

**Constructing Meaning, Piece by Piece:  
A Computational Cognitive Model of  
Human Sentence Comprehension**

by

Peter Lindes

A dissertation submitted in partial fulfillment  
of the requirements for the degree of  
Doctor of Philosophy  
(Computer Science and Engineering)  
in the University of Michigan  
2022

Doctoral Committee:

Professor John E. Laird, Chair  
Associate Professor Jonathan Brennan  
Professor Benjamin Kuipers  
Professor Richard Lewis

Peter Lindes

[plindes@umich.edu](mailto:plindes@umich.edu)

ORCID iD: 0000-0001-6564-0402

© Peter Lindes 2022

## **Dedication**

This dissertation is dedicated to my family:

My wife and eternal companion:

Bianka Elizabeth Lindes

Our children and their spouses:

Deborah Suzanne Lindes Michael Hoffman

David Alexander Lindes

Analucía Ramírez Pickett David Brian Pickett

Ángel David Lindes Noel Grace Lindes

Our grandchildren:

Matthew Hoffman Lauren Hoffman

Elías Peter Lindes-Frost Allyson Bianka Ramírez

Diego Eric Lindes-Frost Adelyn Eloisa Ramírez

Savannah Lucía Lindes-Frost Aleyna Itzelt Ramírez

Raquel Elizabeth Lindes-Frost

## Acknowledgements

John Laird is my adviser, and he's been amazing. Besides providing the position and the funding, he is always available with ideas and advice. We don't always agree on everything, but we have enjoyed learning a lot together. He has diligently and patiently worked hard with me to get this thesis together.

The rest of my committee members have also been a big help in getting to this point. Ben Kuipers has helped me a lot as a friend to adjust to the rigors of being a grad student. I appreciate many stimulating conversations we have had. Rick Lewis has been a major inspiration with his thesis, and he once said I was one of three people in the world to have read it. And I have! He has also offered good guidance in the several meetings we have had. Jon Brennan is the linguist on my team. Conversations with him have broadened my perspective, and he has helped me keep connected with inspiring meetings of the Psycholinguistics group.

The whole group of Soar grad students have been a very important source of moral support and practical knowledge. James Kirk and Aaron Mininger have always been very open to helping me understand their work in Rosie. Steven Jones worked hard to get my extensions to Soar implemented in the kernel. Bryan Stearns has helped me many times to understand some of the subtleties of the Soar architecture. They and Mazin Assanie, James Boggs, Lizzie Goeddel, Ion Jovina, Preeti Ramaraj, and Jule Schatz, are all great people to discuss both technology and what's going on in the world with, and to learn from.

The CSE staff, including Ashley Andreeae, Kimberley Clark, Yolonda Coleman, Kelly Cormier, Therese Foran, Jamie Goldsmith, George Kroslak, Karen Liska, Stephen Reger, Brian Rice, and Jasmin Stubblefield were all always very helpful. I couldn't have done it without them.

I owe a special debt of gratitude to Bill and Rozie Meyer, who generously hosted me as a housemate for six years, in spite of my strange habits. Their home has been a wonderfully warm and comfortable place to spend these years. Also, Kendall Teichert and Javon Mitchell were good roommates for my first two years in Ann Arbor, and they're still good friends.

I am especially grateful for my family, all those mentioned in the Dedication and the rest of the extended family. They have been cheering me on all the way. Above all is my dear wife Bianka. She couldn't come to live with me in Ann Arbor, but she has endured eight years of frequent and sometimes long separations. Thanks to her, our family is still strong and our love is stronger than ever.

My eternal gratitude goes to my Heavenly Father, without whose inspiration, guidance, and strength I never could have even started, let alone finished, this long, challenging, and rewarding project.

The work described here was supported in part by AFOSR under Grant Number FA9550-18-1-0168 for Interactive Task Learning. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressly or implied, of AFOSR or the U.S. Government.

## Preface

Watching my grandchildren is fascinating! One of the most fascinating parts is seeing how they use language. Our six-year-old can carry on a very complex conversation, even about abstract things like future dates. She speaks mainly in English, but she understands a lot of Spanish. Her two older sisters prefer English, but they understand and speak Spanish very well. They can even do simultaneous interpretation when necessary. How do they do all that? How did they learn to do that?

Another fascinating thing is artificial intelligence. A dream I have is to be able to build a robot that can do our household chores: make the beds, wash the dishes and put them away, mop the floors, clean the bathrooms, and on and on. To be useful, this robot will have to be able to understand the instruction, given in natural language, that it gets from my wife or from me. It will also have to be able to get along with those grandchildren. How can we make an artificial agent that can understand and learn from situated, interactive natural language instruction and then carry out the tasks it has learned?

This thesis attempts to make a contribution to understanding these big questions by building a model we call Lucia. Lucia tries to model human comprehension processing computationally, giving us a window into what might be happening in the human mind and brain. At the same time, it tackles the challenge of using this same Lucia model to work inside a robotic agent called Rosie to give it the ability to learn new tasks from interactive instruction by a human. It leaves many questions unanswered, but also gives insights that have not been available before. I hope this new knowledge will eventually bless those grandchildren, and maybe even help get them the sort of robot helpers I have dreamed of.

## **Table of Contents**

Dedication	ii
Acknowledgements	iii
Preface	v
List of Tables	xiii
List of Figures	xv
List of Appendices	xviii
Acronyms	xx
Abstract	xxi
Chapter 1 Introduction	1
1.1 Defining the Problem	2
1.2 The Lucia Comprehension System	5
1.2.1 E3C: Embodied, End-to-End Comprehension	5
1.2.2 CKM: Composable Knowledge of Meaning	6
1.2.3 I3P: Incremental, Immediate Interpretation Processing	7
1.2.4 GCM: General Cognitive Mechanisms	8
1.2.5 LAE: Language is Acquired from Experience	10
1.3 Related Work	10
1.3.1 Classic systems	10
1.3.2 Systems in Robots (E3C)	11
1.3.3 Construction Grammar Systems (CKM)	12

1.3.4 Systems Developed in a Cognitive Architecture (GCM)	12
1.3.5 Lucia	13
1.3.6 Summary	14
1.3.7 Machine Learning Systems	15
1.4 Contributions and Explorations	15
1.5 Thesis Preview	16
<b>Chapter 2 Embodied, End-to-End Comprehension</b>	<b>18</b>
2.1 Defining the Problem	19
2.1.1 Lucia in context	20
2.1.2 Games and Puzzles	22
2.1.3 Robot tasks	24
2.2 Sentence Corpora	26
2.3 Knowledge of the World	28
2.3.1 Perceptual properties	29
2.3.2 Objects and relations in the World Model	29
2.3.3 Actions, concepts, and the environment	31
2.4 Messages as Sentence Meanings	32
2.4.1 Simple messages and quoted sentences	33
2.4.2 Declarative sentences	33
2.4.3 Commands	35
2.4.4 Conditionals	38
2.4.5 Questions	39
2.5 Evaluation	40
<b>Chapter 3 Composable Knowledge of Meaning</b>	<b>43</b>

3.1 Defining the Problem	44
3.2 Related Work	45
3.2.1 Generative Linguistics	45
3.2.2 Cognitive Linguistics	45
3.2.3 Construction Grammar Theory	47
3.2.4 Goldberg's Five Principles	47
3.2.5 ECG in Context	48
3.3 Embodied Construction Grammar in Lucia	49
3.3.1 An Example Sentence	50
3.3.2 Types of Constructions	51
3.3.3 An ECG Example	51
3.3.4 Hierarchies and Recursion	52
3.3.5 The ECG Formalism	52
3.4 Lucia's ECG Grammar for Rosie	54
3.4.1 Lexical Items	54
3.4.2 Referring Expressions	55
3.4.3 Imperative Sentences	58
3.4.4 Declarative Sentences	59
3.4.5 Questions and Conditionals	61
3.5 The Grammar Development Process	63
3.6 Evaluation	67
3.6.1 Composability	68
3.6.2 Coverage	68
3.6.3 Generality	71

