the answer. These frames will be converted into RelEx format by the RelEx2Frames rule based system as described above, and NLGen will generate the expected sentence "the red ball is next to the tree".

47.5.4 Example of the PetBrain Language Generation Pipeline

To illustrate the process of language generation using NLGen, as utilized in the context of PetBrain query response, consider the sentence "The red ball is near the tree". When parsed by RelEx, this sentence is converted to:

```
_obj(near, tree)
_subj(near, ball)
imperative(near)
hyp(near)
definite(tree)
singular(tree)
_to-do(be, near)
_subj(be, ball)
present(be)
definite(ball)
singular(ball)
```

So, if sentences with this format are in the system's experience, these relations are stored by NLGen and will be used to match future relations that must be converted into natural language. NLGen matches at an abstract level, so sentences like "The stick is next to the fountain" will also be matched even if the corpus contain only the sentence "The ball is near the tree".

If the agent wants to say that "The red ball is near the tree", it must invoke NLGen with the above RelEx contents as input. However, the knowledge that the red ball is near the tree is stored as frames, and not as RelEx format. More specifically, in this case the related frame stored is the *Locative_relation* one, containing the following elements and respective values: Figure → red ball, Ground → tree, *Relation_type* → near.

So we must convert these frames and their elements' values into the RelEx format accept by NLGen. For AGI purposes, a system must learn how to perform this conversion in a flexible and context-appropriate way. In our current system, however, we have implemented a temporary

EFTA00624606

short-cut: a system of hand-coded rules, in which the preconditions are the required frames and the output is the corresponding RelEx format that will generate the sentence that represents the frames. The output of a rule may contains variables that must be replaced by the frame elements' values. For the example above, the output $_subj(be, ball)$ is generated from the rule output $_subj(be, \$var1)$ with the $\$var1$ replaced by the Figure element value.

Considering specifically question-answering (QA), the PetBrain's Language Comprehension module represents the answer to a question as a list of frames. In this case, we may have the following situations:

- The frames match a precondition and the RelEx output is correctly recognized by NLGen, which generates the expected sentence as the answer;
- The frames match a precondition, but NLGen did not recognize the RelEx output generated. In this case, the answer will be "I know the answer, but I don't know how to say it", which means that the question was answered correctly by the Language Comphrehension, but the NLGen could not generate the correct sentence;
- The frames didn't match any precondition; then the answer will also be "I know the answer, but I don't know how to say it".
- Finally, if no frames are generated as answer by the Language Comprehension module, the agent's answer will be "I don't know".

If the question is a truth-question, then NLGen is not required. In this case, the creation of frames as answer is considered as a "Yes", otherwise, the answer will be "No" because it was not possible to find the corresponding frames as the answer.

47.6 The Prospect of Massively Multiplayer Language Teaching

Now we tie in the theme of embodied language learning with more general considerations regarding embodied experiential learning.

Potentially, this may provide a means to facilitate robust language learning on the part of virtually embodied agents, and lead to an experientially-trained AGI language facility that can then be used to power other sorts of agents such as virtual babies, and ultimately virtual adult-human avatars that can communicate with experientially-grounded savvy rather than in the manner of chat-bots.

As one concrete, evocative example, imagine millions of talking parrots spread across different online virtual worlds – all communicating in simple English. Each parrot has its own local memories, its own individual knowledge and habits and likes and dislikes – but there's also a common knowledge-base underlying all the parrots, which includes a common knowledge of English.

The interest of many humans in interacting with chatbots suggests that virtual talking parrots or similar devices would be likely to meet with a large and enthusiastic audience.

Yes, humans interacting with parrots in virtual worlds can be expected to try to teach the parrots ridiculous things, obscene things, and so forth. But still, when it comes down to it, even pranksters and jokesters will have more fun with a parrot that can communicate better, and will prefer a parrot whose statements are comprehensible.

And for a virtual parrot, the test of whether it has used English correctly, in a given instance, will come down to whether its human friends have rewarded it, and whether it has gotten what

EFTA00624607

it wanted. If a parrot asks for food incoherently, it's less likely to get food – and since the virtual parrots will be programmed to want food, they will have motivation to learn to speak correctly. If a parrot interprets a human-controlled avatar's request "Fetch my hat please" incorrectly, then it won't get positive feedback from the avatar – and it will be programmed to want positive feedback.

And of course parrots are not the end of the story. Once the collective wisdom of throngs of human teachers has induced powerful language understanding in the collective bird-brain, this language understanding (and the commonsense understanding coming along with it) will be useful for many, many other purposes as well. Humanoid avatars – both human-baby avatars that may serve as more rewarding virtual companions than parrots or other virtual animals; and language-savvy human-adult avatars serving various useful and entertaining functions in online virtual worlds and games. Once AIs have learned enough that they can flexibly and adaptively explore online virtual worlds and gather information from human-controlled avatars according to their own goals using their linguistic facilities, it's easy to envision dramatic acceleration in their growth and understanding.

A baby AI has numerous disadvantages compared to a baby human being: it lacks the intricate set of inductive biases built into the human brain, and it also lacks a set of teachers with a similar form and psyche to it... and for that matter, it lacks a really rich body and world. However, the presence of thousands to millions of teachers constitutes a large advantage for the AI over human babies. And a flexible AGI framework will be able to effectively exploit this advantage. If nonlinguistic learning mechanisms like the ones we've described here, utilized in a virtually-embodied context, can go beyond enabling interestingly trainable virtual animals and catalyze the process of language learning – then, within a few years time, we may find ourselves significantly further along the path to AGI than most observers of the field currently expect.

EFTA00624608

Chapter 48
Natural Language Dialogue

Co-authored with Ruiting Lian

48.1 Introduction

Language evolved for dialogue – not for reading, writing or speechifying. So it's natural that dialogue is broadly considered a critical aspect of humanlike AGI – even to the extent that (for better or for worse) the conversational "Turing Test" is the standard test of human-level AGI.

Dialogue is a high-level functionality rather than a foundational cognitive process, and in the CogPrime approach it is something that must largely be learned via experience rather than being programmed into the system. In that sense, it may seem odd to have a chapter on dialogue in a book section focused on *engineering* aspects of general intelligence. One might think: Dialogue is something that should emerge from an intelligent system in conjunction with other intelligent systems, not something that should need to be engineered. And this is certainly a reasonable perspective! We do think that, as a CogPrime system develops, it will develop its own approach to natural language dialogue, based on its own embodiment, environment and experience – with similarities and differences to human dialogue.

However, we have also found it interesting to design a natural language dialogue system based on CogPrime, with the goal **not** of emulating human conversation, but rather of enabling interesting and intelligent conversational interaction with CogPrime systems. We call this system "ChatPrime" and will describe its architecture in this chapter. The components used in ChatPrime may also be useful for enabling CogPrime systems to carry out more humanlike conversation, via their incorporation in learned schemata; but we will not focus on that aspect here. In addition to its intrinsic interest, consideration of ChatPrime sheds much light on the conceptual relationship between NLP and other aspects of CogPrime.

We are very aware that there is an active subfield of computational linguistics focused on dialogue systems [Wah06, LDA05], however we will not draw significantly on that literature here. Making practical dialogue systems in the absence of a generally functional cognitive engine is a subtle and difficult art, which has been addressed in a variety of ways; however, we have found that designing a dialogue system within the context of an integrative cognitive engine like CogPrime is a somewhat different sort of endeavor.

EFTA00624610

48.1.1 Two Phases of Dialogue System Development

In practical terms, we envision the ChatPrime system as possessing two phases of development:

1. **Phase 1**:
   - "Lower levels" of NL comprehension and generation executed by a relatively traditional approach incorporating statistical and rule-based aspects (the RelEx and NLGen systems)
   - Dialogue control utilizes hand-coded procedures and predicates (SpeechActSchema and SpeechActTriggers) corresponding to fine-grained types of speech act
   - Dialogue control guided by general cognitive control system (OpenPsi, running within OpenCog)
   - SpeechActSchema and SpeechActTriggers, in some cases, will internally consult probabilistic inference, thus supplying a high degree of adaptive intelligence to the conversation
2. **Phase 2**:
   - "Lower levels" of NL comprehension and generation carried out within primary cognition engine, in a manner enabling their underlying rules and probabilities to be modified based the system's experience. Concretely, one way this could be done in OpenCog would be via
     - Implementing the RelEx and RelEx2Frame rules as PLN implications in the Atomspace
     - Implementing parsing via expressing the link parser dictionary as Atoms in the Atomspace, and using the SAT link parser to do parsing as an example of logical unification (carried out by a MindAgent wrapping an SAT solver)
     - Implementing NLGen within the OpenCog core, via making NLGen's sentence database a specially indexed Atomspace, and wrapping the NLGen operations in a MindAgent
   - Reimplement the SpeechActSchema and SpeechActTriggers in an appropriate combination of Combo and PLN logical link types, so they are susceptible to modification via inference and evolution

It's worth noting that the work required to move from Phase 1 to Phase 2 is essentially software development and computer science algorithm optimization work, rather than computational linguistics or AI theory. Then after the Phase 2 system is built there will, of course, be significant work involved in "tuning" PLN, MOSES and other cognitive algorithms to experientially adapt the various portions of the dialogue system that have been moved into the OpenCog core and refactored for adaptiveness.

48.2 Speech Act Theory and its Elaboration

We review here the very basics of speech act theory, and then the specific variant of speech act theory that we feel will be most useful for practical OpenCog dialogue system development.

EFTA00624611

The core notion of speech act theory is to analyze linguistic behavior in terms of discrete speech acts aimed at achieving specific goals. This is a convenient theoretical approach in an OpenCog context, because it pushes us to treat speech acts just like any other acts that an OpenCog system may carry out in its world, and to handle speech acts via the standard OpenCog action selection mechanism.

Searle, who originated speech act theory, divided speech acts according to the following (by now well known) ontology:

- **Assertives** : The speaker commits herself to something being true. *The sky is blue.*
- **Directives**: The speaker attempts to get the hearer to do something. *Clean your room!*
- **Commissives**: The speaker commits to some future course of action. *I will do it.*
- **Expressives**: The speaker expresses some psychological state. *I'm sorry.*
- **Declarations**: The speaker brings about a different state of the world. *The meeting is adjourned.*

Inspired by this ontology, Twitchell and Nunamaker (in their 2004 paper "Speech Act Profiling: A Probabilistic Method for Analyzing Persistent Conversations and Their Participants") created a much more fine-grained ontology of 42 kinds of speech acts, called SWBD-DAMSL (DAMSL = Dialogue Act Markup in Several Layers). Nearly all of their 42 speech act types can be neatly mapped into one of Searle's 5 high level categories, although a handful don't fit Searle's view and get categorized as "other." Figures 48.1 and 48.2 depict the 42 acts and their relationship to Searle's categories.

48.3 Speech Act Schemata and Triggers

In the suggested dialogue system design, multiple SpeechActSchema would be implemented, corresponding *roughly* to the 42 SWBD-DAMSL speech acts. The correspondence is "rough" because

- we may wish to add new speech acts not in their list
- sometimes it may be most convenient to merge 2 or more of their speech acts into a single SpeechActSchema. For instance, it's probably easiest to merge their YES ANSWER and NO ANSWER categories into a single TRUTH VALUE ANSWER schema, yielding affirmative, negative, and intermediate answers like "probably", "probably not", "I'm not sure", etc.
- sometimes it may be best to split one of their speech acts into several, e.g. to separately consider STATEMENTs which are responses to statements, versus statements that are unsolicited disbursements of "what's on the agent's mind."

Overall, the SWBD-DAMSL categories should be taken as guidance rather than doctrine. However, they are valuable guidance due to their roots in detailed analysis of real human conversations, and their role as a bridge between concrete conversational analysis and the abstractions of speech act theory.

Each SpeechActSchema would take in an input consisting of a DialogueNode, a Node type possessing a collection of links to

- a series of past statements by the agent and other conversation participants, with
  - each statement labeled according to the utterer

EFTA00624612

Tag Name	Tag	Example
STATEMENT-NON-OPINION	sd	Me, I'm in the legal department.
ACKNOWLEDGE (BACKCHANNEL)	b	Uh-huh.
STATEMENT-OPINION	sv	I think it's great.
AGREE/ACCEPT	aa	That's exactly it.
ABANDONED, TURN-EXIT OR UNINTERPRETABLE	%	So,-
APPRECIATION	ba	I can imagine.
YES-NO-QUESTION	qy	Do you have to have any special training?
NON-VERBAL	x	[Laughter], [Throat-clearing]
YES ANSWERS	ny	Yes.
CONVENTIONAL-CLOSING	fc	Well, it's been nice talking to you.
WH-QUESTION	qw	Well, how old are you?
NO ANSWERS	nn	No.
RESPONSE ACKNOWLEDGEMENT	bk	Oh, okay.
HEDGE	h	I don't know if I'm making any sense or not.
DECLARATIVE YES-NO-QUESTION	qyCd	So you can afford to get a house?
OTHER	other	Well give me a break, you know.
BACKCHANNEL IN QUESTION FORM	bh	Is that right?
QUOTATION	Cq	You can't be pregnant and have cats.
SUMMARIZE/REFORMULATE	bf	Oh, you mean you switched schools for the kids.
AFFIRMATIVE NON-YES ANSWERS	na	It is.
ACTION-DIRECTIVE	ad	Why don't you go first
COLLABORATIVE COMPLETION	C2	Who aren't contributing.
REPEAT-PHRASE	bCm	Oh, fajitas
OPEN-QUESTION	qo	How about you?
RHETORICAL-QUESTIONS	qh	Who would steal a newspaper?
HOLD BEFORE ANSWER/AGREEMENT	Ch	I'm drawing a blank.
REJECT	ar	Well, no
NEGATIVE NON-NO ANSWERS	ng	Uh, not a whole lot.
SIGNAL-NON-UNDERSTANDING	br	Excuse me?
OTHER ANSWERS	no	I don't know
CONVENTIONAL-OPENING	fp	How are you?
OR-CLAUSE	qrr	or is it more of a company?
DISPREFERRED ANSWERS	arp	Well, not so much that.
3RD-PARTY-TALK	t3	My goodness, Diane, get down from there.
OFFERS, OPTIONS COMMITS	commits	I'll have to check that out
SELF-TALK	t1	What's the word I'm looking for
DOWNPLAYER	bd	That's all right.
MAYBE/ACCEPT-PART	aap	Something like that
TAG-QUESTION	Cg	Right?
DECLARATIVE WH-QUESTION	qwCd	You are what kind of buff?
APOLOGY	fa	I'm sorry.
THANKING	ft	Hey thanks a lot

Fig. 48.1: The 42 DAMSL speech act categories.

Assertives	Expressives	Directives
STATEMENT	OPINION	YES-NO-QUESTION
YES ANSWERS	ABANDONED/UNINTERPRETABLE	WH-QUESTION
NO ANSWERS	BACKCHANNEL/ACKNOWLEDGE	DECLARATIVE YES-NO-QUESTION
QUOTATION	RESPONSE ACKNOWLEDGEMENT	BACKCHANNEL-QUESTION
AFFIRMATIVE NON-YES ANSWERS	SIGNAL-NON-UNDERSTANDING	SUMMARIZE/REFORMULATE
COLLABORATIVE COMPLETION	AGREEMENT/ACCEPT	ACTION-DIRECTIVE
RHETORICAL-QUESTIONS	APPRECIATION	OPEN-QUESTION
NEGATIVE NON-NO ANSWERS	CONVENTIONAL-CLOSING	TAG-QUESTION
OTHER ANSWERS	HEDGE	DECLARATIVE WH-QUESTION
OR-CLAUSE	HOLD BEFORE ANSWER/AGREEMENT	
DISPREFERRED ANSWERS	REJECT	Other
	CONVENTIONAL-OPENING	OTHER
Commissives	DOWNPLAYER	THIRD-PARTY TALK
OFFERS, OPTIONS, & COMMITS	MAYBE/ACCEPT-PART	NONVERBAL
	APOLOGY	
Declarations[1]	THANKING	
	REPEAT-PHRASE	

[1]None of the 42 dialogue acts could be described as a declaration

Fig. 48.2: Connecting the 42 DAMSL speech act categories to Searle's 5 higher-level categories.

– each statement uttered by the agent, labeled according to which SpeechActSchema was used to produce it, plus (see below) which SpeechActTrigger and which response generator was involved

EFTA00624613

- a set of Atoms comprising the context of the dialogue. These Atoms may optionally be linked to some of the Atoms representing some of the past statements. If they are not so linked, they are considered as general context.

The enaction of SpeechActSchema would be carried out via PredictiveImplicationLinks embodying "Context AND Schema → Goal" schematic implications, of the general form

```
PredictiveImplication
   AND
        Evaluation
           SpeechActTrigger T
           DialogueNode D
        Execution
          SpeechActSchema S
          DialogueNode D
   Evaluation
        Evaluation
           Goal G
```

with

```
ExecutionOutput
     SpeechActSchema S
     DialogueNode D
     UtteranceNode U
```

being created as a result of the enaction of the SpeechActSchema. (An UtteranceNode is a series of one or more SentenceNodes.)

A single SpeechActSchema may be involved in many such implications, with different probabilistic weights, if it naturally has many different Trigger contexts.

Internally each SpeechActSchema would contain a set of one or more response generators, each one of which is capable of independently producing a response based on the given input. These may also be weighted, where the weight determines the probability of a given response generation process being chosen in preference to the others, once the choice to enact that particular SpeechActSchema has already been made.

48.3.1 Notes Toward Example SpeechActSchema

To make the above ideas more concrete, let's consider a few specific SpeechActSchema. We won't fully specify them here, but will outline them sufficiently to make the ideas clear.

48.3.1.1 TruthValueAnswer

The TruthValueAnswer SpeechActSchema would encompass SWBD-DAMSL's YES ANSWER and NO ANSWER, and also more flexible truth value based responses.

EFTA00624614

Trigger context

: when the conversation partner produces an utterance that RelEx maps into a truth-value query (this is simple as truth-value-query is one of RelEx's relationship types).

Goal

: the simplest goal relevant here is pleasing the conversation partner, since the agent may have noticed in the past that other agents are pleased when their questions are answers. (More advanced agents may of course have other goals for answering questions, e.g. providing the other agent with information that will let it be more useful in future.)

Response generation schema

: for starters, this SpeechActSchema could simply operate as follows. It takes the relationship (Atom) corresponding to the query, and uses it to launch a query to the pattern matcher or PLN backward chainer. Then based on the result, it produces a relationship (Atom) embodying the answer to the query, or else updates the truth value of the existing relationship corresponding to the answer to the query. This "answer" relationship has a certain truth value. The schema could then contain a set of rules mapping the truth values into responses, with a list of possible responses for each truth value range. For example a very high strength and high confidence truth value would be mapped into a set of responses like {definitely, certainly, surely, yes, indeed}.

This simple case exemplifies the overall Phase 1 approach suggested here. The conversation will be guided by fairly simple heuristic rules, but with linguistic sophistication in the comprehension and generation aspects, and potentially subtle inference invoked within the SpeechActSchema or (less frequently) the Trigger contexts. Then in Phase 2 these simple heuristic rules will be refactored in a manner rendering them susceptible to experiential adaptation.

48.3.1.2 Statement: Answer

The next few SpeechActSchema (plus maybe some similar ones not given here) are intended to collectively cover the ground of SWBD-DAMSL's STATEMENT OPINION and STATEMENT NON-OPINION acts.

Trigger context

: The trigger is that the conversation partner asks a wh- question

EFTA00624615

Goal

: Similar to the case of a TruthValueAnswer, discussed above

Response generation schema

: When a wh- question is received, one reasonable response is to produce a statement comprising an answer. The question Atom is posed to the pattern matcher or PLN, which responds with an Atom-set comprising a putative answer. The answer Atoms are then pared down into a series of sentence-sized Atom-sets, which are articulated as sentences by NLGen. If the answer Atoms have very low-confidence truth values, or if the Atomspace contains knowledge that other agents significantly disagree with the agent's truth value assessments, then the answer Atom-set may have Atoms corresponding to "I think" or "In my opinion" etc. added onto it (this gives an instance of the STATEMENT NON-OPINION act).

48.3.1.3 Statement: Unsolicited Observation

Trigger context

: when in the presence of another intelligent agent (human or AI) and nothing has been said for a while, there is a certain probability of choosing to make a "random" statement.

Goal 1

: Unsolicited observations may be made with a goal of pleasing the other agent, as it may have been observed in the past that other agents are happier when spoken to

Goal 2

: Unsolicited observations may be made with goals of increasing the agent's own pleasure or novelty or knowledge – because it may have been observed that speaking often triggers conversations, and conversations are often more pleasurable or novel or educational than silence

Response generation schema

: One option is a statement describing something in the mutual environment, another option is a statement derived from high-STI Atoms in the agent's Atomspace. The particulars are similar to the "Statement: Answer" case.

EFTA00624616

48.3.1.4 Statement: External Change Notification

Trigger context

: when in a situation with another intelligent agent, and something significant changes in the mutually perceived situation, a statement describing it may be made.

Goal 1

: External change notification utterances may be made for the same reasons as Unsolicited Observations, described above.

Goal 2

: The agent may think a certain external change is important to the other agent it is talking to, for some particular reason. For instance, if the agent sees a dog steal Bob's property, it may wish to tell Bob about this.

Goal 3

: The change may be important to the agent itself – and it may want its conversation partner to do something relevant to an observed external change ... so it may bring the change to the partner's attention for this reason. For instance, "Our friends are leaving. Please try to make them come back."

Response generation schema

: The Atom-set for expression characterizes the change observed. The particulars are similar to the "Statement: Answer" case.

48.3.1.5 Statement: Internal Change Notification

Trigger context 1

: when the importance level of an Atom increases dramatically while in the presence of another intelligent agent, a statement expressing this Atom (and some of its currently relevant surrounding Atoms) may be made

EFTA00624617

Trigger context 2

: when the truth value of a reasonably important Atom changes dramatically while in the presence of another intelligent agent, a statement expressing this Atom and its truth value may be made

Goal

: Similar goals apply here as to External Change Notification, considered above

Response generation schema

: Similar to the "Statement: External Change Notification" case.

48.3.1.6 WHQuestion

Trigger context

: being in the presence of an intelligent agent thought capable of answering questions

Goal 1

: the general goal of increasing the agent's total knowledge

Goal 2

: the agent notes that, to achieve one of its currently important goals, it would be useful to possess a Atom fulfilling a certain specification

Response generation schema

: Formulate a query whose answer would be an Atom fulfilling that specification, and then articulate this logical query as an English question using NLGen

48.4 Probabilistic Mining of Trigger contexts

One question raised by the above design sketch is where the Trigger contexts come from. They may be hand-coded, but this approach may suffer from excessive brittleness. The approach suggested by Twitchell and Nunamaker's work (which involved modeling human dialogues rather

EFTA00624618

than automatically generating intelligent dialogues) is statistical. That is, they suggest marking up a corpus of human dialogues with tags corresponding to the 42 speech acts, and learning from this annotated corpus a set of Markov transition probabilities indicating which speech acts are most likely to follow which others. In their approach the transition probabilities refer only to series of speech acts.

In an OpenCog context one could utilize a more sophisticated training corpus in a more sophisticated way. For instance, suppose one wants to build a dialogue system for a game character conversing with human characters in a game world. Then one could conduct experiments in which one human controls a "human" game character, and another human puppeteers an "AI" game character. That is, the puppeteered character funnels its perceptions to the AI system, but has its actions and verbalizations controlled by the human puppeteer. Given the dialogue from this sort of session, one could then perform markup according to the 42 speech acts.