3.7 Conclusions and Implications	78
Chapter 4 Constructing Meaning, Piece by Piece	79
4.1 Research Background	80
4.2 Defining the Problem	81
4.3 Overview of the Lucia Processing Algorithm	83
4.4 Piece-by-Piece Processing	86
4.4.1 An example sentence comprehension	86
4.4.2 A tree of chunks with a root	87
4.4.3 Evolution of the tree with rapid composition	89
4.5 Immediate Interpretation	92
4.5.1 Integration	92
4.5.2 Details of nodes in the tree	93
4.5.3 Grounding	96
4.6 Achieving a Single Path	96
4.6.1 Lexical ambiguities	97
4.6.2 Grammatical ambiguities	100
4.6.3 Structural ambiguities	101
4.6.4 Semantic ambiguities	103
4.6.5 Grounding ambiguities	104
4.7 Evaluation	104
Chapter 5 Lucia in a Cognitive Architecture	106
5.1 Cognitive Architectures	107
5.1.1 The Common Model of Cognition	108
5.1.2 The Soar Cognitive Architecture	109

5.1.3 ACT-R	111
5.2 Language Comprehension in Cognitive Architectures	113
5.3 Lucia's Implementation in Soar	115
5.3.1 Knowledge of the meaning of language in Soar	116
5.3.2 Overall control structure	116
5.3.3 The selection phase	122
5.3.4 The integration phase	124
5.3.5 The grounding phase	128
5.4 Ambiguity and Local Repairs	131
5.4.1 Lexical selection	131
5.4.2 Grounding pronouns	134
5.4.3 Local repairs	135
5.5 Difficult Sentences	140
5.6 Evaluation	144
5.6.1 Basic E3C-CKM-I3P performance	144
5.6.2 Real-time performance	145
5.6.3 Limitations	147
Chapter 6 Exploring Grammar, Architecture, and Acquisition	148
6.1 Events and Difficult Sentences	148
6.1.1 Grammar for event descriptions and relative clauses	149
6.1.2 Garden-path sentences	152
6.1.3 Parsing breakdown	154
6.2 System B: Language Knowledge in Semantic Memory	159
6.2.1 Overview of System B	160

6.2.2 System B control structure	161
6.2.3 Using semantic memory and spreading activation	162
6.2.4 Learning procedural skill	168
6.2.5 Evaluation of System B	170
6.3 Language Acquisition from Experience	173
6.3.1 Defining the problem	174
6.3.2 Related theoretical work	175
6.3.3 Toward a computational cognitive model of LAE	179
Chapter 7 Conclusions	182
7.1 What has been accomplished	182
7.1.1 E3C: Embodied End-to-End Comprehension	182
7.1.2 CKM: Composable Knowledge of the Meaning of Language	183
7.1.3 I3P: Incremental Immediate Interpretation Processing	183
7.1.4 GCM: General Cognitive Mechanisms	184
7.1.5 LAE: Language is Acquired from Experience	185
7.2 The Model as a Basis for Future Research	185
7.2.1 A model to build AI systems on	186
7.2.2 A basis for research on human comprehension	186
7.2.3 A basis for future architecture research	187
7.3 Lessons Learned about Doing Research	188
7.3.1 A holistic approach to development	188
7.3.2 The centrality of meaning	189
7.3.3 Modeling human behavior qualitatively	189
Appendices	191



## **List of Tables**

Table 1-1: Lucia compared to AI models .....	14
Table 2-1: Handles for perceptual properties .....	29
Table 2-2: Objects and locations in a simple World Model.....	30
Table 2-3: Spatial relations in a simple World Model .....	31
Table 2-4: Examples of grounding to the World Model .....	31
Table 2-5: Results of testing .....	42
Table 3-1: Corpus development statistics .....	64
Table 3-2: Productivity of selected constructions .....	76
Table 5-1: Results of testing .....	145
Table 6-1: Grammar for event descriptions .....	149
Table 6-2: Grammar for relative clauses .....	151
Table 6-3: A construction with and without recursion .....	157
Table 6-4: Composite selection based on spreading activation .....	167
Table 6-5: System B parsing statistics .....	170
Table 6-6: System B lexical failures .....	171
Table 6-7: System B composite failures .....	171
Table 6-8: Results of ablation studies of System B.....	172
Table A2-1: Numbers or ECG Items.....	212
Table A2-2: Lexical Constructions by Category .....	213
Table A2-3: Lexical Items with Multiple or No Types .....	214
Table A2-4: Composite Constructions and the Composition Hierarchy.....	215
Table A2-5: General Constructions and the Type Hierarchy .....	218
Table A2-6: Meaning Schemas .....	220
Table A2-7: The RefExpr Hierarchy .....	223
Table A2-8: Declarative Sentences.....	225
Table A2-9: Imperative Sentences.....	226

Table A2-10: Questions .....	228
Table A2-11: Conditional Sentences .....	228
Table A2-12: Relative Clauses .....	229
Table A2-13: Until Clauses.....	229
Table A3-1: Constructions for CS-A3.1 .....	232
Table A3-2: Schemas for CS-A3.1 .....	233

## List of Figures

Figure 1-1: Comprehension in an autonomous agent .....	5
Figure 1-2: Results of processing a simple sentence .....	7
Figure 1-3: Incremental processing cycles .....	8
Figure 1-4: General cognitive mechanisms in the CMC .....	9
Figure 2-1: An abstract ITL agent.....	20
Figure 2-2: Lucia embodied in Rosie.....	21
Figure 2-3: Schematic of Lucia in Rosie.....	22
Figure 2-4: Rosie's games and puzzles.....	23
Figure 2-5: Rosie solving Tower-of-Hanoi.....	23
Figure 2-6: A script for teaching Tower-of-Hanoi.....	24
Figure 2-7: Rosie working in the kitchen .....	25
Figure 2-8: Rosie's <i>kitchen</i> task .....	26
Figure 2-9: A comprehension algorithm for Rosie .....	27
Figure 2-10: Ways to test Lucia's output .....	40
Figure 3-1: Results of processing a simple sentence .....	50
Figure 3-2: An example of ECG items .....	51
Figure 3-3: A formal definition of the ECG language .....	53
Figure 3-4: Lexical constructions for CS-3.1 .....	55
Figure 3-5: Meaning structures for CS-3.1 .....	56
Figure 3-6: The network of constructions for a complex referring expression .	57
Figure 3-7: Details of syntax with semantic precision .....	58
Figure 3-8: Construction network for a simple command.....	59
Figure 3-9: Construction network for a complex declarative.....	61
Figure 3-10: Construction network for questions.....	62
Figure 3-11: Construction network for conditional sentences .....	63
Figure 3-12: Growth of ECG items for the Baseline corpus .....	64

Figure 3-13: Growth of ECG items for the Games corpus.....	66
Figure 3-14: Growth of ECG items for the Robot corpus .....	66
Figure 3-15: Generality for sentence forms .....	74
Figure 3-16: Generality for all sentences .....	75
Figure 4-1: Comprehension in context.....	83
Figure 4-2: Cycles in incremental comprehension processing .....	84
Figure 4-3: The grounded meaning of a sentence.....	87
Figure 4-4: The final structure of the example sentence.....	88
Figure 4-5: Fifteen construction cycles for the example sentence .....	90
Figure 4-6: Full tree for a simple command .....	93
Figure 4-7: Details of construction instances.....	94
Figure 4-8: Details of schema instances .....	95
Figure 4-9: A local repair.....	102
Figure 5-1: General cognitive mechanisms in the CMC (from Laird, Lebiere, et al., 2017) .....	109
Figure 5-2: The Soar Cognitive Architecture (from Laird et al., 2017) .....	110
Figure 5-3: The Soar Decision Cycle .....	111
Figure 5-4: The ACT-R Cognitive Architecture (from Laird, Lebiere, et al., 2017)	
.....	112
Figure 5-5: Lucia as embedded within Rosie.....	115
Figure 5-6: The ECGtoSoar translator .....	116
Figure 5-7: Soar trace for a simple sentence .....	117
Figure 5-8: Inside a match-construction decision cycle.....	119
Figure 5-9: Construction cycles for a sentence with local repair.....	121
Figure 5-10: A prepositional phrase requiring a local repair.....	136
Figure 5-11: Using a snip for a local repair.....	137
Figure 5-12: A repair combined with lexical selection .....	139
Figure 5-13: A deliberative repair .....	142
Figure 6-1: Garden path failure and recovery .....	152
Figure 6-2: A parse that succeeds and one that fails.....	156
Figure 6-3: Semantics of a parsing breakdown .....	158

Figure 6-4: Control sequences for System A and System B .....	161
Figure 6-5: ECG lexical constructions and context concepts .....	164
Figure 6-6: Composite selection for a local repair .....	166
Figure 6-7: Spread of attention in working memory .....	167
Figure 6-8: An example of skill learning .....	169
Figure 6-9: Learning from situated language episodes (from Mok, 2009, p. 8) .....	177
Figure 6-10: The three-phase theory of language acquisition .....	179
Figure A3-1: Analysis of CS-A3.1.....	231
Figure A3-2: Complete meaning structure for CS-A3.1 .....	233
Figure A3-3: Structures built for CS-1.....	234
Figure A3-4: A local repair.....	236
Figure A3-5: Syntactic structure of a monster sentence.....	252
Figure A3-6: Semantic structure of a monster sentence.....	253

## **List of Appendices**

Appendix 1 The Rosie Corpora	192
A1.1 The Baseline Corpus	192
A1.2 The Games Corpus	199
A1.3 The Robot Corpus	207
Appendix 2 The Lucia ECG Grammar for Rosie	212
A2.1 Lexical Constructions	212
A2.2 Composite Constructions	215
A2.3 General Constructions	218
A2.4 Meaning Schemas	220
A2.5 Referring Expressions	223
A2.6 Sentences	224
A2.6.1 Declarative sentences	225
A2.6.2 Imperative sentences	226
A2.6.3 Questions	228
A2.6.4 Conditional sentences	228
A2.7 Subordinate Clauses	229
Appendix 3 Case Studies	230
A3.1: Simple Commands	231
A3.2: Complex Referring Expressions	235

A3.3: Simple Declaratives	241
A3.4: Complex Sentences	242
A3.5: Sentences That Are Problematic for Humans	249

## **Acronyms**

**CKM** – Composable knowledge of the meaning of language

**E3C** – Embodied, end-to-end comprehension

**ECG** – Embodied Construction Grammar

**GCM** – General cognitive mechanisms

**I3P** – Incremental, immediate interpretation processing

**LAE** – Language acquisition from experience

## **Abstract**

AI systems with language for robots don't try to model human processing. Psycholinguistic models of human language processing don't have operational computational models. To solve these problems, this thesis contributes to progress in answering two interlocking scientific questions: how does the human mind do sentence comprehension, and how can we enable artificial agents to use natural language to collaborate with humans. We do this with a system called Lucia, which is a computational cognitive model of human sentence comprehension that works by constructing the meaning of a sentence piece by piece.

The Lucia model is designed according to five overriding qualitative principles of human language comprehension. To show that its results are useful, it does embodied, end-to-end comprehension (E3C) within an artificial agent called Rosie. To model key characteristics of human comprehension, it draws on research in cognitive linguistics, psycholinguistics, artificial intelligence, and robotics to: represent composable knowledge of the meaning of linguistic forms (CKM), do incremental, immediate interpretation processing (I3P), and do it using general cognitive mechanisms (GCM). The model leads to a theory of language acquisition from experience (LAE), some parts of which have been implemented experimentally.

To conform to these principles, the Lucia model is implemented in a robotic agent called Rosie to do E3C. It uses Embodied Construction Grammar (ECG) as its method of representing composable knowledge of meaning (CKM), and demonstrates that this knowledge can be processed incrementally (I3P) using a novel comprehension algorithm that relies on the general cognitive mechanisms (GCM) of the Soar cognitive architecture to produce embodied, end-to-end comprehension (E3C).

Lucia makes several contributions to answering the broader scientific questions. It provides a novel theory for incremental processing (I3P) based on a three-phase construction cycle. It provides a theory of how memories interact during comprehension. It demonstrates grounded comprehension in an embodied robotic agent. Finally, it provides a detailed, functional model of cognitive E3C processing that can serve as a basis for further research in modeling human language processing in the brain and in designing larger-scale language models for artificial agents.

## **Chapter 1 Introduction**

This thesis addresses the question of how a computational cognitive model of human sentence comprehension within an autonomous robotic agent can model qualitatively certain aspects of human comprehension in the context of understanding natural language instructions from a human instructor. Our approach is to draw inspiration from human language processing to develop a computational model that is useful for an autonomous robot.

Previous research has led to many insights on specific aspects of human language processing. Still missing is a comprehensive model of comprehension that addresses five key qualitative principles of human processing: 1) the external functional property of embodied, end-to-end comprehension; the internal *cognitive* properties of 2) representing composable knowledge of the meaning of language to provide generality, 3) incremental processing with immediate interpretation, and 4) performing real-time comprehension using domain-general cognitive mechanisms; and 5) the ability to acquire language from experience.

This thesis describes *Lucia*, an implemented theory of sentence comprehension that serves as the language comprehension component of an embodied autonomous agent called *Rosie*, which learns new tasks from situated interactive instruction. Lucia conforms to the first four of these qualitative principles and provides insight for pursuing the fifth. This research gives insight into two important questions about human-like language comprehension.

### **Q1: How do humans comprehend sentences?**

Natural language is an important part of human cognition. Much research has been done on ways of measuring human language processing from the

outside. A problem for cognitive science is the lack of the ability to see inside the mind and brain to understand in detail their inner workings. This leaves the important question of how the internal mechanisms of human sentence comprehension work. Lucia provides some insight into this question by providing a working model of the computational processing required for language comprehension.

**Q2:** *How can robots comprehend sentences well enough to collaborate with humans?*

A computational cognitive model of human sentence processing could guide artificial agent development. Lucia is an implemented model that conforms to the first four functional and cognitive principles mentioned, and is designed to serve as a basis for future work on acquisition. It satisfies the sentence comprehension needs of a robotic agent within the domain considered here. Lucia does not attempt to model all the quantitative details of human processing; rather it satisfies the functional principle while being implemented according to the three cognitive principles.

### **1.1 Defining the Problem**

This section further defines the five qualitative principles listed above, which become constraints, or goals, for the model of sentence comprehension presented in this thesis. Satisfying each of them will contribute to answering the cognitive science question (Q1) and provide guidance for developing artificial agents (Q2). The thesis demonstrates how the implemented Lucia system satisfies the four functional and cognitive principles, and provides a theory for the acquisition principle, only small parts of which have been implemented.

**E3C: Embodied, End-to-End Comprehension – Human-like comprehension occurs within an embodied agent, human or artificial, so that the agent can act in the world in a way corresponding to the intent of the speaker and the goals of the agent.**

Human-like comprehension by either a human or an artificial agent collaborating with a human instructor in a real-world environment requires transforming each input sentence into an internal meaning representation

grounded to the agent's perception, action abilities, and knowledge of the world in order to take internal or external action based on the meaning representation.

**CKM: Composable Knowledge of the Meaning of Language** – *Knowledge of the meaning of language is represented in composable units, called constructions, in order to have generality, so that it can be used to comprehend (E3C) many sentences that have never been previously experienced.*

Compositionality is necessary to achieve the generality and creativity of human language use, and to give flexibility and generality to the language capability of an artificial agent. Knowledge of the meaning of language maps small composable units of linguistic form to their internal meanings with form-meaning mappings that can be composed into larger and larger units, to ultimately construct the meaning of a sentence.

**I3P: Incremental, Immediate Interpretation Processing** *The meaning of a sentence is constructed incrementally in real time, each new construction unit (CKM) added is immediately grounded (as possible) to embodied world knowledge (I3P), and the meaning of each full sentence is interpreted as an action message (E3C).*

Psycholinguistic evidence given in Chapter 4 shows that during sentence comprehension humans immediately ground the meanings of words and phrases to their internal representations of the world, sometimes making commitments that must later be retracted when more of the sentence is processed. This key property of human comprehension is a challenge for computational models that Lucia is designed to meet.

**GCM: General Cognitive Mechanisms** – *Comprehension processing is done using domain-general mechanisms of cognition. In practice, this implies that our computational cognitive model will be built using a cognitive architecture.*

This work assumes that human language processing uses domain-general cognitive mechanisms rather than any innate language-specific faculty in the brain. Many researchers agree with this assumption (Christiansen & Chater, 2016; Dąbrowska, 2015; Newell, 1990), while others disagree (Chomsky, 1986).

We argue that the success of this work can lend support to our assumption. Thus, we create our model within a cognitive architecture.