As a simple example, consider the following brief snippet of annotated conversation:

speaker	**utterance**	**speech act type**
Ben	Go get me the ball	ad
AI	Where is it?	qw
Ben	Over there [points]	sd
AI	By the table?	qy
Ben	Yeah	ny
AI	Thanks	ft
AI	I'll get it now.	commits

A DialogueNode object based on this snippet would contain the information in the table, plus some physical information about the situation, such as, in this case: predicates describing the relative locations of the two agents, the ball an the table (e.g. the two agents are very near each other, the ball and the table are very near each other, but these two groups of entities are only moderately near each other); and, predicates involving

Then, one could train a machine learning algorithm such as MOSES to predict the probability of speech act type S_1 occurring at a certain point in a dialogue history, based on the prior history of the dialogue. This prior history could include percepts and cognitions as well as utterances, since one has a record of the AI system's perceptions and cognitions in the course of the marked-up dialogue.

One question is whether to use the 42 SWBD-DAMSL speech acts for the creation of the annotated corpus, or whether instead to use the modified set of speech acts created in designing SpeechActSchema. Either way could work, but we are mildly biased toward the former, since this specific SWBD-DAMSL markup scheme has already proved its viability for marking up conversations. It seems unproblematic to map probabilities corresponding to these speech acts into probabilities corresponding to a slightly refined set of speech acts. Also, this way the corpus would be valuable independently of ongoing low-level changes in the collection of SpeechActSchema.

In addition to this sort of supervised training in advance, it will be important to enable the system to learn Trigger contexts online as a consequence of its life experience. This learning may take two forms:

1. Most simply, adjustment of the probabilities associated with the PredictiveImplicationLinks between SpeechActTriggers and SpeechActSchema

EFTA00624619

2. More sophisticatedly, learning of new SpeechActTrigger predicates, using an algorithm such as MOSES for predicate learning, based on mining the history of actual dialogues to estimate fitness

In both cases the basis for learning is information regarding the extent to which system goals were fulfilled by each past dialogue. PredictiveImplications that correspond to portions of successful dialogues will be have their truth values increased, and those corresponding to portions of unsuccessful dialogues will have their truth values decreased. Candidate SpeechActTriggers will be valued based on the observed historical success of the responses they would have generated based on historically perceived utterances; and (ultimately) more sophisticatedly, based on the estimated success of the responses they generate. Note that, while somewhat advanced, this kind of learning is much easier than the procedure learning required to learn new SpeechActSchema.

48.5 Conclusion

While the underlying methods are simple, the above methods appear capable of producing arbitrarily complex dialogues about any subject that is represented by knowledge in the AtomSpace. There is no reason why dialogue produced in this manner should be indistinguishable from human dialogue; but it may nevertheless be humanly comprehensible, intelligent and insightful. What is happening in this sort of dialogue system is somewhat similar to current natural language query systems that query relational databases, but the "database" in question is a dynamically self-adapting weighted labeled hypergraph rather than a static relational database, and this difference means a much more complex dialogue system is required, as well as more flexible language comprehension and generation components.

Ultimately, a CogPrime system – if it works as desired – will be able to learn increased linguistic functionality, and new languages, on its own. But this is not a prerequisite for having intelligent dialogues with a CogPrime system. Via building a ChatPrime type system, as outlined here, intelligent dialogue can occur with a CogPrime system while it is still at relatively early stages of cognitive development, and even while the underlying implementation of the CogPrime design is incomplete. This is not closely analogous to human cognitive and linguistic development, but, it can still be pursued in the context of a CogPrime development plan that follows the overall arc of human developmental psychology.

EFTA00624620

Section VIII
From Here to AGI

Chapter 49
Summary of Argument for the CogPrime Approach

49.1 Introduction

By way of conclusion, we now return to the "key claims" that were listed at the end of Chapter 1 of Part 1. Quite simply, this is a list of claims such that – roughly speaking – if the reader accepts these claims, they should accept that the CogPrime approach to AGI is a viable one. On the other hand if the reader rejects one or more of these claims, they may well find one or more aspects of CogPrime unacceptable for some related reason. In Chapter 1 of Part 1 we merely listed these claims; here we briefly discuss each one in the context of the intervening chapters, giving each one its own section or subsection.

As we clarified at the start of Part 1, we don't fancy that we have provided an ironclad argument that the CogPrime approach to AGI is guaranteed to work as hoped, once it's fully engineered, tuned and taught. Mathematics isn't yet adequate to analyze the real-world behavior of complex systems like these; and we have not yet implemented, tested and taught enough of CogPrime to provide convincing empirical validation. So, most of the claims listed here have not been rigorously demonstrated, but only heuristically argued for. That is the reality of AGI work right now: one assembles a design based on the best combination of rigorous and heuristic arguments one can, then proceeds to create and teach a system according to the design, adjusting the details of the design based on experimental results as one goes along. = For an uncluttered list of the claims, please refer back to Chapter 1 of Part 1; here we will review the claims integrated into the course of discussion.

The following chapter, aimed at the more mathematically-minded reader, gives a list of formal propositions echoing many of the ideas in the chapter – propositions such that, if they are true, then the success of CogPrime as an architecture for general intelligence is likely.

49.2 Multi-Memory Systems

The first of our key claims is that *to achieve general intelligence in the context of human-intelligence-friendly environments and goals using feasible computational resources, it's important that an AGI system can handle different kinds of memory (declarative, procedural, episodic, sensory, intentional, attentional) in customized but interoperable ways.* The basic idea is that

EFTA00624624

these different kinds of knowledge have very different characteristics, so that trying to handle them all within a single approach, while surely possible, is likely to be unacceptably inefficient.

The tricky issue in formalizing this claim is that "single approach" is an ambiguous notion: for instance, if one has a wholly logic-based system that represents all forms of knowledge using predicate logic, then one may still have specialized inference control heuristics corresponding to the different kinds of knowledge mentioned in the claim. In this case one has "customized but interoperable ways" of handling the different kinds of memory, and one doesn't really have a "single approach" even though one is using logic for everything. To bypass such conceptual difficulties, one may formalize cognitive synergy using a geometric framework as discussed in Appendix B, in which different types of knowledge are represented as metrized categories, and cognitive synergy becomes a statement about paths to goals being shorter in metric spaces combining multiple knowledge types than in those corresponding to individual knowledge types.

In CogPrime we use a complex combination of representations, including the Atomspace for declarative, attentional and intentional knowledge and some episodic and sensorimotor knowledge, Combo programs for procedural knowledge, simulations for episodic knowledge, and hierarchical neural nets for some sensorimotor knowledge (and related episodic, attentional and intentional knowledge). In cases where the same representational mechanism is used for different types of knowledge, different cognitive processes are used, and often different aspects of the representation (e.g. attentional knowledge is dealt with largely by ECAN acting on AttentionValues and HebbianLinks in the Atomspace; whereas declarative knowledge is dealt with largely by PLN acting on TruthValues and logical links, also in the AtomSpace). So one has a mix of the "different representations for different memory types" approach and the "different control processes on a common representation for different memory types" approach.

It's unclear how closely dependent the need for a multi-memory approach is on the particulars of "human-friendly environments." We argued in Chapter 9 of Part 1 that one factor militating in favor of a multi-memory approach is the need for multimodal communication: declarative knowledge relates to linguistic communication; procedural knowledge relates to demonstrative communication; attentional knowledge relates to indicative communication; and so forth. But in fact the multi-memory approach may have a broader importance, even to intelligences without multimodal communication. This is an interesting issue but not particularly critical to the development of human-like, human-level AGI, since in the latter case we are specifically concerned with creating intelligences that *can* handle multimodal communication. So if for no other reason, the multi-memory approach is worthwhile for handling multi-modal communication.

Pragmatically, it is also quite clear that the human brain takes a multi-memory approach, e.g. with the cerebellum and closely linked cortical regions containing special structures for handling procedural knowledge, with special structures for handling motivational (intentional) factors, etc. And (though this point is certainly not definitive, it's meaningful in the light of the above theoretical discussion) decades of computer science and narrow-AI practice strongly suggest that the "one memory structure fits all" approach is not capable of leading to effective real-world approaches.

49.3 Perception, Action and Environment

The more we understand of human intelligence, the clearer it becomes how closely it has evolved to the particular goals and environments for which the human organism evolved. This is true

EFTA00624625

in a broad sense, as illustrated by the above issues regarding multi-memory systems, and is also true in many particulars, as illustrated e.g. by Changizi's [Cha09] evolutionary analysis of the human visual system. While it might be possible to create a human-like, human-level AGI by abstracting the relevant biases from human biology and behavior and explicitly encoding them in one's AGI architecture, it seems this would be an inordinately difficult approach in practice, leading to the claim that *to achieve human-like general intelligence, it's important for an intelligent agent to have sensory data and motoric affordances that roughly emulate those available to humans*. We don't claim this is a necessity – just a dramatic convenience. And if one accepts this point, it has major implications for what sorts of paths toward AGI it makes most sense to follow.

Unfortunately, though, the idea of a "human-like" set of goals and environments is fairly vague; and when you come right down to it, *we don't know exactly how close the emulation needs to be* to form a natural scenario for the maturation of human-like, human-level AGI systems. One could attempt to resolve this issue via a priori theory, but given the current level of scientific knowledge it's hard to see how that would be possible in any definitive sense ... which leads to the conclusion that *our AGI systems and platforms need to support fairly flexible experimentation with virtual-world and/or robotic infrastructures.*

Our own intuition is that currently neither current virtual world platforms, nor current robotic platforms, are *quite* adequate for the development of human-level, human-like AGI. Virtual worlds would need to become a lot more like robot simulators, allowing more flexible interaction with the environment, and more detailed control of the agent. Robots would need to become more robust at moving and grabbing – e.g. with Big Dog's movement ability but the grasping capability of the best current grabber arms. We do feel that development of adequate virtual world or robotics platforms is quite possible using current technology, and could be done at fairly low cost if someone were to prioritize this. Even without AGI-focused prioritization, it seems that the needed technological improvements are likely to happen during the next decade for other reasons. So at this point we feel it makes sense for AGI researchers to focus on AGI and exploit embodiment-platform improvements as they come along – at least, this makes sense in the case of AGI approaches (like CogPrime) that can be primarily developed in an embodiment-platform-independent manner.

49.4 Developmental Pathways

But if an AGI system is going to live in human-friendly environments, what should it do there? No doubt very many pathways leading from incompetence to adult-human-level general intelligence exist, but one of them is much better understood than any of the others, and that's the one normal human children take. Of course, given their somewhat different embodiment, it doesn't make sense to try to force AGI systems to take *exactly* the same path as human children, but having AGI systems follow a fairly close approximation to the human developmental path seems the smoothest developmental course ... a point summarized by the claim that: *To work toward adult human-level, roughly human-like general intelligence, one fairly easily comprehensible path is to use environments and goals reminiscent of human childhood, and seek to advance one's AGI system along a path roughly comparable to that followed by human children.*

Human children learn via a rich variety of mechanisms; but broadly speaking one conclusion one may drawn from studying human child learning is that it may make sense to *teach an*

EFTA00624626

AGI system aimed at roughly human-like general intelligence via a mix of spontaneous learning and explicit instruction, and to instruct it via a combination of imitation, reinforcement and correction, and a combination of linguistic and nonlinguistic instruction. We have explored exactly what this means in Chapter 31 and others, via looking at examples of these types of learning in the context of virtual pets in virtual worlds, and exploring how specific CogPrime learning mechanisms can be used to achieve simple examples of these types of learning.

One important case of learning that human children are particularly good at is language learning; and we have argued that this is a case where it may pay for AGI systems to take a route somewhat different from the one taken by human children. Humans seem to be born with a complex system of biases enabling effective language learning, and it's not yet clear exactly what these biases are nor how they're incorporated into the learning process. It is very tempting to give AGI systems a "short cut" to language proficiency via making use of existing rule-based and statistical-corpus-analysis-based NLP systems; and we have fleshed out this approach sufficiently to have convinced ourselves it makes practical as well as conceptual sense, in the context of the specific learning mechanisms and NLP tools built into OpenCog. Thus we have provided a number of detailed arguments and suggestions in support of our claim that *one effective approach to teaching an AGI system human language is to supply it with some in-built linguistic facility, in the form of rule-based and statistical-linguistics-based NLP systems, and then allow it to improve and revise this facility based on experience.*

49.5 Knowledge Representation

Many knowledge representation approaches have been explored in the AI literature, and ultimately many of these could be workable for human-level AGI if coupled with the right cognitive processes. The key goal for a knowledge representation for AGI should be *naturalness* with respect to the AGI's cognitive processes – i.e. the cognitive processes shouldn't need to undergo complex transformative gymnastics to get information in and out of the knowledge representation in order to do their cognitive work. Toward this end we have come to a similar conclusion to some other researchers (e.g. Joscha Bach and Stan Franklin), and concluded that *given the strengths and weaknesses of current and near-future digital computers, a (loosely) neural-symbolic network is a good representation for directly storing many kinds of memory, and interfacing between those that it doesn't store directly.* CogPrime's AtomSpace is a neural-symbolic network designed to work nicely with PLN, MOSES, ECAN and the other key CogPrime cognitive processes; it supplies them with what they need without causing them undue complexities. It provides a platform that these cognitive processes can use to adaptively, automatically construct specialized knowledge representations for particular sorts of knowledge that they encounter.

49.6 Cognitive Processes

The crux of intelligence is dynamics, learning, adaptation; and so the crux of an AGI design is the set of cognitive processes that the design provides. These processes must collectively allow the AGI system to achieve its goals in its environments using the resources at hand. Given

EFTA00624627

CogPrime's multi-memory design, it's natural to consider CogPrime's cognitive processes in terms of which memory subsystems they focus on (although, this is not a perfect mode of analysis, since some of the cognitive processes span multiple memory types).

49.6.1 Uncertain Logic for Declarative Knowledge

One major decision made in the creation of CogPrime was that *given the strengths and weaknesses of current and near-future digital computers, uncertain logic is a good way to handle declarative knowledge*. Of course this is not obvious nor is it the only possible route. Declarative knowledge can potentially be handled in other ways; e.g. in a hierarchical network architecture, one can make declarative knowledge emerge automatically from procedural and sensorimotor knowledge, as is the goal in the Numenta and DeSTIN designs reviewed in Chapter 4 of Part 1. It seems clear that the human brain doesn't contain anything closely parallel to formal logic – even though one can ground logic operations in neural-net dynamics as explored in Chapter 34, this sort of grounding leads to "uncertain logic enmeshed with a host of other cognitive dynamics" rather than "uncertain logic as a cleanly separable cognitive process."

But contemporary digital computers are not brains – they lack the human brain's capacity for cheap massive parallelism, but have a capability for single-operation speed and precision far exceeding the brain's. In this way computers and formal logic are a natural match (a fact that's not surprising given that Boolean logic lies at the foundation of digital computer operations). Using *uncertain logic* is a sort of compromise between brainlike messiness and fuzziness, and computerlike precision. An alternative to using uncertain logic is using crisp logic and incorporating uncertainty as content within the knowledge base – this is what SOAR does, for example, and it's not a wholly unworkable approach. But given that the vast mass of knowledge needed for confronting everyday human reality is highly uncertain, and that this knowledge often needs to be manipulated efficiently in real-time, it seems to us there is a strong argument for embedding uncertainty in the logic.

Many approaches to uncertain logic exist in the literature, including probabilistic and fuzzy approaches, and one conclusion we reached in formulating CogPrime is that none of them was adequate on its own – leading us, for example, to the conclusion that *to deal with the problems facing a human-level AGI, an uncertain logic must integrate imprecise probability and fuzziness with a broad scope of logical constructs*. The arguments that both fuzziness and probability are needed seem hard to counter – these two notions of uncertainty are qualitatively different yet both appear cognitively necessary.

The argument for using probability in an AGI system is assailed by some AGI researchers such as Pei Wang, but we are swayed by the theoretical arguments in favor of probability theory's mathematically fundamental nature, as well as the massive demonstrated success of probability theory in various areas of narrow AI and applied science. However, we are also swayed by the arguments of Pei Wang, Peter Walley and others that using single-number probabilities to represent truth values leads to untoward complexities related to the tabulation and manipulation of amounts of evidence. This has led us to an imprecise probability based approach; and then technical arguments regarding the limitations of standard imprecise probability formalisms has led us to develop our own "indefinite probabilities" formalism.

The PLN logic framework is one way of integrating imprecise probability and fuzziness in a logical formalism that encompasses a broad scope of logical constructs. It integrates term logic

EFTA00624628

and predicate logic – a feature that we consider not necessary, but very convenient, for AGI. Either predicate or term logic on its own would suffice, but each is awkward in certain cases, and integrating them as done in PLN seems to result in more elegant handling of real-world inference scenarios. Finally, PLN also integrates intensional inference in an elegant manner that demonstrates integrative intelligence – it defines intension using *pattern theory*, which binds inference to pattern recognition and hence to other cognitive processes in a conceptually appropriate way.

Clearly PLN is not the only possible logical formalism capable of serving a human-level AGI system; however, we know of no other existing, fleshed-out formalism capable of fitting the bill. In part this is because PLN has been developed as part of an integrative AGI project whereas other logical formalisms have mainly been developed for other purposes, or purely theoretically. Via using PLN to control virtual agents, and integrating PLN with other cognitive processes, we have tweaked and expanded the PLN formalism to serve all the roles required of the "declarative cognition" component of an AGI system with reasonable elegance and effectiveness.

49.6.2 Program Learning for Procedural Knowledge

Even more so than declarative knowledge, procedural knowledge is represented in many different ways in the AI literature. The human brain also apparently uses multiple mechanisms to embody different kinds of procedures. So the choice of how to represent procedures in an AGI system is not particularly obvious. However, there is one particular representation of procedures that is particularly well-suited for current computer systems, and particularly well-tested in this context: *programs*. In designing CogPrime, we have acted based on the understanding that *programs are a good way to represent procedures – including both cognitive and physical-action procedures, but perhaps not including low-level motor-control procedures.*

Of course, this begs the question of *programs in what programming language*, and in this context we have made a fairly traditional choice, using a special language called Combo that is essentially a minor variant of LISP, and supplying Combo with a set of customized primitives intended to reduce the length of the typical programs CogPrime needs to learn and use. What differentiates this use of LISP from many traditional uses of LISP in AI is that we are *only* using the LISP-ish representational style for procedural knowledge, rather than trying to use it for everything.

One test of whether the use of Combo programs to represent procedural knowledge makes sense is whether the procedures useful for a CogPrime system in everyday human environments have short Combo representations. We have worked with Combo enough to validate that they generally do in the virtual world environment – and also in the physical-world environment *if* lower-level motor procedures are supplied as primitives. That is, we are not convinced that Combo is a good representation for the procedure a robot needs to do to move its fingers to pick up a cup, coordinating its movements with its visual perceptions. It's certainly possible to represent this sort of thing in Combo, but Combo may be an awkward tool. However, if one represents low-level procedures like this using another method, e.g. learned cell assemblies in a hierarchical network like DeSTIN, then it's very feasible to make Combo programs that invoke these low-level procedures, and encode higher-level actions like "pick up the cup in front of you slowly and quietly, then hand it to Jim who is standing next to you."

EFTA00624629

Having committed to use programs to represent many procedures, the next question is how to learn programs. One key conclusion we have come to via our empirical work in this area is that some form of powerful *program normalization* is essential. Without normalization, it's too hard for existing learning algorithms to generalize from known, tested programs and draw useful uncertain conclusions about untested ones. We have worked extensively with a generalization of Holman's "Elegant Normal Form" in this regard.

For learning normalized programs, we have come to the following conclusions:

- for relatively straightforward procedure learning problems, *hillclimbing with random restart and a strong Occam bias is an effective method*
- for more difficult problems that elude hillclimbing, *probabilistic evolutionary program learning is an effective method*

The probabilistic evolutionary program learning method we have worked with most in OpenCog is MOSES, and significant evidence has been gathered showing it to be dramatically more effective than genetic programming on relevant classes of problems. However, more work needs to be done to evaluate its progress on complex and difficult procedure learning problems. Alternate, related probabilistic evolutionary program learning algorithms such as PLEASURE have also been considered and may be implemented and tested as well.

49.6.3 Attention Allocation

There is significant evidence that the brain uses some sort of "activation spreading" type method to allocate attention, and many algorithms in this spirit have been implemented and utilized in the AI literature. So, we find ourselves in agreement with many others that *activation spreading is a reasonable way to handle attentional knowledge* (though other approaches, with greater overhead cost, may provide better accuracy and may be appropriate in some situations). We also agree with many others who have chosen **Hebbian learning as one route of learning associative relationships**, with more sophisticated methods such as information-geometric ones potentially also playing a role.

Where CogPrime differs from standard practice is in the use of an economic metaphor to regulate activation spreading. In this matter CogPrime is broadly in agreement with Eric Baum's arguments about the value of economic methods in AI, although our specific use of economic methods is very different from his. Baum's work (e.g. Hayek [Bau04]) embodies more complex and computationally expensive uses of artificial economics, whereas we believe that *in the context of a neural-symbolic network, artificial economics is an effective approach to* **activation spreading**; and CogPrime's ECAN framework seeks to embody this idea. ECAN can also make use of more sophisticated and expensive uses of artificial currency when large amount of system resources are involved in a single choice, rendering the cost appropriate.

One major choice made in the CogPrime design is *to focus on two kinds of attention: processor (represented by ShortTermImportance) and memory (represented by LongTermImportance).* This is a direct reflection of one of the key differences between the von Neumann architecture and the human brain: in the former but not the latter, there is a strict separation between memory and processing in the underlying compute fabric. We carefully considered the possibility of using a larger variety of attention values, and in Chapter 23 we presented some mathematics and concepts that could be used in this regard, but for reasons of simplicity and computational

EFTA00624630

efficiency we are currently using only STI and LTI in our OpenCogPrime implementation, with the possibility of extending further if experimentation proves it necessary.

49.6.4 Internal Simulation and Episodic Knowledge

For episodic knowledge, as with declarative and procedural knowledge, CogPrime has opted for a solution motivated by the particular strengths of contemporary digital computers. When the human brain runs through a "mental movie" of past experiences, it doesn't do any kind of accurate physical simulation of these experiences. But that's not because the brain wouldn't benefit from such – it's because the brain doesn't know how to do that sort of thing! On the other hand, any modern laptop can run a reasonable Newtonian physics simulation of everyday events, and more fundamentally can recall and manage the relative positions and movements of items in an internal 3D landscape paralleling remembered or imagined real-world events. With this in mind, we believe that in an AGI context, *simulation is a good way to handle episodic knowledge; and running an internal "world simulation engine" is an effective way to handle simulation.*

CogPrime can work with many different simulation engines; and since simulation technology is continually advancing independently of AGI technology, this is an area where AGI can buy some progressive advancement for free as time goes on. The subtle issues here regard interfacing between the simulation engine and the rest of the mind: mining meaningful information out of simulations using pattern mining algorithms; and more subtly, figuring out what simulations to run at what times in order to answer the questions most relevant to the AGI system in the context of achieving its goals. We believe we have architected these interactions in a viable way in the CogPrime design, but we have tested our ideas in this regard only in some fairly simple contexts regarding virtual pets in a virtual world, and much more remains to be done here.

49.6.5 Low-Level Perception and Action

The centrality or otherwise of low-level perception and action in human intelligence is a matter of ongoing debate in the AI community. Some feel that the essence of intelligence lies in cognition and/or language, with perception and action having the status of "peripheral devices." Others feel that modeling the physical world and one's actions in it is the essence of intelligence, with cognition and language emerging as side-effects of these more fundamental capabilities. The CogPrime architecture doesn't need to take sides in this debate. Currently we are experimenting both in virtual worlds, and with real-world robot control. The value added by robotic versus virtual embodiment can thus be explored via experiment rather than theory, and may reveal nuances that no one currently foresees.