**LAE: Language is Acquired from Experience** – *Knowledge of how to map form to meaning is learned incrementally from individual experiences where situational knowledge and reasoning are used to learn increments of form-meaning mapping called constructions, and these constructions are modified as indicated by further experiences.*

Research on language acquisition (Goldberg, 2019; Krashen, 2003; Tomasello, 2003) suggests that humans acquire language incrementally from experience. An artificial agent that can learn more language as it interacts with a human collaborator can collaborate better in the future. An automatic system for acquiring Lucia's knowledge of the meaning of language is beyond the scope of this thesis. However, we describe a theory of how this could be done that is consistent with the existing components of Lucia and explore an implementation of some parts of that theory.

Our approach to creating a computational cognitive model to satisfy our qualitative principles comes from a branch of science that overlaps both cognitive science and artificial intelligence: the study of cognitive architectures. The theory developed there is consistent with the *cognitive architecture principle* (Lehman et al., 1996):

$$\text{ARCHITECTURE} + \text{KNOWLEDGE} = \text{BEHAVIOR}$$

The behavior we want to model is embodied, end-to-end comprehension within an agent that performs *interactive task learning* (ITL; Gluck & Laird, 2018). The architecture we use is the Soar cognitive architecture (Laird, 2012), which contains memories and processing mechanisms intended to model the general mechanisms of human cognition. The work of this thesis developed the knowledge necessary to make this architecture produce the desired language comprehension behavior within the ITL behavior of the Rosie agent. That knowledge defines Lucia, which models human comprehension computationally

by constructing meaning, piece by piece. This makes it a computational cognitive model of human sentence comprehension.

## 1.2 The Lucia Comprehension System

This section introduces Lucia and describes abstractly how it satisfies each of the four plus one principles of human-like sentence comprehension we have defined, as well as the implementation commitments we make to achieve this.

### 1.2.1 E3C: Embodied, End-to-End Comprehension

For a robot to perform tasks in the real world based on interactive instruction from a human, it must be able to create a grounded interpretation, called a *message*, for each input sentence. For our purposes we define *language understanding* in an artificial agent to mean the agent can act appropriately in the world based on its language input. We define *end-to-end comprehension* as the ability to transform each input sentence into an internal message grounded to the agent's knowledge such that the agent can then interpret that message to take appropriate internal or external action.

Figure 1-1 shows how an agent could achieve language understanding by coordinating, using shared knowledge, a Comprehender subsystem with an Operations subsystem, each of which has its own private knowledge.

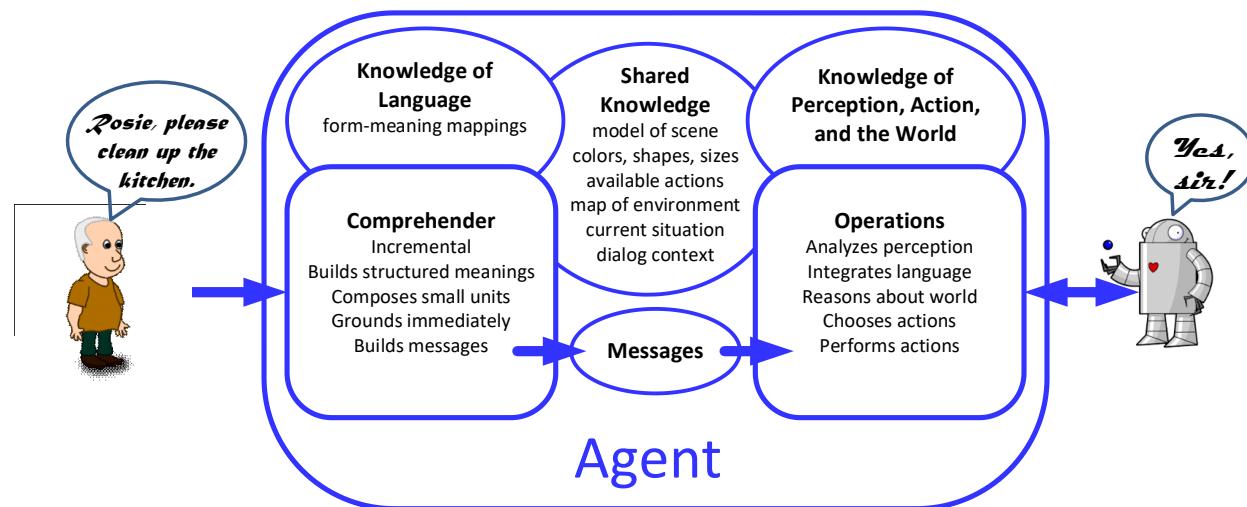


Figure 1-1: Comprehension in an autonomous agent

The Comprehender subsystem, of which Lucia is one possible implementation, does the sentence comprehension part. Consider sentence (1) which a human instructor might say to Rosie:

- (1) Pick up the green sphere.

For Rosie to act on this sentence, it needs a message defining a command with internal symbols (which we call *identifiers*) that identify the action to take and an object to act on, as in (2):

- (2) command(op\_pick-up1, 015)

This message is grounded to the agent's knowledge and is actionable, leading to:

**Commitment 1:** *Lucia performs E3C by being embodied in Rosie and producing an internal message for each input sentence as needed for ITL.*

Chapter 2 describes the sentences Rosie uses to learn a variety of new tasks, the interfaces between Lucia and Rosie, the short- and long-term knowledge they share, and the kinds of messages Lucia can send to Rosie.

### **1.2.2 CKM: Composable Knowledge of Meaning**

Comprehending a sentence (E3C) involves using CKM to transform its *form*, a sequence of words, into an internal message, which is its *meaning*. A trivial approach to this problem would be to use a lookup table that maps every sentence in the agent's domain to its corresponding message. However, human knowledge of language has the *generality* to comprehend an unbounded number of sentences that have never been heard or seen before, and an artificial agent needs a degree of this generality.

One way to achieve generality is for knowledge of mapping form to meaning to be *composable*, consisting of small units of form-meaning mapping that can be composed into larger structures. A branch of cognitive linguistics is devoted to theories of *construction grammar* (CxG; Hoffman & Trousdale, 2013), where *constructions* are units of form-meaning mapping that can be composed into sentence meanings. Figure 1-2 sketches the mapping of (1) to (2) and the

*comprehension state graph* in between. Each node represents a construction with its meaning, and these are grounded as shown by the green arrows.

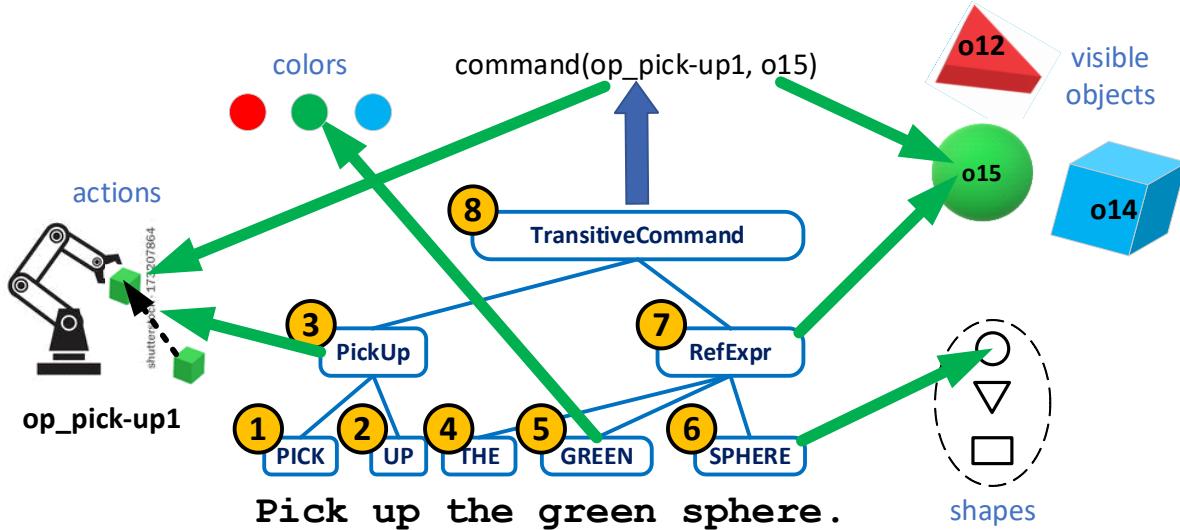


Figure 1-2: Results of processing a simple sentence

Lucia adopts the *Embodied Construction Grammar* (ECG; Bergen & Chang, 2013) theory, in which a *construction* describes an input form, names a *schema* which it *evokes* to represent its meaning, and specifies how to *populate* the parts of that schema with the information needed for grounding.