As noted above, we are unconfident of CogPrime's generic procedure learning or pattern recognition algorithms in terms of their capabilities to handle large amounts of raw sensorimotor data in real time, and so for robotic applications we advocate hybridizing CogPrime with a separate (but closely cross-linked) system better customized for this sort of data, in line with our general hypothesis that *Hybridization of one's integrative neural-symbolic system with a spatiotemporally hierarchical deep learning system is an effective way to handle representation*

EFTA00624631

and learning of low-level sensorimotor knowledge. While this general principle doesn't depend on any particular approach, DeSTIN is one example of a deep learning system of this nature that can be effective in this context

We have not yet done any sophisticated experiments in this regard – our current experiments using OpenCog to control robots involve cruder integration of OpenCog with perceptual and motor subsystems, rather than the tight hybridization described in Chapter 26. Creating such a hybrid system is "just" a matter of software engineering, but testing such a system may lead to many surprises!

49.6.6 Goals

Given that we have characterized general intelligence as "the ability to achieve complex goals in complex environments," it should be plain that goals play a central role in our work. However, we have chosen not to create a separate subsystem for intentional knowledge, and instead have concluded that *one effective way to handle goals is to represent them declaratively, and allocate attention among them economically.* An advantage of this approach is that it automatically provides integration between the goal system and the declarative and attentional knowledge systems.

Goals and subgoals are related using logical links as interpreted and manipulated by PLN, and attention is allocated among goals using the STI dynamics of ECAN, and a specialized variant based on RFS's (requests for service). Thus the mechanics of goal management is handled using uncertain inference and artificial economics, whereas the figuring-out of how to achieve goals is done integratively, relying heavily on procedural and episodic knowledge as well as PLN and ECAN.

The combination of ECAN and PLN seems to overcome the well-known shortcomings found with purely neural-net or purely inferential approaches to goals. Neural net approaches generally have trouble with abstraction, whereas logical approaches are generally poor at real-time responsiveness and at tuning their details quantitatively based on experience. At least in principle, our hybrid approach overcomes all these shortcomings; though of current, it has been tested only in fairly simple cases in the virtual world.

49.7 Fulfilling the "Cognitive Equation"

A key claim based on the notion of the "Cognitive Equation" posited in *Chaotic Logic* [Goe94] is that *it is important for an intelligent system to have some way of recognizing large-scale patterns in itself, and then embodying these patterns as new, localized knowledge items in its memory.* This dynamic introduces a feedback dynamic between emergent pattern and substrate, which is hypothesized to be critical to general intelligence under feasible computational resources. It also ties in nicely with the notion of "glocal memory" – essentially positing a localization of some global memories, which naturally will result in the formation of some glocal memories. One of the key ideas underlying the CogPrime design is that *given the use of a neural-symbolic network for knowledge representation, a graph-mining based "map formation" heuristic is one good way to do this.*

EFTA00624632

Map formation seeks to fulfill the Cognitive Equation quite directly, probably more directly than happens in the brain. Rather than relying on other cognitive processes to implicitly recognize overall system patterns and embody them in the system as localized memories (though this implicit recognition may also happen), the MapFormation MindAgent explicitly carries out this process. Mostly this is done using fairly crude greedy pattern mining heuristics, though if really subtle and important patterns seem to be there, more sophisticated methods like evolutionary pattern mining may also be invoked.

It seems possible that this sort of explicit approach could be less efficient than purely implicit approaches; but, there is no evidence for this, and it may actually provide *increased* efficiency. And in the context of the overall CogPrime design, the explicit MapFormation approach seems most natural.

49.8 Occam's Razor

The key role of "Occam's Razor" or the urge for simplicity in intelligence has been observed by many before (going back at least to Occam himself, and probably earlier!), and is fully embraced in the CogPrime design. Our theoretical analysis of intelligence, presented in Chapter 2 of Part 1 and elsewhere, portrays intelligence as closely tied to the creation of procedures that achieve goals in environments *in the simplest possible way*. And this quest for simplicity is present in many places throughout the CogPrime design, for instance

- In MOSES and hillclimbing, where program compactness is an explicit component of program tree fitness
- In PLN, where the backward and forward chainers explicitly favor shorter proof chains, and intensional inference explicitly characterizes entities in terms of their patterns (where patterns are defined as compact characterizations)
- In pattern mining heuristics, which search for compact characterizations of data
- In the forgetting mechanism, which seeks the smallest set of Atoms that will allow the regeneration of a larger set of useful Atoms via modestly-expensive application of cognitive processes
- Via the encapsulation of procedural and declarative knowledge in simulations, which in many cases provide a vastly compacted form of storing real-world experiences

Like cognitive synergy and emergent networks, Occam's Razor is not something that is implemented in a single place in the CogPrime design, but rather an overall design principle that underlies nearly every part of the system.

49.8.1 Mind Geometry

The three mind-geometric principles outlined in Appendix **??** are:

- syntax-semantics correlation
- cognitive geometrodynamics
- cognitive synergy

EFTA00624633

?

The key role of syntax-semantics correlation in CogPrime is clear. It plays an explicit role in MOSES. In PLN, it is critical to inference control, to the extent that inference control is based on the extraction of patterns from previous inferences. The syntactic structures are the inference trees, and the semantic structures are the inferential conclusions produced by the trees. History-guided inference control assumes that prior similar trees will be a good starting-point for getting results similar to prior ones – i.e. it assumes a reasonable degree of syntax-semantics correlation. Also, without a correlation between the core elements used to generate an episode, and the whole episode, it would be infeasible to use historical data mining to understand what core elements to use to generate a new episode – and creation of compact, easily manipulable seeds for generating episodes would not be feasible.

Cognitive geometrodynamics is about finding the shortest path from the current state to a goal state, where distance is judged by an appropriate metric including various aspects of computational effort. The ECAN and effort management frameworks attempt to enforce this, via minimizing the amount of effort spent by the system in getting to a certain conclusion. MindAgents operating primarily on one kind of knowledge (e.g. MOSES, PLN) may for a time seek to follow the shortest paths within their particular corresponding memory spaces; but then when they operate more interactively and synergetically, it becomes a matter of finding short paths in the composite mindspace corresponding to the combination of the various memory types.

Finally, cognitive synergy is thoroughly and subtly interwoven throughout CogPrime. In a way the whole design is about cognitive synergy – it's critical for the design's functionality that *it's important that the cognitive processes associated with different kinds of memory can appeal to each other for assistance in overcoming bottlenecks in a manner that: a) works in "real time", i.e. on the time scale of the cognitive processes internal processes; b) enables each cognitive process to act in a manner that is sensitive to the particularities of each others' internal representations.*

Recapitulating in a bit more depth, recall that another useful way to formulate cognitive synergy as follows. Each of the key learning mechanisms underlying CogPrime is susceptible to combinatorial explosions. As the problems they confront become larger and larger, the performance gets worse and worse at an exponential rate, because the number of combinations of items that must be considered to solve the problems grows exponentially with the problem size. This could be viewed as a deficiency of the fundamental design, but we don't view it that way. Our view is that combinatorial explosion is intrinsic to intelligence. The task at hand is to dampen it sufficiently that realistically large problems can be solved, rather than to eliminate it entirely. One possible way to dampen it would be to design a single, really clever learning algorithm – one that was still susceptible to an exponential increase in computational requirements as problem size increases, but with a surprisingly small exponent. Another approach is the mirrorhouse approach: Design a bunch of learning algorithms, each focusing on different aspects of the learning process, and design them so that they each help to dampen each others' combinatorial explosions. This is the approach taken within CogPrime. The component algorithms are clever on their own – they are less susceptible to combinatorial explosion than many competing approaches in the narrow-AI literature. But the real meat of the design lies in the intended interactions between the components, manifesting cognitive synergy.

EFTA00624634

49.9 Cognitive Synergy

To understand more specifically how cognitive synergy works in CogPrime, in the following subsections we will review some synergies related to the key components of CogPrime as discussed above. These synergies are absolutely critical to the proposed functionality of the CogPrime system. Without them, the cognitive mechanisms are not going to work adequately well, but are rather going to succumb to combinatorial explosions. The other aspects of CogPrime - the cognitive architecture, the knowledge representation, the embodiment framework and associated developmental teaching methodology – are all critical as well, but none of these will yield the critical emergence of intelligence without cognitive mechanisms that effectively scale. And, in the absence of cognitive mechanisms that effectively scale *on their own*, we must rely on cognitive mechanisms that *effectively help each other to scale*. The reasons why we believe these synergies will exist are essentially qualitative: we have not proved theorems regarded these synergies, and we have observed them in practice only in simple cases so far. However, we do have some ideas regarding how to potentially prove theorems related to these synergies, and some of these are described in Appendix H.

49.9.1 Synergies that Help Inference

The combinatorial explosion in PLN is obvious: forward and backward chaining inference are both fundamentally explosive processes, reined in only by pruning heuristics. This means that for nontrivial complex inferences to occur, one needs **really, really clever** pruning heuristics. The CogPrime design combines simple heuristics with pattern mining, MOSES and economic attention allocation as pruning heuristics. Economic attention allocation assigns importance levels to Atoms, which helps guide pruning. Greedy pattern mining is used to search for patterns in the stored corpus of inference trees, to see if there are any that can be used as analogies for the current inference. And MOSES comes in when there is not enough information (from importance levels or prior inference history) to make a choice, yet exploring a wide variety of available options is unrealistic. In this case, MOSES tasks may be launched, pertinently to the leaves at the fringe of the inference tree, under consideration for expansion. For instance, suppose there is an Atom A at the fringe of the inference tree, and its importance hasn't been assessed with high confidence, but a number of items B are known so that:

```
MemberLink A B
```

Then, MOSES may be used to learn various relationships characterizing A, based on recognizing patterns across the set of B that are suspected to be members of A. These relationships may then be used to assess the importance of A more confidently, or perhaps to enable the inference tree to match one of the patterns identified by pattern mining on the inference tree corpus. For example, if MOSES figures out that:

```
SimilarityLink G A
```

then it may happen that substituting G in place of A in the inference tree, results in something that pattern mining can identify as being a good (or poor) direction for inference.

EFTA00624635

49.10 Synergies that Help MOSES

MOSES's combinatorial explosion is obvious: the number of possible programs of size N increases very rapidly with N. The only way to get around this is to utilize prior knowledge, and as much as possible of it. When solving a particular problem, the search for new solutions must make use of prior candidate solutions evaluated for that problem, and also prior candidate solutions (including successful and unsuccessful ones) evaluated for other related problems.

But, extrapolation of this kind is in essence a contextual analogical inference problem. In some cases it can be solved via fairly straightforward pattern mining; but in subtler cases it will require inference of the type provided by PLN. Also, attention allocation plays a role in figuring out, for a given problem A, which problems B are likely to have the property that candidate solutions for B are useful information when looking for better solutions for A.

49.10.1 Synergies that Help Attention Allocation

Economic attention allocation, without help from other cognitive processes, is just a very simple process analogous to "activation spreading" and "Hebbian learning" in a neural network. The other cognitive processes are the things that allow it to more sensitively understand the attentional relationships between different knowledge items (e.g. which sorts of items are often usefully thought about in the same context, and in which order).

49.10.2 Further Synergies Related to Pattern Mining

Statistical, greedy pattern mining is a simple process, but it nevertheless can be biased in various ways by other, more subtle processes.

For instance, if one has learned a population of programs via MOSES, addressing some particular fitness function, then one can study which items tend to be utilized in the same programs in this population. One may then direct pattern mining to find patterns combining these items found to be in the MOSES population. And conversely, relationships denoted by pattern mining may be used to probabilistically bias the models used within MOSES.

Statistical pattern mining may also help PLN by supplying it with information to work on. For instance, conjunctive pattern mining finds conjunctions of items, which may then be combined with each other using PLN, leading to the formation of more complex predicates. These conjunctions may also be fed to MOSES as part of an initial population for solving a relevant problem.

Finally, the main interaction between pattern mining and MOSES/PLN is that the former may recognize patterns in links created by the latter. These patterns may then be fed back into MOSES and PLN as data. This virtuous cycle allows pattern mining and the other, more expensive cognitive processes to guide each other. Attention allocation also gets into the game, by guiding statistical pattern mining and telling it which terms (and which combinations) to spend more time on.

EFTA00624636

49.10.3 Synergies Related to Map Formation

The essential synergy regarding map formation is obvious: Maps are formed based on the HebbianLinks created via PLN and simpler attentional dynamics, which are based on which Atoms are usefully used together, which is based on the dynamics of the cognitive processes doing the "using." On the other hand, once maps are formed and encapsulated, they feed into these other cognitive processes. This synergy in particular is critical to the emergence of self and attention.

What has to happen, for map formation to work well, is that the cognitive processes must *utilize* encapsulated maps in a way that gives rise overall to relatively clear clusters in the network of HebbianLinks. This will happen if the encapsulated maps are not too complex for the system's other learning operations to understand. So, there must be useful coordinated attentional patterns whose corresponding encapsulated-map Atoms are not too complicated. This has to do with the system's overall parameter settings, but largely with the settings of the attention allocation component. For instance, this is closely tied in with the limited size of "attentional focus" (the famous 7 +/- 2 number associated with humans' and other mammals short term memory capacity). If only a small number of Atoms are typically very important at a given point in time, then the maps formed by grouping together all simultaneously highly important things will be relatively small predicates, which will be easily reasoned about - thus keeping the "virtuous cycle" of map formation and comprehension going effectively.

49.11 Emergent Structures and Dynamics

We have spent much more time in this book on the engineering of cognitive processes and structures, than on the cognitive processes and structures that must *emerge* in an intelligent system for it to display human-level AGI. However, this focus should not be taken to represent a lack of appreciation for the importance of emergence. Rather, it represents a practical focus: engineering is what we must do to create a software system potentially capable of AGI, and emergence is then what happens inside the engineered AGI to allow it to achieve intelligence. Emergence must however be taken carefully into account when deciding what to engineer!

One of the guiding ideas underlying the CogPrime design is that *an AGI system with adequate mechanisms for handling the key types of knowledge mentioned above, and the capability to explicitly recognize large-scale pattern in itself, should* **upon sustained interaction with an appropriate environment in pursuit of appropriate goals**, *emerge a variety of complex structures in its internal knowledge network, including (but not limited to): a hierarchical network, representing both a spatiotemporal hierarchy and an approximate "default inheritance" hierarchy, cross-linked; a heterarchical network of associativity, roughly aligned with the hierarchical network; a self network which is an approximate micro image of the whole network; and inter-reflecting networks modeling self and others, reflecting a "mirrorhouse" design pattern.*

The dependence of these posited emergences on the environment and goals of the AGI system should not be underestimated. For instance, PLN and pattern mining don't have to lead to a hierarchical structured Atomspace, but if the AGI system is placed in an environment which is itself hierarchically structured, then they very likely will do so. And if this environment consists of hierarchically structured *language and culture*, then what one has is a system of minds with hierarchical networks, each reinforcing the hierarchality of each others' networks. Similarly,

EFTA00624637

integrated cognition doesn't have to lead to mirrorhouse structures, but integrated cognition about situations involving other minds studying and predicting and judging each other, is very likely to do so. What is needed for appropriate emergent structures to arise in a mind, is mainly that the knowledge representation is sufficiently flexible to allow these structures, and the cognitive processes are sufficiently intelligent to observe these structures in the environment and then mirror them internally. Of course, it also doesn't hurt if the internal structures and processes are at least slightly *biased* toward the origination of the particular high-level emergent structures that are characteristic of the system's environment/goals; and this is indeed the case with CogPrime – biases toward hierarchical, heterarchical, dual and mirrorhouse networks are woven throughout the system design, in a thoroughgoing though not extremely systematic way.

49.12 Ethical AGI

Creating an AGI with guaranteeably ethical behavior seems an infeasible task; but of course, no human is guaranteeably ethical either, and in fact it seems almost guaranteed that in any moderately large group of humans there are going to be some with strong propensities for extremely unethical behaviors, according to any of the standard human ethical codes. One of our motivations in developing CogPrime has been the belief that *an AGI system, if supplied with a commonsensically ethical goal system and an intentional component based on rigorous uncertain inference, should be able to reliably achieve a much higher level of commonsensically ethical behavior than any human being.*

Our explorations in the detailed design of CogPrime's goal system have done nothing to degrade this belief. While we have not yet developed any CogPrime system to the point where experimenting with its ethics is meaningful, based on our understanding of the current design it seems to us that

- a typical CogPrime system will display a much more consistent and less conflicted and confused motivational system than any human being, due to its explicit orientation toward carrying out actions that (based on its knowledge) rationally seem most likely to lead to achievement of its goals
- if a CogPrime system is given goals that are consistent with commonsensical human ethics (say, articulated in natural language), and then educated in an ethics-friendly environment such as a virtual or physical school, then it is reasonable to expect the CogPrime system will ultimately develop an advanced (human adult level or beyond) form of commmonsensical human ethics

Human ethics is itself wracked with inconsistencies, so one cannot expect a rationality-based AGI system to precisely mirror the ethics of any particular human individual or cultural system. But given the degree to which general intelligence represents adaptation to its environment, and interpretation of natural language depends on life history and context, it seems very likely to us that a CogPrime system, if supplied with a human-commonsense-ethics based goal system and then raised by compassionate and intelligent humans in a school-type environment, would arrive at its own variant of human-commonsense-ethics. The AGI system's ethics would then interact with human ethical systems in complex ways, leading to ongoing evolution of both systems and the development of new cultural and ethical patterns. Predicting the future is

EFTA00624638

difficult even in the absence of radical advanced technologies, but our intuition is that this path has the potential to lead to beneficial outcomes for both human and machine intelligence.

49.13 Toward Superhuman General Intelligence

Human-level AGI is a difficult goal, relative to the current state of scientific understanding and engineering capability, and most of this book has been focused on our ideas about how to achieve it. However, we also suspect the CogPrime architecture has the ultimate potential to push beyond the human level in many ways. As part of this suspicion we advance the claim that *once sufficiently advanced, a CogPrime system should be able to radically self-improve via a variety of methods, including supercompilation and automated theorem-proving.*

Supercompilation allows procedures to be automatically replaced with equivalent but massively more time-efficient procedures. This is particularly valuable in that it allows AI algorithms to learn new procedures without much heed to their efficiency, since supercompilation can always improve the efficiency afterwards. So it is a real boon to automated program learning.

Theorem-proving is difficult for current narrow-AI systems, but for an AGI system with a deep understanding of the context in which each theorem exists, it should be much easier than for human mathematicians. So we envision that ultimately an AGI system will be able to design itself new algorithms and data structures via proving theorems about which ones will best help it achieve its goals in which situations, based on mathematical models of itself and its environment. Once this stage is achieved, it seems that machine intelligence may begin to vastly outdo human intelligence, leading in directions we cannot now envision.

While such projections may seem science-fictional, we note that the CogPrime architecture explicitly supports such steps. If human-level AGI is achieved within the CogPrime framework, it seems quite feasible that profoundly self-modifying behavior could be achieved fairly shortly thereafter. For instance, one could take a human-level CogPrime system and teach it computer science and mathematics, so that it fully understood the reasoning underlying its own design, and the whole mathematics curriculum leading up the the algorithms underpinning its cognitive processes.

49.13.1 Conclusion

What we have sought to do in these pages is, mainly,

- to articulate a theoretical perspective on general intelligence, according to which the creation of a human-level AGI doesn't require anything *that* extraordinary, but "merely" an appropriate combination of closely interoperating algorithms operating on an appropriate multi-type memory system, utilized to enable a system in an appropriate body and environment to figure out how to achieve its given goals
- to describe a software design (CogPrime) that, according to this somewhat mundane but theoretically quite well grounded vision of general intelligence, appears likely (according to a combination of rigorous and heuristic arguments) to be able to lead to human-level AGI using feasible computational resources

EFTA00624639

- to describe some of the preliminary lessons we've learned via implementing and experimenting with aspects of the CogPrime design, in the OpenCog system

In this concluding chapter we have focused on the "combination of rigorous and heuristic arguments" that lead us to consider it likely that CogPrime has the potential to lead to human-level AGI using feasible computational resources.

We also wish to stress that *not all of our arguments and ideas need to be 100% correct in order for the project to succeed*. The quest to create AGI is a mix of theory, engineering, and scientific and unscientific experimentation. If the current CogPrime design turns out to have significant shortcomings, yet still brings us a significant percentage of the way toward human-level AGI, the results obtained along the path will very likely give us clues about how to tweak the design to more effectively get the rest of the way there. And the OpenCog platform is extremely flexible and extensible, rather than being tied to the particular details of the CogPrime design. While we do have faith that the CogPrime design as described here has human-level AGI potential, we are also pleased to have a development strategy and implementation platform that will allow us to modify and improve the design in whatever suggestions are made by our ongoing experimentation.

Many great achievements in history have seemed more magical before their first achievement than afterwards. Powered flight and spaceflight are the most obvious examples, but there are many others such as mobile telephony, prosthetic limbs, electronically deliverable books, robotic factory workers, and so on. We now even have wireless transmission of power (one can recharge cellphones via wifi), though not yet as ambitiously as Tesla envisioned. We very strongly suspect that human-level AGI is in the same category as these various examples: an exciting and amazing achievement, which however is achievable via systematic and careful application of fairly mundane principles. We believe computationally feasible human-level intelligence is both *complicated* (involving many interoperating parts, each sophisticated in their own right) and *complex* (in the sense of involving many emergent dynamics and structures whose details are not easily predictable based on the parts of the system) ... but that neither the complication nor the complexity is an obstacle to engineering human-level AGI.

Furthermore, while ethical behavior is a complex and subtle matter for humans *or* machines, we believe that the production of human-level AGIs that are not only intelligent but also *beneficial* to humans and other biological sentiences, is something that is probably tractable to achieve based on a combination of careful AGI design and proper AGI education and "parenting." One of the motivations underlying our design has been to create an artificial mind that has broadly humanlike intelligence, yet has a more rational and self-controllable motivational system than humans, thus ultimately having the potential for a greater-than-human degree of ethical reliability alongside its greater-than-human intelligence.

In our view, what is needed to create human-level AGI is not a new scientific breakthrough, nor a miracle, but "merely" a sustained effort over a number of years by a moderate-sized team of appropriately-trained professionals, completing the implementation of the design in this book and then parenting and educating the resulting implemented system. CogPrime is by no means the only possible path to human-level AGI, but we believe it is considerably more fully thought-through and fleshed-out than any available alternatives. Actually, we would love to see CogPrime and a dozen alternatives simultaneously pursued – this may seem ambitious, but it would cost a fraction of the money currently spent on other sorts of science or engineering, let alone the money spent on warfare or decorative luxury items. We strongly suspect that, in hindsight, our human and digital descendants will feel amazed that their predecessors allocated

EFTA00624640

so few financial and attentional resources to the creation of powerful AGI, and consequently took *so long* to achieve such a fundamentally straightforward thing.

EFTA00624641

Chapter 50
Build Me Something I Haven't Seen: A CogPrime Thought Experiment

50.1 Introduction

AGI design necessarily leads one into some rather abstract spaces – but being a human-like intelligence in the everyday world is a pretty concrete thing. If the CogPrime research program is successful, it will result not just in abstract ideas and equations, but rather in real AGI robots carrying out tasks in the world, and AGI agents in virtual worlds and online digital spaces conducting important business, doing science, entertaining and being entertained by us, and so forth. With this in mind, in this final chapter we will bring the discussion closer to the concrete and everyday, and pursue a thought experiment of the form "How would a completed CogPrime system carry out this specific task?"