**Commitment 2:** Lucia uses ECG to represent composable knowledge of the meaning of language (CKM).

Chapter 3 gives details of the ECG formalism (Bryant, 2008) used to define a large network of ECG elements called a *grammar*. Lucia uses that formalism to represent composable knowledge and to comprehend the sentences needed for Rosie's ITL.

### 1.2.3 I3P: Incremental, Immediate Interpretation Processing

Evidence discussed in Chapter 4 suggests that humans achieve real-time comprehension by *incremental*, word-by-word processing with *immediate interpretation*, grounding each word or phrase to the current context as it is processed. In Figure 1-2, the nodes are numbered in the order they are created,

with one lexical node for each word and additional composite nodes added: node 3 while processing *up*, and nodes 7 and 8 while processing *sphere*.

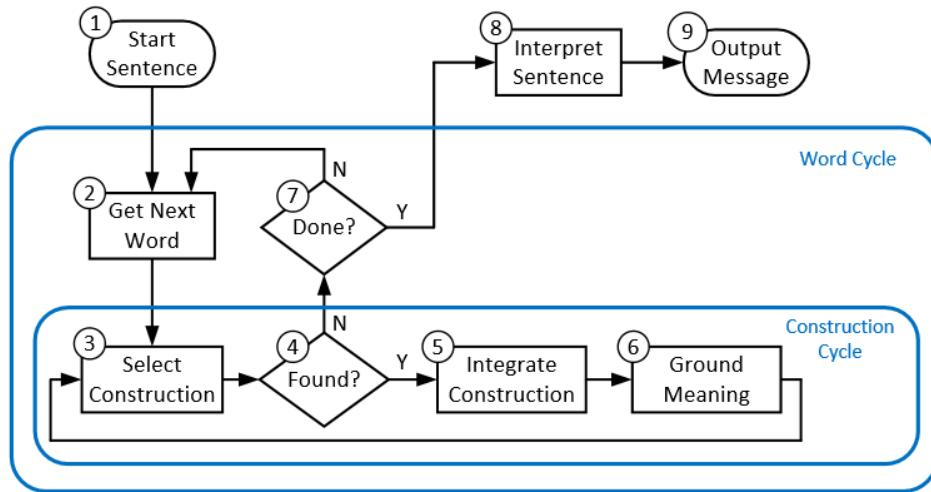


Figure 1-3: Incremental processing cycles

Lucia constructs meaning piece by piece using a word cycle and a construction cycle, as shown in Figure 1-3. Each construction cycle has three phases: a *selection phase* chooses which construction to apply next, an *integration phase* builds a new form-meaning node and adds it to the graph, and a *grounding phase* grounds the node's meaning to the agent's world knowledge. The selection phase resolves many local ambiguities to result in a single grounded path through the space of possible comprehensions, and local repairs are made later if subsequent input indicates that a previous choice was incorrect.

**Commitment 3:** *Lucia does immediate interpretation processing (I3P) using construction cycles that select, integrate, and ground one construction at a time, often with multiple construction cycles for a single input word.*

Chapter 4 describes this incremental processing algorithm in detail, using a number of case studies to show how ambiguities are resolved in construction selection and how local repairs are made when necessary.

#### 1.2.4 GCM: General Cognitive Mechanisms

Human language processing is done using the cognitive mechanisms of the human mind and brain, and we model these using the mechanisms of a cognitive

architecture. Following decades of work on cognitive architectures (Kotseruba & Tsotsos, 2020), Laird, Lebiere, and Rosenbloom (2017) abstracted ideas from several of the prominent architectures to produce a theoretical model now called *The Common Model of Cognition (CMC)*. We take this as a guide to general cognitive mechanisms.

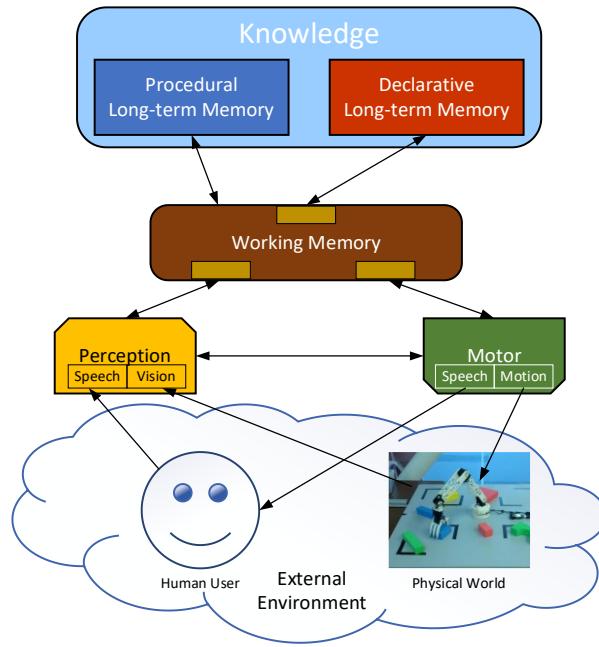


Figure 1-4: General cognitive mechanisms in the CMC

Figure 1-4, adapted from Laird et al. (2017), shows the structure of the CMC with a working memory (WM) at its center that communicates with knowledge in long-term memories as well as perception and motor modules. Rules in procedural long-term memory that match against WM are selected to fire and change WM in a *cognitive cycle*, approximately every 50ms in humans. Rosie and Lucia are implemented in the Soar cognitive architecture (Laird, 2012), one of those on which the CMC is based.

**Commitment 4:** *Lucia is built within the Soar cognitive architecture.*

Lucia's CKM is implemented as procedures stored in Soar's procedural long-term memory that implement each element of the ECG grammar. Chapter 5 describes in detail how Lucia uses the general cognitive mechanisms in Soar

to achieve end-to-end comprehension and discusses Lucia’s real-time performance. Chapter 6 describes an experimental version where CKM is stored in Soar’s declarative long-term semantic memory and through run-time interpretation and compilation is transformed into procedural knowledge by Soar’s learning mechanism.

The above four principles have been inspired by the literature on human comprehension, and the thesis demonstrates that Lucia does in fact function in all these ways. However, circling back to compare Lucia’s results on these points to data on human performance is something we leave for future work.

### **1.2.5 LAE: Language is Acquired from Experience**

The Lucia system as implemented does not include automatic LAE. We have explored a theory and practice toward implementing such a capability, which is described in Chapter 6.

## **1.3 Related Work**

From the literature on computational language comprehension, we have assembled a list of systems that attempt to satisfy two or more of the human-like principles defined above. This section analyzes briefly the goals, approach, implementation, and results of each system, comparing them to our five principles. Many of these systems differ from Lucia primarily in their ad-hoc approach to knowledge of the meaning of language as compared to our use of construction grammar theory, and that many do not do incremental processing or use a cognitive architecture. Table 1-1 summarizes our analysis.

### **1.3.1 Classic systems**

SHRDLU (Winograd, 1972) is “a computer system for understanding English” built with modules written in LISP for syntax, semantics, action planning, and language generation. It succeeds in processing complex dialogs in a simulated blocks-world environment. It has a procedural language for syntactic analysis that parses incrementally left-to-right, but without the generality from a compositional theory of form-meaning mapping. Its grounding is done after the

fact, rather than immediately. It makes no attempt to model human cognitive mechanism or propose a theory of language acquisition.

LAS/HAM (Anderson, 1977) is a system for “The study of language acquisition.” The system, also written in LISP, is based on a general theory of mathematical induction and a memory system called HAM that is intended to model human associative memory, but is not a complete cognitive architecture. It has program code to construct “HAM conceptualizations” of meaning for comprehension that are far from actionable sentence meanings. It uses an “augmented transition network” to represent syntax, but without compositional mapping to meaning. It parses left-to-right, but does not do grounding or I3P. It succeeds in inducing syntax rules for a very limited context-free grammar.

### **1.3.2 Systems in Robots (E3C)**

Tellex et al. (2011) present a system that does E3C in a robot using a probabilistic language model. Later work in this group (Gopalan et al., 2020; Nguyen et al., 2020) replaces this model with one using a recurrent neural network. Both do E3C in their limited domains and some language learning. Neither of these approaches models the cognitive aspects of CKM, I3P, or GCM.