The task we will use for this thought-experiment is one we have used as a running example now and then in the preceding chapters. We consider the case of a robotically or virtually embodied CogPrime system, operating in a preschool type environment, interacting with a human whom it already knows and given the task of "Build me something with blocks that I haven't seen before."

This target task is fairly simple, but it is complex enough to involve essentially every one of CogPrime's processes, interacting in a unified way. It involves simple, grounded creativity of the sort that normal human children display every day – and which, we conjecture, is structurally and dynamically basically the same as the creativity underlying the genius of adult human creators like Einstein, Dali, Dostoevsky, Hendrix, and so forth ... and as the creativity that will power massively capable genius machines in future.

We will consider the case of a simple interaction based on the above task where:

1. The human teacher tells the CogPrime agent "Build me something with blocks that I haven't seen before."
2. After a few false starts, the agent builds something it thinks is appropriate and says "Do you like it?"
3. The human teacher says "It's beautiful. What is it?"
4. The agent says "It's a car man" [and indeed, the construct has 4 wheels and a chassis vaguely like a car, but also a torso, arms and head vaguely like a person]

Of course, a complex system like CogPrime could carry out an interaction like this internally in many different ways, and what is roughly described here is just one among many possibilities.

EFTA00624642

First we will enumerate a number of CogPrime processes and explain some ways that each one may help CogPrime carry out the target task. Then we will give a more evocative narrative, conveying the dynamics that would occur in CogPrime while carrying out the target task, and mentioning how each of the enumerated cognitive processes as it arises in the narrative.

50.2 Roles of Selected Cognitive Processes

Now we review a number of the more interesting CogPrime cognitive processes mentioned in previous chapters of the book, for each one indicating one or more of the roles it might play in helping a CogPrime system carry out the target task. Note that this list is incomplete in many senses, e.g. it doesn't list all the cognitive processes, nor all the roles played by the ones listed. The purpose is to give an evocative sense of the roles played by the different parts of the design in carrying out the task.

- Chapter 19 (OpenCog Framework)
  - **Freezing/defrosting**.
    - When the agent builds a structure from blocks and decides it's not good enough to show off to the teacher, what does it do with the detailed ideas and thought process underlying the structure it built? If it doesn't like the structure so much, it may just leave this to the generic forgetting process. But if it likes the structure a lot, it may want to increase the VLTI (Very Long Term Importance) of the Atoms related to the structure in question, to be sure that these are stored on disk or other long-term storage, even after they're deemed sufficiently irrelevant to be pushed out of RAM by the forgetting mechanism.
    - When given the target task, the agent may decide to revive from disk the mind-states it went through when building crowd-pleasing structures from blocks before, so as to provide it with guidance.
- Chapter 22 (Emotion, Motivation, Attention and Control)
  - **Cognitive cycle**.
    - While building with blocks, the agent's cognitive cycle will be dominated by perceiving, acting on, and thinking about the blocks it is building with.
    - When interacting with the teacher, then interaction-relevant linguistic, perceptual and gestural processes will also enter into the cognitive cycle.
  - **Emotion**. The agent's emotions will fluctuate naturally as it carries out the task.
    - If it has a goal of pleasing the teacher, then it will experience happiness as its expectation of pleasing the teacher increases.
    - If it has a goal of experiencing novelty, then it will experience happiness as it creates structures that are novel in its experience.
    - If it has a goal of learning, then it will experience happiness as it learns new things about blocks construction.
    - On the other hand, it will experience unhappiness as its experienced or predicted satisfaction of these goals decreases.
  - **Action selection**

EFTA00624643

- In dialoguing with the teacher, action selection will select one or more DialogueController schema to control the conversational interaction (based on which DC schema have proved most effective in prior similar situations.
  - When the agent wants to know the teacher's opinion of its construct, what is happening internally is that the "please teacher" Goal Atom gets a link of the conceptual form (Implication "find out teacher's opinion of my current construct" "please teacher"). This link may be created by PLN inference, probably largely by analogy to previously encountered similar situations. Then, GoalImportance is spread from the "please teacher" Goal Atom to the "find out teacher's opinion of my current construct" Atom (via the mechanism of sending an RFS package to the latter Atom). More inference causes a link (Implication "ask the teacher for their opinion of my current construct" "find out teacher's opinion of my current construct") to be formed, and the "ask the teacher for their opinion of my current construct" Atom to get GoalImportance also. Then PredicateSchematization causes the predicate "ask the teacher for their opinion of my current construct" to get turned into an actionable schema, which gets GoalImportance, and which gets pushed into the ActiveSchemaPool via Goal-driven action selection. Once the schema version of "ask the teacher for their opinion of my current construct" is in the ActiveSchemaPool, it then invokes natural language generation Tasks, which lead to the formulation of an English sentence such as "Do you like it?"
  - When the teacher asks "It's beautiful. What is it?", then the NL comprehension MindAgent identifies this as a question, and the "please teacher" Goal Atom gets a link of the conceptual form (Implication "answer the question the teacher just asked" "please teacher"). This follows simply from the knowledge (Implication ("teacher has just asked a question" AND "I answer the teacher's question") ("please teacher")), or else from more complex knowledge refining this Implication. From this point, things proceed much as in the case "Do you like it?" described just above.
- Consider a schema such as "pick up a red cube and place it on top of the long red block currently at the top of the structure" (let's call this P). Once P is placed in the ActiveSchemaPool, then it runs and generates more specific procedures, such as the ones needed to find a red cube, to move the agent's arm toward the red cube and grasp it, etc. But the execution of these specific low-level procedures is done via the ExecutionManager, analogously to the execution of the specifics of generating a natural language sentence from a collection of semantic relationships. Loosely speaking, reaching for the red cube and turning simple relationships into a simple sentences, are considered as "automated processes" not requiring holistic engagement of the agent's mind. What the generic, more holistic Action Selection mechanism does in the present context is to figure out to put P in the ActiveSchemaPool in the first place. This occurs because of a chain such as: P predictively implies (with a certain probabilistic weight) "completion of the car-man structure", which in turn predictively implies "completion of a structure that is novel to the teacher," which in turn predictively implies "please the teacher," which in turn implies "please others," which is assumed an Ubergoal (a top-level system goal).

EFTA00624644

  - **Goal Atoms**. As the above items make clear, the scenario in question requires the initial Goal Atoms to be specialized, via the creation of more and more particular subgoals suiting the situation at hand.
  - **Context Atoms**.
    - Knowledge of the context the agent is in can help it disambiguate language it hears, e.g. knowing the context is blocks-building helps it understand which sense of the word "blocks" is meant.
    - On the other hand, if the context is that the teacher is in a bad mood, then the agent might know via experience that in this context, the strength of (Implication "ask the teacher for their opinion of my current construct" "find out teacher's opinion of my current construct") is lower than in other contexts.
  - **Context formation**.
    - A context like blocks-building or "teacher in a bad mood" may be formed by clustering over multiple experience-sets, i.e. forming Atoms that refer to spatiotemporally grouped sets of percepts/concepts/actions, and grouping together similar Atoms of this nature into clusters.
    - The Atom referring to the cluster of experience-sets involving blocks-building will then survive as an Atom if it gets involved in relationships that are important or have surprising truth values. If many relationships have significantly different truth-value inside the blocks-building context than outside it, this means it's likely that the blocks-building ConceptNode will remain as an Atom with reasonably high LTI, so it can be used as a context in future.
  - **Time-dependence of goals**. Many of the agent's goals in this scenario have different importances over different time scales. For instance "please the teacher" is important on multiple time-scales: the agent wants to please the teacher in the near term but also in the longer term. But a goal like "answer the question the teacher just asked" has an intrinsic time-scale to it; if it's not fulfilled fairly rapidly then its importance goes away.

- Chapter 23 (Attention allocation)
  - **ShortTermImportance versus LongTermImportance**. While conversing, the concepts and immediately involved in the conversation (including the Atoms describing the agents in the conversation) have very high STI. While building, Atoms representing to the blocks and related ideas about the structures being built (e.g. images of cars and people perceived or imagined in the past) have very high STI. But the reason these Atoms are in RAM prior to having their STI boosted due to their involvement in the agent's activities, is because they had their LTI boosted at some point in the past. And after these Atoms leave the AttentionalFocus and their STI reduces, they will have boosted LTI and hence likely remain in RAM for a long while, to be involved in "background thought", and in case they're useful in the AttentionalFocus again.
  - **HebbianLink formation**. As a single example, the car-man has both wheels and arms, so now a Hebbian association between wheels and arms will exist in the agent's memory, to potentially pop up again and guide future thinking. The very idea of a car-man likely emerged partly due to previously formed HebbianLinks – because people were often seen sitting in cars, the association between person and car existed, which made the car concept and the human concept natural candidates for blending.
  - **Data mining the System Activity Table**. The HebbianLinks mentioned above may have been formed via mining the SystemActivityTable

EFTA00624645

  - **ECAN based associative memory**. When the agent thinks about making a car, this spreads importance to various Atoms related to the car concept, and one thing this does is lead to the emergence of the car attractor into the AttentionalFocus. The different aspects of a car are represented by heavily interlinked Atoms, so that when some of them become important, there's a strong tendency for the others to also become important – and for "car" to then emerge as an attractor of importance dynamics.
  - **Schema credit assignment**.
    - Suppose the agent has a subgoal of placing a certain blue block on top of a certain red block. It may use a particular motor schema for carrying out this action – involving, for instance, holding the blue block above the red block and then gradually lowering it. If this schema results in success (rather than in, say, knocking down the red block), then it should get rewarded via having its STI and LTI boosted and also having the strength of the link between it and the subgoal increased.
    - Next, suppose that a certain cognitive schema (say, the schema of running multiple related simulations and averaging the results, to estimate the success probability of a motor procedure) was used to arrive at the motor schema in question. Then this cognitive schema may get passed some importance from the motor schema, and it will get the strength of its link to the goal increased. In this way credit passes backwards from the goal to the various schema directly or indirectly involved in fulfilling it.
  - **Forgetting**. If the agent builds many structures from blocks during its lifespan, it will accumulate a large amount of perceptual memory.

- Chapter 24 (Goal and Action Selection). Much of the use of the material in this chapter was covered above in the bullet point for Chapter 22, but a few more notes are:
  - **Transfer of RFS between goals**. Above it was noted that the link (Implication "ask the teacher for their opinion of my current construct" "find out teacher's opinion of my current construct") might be formed and used as a channel for GoalImportance spreading.
  - **Schema Activation**. Supposing the agent is building a man-car, it may have car-building schema and man-building schema in its ActiveSchemaPool at the same time, and it may enact both of them in an interleaved manner. But if each tend to require two hands for their real-time enaction, then schema activation will have to pass back and forth between the two of them, so that at any one time, one is active whereas the other one is sitting in the ActiveSchemaPool waiting to get activated.
  - **Goal Based Schema Learning**. To take a fairly low-level example, suppose the agent has the (sub)goal of making an arm for a blocks-based person (or man-car), given the presence of a blocks-based torso. Suppose it finds a long block that seems suitable to be an arm. It then has the problem of figuring out how to attach the arm to the body. It may try out several procedures in its internal simulation world, until it finds one that works: *hold the arm in the right position while one end of it rests on top of some block that is part of the torso, then place some other block on top of that end, then slightly release the arm and see if it falls. If it doesn't fall, leave it. If it seems about to fall, then place something heavier atop it, or shove it further in toward the center of the torso.* The procedure learning process could be MOSES here, or it could be PLN.

- Chapter 25 (Procedure Evaluation)

EFTA00624646

  - **Inference Based Procedure Evaluation**. A procedure for man-building such as "first put up feet, then put up legs, then put up torso, then put up arms and head" may be synthesized from logical knowledge (via predicate schematization) but without filling in the details of how to carry out the individual steps, such as "put up legs." If a procedure with abstract (ungrounded) schema like PutUpTorso is chosen for execution and placed into the ActiveSchemaPool, then in the course of execution, inferential procedure evaluation must be used to figure out how to make the abstract schema actionable. The GoalDrivenActionSelection MindAgent must make the choice whether to put a not-fully-grounded schema into the ActiveSchemaPool, rather than grounding it first and then making it active; this is the sort of choice that may be made effectively via learned cognitive schema.

- Chapter 26 (Perception and Action)
  - **ExperienceDB**. No person remembers every blocks structure they ever saw or built, except maybe some autists. But a CogPrime can store all this information fairly easily, in its ExperienceDB, even if it doesn't keep it all in RAM in its AtomSpace. It can also store everything anyone ever said about blocks structures in its vicinity.
  - **Perceptual Pattern Mining**.
  - **Object Recognition**. Recognizing structures made of blocks as cars, people, houses, etc. requires fairly abstract object recognition, involving identifying the key shapes and features involved in an object-type, rather than just going by simple visual similarity.
  - **Hierarchical Perception Networks**. If the room is well-lit, it's easy to visually identify individual blocks within a blocks structure. If the room is darker, then more top-down processing may be needed – identifying the overall shape of the blocks structure may guide one in making out the individual blocks.
  - **Hierarchical Action Networks**. Top-down action processing tells the agent that, if it wants to pick up a block, it should move its arm in such a way as to get its hand near the block, and then move its hand. But if it's still learning how to do that sort of motion, more likely it will do this, but then start moving its its hand and find that it's hard to get a grip on the block – and then have to go back and move its arm a little differently. Iterating between broader arm/hand movements and more fine-grained hand/finger movements is an instance of information iteratively passing up and down a hierarchical action network.
  - **Coupling of Perception and Action Networks**. Picking up a block in the dark is a perfect example of rich coupling of perception and action networks. Feeling the block with the fingers helps with identifying blocks that can't be clearly seen.

- Chapter 30 (Procedure Learning)
  - **Specification Based Procedure Learning**.
    - Suppose the agent has never seen a horse, but the teacher builds a number of blocks structures and calls them horses, and draws a number of pictures and calls them horses. This may cause a procedure learning problem to be spawned, where the fitness function is accuracy at distinguishing horses from non-horses.
    - Learning to pick up a block is specification-based procedure learning, where the specification is to pick up the block and grip it and move it without knocking down the other stuff near the block.
  - **Representation Building**.

EFTA00624647

    - In the midst of building a procedure to recognize horses, MOSES would experimentally vary program nodes recognizing visual features into other program nodes recognizing other visual features
    - In the midst of building a procedure to pick up blocks, MOSES would experimentally vary program nodes representing physical movements into other nodes representing physical movements
    - In both of these cases, MOSES would also carry out the standard experimental variations of mathematical and control operators according to its standard representation-building framework

- Chapter 31 (Imitative, Reinforcement and Corrective Learning)
  - **Reinforcement Learning**.
    - Motor procedures for placing blocks (in simulations or reality) will get rewarded if they don't result in the blocks structure falling down, punished otherwise.
    - Procedures leading to the teacher being pleased, in internal simulations (or in repeated trials of scenarios like the one under consideration), will get rewarded; procedures leading to the teacher being displeased will get punished.
  - **Imitation Learning**. If the agent has seen others build with blocks before, it may summon these memories and then imitate the actions it has seen others take.
  - **Corrective Learning**. This would occur if the teacher intervened in the agent's block-building and guided him physically – e.g. steadying his shaky arm to prevent him from knocking the blocks structure over.

- Chapter 32 (Hillclimbing)
  - **Complexity Penalty**. In learning procedures for manipulating blocks, the complexity penalty will militate against procedures that contain extraneous steps.

- Chapter 33 (Probabilistic Evolutionary Procedure Learning)
  - **Supplying Evolutionary Learning with Long-Term Memory**. Suppose the agent has previously built people from clay, but never from blocks. It may then have learned a "classification model" predicting which clay people will look appealing to humans, and which won't. It may then transfer this knowledge, using PLN, to form a classification model predicting which blocks-people will look appealing to humans, and which won't.
  - **Fitness Function Estimation via Integrative Intelligence**. To estimate the fitness of a procedure for, say, putting an arm on a blocks-built human, the agent may try out the procedure in the internal simulation world; or it may use PLN inference to reason by analogy to prior physical situations it's observed. These allow fitness to be estimated without actually trying out the procedure in the environment.

- Chapter 34 (Probabilistic Logic Networks)
  - **Deduction**. This is a tall skinny structure; tall skinny structures fall down easily; thus this structure may fall down easily.
  - **Induction**. This teacher is talkative; this teacher is friendly; therefore the talkative are generally friendly.
  - **Abduction**. This structure has a head and arms and torso; a person has a head and arms and torso; therefore this structure is a person.

EFTA00624648

  - **PLN forward chaining**. What properties might a car-man have, based on inference from the properties of cars and the properties of men?
  - **PLN backward chaining**.
    - An inference target might be: *Find X so that X looks something like a wheel and can be attached to this blocks-chassis, and I can find four fairly similar copies.*
    - Or: *Find the truth value of the proposition that this structure looks like a car.*
  - **Indefinite truth values**. Consider the deductive inference "This is a tall skinny structure; tall skinny structures fall down easily; thus this structure may fall down easily." In this case, the confidence of the second premise may be greater than the confidence of the first premise, which may result in an intermediate confidence for the conclusion, according to the propagation of indefinite probabilities through the PLN deduction rule.
  - **Intensional inference**. Is the blocks-structure a person? According to the definition of intensional inheritance, it shares many informative properties with people (e.g. having arms, torso and head), so to a significant extent, it is a person.
  - **Confidence decay**. The agent's confidence in propositions regarding building things with blocks should remain nearly constant. The agent's confidence in propositions regarding the teacher's taste should decay more rapidly. This should occur because the agent should observe that, in general, propositions regarding physical object manipulation tend to retain fairly constant truth value, whereas propositions regarding human tastes tend to have more rapidly decaying truth value.

- Chapter 35 (Spatiotemporal Inference)
  - **Temporal reasoning**. Suppose, after the teacher asks "What is it?", the agent needs to think a while to figure out a good answer. But maybe the agent knows that it's rude to pause too long before answering something to a direct question. Temporal reasoning helps figure out "how long is too long" to wait before answering.
  - **Spatial reasoning**. Suppose the agent puts shoes on the wheels of the car. This is a joke relying on the understanding that wheels hold a car up, whereas feet hold a person up, and the structure is a car-man. But it also relies on the spatial inferences that: the car's wheels are in the right position for the man's feet (below the torso); and, the wheels are below the car's chassis just like a person's feet are below its torso.

- Chapter 36 (Inference Control)
  - **Evaluator Choice as a Bandit Problem**. In doing inference regarding how to make a suitably humanlike arm for the blocks-man, there may be a choice between multiple inference pathways, perhaps one that relies on analogy to other situations building arms, versus one that relies on more general reasoning about lengths and weights of blocks. The choice between these two pathways will be made randomly with a certain probabilistic bias assigned to each one, via prior experience.
  - **Inference Pattern Mining**. The probabilities used in choosing which inference path to take, are determined in part by prior experience – e.g. maybe it's the case that in prior situations of building complex blocks structures, analogy has proved a better guide than naive physics, thus the prior probability of the analogy inference pathway will be nudged up.
  - **PLN and Bayes Nets**. What's the probability that the blocks-man's hat will fall off if the man-car is pushed a little bit to simulate driving? This question could be resolved in many ways (e.g. by internal simulation), but one possibility is inference. If

EFTA00624649

this is resolved by inference, it's the sort of conditional probability calculation that could potentially be done faster if a lot of the probabilistic knowledge from the AtomSpace were summarized in a Bayes Net. Updating the Bayes net structure can be slow, so this is probably not appropriate for knowledge that is rapidly shifting; but knowledge about properties of blocks structures may be fairly persistent after the agent has gained a fair bit of knowledge by playing with blocks a lot.

- Chapter 37 (Pattern Mining)
  - **Greedy Pattern Mining**.
    - "Push a tall structure of blocks and it tends to fall down" is the sort of repetitive pattern that could easily be extracted from a historical record of perceptions and (the agent's and others') actions via simple greedy pattern mining algorithm.
    - If there is a block that is shaped like a baby's rattle, with a long slender handle and then a circular shape at the end, then greedy pattern mining may be helpful due to having recognized the pattern that structures like this are sometimes rattles – and also that structures like this are often stuck together, with the handle part connected sturdily to the circular part.
  - **Evolutionary Pattern Mining**. "Push a tall structure of blocks with a wide base and a gradual narrowing toward the top and it may not fall too badly" is a more complex pattern that may not be found via greedy mining, unless the agent has dealt with a lot of pyramids.
- Chapter 38 (Concept Formation)
  - **Formal Concept Analysis**. Suppose there are many long, slender blocks of different colors and different shapes (some cylindrical, some purely rectangular for example). Learning this sort of concept based on common features is exactly what FCA is good at (and when the features are defined fuzzily or probabilistically, it's exactly what uncertain FCA is good at). Learning the property of "slender" itself is another example of something uncertain FCA is good at – it would learn this if there were many concepts that preferentially involved slender things (even though formed on the basis of concepts other than slenderness)
  - **Conceptual Blending**. The concept of a "car-man" or "man-car" is an obvious instance of conceptual blending. The agents know that building a man won't surprise the teacher, and nor will building a car ... but both "man" and "car" may pop to the forefront of its mind (i.e. get a briefly high STI) when it thinks about what to build. But since it knows it has to do something new or surprising, there may be a cognitive schema that boosts the amount of funds to the ConceptBlending MindAgent, causing it to be extra-active. In any event, the ConceptBlending agent seeks to find ways to combine important concepts; and then PLN explores these to see which ones may be able to achieve the given goal of surprising the teacher (which includes subgoals such as actually being buildable).
- Chapter 39 (Dimensional Embedding)
  - **Dimensional Embedding**. When the agent needs to search its memory for a previously seen blocks structure similar to the currently observed one – or for a previously articulated thought similar to the one it's currently trying to articulate – then it needs to to a search through its large memory for "an entity similar to X" (where X is a

EFTA00624650

structure or a thought). This kind of search can be quite computationally difficult – but if the entities in question have been projected into an embedding space, then it's quite rapid. (The cost is shifted to the continual maintenance of the embedding space, and its periodic updating; and there is some error incurred in the projection, but in many cases this error is not a show-stopper.)
  - **Embedding Based Inference Control**. Rapid search for answers to similarity or inheritance queries can be key for guiding inference in appropriate directions; for instance reasoning about how to build a structure with certain properties, can benefit greatly from rapid search for previously-encountered substructures currently structurally or functionally similar to the substructures one desires to build.

- Chapter 40 (Simulation and Episodic Memory)
  - **Fitness Estimation via Simulation** . One way to estimate whether a certain blocks structure is likely to fall down or not, is to build it in one's "mind's eye" and see if the physics engine in one's mind's-eye causes it to fall down. This is something that in many cases will work better for CogPrime than for humans, because CogPrime has a more mathematically accurate physics engine than the human mind does; however, in cases that rely heavily on naive physics rather than, say, direct applications of Newton's Laws, then CogPrime's simulation engine may underperform the typical human mind.
  - **Concept Formation via Simulation** . Objects may be joined into categories using uncertain FCA, based on features that they are identified to have via "simulation experiments" rather than physical world observations. For instance, it may be observed that pyramid-shaped structures fall less easily than pencil-shaped tower structures – and the concepts corresponding to these two categories may be formed – from experiments run in the internal simulation world, perhaps inspired by isolated observations in the physical world.
  - **Episodic Memory**. Previous situations in which the agent has seen similar structures built, or been given similar problems to solve, may be brought to mind as "episodic movies" playing in the agent's memory. By watching what happens in these replayed episodic movies, the agent may learn new declarative or procedural knowledge about what to do. For example, maybe there was some situation in the agent's past where it saw someone asked to do something surprising, and that someone created something funny. This might (via a simple PLN step) bias the agent to create something now, which it has reason to suspect will cause others to laugh.

- Chapter 41 (Integrative Procedure Learning)
  - **Concept-Driven Procedure Learning**. Learning the concept of "horse", as discussed above in the context of Chapter 30, is an example of this.
  - **Predicate Schematization**. The synthesis of a schema for man-building, as discussed above in the context of Chapter 25, is an example of this.

- Chapter 42 (Map Formation)
  - **Map Formation**. The notion of a car involves many aspects: the physical appearance of cars, the way people get in and out of cars, the ways cars drive, the noises they make, etc. All these aspects are represented by Atoms that are part of the car map, and are richly interconnected via HebbianLinks as well as other links.

EFTA00624651

  - **Map Encapsulation** . The car map forms implicitly via the interaction of multiple cognitive dynamics, especially ECAN. But then the MapEncapsulation MindAgent may do its pattern mining and recognize this map explicitly, and form a PredicateNode encapsulating it. This PredicateNode may then be used in PLN inference, conceptual blending, and so forth (e.g. helping with the formation of a concept like car-man via blending).

- Chapter 44 (Natural Language Comprehension)
  - **Experience Based Disambiguation**. The particular dialogue involved in the present example doesn't require any nontrivial word sense disambiguation. But it does require parse selection, and semantic interpretation selection:
    - In "Build me something with blocks," the agent has no trouble understanding that "blocks" means "toy building blocks" rather than, say, "city blocks", based on many possible mechanisms, but most simply importance spreading.
    - "Build me something with blocks" has at least three interpretations: the building could be carried out using blocks with a tool; or the thing built could be presented alongside blocks; or the thing built could be composed of blocks. The latter is the most commonsensical interpretation for most humans, but that is because we have heard the phrase "building with blocks" used in a similarly grounded way before (as well as other similar phrases such as "playing with Legos", etc., whose meaning helps militate toward the right interpretation via PLN inference and importance spreading). So here we have a simple example of experience-based disambiguation, where experiences at various distances of association from the current one are used to help select the correct parse.
    - A subtler form of semantic disambiguation is involved in interpreting the clause "that I haven't seen before." A literal-minded interpretation would say that this requirement is fulfilled by any blocks construction that's not precisely identical to one the teacher has seen before. But of course, any sensible human knows this is an idiomatic clause that means "significantly different from anything I've seen before." This could be determined by the CogPrime agent if it has heard the idiomatic clause before, or if it's heard a similar idiomatic phrase such as "something I've never done before." Or, even if the agent has never heard such an idiom before, it could potentially figure out the intended meaning simply because the literal-minded interpretation would be a pointless thing for the teacher to say. So if it knows the teacher usually doesn't add useless modificatory clauses onto their statements, then potentially the agent could guess the correct meaning of the phrase.

- Chapter 46 (Language Generation)
  - **Experience-Based Knowledge Selection for Language Generation**. When the teacher asks "What is it?", the agent must decide what sort of answer to give. Within the confines of the QuestionAnswering DialogueController, the agent could answer "A structure of blocks", or "A part of the physical world", or "A thing", or "Mine." (Or, if it were running another DC, it could answer more broadly, e.g. "None of your business," etc.). However, the QA DC tells it that, in the present context, the most likely desired answer is one that the teacher doesn't already know; and the most important property of the structure that the teacher doesn't obviously already know is the fact that it

EFTA00624652

depicts a "car man." Also, memory of prior conversations may bring up statements like "It's a horse" in reference to a horse built of blocks, or a drawing of a horse, etc.

- **Experience-Based Guidance of Word and Syntax Choice** . The choice of phrase "car man" requires some choices to be made. The agent could just as well say "It's a man with a car for feet" or "It's a car with a human upper body and head" or "It's a car centaur," etc. A bias toward simple expressions would lead to "car man." If the teacher were known to prefer complex expressions, then the agent might be biased toward expressing the idea in a different way.

- Chapter 48 (Natural Language Dialogue)
  - **Adaptation of Dialogue Controllers**. The QuestionAsking and QuestionAnswering DialogueControllers both get reinforcement from this interaction, for the specific internal rules that led to the given statements being made.

50.3 A Semi-Narrative Treatment

Now we describe how a CogPrime system might carry out the specified task in a semi-narrative form, weaving in the material from the previous section as we go along, and making some more basic points as well. The semi-narrative covers most but not all of the bullet points from the previous section, but with some of the technical details removed; and it introduces a handful of new examples not given in the bullet points.

The reason this is called a semi-narrative rather than a narrative is that there is no particular linear order to the processes occurring in each phase of the situation described here. CogPrime's internal cognitive processes do not occur in a linear narrative; rather, what we have is a complex network of interlocking events. But still, describing some of these events concretely in a manner correlated with the different stages of a simple interaction, may have some expository value.

The human teacher tells the CogPrime agent "Build me something with blocks that I haven't seen before."

Upon hearing this, the agent's cognitive cycles are dominated by language processing and retrieval from episodic and sensory memory.

The agent may decide to revive from disk the mind-states it went through when building human-pleasing structures from blocks before, so as to provide it with guidance

It will likely experience the emotion of happiness, because it anticipates the pleasure of getting rewarded for the task in future.

The ubergoal of pleasing the teacher gets active (gets funded significantly with STI currency), as it becomes apparent there are fairly clear ways of fulfilling that goal (via the subgoal S of building blocks structures that will get positive response from the teacher). Other ubergoals like gaining knowledge are not funded as much with STI currency just now, as they are not immediately relevant.

Action selection, based on ImplicationLinks derived via PLN (between various possible activities and the subgoal S) causes it to start experimentally building some blocks structures. Past experience with building (turned into ImplicationLinks via mining the SystemActivityTable) tells it that it may want to build a little bit in its internal simulation world before building in the external world, causing STI currently to flow to the simulation MindAgent.

EFTA00624653

The Atom corresponding to the context blocks-building gets high STI and is pushed into the AttentionalFocus, making it likely that many future inferences will occur in this context. Other Atoms related to this one also get high STI (the ones in the blocks-building map, and others that are especially related to blocks-building in this particular context).

After a few false starts, the agent builds something it thinks is appropriate and says "Do you like it?"

Now that the agent has decided what to do to fulfill its well-funded goal, its cognitive cycles are dominated by action, perception and related memory access and concept creation.

An obvious subgoal is spawned: build a new structure now, and make this particular structure under construction appealing and novel to the teacher. This subgoal has a shorter time scale than the high level goal. The subgoal gets some currency from its supergoal using the mechanism of RFS spreading.

Action selection must tell it when to continue building the same structure and when to try a new one, as well as more micro level choices.

Atoms related to the currently pursued blocks structure get high STI.

After a failed structure (a "false start") is disassembled, the corresponding Atoms lose STI dramatically (leaving AF) but may still have significant LTI, so they can be recalled later as appropriate. They may also have VLTI so they will be saved to disk later on if other things push them out of RAM due to getting higher LTI.

Meanwhile everything that's experienced from the external world goes into the ExperienceDB.

Atoms representing different parts of aspects of the same blocks structure will get Hebbian links between them, which will guide future reasoning and importance spreading.

Importance spreading helps the system go from an idea for something to build (say, a rock or a car) to the specific plans and ideas about how to build it, via increasing the STI of the Atoms that will be involved in these plans and ideas.

If something apparently good is done in building a blocks structure, then other processes and actions that helped lead to or support that good thing, get passed some STI from the Atoms representing the good thing, and also may get linked to the Goal Atom representing "good" in this context. This leads to reinforcement learning.

The agent may play with building structures and then seeing what they most look like, thus exercising abstract object recognition (that uses procedures learned by MOSES or hillclimbing, or uncertain relations learned by inference, to guess what object category a given observed collection of percepts most likely falls into).

Since the agent has been asked to come up with something surprising, it knows it should probably try to formulate some new concepts – because it has learned in the past, via SystemActivityTable mining, that often newly formed concepts are surprising to others. So, more STI currency is given to concept formation MindAgents, such as the ConceptualBlending Mind Agent (which, along with a lot of stuff that gets thrown out or stored for later use, comes up with "car-man").

When the notion of "car" is brought to mind, the distributed map of nodes corresponding to "car" get high STI. When car-man is formed, it is reasoned about (producing new Atoms), but it also serves as a nexus of importance-spreading, causing the creation of a distributed car-man map.

If the goal of making an arm for a man-car occurs, then goal-driven schema learning may be done to learn a procedure for arm-making (where the actual learning is done by MOSES or hill-climbing).

EFTA00624654

If the agent is building a man-car, it may have man-building and car-building schema in its ActiveSchemaPool at the same time, and SchemaActivation may spread back and forth between the different modules of these two schema.

If the agent wants to build a horse, but has never seen a horse made of blocks (only various pictures and movies of horses), it may uses MOSES or hillclimbing internally to solve the problem of creating a horse-recognizer or a horse-generator which embodies appropriate abstract properties of horses. Here as in all cases of procedure learning, a complexity penalty rewards simpler programs, from among all programs that approximately fulfill the goals of the learning process.

If a procedure being executed has some abstract parts, then these may be executed by inferential procedure evaluation (which makes the abstract parts concrete on the fly in the course of execution).

To guess the fitness of a procedure for doing something (say, building an arm or recognizing a horse), inference or simulation may be used, as well as direct evaluation in the world.

Deductive, inductive and abductive PLN inference may be used in figuring out what a blocks structure will look or act like like before building it (it's tall and thin so it may fall down; it won't be bilaterally symmetric so it won't look much like a person; etc.)

Backward-chaining inference control will help figure out how to assemble something matching a certain specification e.g. how to build a chassis based on knowledge of what a chassis looks like. Forward chaining inference (critically including intensional relationships) will be used to estimate the properties that the teacher will perceive a given specific structure to have. Spatial and temporal algebra will be used extensively in this reasoning, within the PLN framework.

Coordinating different parts of the body – say an arm and a hand – will involve importance spreading (both up and down) within the hierarchical action network, and from this network to the hierarchical perception network and the heterarchical cognitive network.

In looking up Atoms in the AtomSpace, some have truth values whose confidences have decayed significantly (e.g. those regarding the teacher's tastes), whereas others have confidences that have hardly decayed at all (e.g. those regarding general physical properties of blocks).

Finding previous blocks structures similar to the current one (useful for guiding building by analogy to past experience) may be done rapidly by searching the system's internal dimensional-embedding space.

As the building process occurs, patterns mined via past experience (tall things often fall down) are used within various cognitive processes (reasoning, procedure learning, concept creation, etc.); and new pattern mining also occurs based on the new observations made as different structures are build and experimented with and destroyed.

Simulation of teacher reactions, based on inference from prior examples, helps with the evaluation of possible structures, and also of procedures for creating structures.

As the agent does all this, it experiences the emotion of curiosity (likely among other emotions), because as it builds each new structure it has questions about what it will look like and how the teacher would react to it.

The human teacher says "It's beautiful. What is it?" The agent says "It's a car man"

Now that the building is done and the teacher says something, the agent's cognitive cycles are dominated by language understanding and generation. The Atom representing the context of talking to the teacher gets high STI, and is used as the context for many ensuing inferences.

EFTA00624655

Comprehension of "it" uses anaphor resolution based on a combination of ECAN and PLN inference based on a combination of previously interpreted language and observation of the external world situation.

The agent experiences the emotion of happiness because the teacher has called its creation beautiful, which is recognizes as a positive evaluation – so the agent knows one of its ubergoals ("please the teacher") has been significantly fulfilled.

The goal of pleasing the teacher causes the system to want to answer the question. So the QuestionAnswering DialogueController schema gets paid a lot and gets put into the ActiveSchemaPool. In reaction to the question asked, this DC chooses a semantic graph to speak, then invokes NL generation to say it.

NL generation chooses the most compact expression that seems to adequately convey the intended meaning, so it decides on "car man" as the best simple verbalization to match the newly created conceptual blend that it thinks effectively describes the newly created blocks structure.

The positive feedback from the user leads to reinforcement of the Atoms and processes that led to the construction of the blocks structure that has been judged beautiful (via importance spreading and SystemActivityTable mining).

50.4 Conclusion

The simple situation considered in this chapter is complex enough to involve nearly all the different cognitive processes in the CogPrime system – and many interactions between these processes. This fact illustrates one of the main difficulties of designing, building and testing an artificial mind like CogPrime – until nearly all of the system is build and made to operate in an integrated way, it's hard to do any meaningful test of the system. Testing PLN or MOSES or conceptual blending in isolation may be interesting computer science, but it doesn't tell you much about CogPrime as a design for a thinking machine.

According to the CogPrime approach, getting a simple child-like interaction like "build me something with blocks that I haven't seen before" to work properly requires a holistic, integrated cognitive system. Once one has built a system capable of this sort of simple interaction then, according to the theory underlying CogPrime, one is not that far from a system with adult human-level intelligence. And once one has an adult human-level AGI built according to a highly flexible design like CogPrime, given the potential of such systems to self-analyze and self-modify, one is not far off from a dramatically powerful Genius Machine. Of course there will be a lot of work to do to get from a child-level system to an adult-level system – it won't necessarily unfold as "automatically" as seems to happen with a human child, because CogPrime lacks the suite of developmental processes and mechanisms that the young human brain has. But still, a child CogPrime mind capable of doing the things outlined in this chapter will have all the basic components and interactions in place, all the ones that are needed for a much more advanced artificial mind.

Of course, one could concoct a narrow-AI system carrying out the specific activities described in this chapter, much more simply than one could build a CogPrime system capable of doing these activities. But that's not the point – the point of this chapter is not to explain how to achieve some particular narrow set of activities "by any means necessary", but rather to explain

EFTA00624656

how these activities might be achieved within the CogPrime framework, which has been designed with much more generality in mind.

It would be worthwhile to elaborate a number of other situations similar to the one described in this chapter, and to work through the various cognitive processes and structures in CogPrime carefully in the context of each of these situations. In fact this sort of exercise has frequently been carried out informally in the context of developing CogPrime. But this book is already long enough, so we will end here, and leave the rest for future works – emphasizing that it is via intimate interplay between concrete considerations like the ones presented in this chapter, and general algorithmic and conceptual considerations as presented in most of the chapters of this book, that we have the greatest hope of creating advanced AGI. The value of this sort of interplay actually follows from the theory of real-world general intelligence presented in Part 1 of the book. Thoroughly general intelligence is only possible given unrealistic computational resources, so real-world general intelligence is about achieving high generality given limited resources relative to the specific classes of environments relevant to a given agent. Specific situations like building surprising things with blocks are particularly important insofar as they embody broader information about the classes of environments relevant to broadly *human-like* general intelligence.

No doubt, once a CogPrime system is completed, the specifics of its handling of the situation described here will differ somewhat from the treatment presented in this chapter. Furthermore, the final CogPrime system may differ algorithmically and structurally in some respects from the specifics given in this book – it would be surprising if the process of building, testing and interacting with CogPrime didn't teach us some new things about various of the topics covered. But our conjecture is that, if sufficient effort is deployed appropriately, then a system much like the CogPrime system described in this book will be able to handle the situation described in this chapter in a roughly similar manner to the one described in this chapter – and that this will serve as a natural precursor to much more dramatic AGI achievements.

EFTA00624657

Appendix A
Glossary

: :

A.1 List of Specialized Acronyms

This includes acronyms that are commonly used in discussing CogPrime, OpenCog and related ideas, plus some that occur here and there in the text for relatively ephemeral reasons.

- **AA**: Attention Allocation
- **ADF**: Automatically Defined Function (in the context of Genetic Programming)
- **AF**: Attentional Focus
- **AGI**: Artificial General Intelligence
- **AV**: Attention Value
- **BD**: Behavior Description
- **C-space**: Configuration Space
- **CBV**: Coherent Blended Volition
- **CEV**: Coherent Extrapolated Volition
- **CGGP**: Contextually Guided Greedy Parsing
- **CSDLN**: Compositional Spatiotemporal Deep Learning Network
- **CT**: Combo Tree
- **ECAN**: Economic Attention Network
- **ECP**: Embodied Communication Prior
- **EPW** : Experiential Possible Worlds (semantics)
- **FCA**: Formal Concept Analysis
- **FI** : Fisher Information
- **FIM**: Frequent Itemset Mining
- **FOI**: First Order Inference
- **FOPL**: First Order Predicate Logic
- **FOPLN**: First Order PLN
- **FS-MOSES**: Feature Selection MOSES (i.e. MOSES with feature selection integrated a la LIFES)
- **GA**: Genetic Algorithms

EFTA00624658

- **GB**: Global Brain
- **GEOP**: Goal Evaluator Operating Procedure (in a GOLEM context)
- **GIS**: Geospatial Information System
- **GOLEM**: Goal-Oriented LEarning Meta-architecture
- **GP**: Genetic Programming
- **HOI**: Higher-Order Inference
- **HOPLN**: Higher-Order PLN
- **HR**: Historical Repository (in a GOLEM context)
- **HTM**: Hierarchical Temporal Memory
- **IA**: (Allen) Interval Algebra (an algebra of temporal intervals)
- **IRC**: Imitation / Reinforcement / Correction (Learning)
- **LIFES**: Learning-Integrated Feature Selection
- **LTI**: Long Term Importance
- **MA**: MindAgent
- **MOSES**: Meta-Optimizing Semantic Evolutionary Search
- **MSH**: Mirror System Hypothesis
- **NARS**: Non-Axiomatic Reasoning System
- **NLGen**: A specific software component within OpenCog, which provides one way of dealing with Natural Language Generation
- **OCP**: OpenCogPrime
- **OP**: Operating Program (in a GOLEM context)
- **PEPL**: Probabilistic Evolutionary Procedure Learning (e.g. MOSES)
- **PLN**: Probabilistic Logic Networks
- **RCC**: Region Connection Calculus
- **RelEx**: A specific software component within OpenCog, which provides one way of dealing with natural language Relationship Extraction
- **SAT**: Boolean SATisfaction, as a mathematical / computational problem
- **SMEPH**: Self-Modifying Evolving Probabilistic Hypergraph
- **SRAM**: Simple Realistic Agents Model
- **STI**: Short Term Importance
- **STV**: Simple Truth VAlue
- **TV**: Truth Value
- **VLTI**: Very Long Term Importances
- **WSPS**: Whole-Sentence Purely-Syntactic Parsing

A.2 Glossary of Specialized Terms

- **Abduction**: A general form of inference that goes from data describing something to a hypothesis that accounts for the data. Often in an OpenCog context, this refers to the PLN abduction rule, a specific First-Order PLN rule (If A implies C, and B implies C, then maybe A is B), which embodies a simple form of abductive inference. But OpenCog may also carry out abduction, as a general process, in other ways.
- **Action Selection**: The process via which the OpenCog system chooses which Schema to enact, based on its current goals and context.
- **Active Schema Pool**: The set of Schema currently in the midst of Schema Execution.

EFTA00624659

- **Adaptive Inference Control**: Algorithms or heuristics for guiding PLN inference, that cause inference to be guided differently based on the context in which the inference is taking place, or based on aspects of the inference that are noted as it proceeds.
- **AGI Preschool**: A virtual world or robotic scenario roughly similar to the environment within a typical human preschool, intended for AGIs to learn in via interacting with the environment and with other intelligent agents.
- **Atom**: The basic entity used in OpenCog as an element for building representations. Some Atoms directly represent patterns in the world or mind, others are components of representations. There are two kinds of Atoms: Nodes and Links.
- **Atom, Frozen**: See Atom, Saved
- **Atom, Realized**: An Atom that exists in RAM at a certain point in time.
- **Atom, Saved**: An Atom that has been saved to disk or other similar media, and is not actively being processed.
- **Atom, Serialized**: An Atom that is serialized for transmission from one software process to another, or for saving to disk, etc.
- **Atom2Link**: A part of OpenCogPrime
s language generation system, that transforms appropriate Atoms into words connected via link parser link types.
- **Atomspace**: A collection of Atoms, comprising the central part of the memory of an OpenCog instance.
- **Attention**: The aspect of an intelligent system's dynamics focused on guiding which aspects of an OpenCog system's memory & functionality gets more computational resources at a certain point in time
- **Attention Allocation**: The cognitive process concerned with managing the parameters and relationships guiding what the system pays attention to, at what points in time. This is a term inclusive of Importance Updating and Hebbian Learning.
- **Attentional Currency**: Short Term Importance and Long Term Importance values are implemented in terms of two different types of artificial money, STICurrency and LTICurrency. Theoretically these may be converted to one another.
- **Attentional Focus**: The Atoms in an OpenCog Atomspace whose ShortTermImportance values lie above a critical threshold (the AttentionalFocus Boundary). The Attention Allocation subsystem treats these Atoms differently. Qualitatively, these Atoms constitute the system's main focus of attention during a certain interval of time, i.e. it's a moving bubble of attention.
- **Attentional Memory**: A system's memory of what it's useful to pay attention to, in what contexts. In CogPrime this is managed by the attention allocation subsystem.
- **Backward Chainer**: A piece of software, wrapped in a MindAgent, that carries out backward chaining inference using PLN.
- **CIM-Dynamic**: Concretely-Implemented Mind Dynamic, a term for a cognitive process that is implemented explicitly in OpenCog (as opposed to allowed to emerge implicitly from other dynamics). Sometimes a CIM-Dynamic will be implemented via a single MindAgent, sometimes via a set of multiple interrelated MindAgents, occasionally by other means.
- **Cognition**: In an OpenCog context, this is an imprecise term. Sometimes this term means any process closely related to intelligence; but more often it's used specifically to refer to more abstract reasoning/learning/etc, as distinct from lower-level perception and action.
- **Cognitive Architecture**: This refers to the logical division of an AI system like OpenCog into interacting parts and processes representing different conceptual aspects of intelligence.

EFTA00624660

It's different from the software architecture, though of course certain cognitive architectures and certain software architectures fit more naturally together.

- **Cognitive Cycle**: The basic "loop" of operations that an OpenCog system, used to control an agent interacting with a world, goes through rapidly each "subjective moment." Typically a cognitive cycle should be completed in a second or less. It minimally involves perceiving data from the world, storing data in memory, and deciding what if any new actions need to be taken based on the data perceived. It may also involve other processes like deliberative thinking or metacognition. Not all OpenCog processing needs to take place within a cognitive cycle.
- **Cognitive Schematic**: An implication of the form "Context AND Procedure IMPLIES goal". Learning and utilization of these is key to CogPrime's cognitive process.
- **Cognitive Synergy**: The phenomenon by which different cognitive processes, controlling a single agent, work together in such a way as to help each other be more intelligent. Typically, if one has cognitive processes that are individually susceptible to combinatorial explosions, cognitive synergy involves coupling them together in such a way that they can help one another overcome each other's internal combinatorial explosions. The CogPrime design is reliant on the hypothesis that its key learning algorithms will display dramatic cognitive synergy when utilized for agent control in appropriate environments.
- **CogPrime** : The name for the AGI design presented in this book, which is designed specifically for implementation within the OpenCog software framework (and this implementation is OpenCogPrime).
- **CogServer**: A piece of software, within OpenCog, that wraps up an Atomspace and a number of MindAgents, along with other mechanisms like a Scheduler for controlling the activity of the MindAgents, and code for important and exporting data from the Atomspace.
- **Cognitive Equation**: The principle, identified in Ben Goertzel's 1994 book "Chaotic Logic", that minds are collections of pattern-recognition elements, that work by iteratively recognizing patterns in each other and then embodying these patterns as new system elements. This is seen as distinguishing mind from "self-organization" in general, as the latter is not so focused on continual pattern recognition. Colloquially this means that "a mind is a system continually creating itself via recognizing patterns in itself."
- **Combo**: The programming language used internally by MOSES to represent the programs it evolves. SchemaNodes may refer to Combo programs, whether the latter are learned via MOSES or via some other means. The textual realization of Combo resembles LISP with less syntactic sugar. Internally a Combo program is represented as a program tree.
- **Composer**: In the PLN design, a rule is denoted a composer if it needs premises for generating its consequent. See generator.
- **CogBuntu**: an Ubuntu Linux remix that contains all required packages and tools to test and develop OpenCog.
- **Concept Creation**: A general term for cognitive processes that create new ConceptNodes, PredicateNodes or concept maps representing new concepts.
- **Conceptual Blending**: A process of creating new concepts via judiciously combining pieces of old concepts. This may occur in OpenCog in many ways, among them the explicit use of a ConceptBlending MindAgent, that blends two or more ConceptNodes into a new one.
- **Confidence**: A component of an OpenCog/PLN TruthValue, which is a scaling into the interval [0,1] of the weight of evidence associated with a truth value. In the simplest case (of a probabilistic Simple Truth Value), one uses confidence c = n / (n+k), where n is

EFTA00624661

the weight of evidence and k is a parameter. In the case of an Indefinite Truth Value, the confidence is associated with the width of the probability interval.