Chai et al. (2016; Liu et al., 2016) have built systems to teach tasks to robots with a combination of language instruction and visual demonstration. These systems do E3C using an off-the-shelf semantic parser. That parser has general knowledge of syntax and semantics, but ignores incremental grounding and modeling cognitive mechanisms. Its knowledge comes from batch learning from large corpora, but not acquisition from individual experiences (LAE).

SGGOL (Chen, 2012; Chen & Mooney, 2011) is a system that learns a lexicon and a semantic parser for understanding navigation instructions in a simulated environment. Their approach has a context-free grammar that is learned from a large dataset of labeled instructions. The semantic lexicon is learned incrementally from individual examples using free-form instructions built by users on Amazon Mechanical Turk. The simulated environment is quite simple. End-to-end performance with the learned knowledge is about 58% for

individual sentences and 20% for complete tasks. No attempt is made to model human processing.

LEIA (McShane & Nirenburg, 2019) is a theory of “language-endowed intelligent agents (LEIAs)” built on the OntoAgent cognitive architecture. This approach satisfies E3C in a robot doing a form of ITL. The overall agent uses a cognitive architecture, but the “Language Understanding Service” uses “externally developed engines” in a separate module that does not do I3P or use GCM. The model has some ability to learn new words.

### **1.3.3 Construction Grammar Systems (CKM)**

ECG-ICSI (Eppe, Trott, & Feldman, 2016; Eppe, Trott, Raghuram, et al., 2016) added a back end to the original ECG processor (Bryant, 2008) to connect to a robot. Bryant’s semantic parser uses ECG as CKM to produce semantic representations of sentences using a global optimization approach, which is not incremental and does not consider GCM. The added back end provides grounding to do E3C with the robot. (Chang, 2008) and (Mok, 2009) have done experiments on language acquisition from experience (LAE) within the ECG context.

Fluid Construction Grammar (FCG; Steels, 2013; Steels & Hild, 2012) is designed to experiment with multiple agents evolving a language from scratch. Its constructions are bi-directional for use with both comprehension and production. Processing is incremental without immediate interpretation. There is no consideration of GCM. FCG has been used for LAE based on “Insight Grammar Learning” (Garcia-Casademont & Steels, 2016; van Eecke & Steels, 2016).

### **1.3.4 Systems Developed in a Cognitive Architecture (GCM)**

NL-Soar (Lewis, 1993) is built in the Soar cognitive architecture and is intended to model human comprehension in a way “that accounts for a broad range of psycholinguistic phenomena,” primarily those involving syntactic structures. It does not include a theory of compositional form-meaning mapping. It shows incremental processing in simulated real time, but has no embodiment to ground

to for E3C or full I3P. It demonstrates parsing using GCM. It does not do full LAE but does do learning of a skill for syntactic processing.

ACT-R-2005 (Lewis & Vasishth, 2005) is a model built in the ACT-R cognitive architecture designed to support a theory of sentence processing based on memory retrievals with the goal of simulating human reading time data. It models cue-based retrievals of long-distance dependencies where other items in memory may interfere. Its simulations match some human data sets well and others not so well. Although it is implemented in a cognitive architecture, it does not do grounding or produce full sentence meanings, and it does not include a theory of how language is acquired.

ACT-R-RR (J. Ball et al., 2010) is an ACT-R model based on the Double-R theory of language comprehension (J. T. Ball, 2004). This model of comprehension has been used in a “synthetic teammate” (Demir et al., 2015; McNeese et al., 2018) where it simulates a human pilot interacting with human teammates in real time. It does E3C and uses concepts similar to construction grammar for CKM. It does not do I3P or LAE.

ACT-R-2020 (Jones, 2020) is an ACT-R model of sentence processing whose goal is to do sentence processing for both English and Korean using the same procedural knowledge for both and language-specific declarative knowledge of words. Jones explores the question of how to represent working memory limits in ACT-R “parsimoniously.” The model represents grammatical structure according to Lexical Functional Grammar (LFG) without any embodiment or grounded semantics. It processes word-by-word, but without immediate interpretation. The paper does not address language acquisition.

### **1.3.5 Lucia**

LUCIA is the reference system described in this thesis and elsewhere (Lindes, 2018, 2019, 2020; Lindes et al., 2017; Lindes & Laird, 2016, 2017a, 2017b). As we show in subsequent chapters, within the limits of the Rosie domain it achieves E3C using CKM, I3P, and GCM. We also present exploration of a theory of LAE, and evidence of implementation of some parts of that theory. Overall, no

other system has attempted to answer our original Q1 or Q2 with all five of the properties of human language comprehension we have defined.

### 1.3.6 Summary

Table 1-1 summarizes our best-effort evaluations for all these systems.

Table 1-1: Lucia compared to AI models

System	E3C	CKM	I3P	GCM	LAE
SHRDLU	+	~	~	-	-
LAS/HAM	-	~	-	~	~
Tellex et al.	+	-	-	-	~
Chai et al.	+	~	-	-	-
SGOLL	~	+	-	-	+
LEIA	+	~	-	-	~
ECG-ICSI	+	+	-	-	~
FCG	+	+	~	-	+
NL-Soar	-	~	~	+	~
ACT-R-2005	-	~	-	+	-
ACT-R-RR	+	~	~	+	-
ACT-R-2020	-	~	~	+	-
LUCIA	+	+	+	+	~

**Key:**  Does this well  Does some of this  Does not do this

### **1.3.7 Machine Learning Systems**

There is a substantial literature on semantic parsers (Artzi et al., 2015; Damonte & Monti, 2021; Goldwasser & Roth, 2011; Liang, 2016) that build up their knowledge of language by statistical analysis of large corpora. These systems produce formal semantic representations from natural language input but without incremental grounding or conforming to any of our cognitive properties. Several of the robotic systems cited above use semantic parsers, while others are not embodied to do E3C, so they are not included in our comparisons.

Today's literature on natural language processing is dominated by large-scale artificial neural networks (Brown et al., 2020; Devlin et al., 2019; Vaswani et al., 2017). These systems do a form of language acquisition by processing corpora of many millions of sentences. They do not produce correct, grounded, actionable messages for individual sentences, they do not create identifiable compositional knowledge of meaning, they do not do incremental, immediate interpretation, nor do they explicitly attempt to model cognitive mechanisms. They do not model incremental acquisition from individual language experiences. Their goals are different ours, so it is difficult to compare them directly.

## **1.4 Contributions and Explorations**

Here are the five principal contributions we see Lucia making, which will emerge in subsequent chapters and are discussed further in Chapter 7:

- C1:** *A demonstration of incremental, cognitive, embodied, end-to-end comprehension (E3C) using Embodied Construction Grammar (ECG).*
- C2:** *A demonstration of incremental, immediate interpretation processing (I3P) based on the construction cycle.*
- C3:** *A theory of how memories interact during comprehension.*
- C4:** *A demonstrated model of grounding in an embodied agent.*
- C5:** *A detailed, functional model of E3C processing that can serve as a basis for future research.*

Chapter 6 presents three initial explorations that introduce possible lines of future research. The first shows how the Lucia's ECG grammar can be extended to cover aspects of human language not present in the sentence

corpora used for Lucia development, and how the Lucia model can explain certain limitations of human processing where seemingly grammatical sentences seem difficult or impossible to understand.

The second explores an alternative way of using the Soar architecture. A new version of Lucia, which we call System B, stores Lucia's grammar as declarative knowledge in Soar's semantic memory. Spreading activation enables bias from context to resolve ambiguity among word senses. For selection of composite constructions, extensions to the architecture were made to add an attention mechanism in WM and allow parallel retrievals. System B also converts the declarative representations of knowledge of meaning of language into procedural knowledge using Soar's existing chunking mechanism.

The third exploration involves the development of a three-phase theory of how CKM is acquired from experience. Deliberate reasoning creates hypotheses for new constructions based on individual episodes of experience, these episodes are generalized into declarative knowledge, and this is then gradually converted into procedural knowledge to become an automatic skill as it is processed. Exploratory experiments on parts of the theory are also covered.

## **1.5 Thesis Preview**

Chapters 2 through 5 discuss Lucia's implementation of E3C, CKM, I3P, and GCM, respectively. Each of these chapters includes a section that reviews briefly the related literature on that subject. After describing the implementation details, data on appropriate quantitative evaluations for that aspect are presented. Additional details are presented in appendices.