- **Confidence Decay**: The process by which the confidence of an Atom decreases over time, as the observations on which the Atom's truth value is based become increasingly obsolete. This may be carried out by a special MindAgent. The rate of confidence decay is subtle and contextually determined, and must be estimated via inference rather than simply assumed a priori.
- **Consciousness**: CogPrime is not predicated on any particular conceptual theory of consciousness. Informally, the AttentionalFocus is sometimes referred to as the "conscious" mind of a CogPrime system, with the rest of the Atomspace as "unconscious" but this is just an informal usage, not intended to tie the CogPrime design to any particular theory of consciousness. The primary originator of the CogPrime
design (Ben Goertzel) tends toward panpsychism, as it happens.
- **Context**: In addition to its general common-sensical meaning, in CogPrime the term Context also refers to an Atom that is used as the first argument of a ContextLink. The second argument of the ContextLink then contains Links or Nodes, with TruthValues calculated restricted to the context defined by the first argument. For instance, (ContextLink USA (InheritanceLink person obese)).
- **Core**: The MindOS portion of OpenCog, comprising the Atomspace, the CogServer, and other associated "infrastructural" code.
- **Corrective Learning**: When an agent learns how to do something, by having another agent explicitly guide it in doing the thing. For instance, teaching a dog to sit by pushing its butt to the ground.
- **CSDLN**: (Compositional Spatiotemporal Deep Learning Network): A hierarchical pattern recognition network, in which each layer corresponds to a certain spatiotemporal granularity, the nodes on a given layer correspond to spatiotemporal regions of a given size, and the children of a node correspond to sub-regions of the region the parent corresponds to. Jeff Hawkins's HTM is one example CSDLN, and Itamar Arel's DeSTIN (currently used in OpenCog) is another.
- **Declarative Knowledge**: Semantic knowledge as would be expressed in propositional or predicate logic facts or beliefs.
- **Deduction**: In general, this refers to the derivation of conclusions from premises using logical rules. In PLN in particular, this often refers to the exercise of a specific inference rule, the PLN Deduction rule (A $\rightarrow$ B, B $\rightarrow$ C, therefore A$\rightarrow$ C)
- **Deep Learning**: Learning in a network of elements with multiple layers, involving feedforward and feedback dynamics, and adaptation of the links between the elements. An example deep learning algorithm is DeSTIN, which is being integrated with OpenCog for perception processing.
- **Defrosting**: Restoring, into the RAM portion of an Atomspace, an Atom (or set thereof) previously saved to disk.
- **Demand**: In CogPrime's OpenPsi subsystem, this term is used in a manner inherited from the Psi model of motivated action. A Demand in this context is a quantity whose value the system is motivated to adjust. Typically the system wants to keep the Demand between certain minimum and maximum values. An Urge develops when a Demand deviates from its target range.
- **Deme**: In MOSES, an "island" of candidate programs, closely clustered together in program space, being evolved in an attempt to optimize a certain fitness function. The idea is that

EFTA00624662

within a deme, programs are generally similar enough that reasonable syntax-semantics correlation obtains.

- **Derived Hypergraph**: The SMEPH hypergraph obtained via modeling a system in terms of a hypergraph representing its internal states and their relationships. For instance, a SMEPH vertex represents a collection of internal states that habitually occur in relation to similar external situations. A SMEPH edge represents a relationship between two SMEPH vertices (e.g. a similarity or inheritance relationship). The terminology "edge /vertex" is used in this context, to distinguish from the "link / node" terminology used in the context of the Atomspace.
- **DeSTIN – Deep SpatioTemporal Inference Network**: A specific CSDLN created by Itamar Arel, tested on visual perception, and appropriate for integration within CogPrime.
- **Dialogue**: Linguistic interaction between two or more parties. In a CogPrime context, this may be in English or another natural language, or it may be in Lojban or Psynese.
- **Dialogue Control**: The process of determining what to say at each juncture in a dialogue. This is distinguished from the linguistic aspects of dialogue, language comprehension and language generation. Dialogue control applies to Psynese or Lojban, as well as to human natural language.
- **Dimensional Embedding**: The process of embedding entities from some non-dimensional space (e.g. the Atomspace) into an n-dimensional Euclidean space. This can be useful in an AI context because some sorts of queries (e.g. "find everything similar to X", "find a path between X and Y") are much faster to carry out among points in a Euclidean space, than among entities in a space with less geometric structure.
- **Distributed Atomspace**: An implementation of an Atomspace that spans multiple computational processes; generally this is done to enable spreading an Atomspace across multiple machines.
- **Dual Network**: A network of mental or informational entities with both a hierarchical structure and a heterarchical structure, and an alignment among the two structures so that each one helps with the maintenance of the other. This is hypothesized to be a critical emergent structure, that must emerge in a mind (e.g. in an Atomspace) in order for it to achieve a reasonable level of human-like general intelligence (and possibly to achieve a high level of pragmatic general intelligence in any physical environment).
- **Efficient Pragmatic General Intelligence**: A formal, mathematical definition of general intelligence (extending the pragmatic general intelligence), that ultimately boils down to: the ability to achieve complex goals in complex environments using limited computational resources (where there is a specifically given weighting function determining which goals and environments have highest priority). More specifically, the definition weighted-sums the system's normalized goal-achieving ability over (goal, environment pairs), and where the weights are given by some assumed measure over (goal, environment pairs), and where the normalization is done via dividing by the (space and time) computational resources used for achieving the goal.
- **Elegant Normal Form (ENF)**: Used in MOSES, this is a way of putting programs in a normal form while retaining their hierarchical structure. This is critical if one wishes to probabilistically model the structure of a collection of programs, which is a meaningful operation if the collection of programs is operating within a region of program space where syntax-semantics correlation holds to a reasonable degree. The Reduct library is used to place programs into ENF.

EFTA00624663

- **Embodied Communication Prior**: The class of prior distributions over (goal, environment pairs), that are imposed by placing an intelligent system in an environment where most of its tasks involve controlling a spatially localized body in a complex world, and interacting with other intelligent spatially localized bodies. It is hypothesized that many key aspects of human-like intelligence (e.g. the use of different subsystems for different memory types, and cognitive synergy between the dynamics associated with these subsystems) are consequences of this prior assumption. This is related to the Mind-World Correspondence Principle.
- **Embodiment**: Colloquially, in an OpenCog context, this usually means the use of an AI software system to control a spatially localized body in a complex (usually 3D) world. There are also possible "borderline cases" of embodiment, such as a search agent on the Internet. In a sense any AI is embodied, because it occupies some physical system (e.g. computer hardware) and has some way of interfacing with the outside world.
- **Emergence**: A property or pattern in a system is emergent if it arises via the combination of other system components or aspects, in such a way that its details would be very difficult (not necessarily impossible in principle) to predict from these other system components or aspects.
- **Emotion**: Emotions are system-wide responses to the system's current and predicted state. Dorner's Psi theory of emotion contains explanations of many human emotions in terms of underlying dynamics and motivations, and most of these explanations make sense in a CogPrime context, due to CogPrime's use of OpenPsi (modeled on Psi) for motivation and action selection.
- **Episodic Knowledge**: Knowledge about episodes in an agent's life-history, or the life-history of other agents. CogPrime includes a special dimensional embedding space only for episodic knowledge, easing organization and recall.
- **Evolutionary Learning**: Learning that proceeds via the rough process of iterated differential reproduction based on fitness, incorporating variations of reproduced entities. MOSES is an explicitly evolutionary-learning-based portion of CogPrime; but CogPrime's dynamics as a whole may also be conceived as evolutionary.
- **Exemplar**: (in the context of imitation learning) - When the owner wants to teach an OpenCog controlled agent a behavior by imitation, he/she gives the pet an exemplar. To teach a virtual pet "fetch" for instance, the owner is going to throw a stick, run to it, grab it with his/her mouth and come back to its initial position.
- **Exemplar**: (in the context of MOSES) – Candidate chosen as the core of a new deme, or as the central program within a deme, to be varied by representation building for ongoing exploration of program space.
- **Explicit Knowledge Representation**: Knowledge representation in which individual, easily humanly identifiable pieces of knowledge correspond to individual elements in a knowledge store (elements that are explicitly there in the software and accessible via very rapid, deterministic operations)
- **Extension**: In PLN, the extension of a node refers to the instances of the category that the node represents. In contrast is the intension.
- **Fishgram (Frequent and Interesting Sub-hypergraph Mining)**: A pattern mining algorithm for identifying frequent and/or interesting sub-hypergraphs in the Atomspace.
- **First-Order Inference (FOI)**: The subset of PLN that handles Logical Links not involving VariableAtoms or higher-order functions. The other aspect of PLN, Higher-Order Inference, uses Truth Value formulas derived from First-Order Inference.

EFTA00624664

- **Forgetting**: The process of removing Atoms from the in-RAM portion of Atomspace, when RAM gets short and they are judged not as valuable to retain in RAM as other Atoms. This is commonly done using the LTI values of the Atoms (removing lowest LTI-Atoms, or more complex strategies involving the LTI of groups of interconnected Atoms). May be done by a dedicated Forgetting MindAgent. VLTI may be used to determine the fate of forgotten Atoms.
- **Forward Chainer**: A control mechanism (MindAgent) for PLN inference, that works by taking existing Atoms and deriving conclusions from them using PLN rules, and then iterating this process. The goal is to derive new Atoms that are interesting according to some given criterion.
- **Frame2Atom**: A simple system of hand-coded rules for translating the output of RelEx2Frame (logical representation of semantic relationships using FrameNet relationships) into Atoms.
- **Freezing**: Saving Atoms from the in-RAM Atomspace to disk.
- **General Intelligence**: Often used in an informal, commonsensical sense, to mean the ability to learn and generalize beyond specific problems or contexts. Has been formalized in various ways as well, including formalizations of the notion of "achieving complex goals in complex environments" and "achieving complex goals in complex environments using limited resources." Usually interpreted as a fuzzy concept, according to which absolutely general intelligence is physically unachievable, and humans have a significant level of general intelligence, but far from the maximally physically achievable degree.
- **Generalized Hypergraph**: A hypergraph with some additional features, such as links that point to links, and nodes that are seen as "containing" whole sub-hypergraphs. This is the most natural and direct way to mathematically/visually model the Atomspace.
- **Generator**: In the PLN design, a rule is denoted a generator if it can produce its consequent without needing premises (e.g. LookupRule, which just looks it up in the AtomSpace). See composer.
- **Global, Distributed Memory**: Memory that stores items as implicit knowledge, with each memory item spread across multiple components, stored as a pattern of organization or activity among them.
- **Glocal Memory**: The storage of items in memory in a way that involves both localized and global, distributed aspects.
- **Goal**: An Atom representing a function that a system (like OpenCog) is supposed to spend a certain non-trivial percentage of its attention optimizing. The goal, informally speaking, is to maximize the Atom's truth value.
- **Goal, Implicit**: A goal that an intelligent system, in practice, strives to achieve; but that is not explicitly represented as a goal in the system's knowledge base.
- **Goal, Explicit**: A goal that an intelligent system explicitly represents in its knowledge base, and expends some resources trying to achieve. Goal Nodes (which may be Nodes or, e.g. ImplicationLinks) are used for this purpose in OpenCog.
- **Goal-Driven Learning**: Learning that is driven by the cognitive schematic i.e. by the quest of figuring out which procedures can be expected to achieve a certain goal in a certain sort of context.
- **Grounded SchemaNode**: See SchemaNode, Grounded.
- **Hebbian Learning**: An aspect of Attention Allocation, centered on creating and updating HebbianLinks, which represent the simultaneous importance of the Atoms joined by the HebbianLink.

EFTA00624665

- **Hebbian Links**: Links recording information about the associative relationship (co-occurrence) between Atoms. These include symmetric and asymmetric HebbianLinks.
- **Heterarchical Network**: A network of linked elements in which the semantic relationships associated with the links are generally symmetrical (e.g. they may be similarity links, or symmetrical associative links). This is one important sort of subnetwork of an intelligent system; see Dual Network.
- **Hierarchical Network**: A network of linked elements in which the semantic relationships associated with the links are generally asymmetrical, and the parent nodes of a node have a more general scope and some measure of control over their children (though there may be important feedback dynamics too). This is one important sort of subnetwork of an intelligent system; see Dual Network.
- **Higher-Order Inference (HOI)**: PLN inference involving variables or higher-order functions. In contrast to First-Order Inference (FOI).
- **Hillclimbing**: A general term for greedy, local optimization techniques, including some relatively sophisticated ones that involve "mildly nonlocal" jumps.
- **Human-Level Intelligence**: General intelligence that's "as smart as" human general intelligence, even if in some respects quite unlike human intelligence. An informal concept, which generally doesn't come up much in CogPrime work, but is used frequently by some other AI theorists.
- **Human-Like Intelligence**: General intelligence with properties and capabilities broadly resembling those of humans, but not necessarily precisely imitating human beings.
- **Hypergraph**: A conventional hypergraph is a collection of nodes and links, where each link may span any number of nodes. OpenCog makes use of generalized hypergraphs (the Atomspace is one of these).
- **Imitation Learning**: Learning via copying what some other agent is observed to do.
- **Implication**: Often refers to an ImplicationLink between two PredicateNodes, indicating an (extensional, intensional or mixed) logical implication.
- **Implicit Knowledge Representation**: Representation of knowledge via having easily humanly identifiable pieces of knowledge correspond to the pattern of organization and/or dynamics of elements, rather than via having individual elements correspond to easily humanly identifiable pieces of knowledge.
- **Importance**: A generic term for the Attention Values associated with Atoms. Most commonly these are STI (short term importance) and LTI (long term importance) values. Other importance values corresponding to various different time scales are also possible. In general an importance value reflects an estimate of the likelihood an Atom will be useful to the system over some particular future time-horizon. STI is generally relevant to processor time allocation, whereas LTI is generally relevant to memory allocation.
- **Importance Decay**: The process of Atom importance values (e.g. STI and LTI) decreasing over time, if the Atoms are not utilized. Importance decay rates may in general be context-dependent.
- **Importance Spreading**: A synonym for Importance Updating, intended to highlight the similarity with "activation spreading" in neural and semantic networks.
- **Importance Updating**: The CIM-Dynamic that periodically (frequently) updates the STI and LTI values of Atoms based on their recent activity and their relationships.
- **Imprecise Truth Value**: Peter Walley's imprecise truth values are intervals [L,U], interpreted as lower and upper bounds of the means of probability distributions in an envelope

EFTA00624666

of distributions. In general, the term may be used to refer to any truth value involving intervals or related constructs, such as indefinite probabilities.

- **Indefinite Probability**: An extension of a standard imprecise probability, comprising a credible interval for the means of probability distributions governed by a given second-order distribution.
- **Indefinite Truth Value**: An OpenCog TruthValue object wrapping up an indefinite probability
- **Induction**: In PLN, a specific inference rule (A → B, A → C, therefore B → C). In general, the process of heuristically inferring that what has been seen in multiple examples, will be seen again in new examples. Induction in the broad sense, may be carried out in OpenCog by methods other than PLN induction. When emphasis needs to be laid on the particular PLN inference rule, the phrase "PLN Induction" is used.
- **Inference**: Generally speaking, the process of deriving conclusions from assumptions. In an OpenCog context, this often refers to the PLN inference system. Inference in the broad sense is distinguished from general learning via some specific characteristics, such as the intrinsically incremental nature of inference: it proceeds step by step.
- **Inference Control**: A cognitive process that determines what logical inference rule (e.g. what PLN rule) is applied to what data, at each point in the dynamic operation of an inference process.
- **Integrative AGI**: An AGI architecture, like CogPrime, that relies on a number of different powerful, reasonably general algorithms all cooperating together. This is different from an AGI architecture that is centered on a single algorithm, and also different than an AGI architecture that expects intelligent behavior to emerge from the collective interoperation of a number of simple elements (without any sophisticated algorithms coordinating their overall behavior).
- **Integrative Cognitive Architecture**: A cognitive architecture intended to support integrative AGI.
- **Intelligence**: An informal, natural language concept. "General intelligence" is one slightly more precise specification of a related concept; "Universal intelligence" is a fully precise specification of a related concept. Other specifications of related concepts made in the particular context of CogPrime research are the pragmatic general intelligence and the efficient pragmatic general intelligence.
- **Intension**: In PLN, the intention of a node consists of Atoms representing properties of the entity the node represents.
- **Intentional memory**: A system's knowledge of its goals and their subgoals, and associations between these goals and procedures and contexts (e.g. cognitive schematics).
- **Internal Simulation World**: A simulation engine used to simulate an external environment (which may be physical or virtual), used by an AGI system as its "mind's eye" in order to experiment with various action‘ q sequences and envision their consequences, or observe the consequences of various hypothetical situations. Particularly important for dealing with episodic knowledge.
- **Interval Algebra**: Allen Interval Algebra, a mathematical theory of the relationships between time intervals. CogPrime utilizes a fuzzified version of classic Interval Algebra.
- **IRC Learning (Imitation, Reinforcement, Correction)**: Learning via interaction with a teacher, involving a combination of imitating the teacher, getting explicit reinforcement signals from the teacher, and having one's incorrect or suboptimal behaviors guided toward betterness by the teacher in real-time. This is a large part of how young humans learn.

EFTA00624667

- **Knowledge Base**: A shorthand for the totality of knowledge possessed by an intelligent system during a certain interval of time (whether or not this knowledge is explicitly represented). Put differently: this is an intelligence's total memory contents (inclusive of all types of memory) during an interval of time.
- **Language Comprehension**: The process of mapping natural language speech or text into a more "cognitive", largely language-independent representation. In OpenCog this has been done by various pipelines consisting of dedicated natural language processing tools, e.g. a pipeline: text → Link Parser → RelEx → RelEx2Frame → Frame2Atom Atomspace; and alternatively a pipeline Link Parser → Link2Atom → Atomspace. It would also be possible to do language comprehension purely via PLN and other generic OpenCog processes, without using specialized language processing tools.
- **Language Generation**: The process of mapping (largely language-independent) cognitive content into speech or text. In OpenCog this has been done by various pipelines consisting of dedicated natural language processing tools, e.g. a pipeline: Atomspace → NLGen → text; or more recently Atomspace → Atom2Link → surface realization → text. It would also be possible to do language generation purely via PLN and other generic OpenCog processes, without using specialized language processing tools.
- **Language Processing**: Processing of human language is decomposed, in CogPrime, into Language Comprehension, Language Generation, and Dialogue Control.
- **Learning**: In general, the process of a system adapting based on experience, in a way that increases its intelligence (its ability to achieve its goals). The theory underlying CogPrime doesn't distinguish learning from reasoning, associating, or other aspects of intelligence.
- **Learning Server**: In some OpenCog configurations, this refers to a software server that performs "offline" learning tasks (e.g. using MOSES or hillclimbing), and is in communication with an Operational Agent Controller software server that performs real-time agent control and dispatches learning tasks to and receives results from the Learning Server.
- **Linguistic Links**: A catch-all term for Atoms explicitly representing linguistic content, e.g. WordNode, SentenceNode, CharacterNode.
- **Link**: A type of Atom, representing a relationship among one or more Atoms. Links and Nodes are the two basic kinds of Atoms.
- **Link Parser**: A natural language syntax parser, created by Sleator and Temperley at Carnegie-Mellon University, and currently used as part of OpenCogPrime's natural language comprehension and natural language generation system.
- **Link2Atom**: A system for translating link parser links into Atoms. It attempts to resolve precisely as much ambiguity as needed in order to translate a given assemblage of link parser links into a unique Atom structure.
- **Lobe**: A term sometimes used to refer to a portion of a distributed Atomspace that lives in a single computational process. Often different lobes will live on different machines.
- **Localized Memory**: Memory that stores each item using a small number of closely-connected elements.
- **Logic**: In an OpenCog context, this usually refers to a set of formal rules for translating certain combinations of Atoms into "conclusion" Atoms. The paradigm case at present is the PLN probabilistic logic system, but OpenCog can also be used together with other logics.
- **Logical Links**: Any Atoms whose truth values are primarily determined or adjusted via logical rules, e.g. PLN's InheritanceLink, SimilarityLink, ImplicationLink, etc. The term isn't usually applied to other links like HebbianLinks whose semantics isn't primarily logic-

EFTA00624668

based, even though these other links can be processed via (e.g. PLN) logical inference via interpreting them logically.

- **Lojban**: A constructed human language, with a completely formalized syntax and a highly formalized semantics, and a small but active community of speakers. In principle this seems an extremely good method for communication between humans and early-stage AGI systems.
- **Lojban++**: A variant of Lojban that incorporates English words, enabling more flexible expression without the need for frequent invention of new Lojban words.
- **Long Term Importance (LTI)**: A value associated with each Atom, indicating roughly the expected utility to the system of keeping that Atom in RAM rather than saving it to disk or deleting it. It's possible to have multiple LTI values pertaining to different time scales, but so far practical implementation and most theory has centered on the option of a single LTI value.
- **LTI**: Long Term Importance
- **Map**: A collection of Atoms that are interconnected in such a way that they tend to be commonly active (i.e. to have high STI, e.g. enough to be in the AttentionalFocus, at the same time).
- **Map Encapsulation**: The process of automatically identifying maps in the Atomspace, and creating Atoms that "encapsulate" them; the Atom encapsulation a map would link to all the Atoms in the map. This is a way of making global memory into local memory, thus making the system's memory glocal and explicitly manifesting the "cognitive equation." This may be carried out via a dedicated MapEncapsulation MindAgent.
- **Map Formation**: The process via which maps form in the Atomspace. This need not be explicit; maps may form implicitly via the action of Hebbian Learning. It will commonly occur that Atoms frequently co-occurring in the AttentionalFocus, will come to be joined together in a map.
- **Memory Types**: In CogPrime
  this generally refers to the different types of memory that are embodied in different data structures or processes in the CogPrime
  architecture, e.g. declarative (semantic), procedural, attentional, intentional, episodic, sensorimotor.
- **Mind-World Correspondence Principle**: The principle that, for a mind to display efficient pragmatic general intelligence relative to a world, it should display many of the same key structural properties as that world. This can be formalized by modeling the world and mind as probabilistic state transition graphs, and saying that the categories implicit in the state transition graphs of the mind and world should be inter-mappable via a high-probability morphism.
- **Mind OS**: A synonym for the OpenCog Core.
- **MindAgent**: An OpenCog software object, residing in the CogServer, that carries out some processes in interaction with the Atomspace. A given conceptual cognitive process (e.g. PLN inference, Attention allocation, etc.) may be carried out by a number of different MindAgents designed to work together.
- **Mindspace**: A model of the set of states of an intelligent system as a geometrical space, imposed by assuming some metric on the set of mind-states. This may be used as a tool for formulating general principles about the dynamics of generally intelligent systems.
- **Modulators**: Parameters in the Psi model of motivated, emotional cognition, that modulate the way a system perceives, reasons about and interacts with the world.

EFTA00624669

- **MOSES (Meta-Optimizing Semantic Evolutionary Search)**: An algorithm for procedure learning, which in the current implementation learns programs in the Combo language. MOSES is an evolutionary learning system, which differs from typical genetic programming systems in multiple aspects including: a subtler framework for managing multiple "demes" or "islands" of candidate programs; a library called Reduct for placing programs in Elegant Normal Form; and the use of probabilistic modeling in place of, or in addition to, mutation and crossover as means of determining which new candidate programs to try.
- **Motoric**: Pertaining to the control of physical actuators, e.g. those connected to a robot. May sometimes be used to refer to the control of movements of a virtual character as well.
- **Moving Bubble of Attention**: The Attentional Focus of a CogPrime system.
- **Natural Language Comprehension**: See Language Comprehension
- **Natural Language Generation**: See Language Generation
- **Natural Language Processing (NLP)**: See Language Processing
- **NLGen**: Software for carrying out the surface realization phase of natural language generation, via translating collections of RelEx output relationships into English sentences. Was made functional for simple sentences and some complex sentences; not currently under active development, as work has shifted to the related Atom2Link approach to language generation.
- **Node**: A type of Atom. Links and Nodes are the two basic kinds of Atoms. Nodes, mathematically, can be thought of as "0-ary" links. Some types of Nodes refer to external or mathematical entities (e.g. WordNode, NumberNode); others are purely abstract, e.g. a ConceptNode is characterized purely by the Links relating it to other atoms. GroundedPredicateNodes and GroundedSchemaNodes connect to explicitly represented procedures (sometimes in the Combo language); ungrounded PredicateNodes and SchemaNodes are abstract and, like ConceptNodes, purely characterized by their relationships.
- **Node Probability**: Many PLN inference rules rely on probabilities associated with Nodes. Node probabilities are often easiest to interpret in a specific context, e.g. the probability P(cat) makes obvious sense in the context of a typical American house, or in the context of the center of the sun. Without any contextual specification, P(A) is taken to mean the probability that a randomly chosen occasion of the system's experience includes some instance of A.
- **Novamente Cognition Engine (NCE)**: A proprietary proto-AGI software system, the predecessor to OpenCog. Many parts of the NCE were open-sourced to form portions of OpenCog, but some NCE code was not included in OpenCog; and now OpenCog includes multiple aspects and plenty of code that was not in NCE.
- **OpenCog**: A software framework intended for development of AGI systems, and also for narrow-AI application using tools that have AGI applications. Co-designed with the CogPrime cognitive architecture, but not exclusively bound to it.
- **OpenCog Prime (OCP)**: The implementation of the CogPrime cognitive architecture within the OpenCog software framework.
- **OpenPsi**: CogPrime's architecture for motivation-driven action selection, which is based on adapting Dorner's Psi model for use in the OpenCog framework.
- **Operational Agent Controller (OAC)**: In some OpenCog configurations, this is a software server containing a CogServer devoted to real-time control of an agent (e.g. a virtual world agent, or a robot). Background, offline learning tasks may then be dispatched to other software processes, e.g. to a Learning Server.

EFTA00624670

- **Pattern**: In a CogPrime context, the term "pattern" is generally used to refer to a process that produces some entity, and is judged simpler than that entity.
- **Pattern Mining**: Pattern mining is the process of extracting an (often large) number of patterns from some body of information, subject to some criterion regarding which patterns are of interest. Often (but not exclusively) it refers to algorithms that are rapid or "greedy", finding a large number of simple patterns relatively inexpensively.
- **Pattern Recognition**: The process of identifying and representing a pattern in some substrate (e.g. some collection of Atoms, or some raw perceptual data, etc.).
- **Patternism**: The philosophical principle holding that, from the perspective of engineering intelligent systems, it is sufficient and useful to think about mental processes in terms of (static and dynamical) patterns.
- **Perception**: The process of understanding data from sensors. When natural language is ingested in textual format, this is generally not considered perceptual. Perception may be taken to encompass both pre-processing that prepares sensory data for ingestion into the Atomspace, processing via specialized perception processing systems like DeSTIN that are connected to the Atomspace, and more cognitive-level process within the Atomspace that is oriented toward understanding what has been sensed.
- **Piagetan Stages**: A series of stages of cognitive development hypothesized by developmental psychologist Jean Piaget, which are easy to interpret in the context of developing CogPrime systems. The basic stages are: Infantile, Pre-operational, Concrete Operational and Formal. Post-formal stages have been discussed by theorists since Piaget and seem relevant to AGI, especially advanced AGI systems capable of strong self-modification.
- **PLN**: short for Probabilistic Logic Networks
- **PLN, First-Order**: See First-Order Inference
- **PLN, Higher-Order**: See Higher-Order Inference
- **PLN Rules**: A PLN Rule takes as input one or more Atoms (the "premises", usually Links), and output an Atom that is a "logical conclusion" of those Atoms. The truth value of the consequence is determined by a PLN Formula associated with the Rule.
- **PLN Formulas**: A PLN Formula, corresponding to a PLN Rule, takes the TruthValues corresponding to the premises and produces the TruthValue corresponding to the conclusion. A single Rule may correspond to multiple Formulas, where each Formula deals with a different sort of TruthValue.
- **Pragmatic General Intelligence**: A formalization of the concept of general intelligence, based on the concept that general intelligence is the capability to achieve goals in environments, calculated as a weighted average over some fuzzy set of goals and environments.
- **Predicate Evaluation**: The process of determining the Truth Value of a predicate, embodied in a PredicateNode. This may be recursive, as the predicate referenced internally by a Grounded PredicateNode (and represented via a Combo program tree) may itself internally reference other PredicateNodes.
- **Probabilistic Logic Networks (PLN)**: A mathematical and conceptual framework for reasoning under uncertainty, integrating aspects of predicate and term logic with extensions of imprecise probability theory. OpenCogPrime's central tool for symbolic reasoning.
- **Procedural Knowledge**: Knowledge regarding which series of actions (or action-combinations) are useful for an agent to undertake in which circumstances. In CogPrime these may be learned in a number of ways, e.g. via PLN or via Hebbian learning of Schema Maps, or via explicit learning of Combo programs via MOSES or hillclimbing. Procedures are represented as SchemaNodes or Schema Maps.

EFTA00624671

- **Procedure Evaluation/Execution**: A general term encompassing both Schema Execution and Predicate Evaluation, both of which are similar computational processes involving manipulation of Combo trees associated with ProcedureNodes.
- **Procedure Learning**: Learning of procedural knowledge, based on any method, e.g. evolutionary learning (e.g. MOSES), inference (e.g. PLN), reinforcement learning (e.g. Hebbian learning).
- **Procedure Node**: A SchemaNode or PredicateNode
- **Psi**: A model of motivated action and emotion, originated by Dietrich Dorner and further developed by Joscha Bach, who incorporated it in his proto-AGI system MicroPsi. OpenCog-Prime's motivated-action component, OpenPsi, is roughly based on the Psi model.
- **Psynese**: A system enabling different OpenCog instances to communicate without using natural language, via directly exchanging Atom subgraphs, using a special system to map references in the speaker's mind into matching references in the listener's mind.
- **Psynet Model**: An early version of the theory of mind underlying CogPrime, referred to in some early writings on the Webmind AI Engine and Novamente Cognition Engine. The concepts underlying the psynet model are still part of the theory underlying CogPrime, but the name has been deprecated as it never really caught on.
- **Reasoning**: See inference
- **Reduct**: A code library, used within MOSES, applying a collection of hand-coded rewrite rules that transform Combo programs into Elegant Normal Form.
- **Region Connection Calculus**: A mathematical formalism describing a system of basic operations among spatial regions. Used in CogPrime as part of spatial inference to provide relations and rules to be referenced via PLN and potentially other subsystems.
- **Reinforcement Learning**: Learning procedures via experience, in a manner explicitly guided to cause the learning of procedures that will maximize the system's expected future reward. CogPrime does this implicitly whenever it tries to learn procedures that will maximize some Goal whose Truth Value is estimated via an expected reward calculation (where "reward" may mean simply the Truth Value of some Atom defined as "reward"). Goal-driven learning is more general than reinforcement learning as thus defined; and the learning that CogPrime does, which is only partially goal-driven, is yet more general.
- **RelEx**: A software system used in OpenCog as part of natural language comprehension, to map the output of the link parser into more abstract semantic relationships. These more abstract relationships may then be entered directly into the Atomspace, or they may be further abstracted before being entered into the Atomspace, e.g. by RelEx2Frame rules.
- **RelEx2Frame**: A system of rules for translating RelEx output into Atoms, based on the FrameNet ontology. The output of the RelEx2Frame rules make use of the FrameNet library of semantic relationships. The current (2012) RelEx2Frame rule-based is problematic and the RelEx2Frame system is deprecated as a result, in favor of Link2Atom. However, the ideas embodied in these rules may be useful; if cleaned up the rules might profitably be ported into the Atomspace as ImplicationLinks.
- **Representation Building**: A stage within MOSES, wherein a candidate Combo program tree (within a deme) is modified by replacing one or more tree nodes with alternative tree nodes, thus obtaining a new, different candidate program within that deme. This process currently relies on hand-coded knowledge regarding which types of tree nodes a given tree node should be experimentally replaced with (e.g. an AND node might sensibly be replaced with an OR node, but not so sensibly replaced with a node representing a "kick" action).

EFTA00624672

- **Request for Services (RFS)**: In CogPrime's Goal-driven action system, a RFS is a package sent from a Goal Atom to another Atom, offering it a certain amount of STI currency if it is able to deliver the goal what it wants (an increase in its Truth Value). RFS's may be passed on, e.g. from goals to subgoals to sub-subgoals, but eventually an RFS reaches a Grounded SchemaNode, and when the corresponding Schema is executed, the payment implicit in the RFS is made.
- **Robot Preschool**: An AGI Preschool in our physical world, intended for robotically embodied AGIs.
- **Robotic Embodiment**: Using an AGI to control a robot. The AGI may be running on hardware physically contained in the robot, or may run elsewhere and control the robot via networking methods such as wifi.
- **Scheduler**: Part of the CogServer that controls which processes (e.g. which MindAgents) get processor time, at which point in time.
- **Schema**: A "script" describing a process to be carried out. This may be explicit, as in the case of a GroundedSchemaNode, or implicit, as the case in Schema maps or ungrounded SchemaNodes.
- **Schema Encapsulation**: The process of automatically recognizing a Schema Map in an Atomspace, and creating a Combo (or other) program embodying the process carried out by this Schema Map, and then storing this program in the Procedure Repository and associating it with a particular SchemaNode. This translates distributed, global procedural memory into localized procedural memory. It's a special case of Map Encapsulation.
- **Schema Execution**: The process of "running" a Grounded Schema, similar to running a computer program. Or, phrased alternately: The process of executing the Schema referenced by a Grounded SchemaNode. This may be recursive, as the predicate referenced internally by a Grounded SchemaNode (and represented via a Combo program tree) may itself internally reference other Grounded SchemaNodes.
- **Schema, Grounded**: A Schema that is associated with a specific executable program (either a Combo program or, say, C++ code)
- **Schema Map**: A collection of Atoms, including SchemaNodes, that tend to be enacted in a certain order (or set of orders), thus habitually enacting the same process. This is a distributed, globalized way of storing and enacting procedures.
- **Schema, Ungrounded**: A Schema that represents an abstract procedure, not associated with any particular executable program.
- **Schematic Implication**: A general, conceptual name for implications of the form ((Context AND Procedure) IMPLIES Goal)
- **SegSim**: A name for the main algorithm underlying the NLGen language generation software. The algorithm is based on segmenting a collection of Atoms into small parts, and matching each part against memory to find, for each part, cases where similar Atom-collections already have known linguistic expression.
- **Self-Modification**: A term generally used for AI systems that can purposefully modify their core algorithms and representations. Formally and crisply distinguishing this sort of "strong self-modification" from "mere" learning is a tricky matter.
- **Sensorimotor**: Pertaining to sensory data, motoric actions, and their combination and intersection.
- **Sensory**: Pertaining to data received by the AGI system from the outside world. In a CogPrime system that perceives language directly as text, the textual input will generally

EFTA00624673

not be considered as "sensory" (on the other hand, speech audio data would be considered as "sensory").

- **Short Term Importance**: A value associated with each Atom, indicating roughly the expected utility to the system of keeping that Atom in RAM rather than saving it to disk or deleting it. It's possible to have multple LTI values pertaining to different time scales, but so far practical implementation and most theory has centered on the option of a single LTI value.
- **Similarity**: a link type indicating the probabilistic similarity between two different Atoms. Generically this is a combination of Intensional Similarity (similarity of properties) and Extensional Similarity (similarity of members).
- **Simple Truth Value**: a TruthValue involving a pair (s,d) indicating strength (e.g. probability or fuzzy set membership) and confidence d. d may be replaced by other options such as a count n or a weight of evidence w.
- **Simulation World**: See Internal Simulation World
- **SMEPH (Self-Modifying Evolving Probabilistic Hypergraphs)**: a style of modeling systems, in which each system is associated with a derived hypergraph
- **SMEPH Edge**: A link in a SMEPH derived hypergraph, indicating an empirically observed relationship (e.g. inheritance or similarity) between two
- **SMEPH Vertex**: A node in a SMEPH derived hypergraph representing a system, indicating a collection of system states empirically observed to arise in conjunction with the same external stimuli
- **Spatial Inference**: PLN reasoning including Atoms that explicitly reference spatial relationships
- **Spatiotemporal Inference**: PLN reasoning including Atoms that explicitly reference spatial and temporal relationships
- **STI**: Shorthand for Short Term Importance
- **Strength**: The main component of a TruthValue object, lying in the interval [0,1], referring either to a probability (in cases like InheritanceLink, SimilarityLink, EquivalenceLink, ImplicationLink, etc.) or a fuzzy value (as in MemberLink, EvaluationLink).
- **Strong Self-Modification**: This is generally used as synonymous with Self-Modification, in a CogPrime context.
- **Subsymbolic**: Involving processing of data using elements that have no correspondence to natural language terms, nor abstract concepts; and that are not naturally interpreted as symbolically "standing for" other things. Often used to refer to processes such as perception processing or motor control, which are concerned with entities like pixels or commands like "rotate servomotor 15 by 10 degrees theta and 55 degrees phi." The distinction between "symbolic" and "subsymbolic" is conventional in the history of AI, but seems difficult to formalize rigorously. Logic-based AI systems are typically considered "symbolic", yet
- **Supercompilation**: A technique for program optimization, which globally rewrites a program into a usually very different looking program that does the same thing. A prototype supercompiler was applied to Combo programs with successful results.
- **Surface Realization**: The process of taking a collection of Atoms and transforming them into a series of words in a (usually natural) language. A stage in the overall process of language generation.
- **Symbol Grounding**: The mapping of a symbolic term into perceptual or motoric entities that help define the meaning of the symbolic term. For instance, the concept "Cat" may be

EFTA00624674

grounded by images of cats, experiences of interactions with cats, imaginations of being a cat, etc.

- **Symbolic**: Pertaining to the formation or manipulation of symbols, i.e. mental entities that are explicitly constructed to represent other entities. Often contrasted with subsymbolic.
- **Syntax-Semantics Correlation**: In the context of MOSES and program learning more broadly, this refers to the property via which distance in syntactic space (distance between the syntactic structure of programs, e.g. if they're represented as program trees) and semantic space (distance between the behaviors of programs, e.g. if they're represented as sets of input/output pairs) are reasonably well correlated. This can often happen among sets of programs that are not too widely dispersed in program space. The Reduct library is used to place Combo programs in Elegant Normal Form, which increases the level of syntax-semantics corellation between them. The programs in a single MOSES deme are often closely enough clustered together that they have reasonably high syntax-semantics correlation.
- **System Activity Table**: An OpenCog component that records information regarding what a system did in the past.
- **Temporal Inference**: Reasoning that heavily involves Atoms representing temporal information, e.g. information about the duration of events, or their temporal relationship (before, after, during, beginning, ending). As implemented in CogPrime, makes use of an uncertain version of Allen Interval Algebra.
- **Truth Value**: A package of information associated with an Atom, indicating its degree of truth. SimpleTruthValue and IndefiniteTruthValue are two common, particular kinds. Multiple truth values associated with the same Atom from different perspectives may be grouped into CompositeTruthValue objects.
- **Universal Intelligence**: A technical term introduced by Shane Legg and Marcus Hutter, describing (roughly speaking) the average capability of a system to carry out computable goals in computable environments, where goal/environment pairs are weighted via the length of the shortest program for computing them.
- **Urge**: In OpenPsi, an Urge develops when a Demand deviates from its target range.
- **Very Long Term Importance (VLTI)**: A bit associated with Atoms, which determines whether, when an Atom is forgotten (removed from RAM), it is saved to disk (frozen) or simply deleted.
- **Virtual AGI Preschool**: A virtual world intended for AGI teaching/training/learning, bearing broad resemblance to the preschool environments used for young humans.
- **Virtual Embodiment**: Using an AGI to control an agent living in a virtual world or game world, typically (but not necessarily) a 3D world with broad similarity to the everyday human world.
- **Webmind AI Engine**: A predecessor to the Novamente Cognition Engine and OpenCog, developed 1997-2001 – with many similar concepts (and also some different ones) but quite different algorithms and software architecture

EFTA00624675

References

ABS+11. Itamar Arel, S Berant, T Slonim, A Moyal, B Li, and K Chai Sim. Acoustic spatiotemporal modeling using deep machine learning for robust phoneme recognition. In *Afeka-AVIOS Speech Processing Conference*, 2011.

All83. James F. Allen. Maintaining knowledge about temporal. *Intervals CACM*, 26:198–3, 1983.

AM01. J. S. Albus and A. M. Meystel. *Engineering of Mind: An Introduction to the Science of Intelligent Systems*. Wiley and Sons, 2001.

Ama85. S. Amari. Differential-geometrical methods in statistics. *Lecture notes in statistics*, 1985.

Ama98. S. Amari. Natural gradient works efficiently in learning. *Neural Computing*, 10:251–276, 1998.

AN00. Shun-ichi Amari and Hiroshi Nagaoka. *Methods of information geometry*. AMS, 2000.

ARC09a. I. Arel, D. Rose, and R. Coop. Destin: A scalable deep learning architecture with application to high-dimensional robust pattern recognition. *Proc. AAAI Workshop on Biologically Inspired Cognitive Architectures*, 2009.

ARC09b. Itamar Arel, Derek Rose, and Robert Coop. A biologically-inspired deep learning architecture with application to high-dimensional pattern recognition. In *Biologically Inspired Cognitive Architectures, 2009*. AAAI Press, 2009.

ARK09. I. Arel, D. Rose, and T. Karnowski. A deep learning architecture comprising homogeneous cortical circuits for scalable spatiotemporal pattern inference. *NIPS 2009 Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

Arn69. Rudolf Arnheim. *Visual Thinking*. University of California Press. Berkeley, 1969.

AS94. Rakesh Agrawal and Ramakrishnan Srikant. Fast algorithms for mining association rules. In *Proc. 20th Int. Conf. Very Large Data Bases*, 1994.

Ash65. Robert B. Ash. *Information Theory*. Dover Publications, 1965.

Bau04. E. B. Baum. *What is Thought?* MIT Press, 2004.

Bau06. E. Baum. A working hypothesis for general intelligence. In *Advances in Artificial General Intelligence: Concepts, Architectures and Algorithms*, 2006.

BE07. Neven Boric and Pablo A. Estevez. Genetic programming-based clustering using an information theoretic fitness measure. In Dipti Srinivasan and Lipo Wang, editors, *2007 IEEE Congress on Evolutionary Computation*, pages 31–38, Singapore, 25-28 September 2007. IEEE Computational Intelligence Society, IEEE Press.

Bel03. Anthony J. Bell. The co-information lattice. *Somewhere or other*, 2003.

Ben94. Brandon Bennett. Spatial reasoning with propositional logics. In *Principles of Knowledge Representation and Reasoning: Proceedings of the 4th International Conference (KR94)*, pages 51–62. Morgan Kaufmann, 1994.

BF97. A Blum and M Furst. Fast planning through planning graph analysis. *Artificial intelligence*, 1997.

BH10. Bundzel and Hashimoto. Object identification in dynamic images based on the memory-prediction theory of brain function. *Journal of Intelligent Learning Systems and Applications*, 2-4, 2010.

Bic08. Derek Bickerton. *Bastard Tongues*. Hill and Wang, 2008.

BKL06. Alina Beygelzimer, Sham Kakade, and John Langford. Cover trees for nearest neighbor. In *Proc. International Conference on Machine Learning*, 2006.

BL99. Avrim Blum and John Langford. Probabilistic planning in the graphplan framework. In *5th European Conference on Planning (ECP '99)*, 1999.

Bor05. Christian Borgelt. Keeping things simple: Finding frequent item sets by recursive elimination. In *Workshop on Open Source Data Mining Software (OSDM'05). Chicago IL*, pages 66–70. 2005.

Car06. Pereira Francisco Cara. *Creativity and Artificial Intelligence: A Conceptual Blending Approach, Applications of Cognitive Linguistics*. Amsterdam: Mouton de Gruyter, 2006.

Cas04. N. L. Cassimatis. Grammatical processing using the mechanisms of physical inferences. In *Proceedings of the Twentieth-Sixth Annual Conference of the Cognitive Science Society*. 2004.

CB00. W. H. Calvin and D. Bickerton. *Lingua ex Machina*. MIT Press, 2000.

CFH97. Eliseo Clementini, Paolino Di Felice, and Daniel HernÃ¡ndez. Qualitative representation of positional information. *Artificial Intelligence*, 95:317–356, 1997.

CGPH09. Lucio Coelho, Ben Goertzel, Cassio Pennachin, and Chris Heward. Classifier ensemble based analysis of a genome-wide snp dataset concerning late-onset alzheimer disease. In *Proceedings of 8th IEEE International Conference on Cognitive Informatics.*, 2009.

Cha08. Gregory Chaitin. *Algorithmic Information Theory*. Cambridge University Press, 2008.

EFTA00624676

Cha09. Mark Changizi. *The Vision Revolution*. BenBella Books, 2009.

Che97. K. Chellapilla. Evolving computer programs without subtree crossover. *IEEE Transactions on Evolutionary Computation*, 1997.

Coh95. A.G. Cohn. A hierarchical representation of qualitative shape based on connection and convexity. In *Proc COSIT95, LNCS*, pages 311–326. Springer Verlag, 1995.

Cox61. Richard Cox. *The Algebra of Probable Inference*. Johns Hopkins University Press, 1961.

CS10. Shay B. Cohen and Noah A. Smith. Covariance in unsupervised learning of probabilistic grammars. *Journal of Machine Learning Research*, 11:3117–3151, 2010.

CSZ06. Olivier Chapelle, Bernhard Schakopf, and Alexander Zien. *Semi-Supervised Learning*. MIT Press, 2006.

CXYM05. Yun Chi, Yi Xia, Yirong Yang, and Richard R Muntz. Mining closed and maximal frequent subtrees from databases of labeled rooted trees.. *IEEE Trans. Knowledge and Data Engineering*, 2005.

Dab99. A.G. Dabak. *A Geometry for Detection Theory*. PhD Thesis, Rice U., 1999.