Chapter 6 begins with a brief summary of the achievements of the Lucia model, and some of its limitations. It then presents a more detailed description of our three exploratory experiments: extending the grammar for wider coverage and using it to model human parsing limitations, a System B with the grammar in semantic memory, and our theory of LAE and how it relates to the development and implementation of Lucia.

In Chapter 7, we discuss further contributions of this thesis and suggest future work. We discuss ways to scale up the model to much larger language coverage, ways to drill down to detailed modeling of human reading times and brain data, and how to model human language acquisition. Specific research questions are suggested.

Now we're ready to dive in. Happy reading!

## **Chapter 2 Embodied, End-to-End Comprehension**

This chapter presents the environment surrounding the computer program that is the computational implementation of Lucia's theoretical model of human sentence comprehension. Lucia is part of an AI agent called Rosie that has been developed over a number of years by John Laird's Soar research group at the University of Michigan (Laird et al., 2018). Rosie's purpose is to learn tasks in certain well-defined domains in the physical world, or a simulation of it, using Interactive Task Learning (ITL; Gluck & Laird, 2018; Laird et al., 2017). As a major component of Rosie, Lucia takes each input sentence and transforms it into an internal message that enables Rosie to act, either internally to increase its knowledge of the task being taught, or externally in the world in which it performs that task. In summary Lucia, performs *embodied, end-to-end comprehension* (E3C).

**E3C: Embodied, End-to-End Comprehension** – *Human-like comprehension occurs within an embodied agent, human or artificial, so that the agent can act in the world in a way corresponding to the intent of the speaker and the goals of the agent.*

Showing that Lucia's comprehension is sufficient to enable Rosie to learn and perform new tasks taught by a human instructor using situated interactive instruction is evidence that Lucia does embodied, end-to-end comprehension.

This chapter describes the kinds of ITL tasks Rosie performs and the messages Lucia produces in order for Rosie to be successful in learning and performing these tasks. We discuss the various kinds of world knowledge that the agent uses, and how Lucia grounds the language to it. We evaluate end-to-end comprehension based on corpora of input sentences defined by the larger Rosie ITL project. A discussion of other systems that do E3C is in Chapter 1.

## **2.1 Defining the Problem**

Many cognitive psychologists argue that human cognition is deeply connected to the body (Barsalou, 1999; Fincher-Kiefer, 2019; Glenberg, 2015; Lakoff, 2012; van Elk & Bekkering, 2018). Cognitive linguists argue that language is inherently embodied (Johnson, 1987, 2018; Johnson & Lakoff, 2002), and that language for abstract concepts is derived from language for physical perception and action (Dove, 2014; Lakoff & Johnson, 1980). Evidence from neuroscience is that perceptual and motor areas of the brain are activated during comprehension (Bergen, 2012; Kemmerer, 2015; Pylkkänen, 2019). Embedding a comprehension model in an embodied agent helps model this aspect of human comprehension, and gives a direct, practical method to test whether the model produces meaningful messages from each input sentence.

Interactive Task Learning (ITL; Gluck & Laird, 2018) as a field studies “the processes by which new tasks are acquired through natural interaction between humans, humans and agents, and agents.” In this paradigm an agent interacts with both a human instructor and the physical world to learn arbitrary new tasks (Laird, Anderson, et al., 2017). Figure 2-1 gives an abstract overview of an ITL agent.

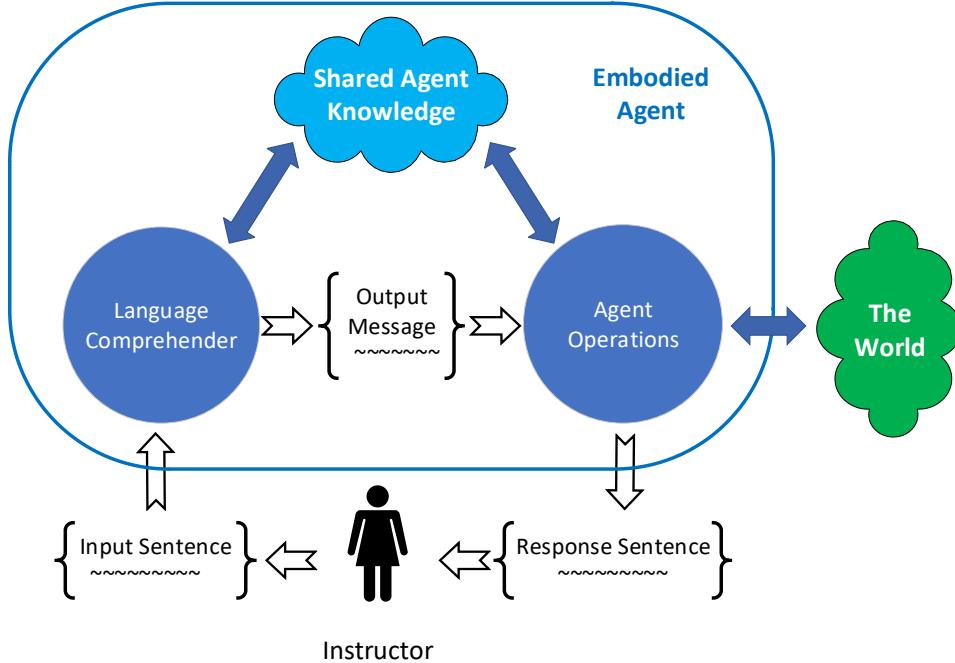


Figure 2-1: An abstract ITL agent

### 2.1.1 Lucia in context

Rosie (Laird et al., 2018) is an artificial agent used for research on a number of topics, including task learning (J. Kirk et al., 2016; J. R. Kirk & Laird, 2016, 2019; Mininger & Laird, 2018), human-robot interaction (Ramaraj, 2021; Ramaraj & Laird, 2018), and language comprehension (Lindes et al., 2017). Two recent dissertations (J. R. Kirk, 2019; Mininger, 2021) describe major ITL projects using Rosie and the language used for these projects. They provide a framework in which to test our theory of E3C. We show that Lucia does E3C by implementing the following commitment:

**Commitment 1:** *Lucia performs E3C by being embodied in Rosie and producing an internal message for each input sentence as needed for ITL.*

Figure 2-2 shows how Lucia fits into the overall Rosie agent, showing that the main interfaces between Lucia and the operational part of Rosie are shared world knowledge and action messages, and the fact that they are both implemented within the same Soar agent.

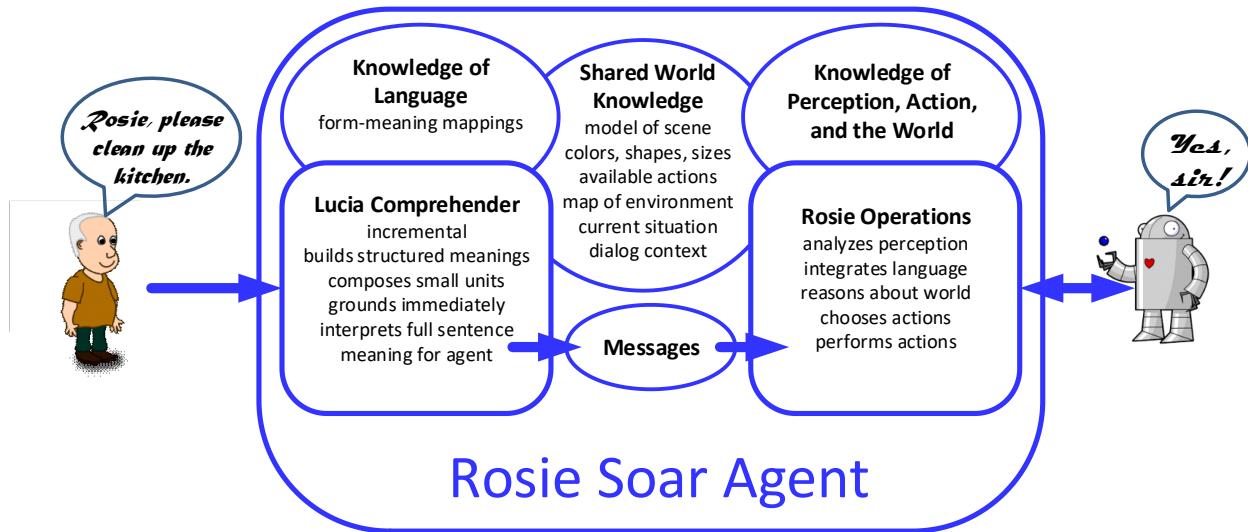


Figure 2-2: Lucia embodied in Rosie

Figure 2-3 is a schematic representation of the major computational processes and knowledge sources in the complete Rosie agent. Lucia uses knowledge of its ECG grammar that has been translated from a set of ECG files. Shared short-term knowledge, primarily about the scene provided by the visual system, is contained in a World Model in short-term memory. References to objects, locations, and their spatial relations are usually grounded to this scene data, and Lucia will add objects to the World Model when a sentence refers to an object not previously seen. Shared long-term knowledge, including bindings between language semantics and perceptual symbols, actions the agent knows how to perform, and knowledge of the larger physical environment, are in an Ontology in semantic memory. This shared world knowledge enables Lucia to do grounded comprehension and Rosie to act in the world based on language input.