Dea98. Terrence Deacon. *The Symbolic Species*. Norton, 1998.

dF37. Bruno de Finetti. La prévision: ses lois logiques, ses sources subjectives,. *Annales de l'Institut Henri Poincaré*, 1937.

DP09. Yassine Djouadi and Henri Prade. Interval-valued fuzzy formal concept analysis. In *ISMIS '09: Proc. of the 18th International Symposium on Foundations of Intelligent Systems*, pages 592–601, Berlin, Heidelberg, 2009. Springer-Verlag.

dS77. Ferdinand de Saussure. *Course in General Linguistics*. Fontana/Collins, 1977. Orig. published 1916 as "Cours de linguistique générale".

EBJ+97. J. Elman, E. Bates, M. Johnson, A. Karmiloff-Smith, D. Parisi, and K. Plunkett. *Rethinking Innateness: A Connectionist Perspective on Development*. MIT Press, 1997.

Ede93. Gerald Edelman. Neural darwinism: Selection and reentrant signaling in higher brain function. *Neuron*, 10, 1993.

FF92. Christian Freksa and Robert Fulton. Temporal reasoning based on semi-intervals. *Artificial Intelligence*, 54(1-2):199 – 227, 1992.

FL12. Jeremy Fishel and Gerald Loeb. Bayesian exploration for intelligent identification of textures. *Frontiers in Neurorobotics 6-4*, 2012.

Fri98. Roy. Frieden. *Physics from Fisher Information*. Cambridge U. Press, 1998.

FT02. G. Fauconnier and M. Turner. *The Way We Think: Conceptual Blending and the Mind's Hidden Complexities*. Basic, 2002.

Gar00. Peter Gardenfors. *Conceptual spaces: the geometry of thought*. MIT Press, 2000.

GBK04. S. Gustafson, E. K. Burke, and G. Kendall. Sampling of unique structures and behaviours in genetic programming. In *European Conf. on Genetic Programming*, 2004.

GCPM06. Ben Goertzel, Lucio Coelho, Cassio Pennachin, and Mauricio Mudada. Identifying Complex Biological Interactions based on Categorical Gene Expression Data. In *Proceedings of Conference on Evolutionary Computing. Vancouver CA*, 2006.

GE01. Roop Goyal and Max Egenhofer. Similarity in cardinal directions. In *In Proc. of the Seventh International Symposium on Spatial and Temporal Databases*, pages 36–55. Springer-Verlag, 2001.

Gea05. Ben Goertzel and et al. Combinations of single nucleotide polymorphisms in neuroendocrine effector and receptor genes predict chronic fatigue syndrome. *Pharmacogenomics*, 2005.

GEA08. Ben Goertzel and Cassio Pennachin Et Al. An integrative methodology for teaching embodied non-linguistic agents, applied to virtual animals in second life. In *Proc.of the First Conf. on AGI*. IOS Press, 2008.

Gea13. Ben Goertzel and et al. The cogprime architecture for embodied artificial general intelligence. In *Proceedings of IEEE Symposium on Human-Level AI, Singapore*, 2013.

GGC+11. Ben Goertzel, Nil Geisweiller, Lucio Coelho, Predrag Janicic, and Cassio Pennachin. *Real World Reasoning*. Atlantis, 2011.

GH11. N Garg and J Henderson. Temporal restricted boltzmann machines for dependency parsing. In *Proc. ACL*, 2011.

GI11. B. Goertzel and M. Iklé. Steps toward a geometry of mind. In J Schmidhuber and K Thorisson, editors, *Subm.to AGI-11*. Springer, 2011.

GIGH08. B. Goertzel, M. Ikle, I. Goertzel, and A. Heljakka. *Probabilistic Logic Networks*. Springer, 2008.

GKD89. D. E. Goldberg, B. Korb, and K. Deb. Messy genetic algorithms: Motivation, analysis, and first results. *Complex Systems*, 1989.

EFTA00624677

GL10. Ben Goertzel and Ruiting Lian. A probabilistic characterization of fuzzy semantics. *Proc. of ICAI-10, Beijing*, 2010.

GLdG+10. Ben Goertzel, Ruiting Lian, Hugo de Garis, Shuo Chen, and Itamar Arel. World survey of artificial brains, part ii: Biologically inspired cognitive architectures. *Neurocomputing*, April 2010.

GMIH08. B. Goertzel, I. Goertzel M. Iklé, and A. Heljakka. *Probabilistic Logic Networks*. Springer, 2008.

GN02. Alfonso Gerevini and Bernhard Nebel. Qualitative spatio-temporal reasoning with rcc-8 and allen's interval calculus: Computational complexity. In Frank van Harmelen, editor, *ECAI*, pages 312–316. IOS Press, 2002.

Goe94. Ben Goertzel. *Chaotic Logic*. Plenum, 1994.

Goe06. Ben Goertzel. *The Hidden Pattern*. Brown Walker, 2006.

Goe08a. B. Goertzel. The pleasure algorithm. *groups.google.com/group/opencog/files*, 2008.

Goe08b. Ben Goertzel. A pragmatic path toward endowing virtually-embodied ais with human-level linguistic capability. IEEE World Congress on Computational Intelligence (WCCI), 2008.

Goe10a. Ben Goertzel. Infinite-order probabilities and their application to modeling self-referential semantics. In *Proceedings of Conference on Advanced Intelligence 2010, Beijing*, 2010.

Goe10b. Ben et al Goertzel. A general intelligence oriented architecture for embodied natural language processing. In *Proc. of the Third Conf. on Artificial General Intelligence (AGI-10)*. Atlantis Press, 2010.

Goe11a. B Goertzel. Integrating a compositional spatiotemporal deep learning network with symbolic representation/reasoning within an integrative cognitive architecture via an intermediary semantic network. In *Proceedings of AAAI Symposium on Cognitive Systems,*, 2011.

Goe11b. Ben Goertzel. Imprecise probability as a linking mechanism between deep learning, symbolic cognition and local feature detection in vision processing. In *Proc. of AGI-11*, 2011.

GPPG06. Ben Goertzel, Hugo Pinto, Cassio Pennachin, and Izabela Freire Goertzel. Using dependency parsing and probabilistic inference to extract relationships between genes, proteins and malignancies implicit among multiple biomedical research abstracts. In *Proc. of Bio-NLP 2006*, 2006.

GR00. Alfonso Gerevini and Jochen Renz. Combining topological and size information for spatial reasoning. *Artificial Intelligence*, 137:2002, 2000.

GSW05. Bernhard Ganter, Gerd Stumme, and Rudolf Wille. *Formal Concept Analysis: Foundations and Applications*. Springer-Verlag, 2005.

HB06. Jeff Hawkins and Sandra Blakeslee. *On Intelligence*. Brown Walker, 2006.

HDY+12. Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, and Tara Sainathand Brian Kingsbury. Deep neural networks for acoustic modeling in speech recognition. *IEEE Signal Processing Magazine*, 2012.

HH07. Barbara Hammer and Pascal Hitzler, editors. *Perspectives of Neural-Symbolic Integration. Studies in Computational Intelligence, Vol. 77*. Springer, 2007.

Hil89. Daniel Hillis. *The Connection Machine*. MIT Press, 1989.

HK02. David Harel and Yehuda Koren. *Graph Drawing by High-Dimensional Embedding*. 2002.

Hob78. J. Hobbs. Resolving pronoun references. *Lingua*, 44:311–338, 1978.

Hof79. Douglas Hofstadter. *Godel, Escher, Bach: An Eternal Golden Braid*. Basic, 1979.

Hol75. J. H. Holland. *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.

Hud84. Richard Hudson. *Word Grammar*. Oxford: Blackwell, 1984.

Hud90. Richard Hudson. *English Word Grammar*. Blackwell Press, 1990.

Hud07a. Richard Hudson. *Language Networks. The new Word Grammar*. Oxford University Press, 2007.

Hud07b. Richard Hudson. *Language Networks: The New Word Grammar*. Oxford Linguistics, 2007.

Hut99. G. Hutton. A tutorial on the universality and expressiveness of fold. *Journal of Functional Programming*, 1999.

Hut05a. Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, 2005.

Hut05b. Marcus Hutter. *Universal Artificial Intelligence: Sequential Decisions based on Algorithmic Probability*. Springer, 2005.

HWP03. Jun Huan, Wei Wang, and Jan Prins. Efficient mining of frequent subgraph in the presence of isomorphism. In *Proceedings of the 3rd IEEE International Conference on Data Mining (ICDM)*, pages 549–552. 2003.

Jac03. Ray Jackendoff. *Foundations of Language: Brain, Meaning, Grammar, Evolution*. Oxford University Press, 2003.

EFTA00624678

JL08. D. J. Jilk and C. Lebiere. and o'reilly. *R. C. and Anderson, J. R. (2008). SAL: An explicitly pluralistic cognitive architecture. Journal of Experimental and Theoretical Artificial Intelligence*, 20:197–218, 2008.

Joh05. Mark Johnson. *LaTeX: A Developmental Cognitive Neuroscience*. Wiley-Blackwell, 2005.

Jol10. I. T. Joliffe. *Principal Component Analysis*. Springer, 2010.

KA95. J. R. Koza and D. Andre. Parallel genetic programming on a network of transputers. Technical report, Stanford University, 1995.

KAR10. Tom Karnowski, Itamar Arel, and D. Rose. Deep spatiotemporal feature learning with application to image classification. In *The 9th International Conference on Machine Learning and Applications (ICMLA'10)*, 2010.

KK01. Michihiro Kuramochi and George Karypis. Frequent subgraph discovery. In *Proceedings of the 2001 IEEE International Conference on Data Mining*, pages 313–320. 2001.

KM04. Dan Klein and Christopher D. Manning. Corpus-based induction of syntactic structure: Models of dependency and constituency. In *ACL '04 Proceedings of the 42nd Annual Meeting on Association for Computational Linguistics*, pages 479–486. Association for Computational Linguistics, 2004.

Koh01. Teuvo Kohonen. *Self-Organizing Maps*. Springer, 2001.

Koz92. J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.

Koz94. J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs*. MIT Press, 1994.

KSPC13. Dimitri Kartsaklis, Mehrnoosh Sadrzadeh, Stephen Pulman, and Bob Coecke. Reasoning about meaning in natural language with compact closed categories and frobenius algebras. 2013.

Kur09. Yohei Kurata. 9-intersection calculi for spatial reasoning on the topological relations between multi-domain objects. USA, June 2009.

Kur12. Ray Kurzweil. *How to Create a Mind*. Viking, 2012.

LA93. C Lebiere and J R Anderson. A connectionist implementation of the act-r production system. In *Proceedings of the Fifteenth Annual Conference of the Cognitive Science Society*, 1993.

Lai12. John E Laird. *The Soar Cognitive Architecture*. MIT Press, 2012.

LBH10. Jens Lehmann, Sebastian Bader, and Pascal Hitzler. Extracting reduced logic programs from artificial neural networks. *Applied Intelligence*, 2010.

LDA05. Ramon Lopez, Cozar Delgado, and Masahiro Araki. *Spoken, Multilingual and Multimodal Dialogue Systems: Development and Assessment*. Wiley, 2005.

Lem10. B. Lemoine. Nlgen2: a linguistically plausible, general purpose natural language generation system. http://www.louisiana.edu/??bal2277/NLGen2, 2010.

Lev94. L. Levin. Randomness and nondeterminism. In *The International Congress of Mathematicians*, 1994.

LGE10. Ruiting Lian, Ben Goertzel, and Al Et. Language generation via glocal similarity matching. *Neurocomputing*, 2010.

LGK+12. Ruiting Lian, Ben Goertzel, Shujing Ke, Jade OÕNeill, Keyvan Sadeghi, Simon Shiu, Dingjie Wang, Oliver Watkins, and Gino Yu. Syntax-semantic mapping for general intelligence: Language comprehension as hypergraph homomorphism, language generation as constraint satisfaction. In *Artificial General Intelligence: Lecture Notes in Computer Science Volume 7716*. Springer, 2012.

LKP+05. Sung Hee Lee, Junggon Kim, Frank Chongwoo Park, Munsang Kim, and James E. Bobrow. Newton-type algorithms for dynamics-based robot movement optimization. *IEEE Transactions on Robotics*, 21(4):657–667, 2005.

LLR09. Weiming Liu, Sanjiang Li, and Jochen Renz. Combining rcc-8 with qualitative direction calculi: Algorithms and complexity. In *IJCAI*, 2009.

LMDK07. Thomas K. Landauer, Danielle S. McNamara, Simon Dennis, and Walter Kintsch. *Handbook of Latent Semantic Analysis*. Psychology Press, 2007.

LN00. George Lakoff and Rafael Nunez. *Where Mathematics Comes From*. Basic Books, 2000.

Loo06. Moshe Looks. *Competent Program Evolution*. PhD Thesis, Computer Science Department, Washington University, 2006.

Loo07a. M. Looks. On the behavioral diversity of random programs. In *Genetic and evolutionary computation conference*, 2007.

Loo07b. M. Looks. Scalable estimation-of-distribution program evolution. In *Genetic and evolutionary computation conference*, 2007.

EFTA00624679

Loo07c. Moshe Looks. Meta-optimizing semantic evolutionary search. In Hod Lipson, editor, *Genetic and Evolutionary Computation Conference, GECCO 2007, Proceedings, London, England, UK, July 7-11, 2007*, page 626. ACM, 2007.

Low99. David Lowe. Object recognition from local scale-invariant features. In *Proc. of the International Conf. on Computer Vision*, pages 1150–1157, 1999.

LP01. Dekang Lin and Patrick Pantel. Dirt: Discovery of inference rules from text. In *Proceedings of the Seventh ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'01)*, pages 323–328. ACM Press, 2001.

LP02. W. B. Langdon and R. Poli. *Foundations of Genetic Programming*. Springer-Verlag, 2002.

Mai00. Monika Maidl. The common fragment of ctl and ltl. In *In IEEE Symposium on Foundations of Computer Science*, pages 643–652, 2000.

May04. M. T. Maybury. *New Directions in Question Answering*. MIT Press, 2004.

Mea07. Reiman E M and et al. Gab2 alleles modify alzheimer's risk in apoe e4 carriers. *Neuron 54 (5)*, 2007.

Mih05. Rada Mihalcea. Unsupervised large-vocabulary word sense disambiguation with graph-based algorithms for sequence data labeling. In *HLT '05: Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*, pages 411–418, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

Mih07. Rada Mihalcea. *Word sense disambiguation. Encyclopedia of Machine Learning*. Springer-Verlag, 2007.

Min88. Marvin Minsky. *The Society of Mind*. MIT Press, 1988.

Mit96. Steven Mithen. *The Prehistory of Mind*. Thames and Hudson, 1996.

MS99. Christopher Manning and Heinrich Scheutze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

MTF04. Rada Mihalcea, Paul Tarau, and Elizabeth Figa. Pagerank on semantic networks, with application to word sense disambiguation. In *COLING '04: Proceedings of the 20th international conference on Computational Linguistics*, Morristown, NJ, USA, 2004. Association for Computational Linguistics.

OCC90. Andrew Ortony, Gerald Clore, and Allan Collins. *The Cognitive Structure of Emotions*. Cambridge University Press, 1990.

Ols95. J. R. Olsson. Inductive functional programming using incremental program transformation. *Artificial Intelligence*, 1995.

PAF00. H. Park, S. Amari, and K. Fukumizu. Adaptive natural gradient learning algorithms for various stochastic models. *Neural Computing*, 13:755–764, 2000.

Pal04. Girish Keshav Palshikar. Fuzzy region connection calculus in finite discrete space domains. *Appl. Soft Comput.*, 4(1):13–23, 2004.

PCP00. Papageorgiou, C., and T. Poggio. *A trainable system for object detection*, volume 38–1. Intl. J ComputerVision, 2000.

PD09. Hoifung Poon and Pedro Domingos. Unsupervised semantic parsing. In *Proceedings of the 2009 Conference on Empirical Methods in Natural Language Processing*, pages 1–10, Singapore, August 2009. Association for Computational Linguistics.

Pei34. C. Peirce. *Collected papers: Volume V. Pragmatism and pragmaticism*. Harvard University Press. Cambridge MA., 1934.

Pel05. Martin Pelikan. *Hierarchical Bayesian Optimization Algorithm: Toward a New Generation of Evolutionary Algorithms*. Springer, 2005.

PJ88a. Pearl and J. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*. Morgan Kaufman, 1988.

PJ88b. Steven Pinker and JacquesMehler. *Connections and Symbols*. MIT Press, 1988.

Pro13. The Univalent Foundations Program. *Homotopy Type Theory: Univalent Foundations of Mathematics*. Institute for Advanced Study, 2013.

RCC93. D. A. Randell, Z. Cui, and A. G. Cohn. A spatial logic based on regions and connection. 1993.

Ros99. J. Rosca. Genetic programming acquires solutions by combining top-down and bottom-up refinement. In *Foundations of Generic Programming*, 1999.

Row90. John Rowan. *Subpersonalities: The People Inside Us*. Routledge Press, 1990.

RVG05. Mike Ross, Linas Vepstas, and Ben Goertzel. Relex semantic relationship extractor. http://opencog.org/wiki/RelEx, 2005.

EFTA00624680

Sch06. J. Schmidhuber. Godel machines: Fully Self-referential Optimal Universal Self-improvers. In B. Goertzel and C. Pennachin, editors, *Artificial General Intelligence*, pages 119–226. 2006.

SDCCK08a. Steven Schockaert, Martine De Cock, Chris Cornelis, and Etienne E. Kerre. Fuzzy region connection calculus: An interpretation based on closeness. *Int. J. Approx. Reasoning*, 48(1):332–347, 2008.

SDCCK08b. Steven Schockaert, Martine De Cock, Chris Cornelis, and Etienne E. Kerre. Fuzzy region connection calculus: Representing vague topological information. *Int. J. Approx. Reasoning*, 48(1):314–331, 2008.

SM07. Ravi Sinha and Rada Mihalcea. Unsupervised graph-basedword sense disambiguation using measures of word semantic similarity. In *ICSC '07: Proceedings of the International Conference on Semantic Computing*, pages 363–369, Washington, DC, USA, 2007. IEEE Computer Society.

SM09. Ravi Sinha and Rada Mihalcea. Unsupervised graph-based word sense disambiguation. In Nicolas Nicolov and John Benjamins Ruslan Mitkov, editors, *Current Issues in Linguistic Theory: Recent Advances in Natural Language Processing*. 2009.

SMI97. F-R. Sinot, Fernandez M., and Mackie I. Efficient reductions with director strings. *Evolutionary Computation*, 1997.

SMK12. Jeremy Stober, Risto Miikkulainen, and Benjamin Kuipers. Learning geometry from sensorimotor experience. In *Proceedings of the First Joint Conference on Development and Learning and Epigenetic Robotics*, 2012.

Sol64a. Ray Solomonoff. *A Formal Theory of Inductive Inference, Part I.* Information and Control, 1964.

Sol64b. Ray Solomonoff. *A Formal Theory of Inductive Inference, Part II.* Information and Control, 1964.

Spe96. L. Spector. Simultaneous evolution of programs and their control structures. In *Advances in Genetic Programming 2*. MIT Press, 1996.

SR04. Murray Shanahan and David A Randell. A logic-based formulation of active visual perception. In *Knowledge Representation*, 2004.

SS03. R. P. Salustowicz and J. Schmidhuber. Probabilistic incremental program evolution. *Lecture Notes in Computer Science vol. 2706*, 2003.

ST91. Daniel Sleator and Davy Temperley. Parsing english with a link grammar. Technical report, Carnegie Mellon University Computer Science technical report CMU-CS-91-196, 1991.

ST93. Daniel Sleator and Davy Temperley. Parsing english with a link grammar. *Third International Workshop on Parsing Technologies.*, 1993.

SV99. A. J. Storkey and R. Valabregue. The basins of attraction of a new hopfield learning rule. *Neural Networks*, 12:869–876, 1999.

SW05. Reza Shadmehr and Steven P. Wise. *The Computational Neurobiology of Reaching and Pointing : A Foundation for Motor Learning*. MIT Press, 2005.

SWM90. Timothy Starkweather, Darrell Whitley, and Keith Mathias. Optimization using disributed genetic algorithms. In *Parallel Problem Solving from Nature, Edited by H Schwefel and R Manner*, 1990.

SZ04. R. Sun and X. Zhang. Top-down versus bottom-up learning in cognitive skill acquisition. *Cognitive Systems Research*, 5, 2004.

Tes59. Lucien Tesnière. *Éléments de syntaxe structurale*. Klincksieck, Paris, 1959.

Tom03. Michael Tomasello. *Constructing a Language: A Usage-Based Theory of Language Acquisition*. 2003.

TSH11. Mohamad Tarifi, Meera Sitharam, and Jeffery Ho. Learning hierarchical sparse representations using iterative dictionary learning and dimension reduction. In *Proc. of BICA 2011*, 2011.

TVCC05. M. Tomassini, L. Vanneschi, P. Collard, and M. Clergue. A study of fitness distance correlation as a difficulty measure in genetic programming. *Evolutionary Computation*, 2005.

VK94. T. Veale and M. T. Keane. *Metaphor and Memory and Meaning in Sapper: A Hybrid Model of Metaphor Interpretation*. Proceedings of the workshop on Hybrid Connectionist Systems of ECAI94, at the 11th European Conference on Artificial Intelligence, 1994.

VO07. Tony Veale and Diarmuid O'Donoghue. *Computation and Blending*. Cognitive Linguistics, 2007.

Wah06. Wolfgang Wahlster. *SmartKom: Foundations of Multimodal Dialogue Systems*. Springer, 2006.

Wan06. Pei Wang. *Rigid Flexibility: The Logic of Intelligence*. Springer, 2006.

WF05. Ian Witten and Eibe Frank. *Data Mining: Practical Machine Learning Tools and Techniques*. Morgan Kaufmann, 2005.

Win95. Stephan Winter. Topological relations between discrete regions. In *Advances in Spatial Databasesâ 4th International Symposium, SSD -5*, pages 310–327. Springer, 1995.

EFTA00624681

Win00. Stephan Winter. Uncertain topological relations between imprecise regions. *Journal of Geographical Information Science*, 14(5):411–430, 2000.

WKB05. Nico Van De Weghe, Bart Kuijpers, and Peter Bogaert. A qualitative trajectory calculus and the composition of its relations. In *Proc. of GeoS*, pages 60–76. Springer-Verlag, 2005.

Yan10. King-Yin Yan. A fuzzy-probabilistic calculus for vagueness. In *Unpublished manuscript*, 2010. Subm.

YKL+04. Sanghoon Yeo, Jinwook Kim, Sung Hee Lee, Frank Chongwoo Park, Wooram Park, Junggon Kim, Changbeom Park, and Intaeck Yeo. A modular object-oriented framework for hierarchical multiresolution robot simulation. *Robotica*, 22(2):141–154, 2004.

Yur98. Deniz Yuret. *Discovery of Linguistic Relations Using Lexical Attraction*. PhD thesis, MIT, 1998.

ZH10. Ruoyu Zou and Lawrence B. Holder. Frequent subgraph mining on a single large graph using sampling techniques. In *International Conference on Knowledge Discovery and Data Mining archive. Proceedings of the Eighth Workshop on Mining and Learning with Graphs. Washington DC*, pages 171–178, 2010.

ZLLY08. Xiaotong Zhang, Weiming Liu, Sanjiang Li, and Mingsheng Ying. Reasoning with cardinal directions: an efficient algorithm. In *AAAI'08: Proc. of the 23rd national conference on Artificial intelligence*, pages 387–392. AAAI Press, 2008.

ZM06. Song-Chun Zhu and David Mumford. A stochastic grammar of images. *Foundations and Trends in Computer Graphics and Vision*, 2006.

EFTA00624682