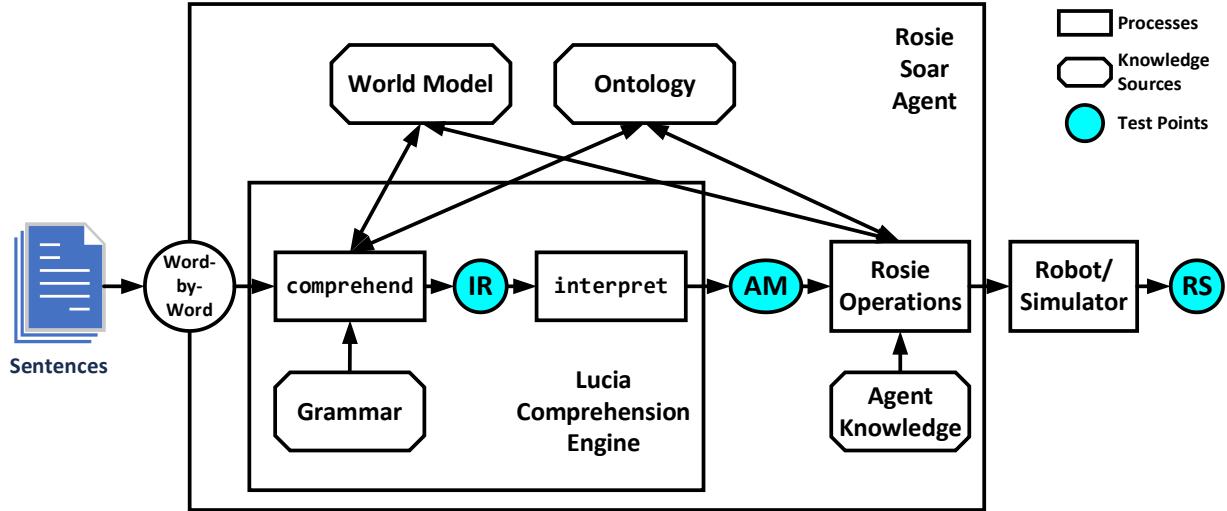


Figure 2-3: Schematic of Lucia in Rosie

Lucia's processing of a sentence consists of two phases: the *comprehend* process that incrementally transforms a sentence into an *internal representation* (*IR*) of the syntax, semantics, and grounding of the sentence, and the *interpret*, or *sentence interpretation*, process that formats an *action message* (*AM*) from the *IR*. The operations part of Rosie does its reasoning and learning and then acts in the world through its embodiment. Performing the task produces a *result state* (*RS*) that contains both the final state of the world after performing the task and the sequence of actions that were taken to get there.

Rosie learns tasks through human instruction, using language represented in three lists, or corpora, defined further below: a Baseline corpus used for initial Lucia development, a Games corpus used for learning games and puzzles, and a Robot corpus used for learning navigation and operations tasks.

### 2.1.2 Games and Puzzles

James Kirk (2019) added learning and reasoning logic into Rosie for learning how to solve puzzles and play games, and used ITL to teach Rosie 60 games and puzzles. Figure 2-4 is a list of the groups of games and puzzles that he implemented and that we used to develop and test Lucia.

**Grid Puzzles (7)**

**Chess Problems (7)**

**Side Swapping Puzzles (4)**

**River Crossing Puzzles (6)**

**Board Games (8)**

**Marking/Logic Puzzles (13)**

**Block Puzzles (9)**

**Solitaires (6)**

Figure 2-4: Rosie's games and puzzles<sup>1</sup>

In the puzzle called Tower of Hanoi, the task is to move a stack of three blocks, one at a time, from one of three locations to another, always maintaining graduated sizes in each stack, to rebuild the stack on a designated goal location.

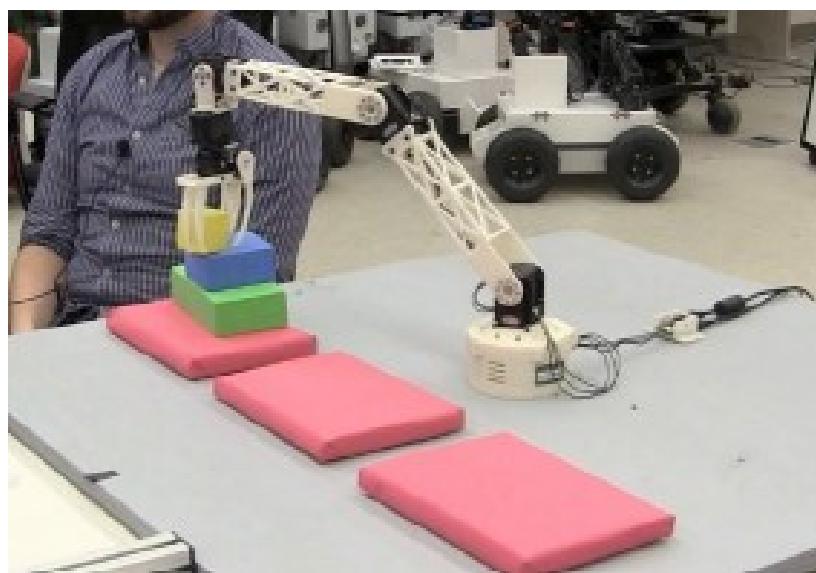


Figure 2-5: Rosie solving Tower-of-Hanoi<sup>2</sup>

Figure 2-5 shows Rosie connected to a table-top arm in the process of solving the Tower-of-Hanoi puzzle with blocks in the real world. James Kirk is the instructor in the background. Figure 2-6 is the script used to teach Rosie the goal and legal actions of the puzzle. Once these are learned, Rosie plans a solution and then moves the physical blocks to solve the puzzle.

---

<sup>1</sup> A complete list is online at <http://www-personal.umich.edu/~jrkirk/ijcai2019.html>, along with an instruction script for each, diagrams of the puzzles and games, and a video example.

<sup>2</sup> Photo from Kirk (2019), p. 10. Video at <https://www.youtube.com/watch?v=N6jOkKnpaHo>.

I: The name of the puzzle is tower-of-hanoi.
<i>The instructor has set up the blocks in the goal state.</i>
R: OK, what are the actions and goals of the game?
I: You can move a clear block onto a clear object that is larger than the block.
R: I don't know the concept clear.
I: If a block is not below an object then it is clear.
R: Ok, I now understand the concept clear.
R: I have learned the action.
I: The goal is that a large block is in a right location and a medium block is on the large block and a small block is on the medium block.
R: I've learned the goal.
<i>The instructor moves the blocks to an initial state.</i>
I: Done.
<i>Rosie moves the blocks, one at a time, according to the rules, until the goal state has been achieved.</i>
R: That was easy!

Figure 2-6: A script for teaching Tower-of-Hanoi<sup>3</sup>

Lines with “I:” are input sentences from the instructor and lines with “R:” are Rosie’s responses. Interleaved lines in italics are comments that describe the actions taken by Rosie or the instructor. Rosie is not solving the puzzle during the teaching phase. Rather, it is learning the task elements, which include task concepts, goals, actions, and failure states. After *Done.* is entered, Rosie plans the actions necessary to solve the puzzle and then executes them.

### 2.1.3 Robot tasks

Aaron Mininger (2021) developed the ITL knowledge in Rosie to learn and perform tasks involving navigating in a building environment and manipulating objects there. Figure 2-7 is a screen shot from a video of Rosie performing a number of tasks in a simulated kitchen.

---

<sup>3</sup> Lucia was tested on a different version of this puzzle script for use in simulation.

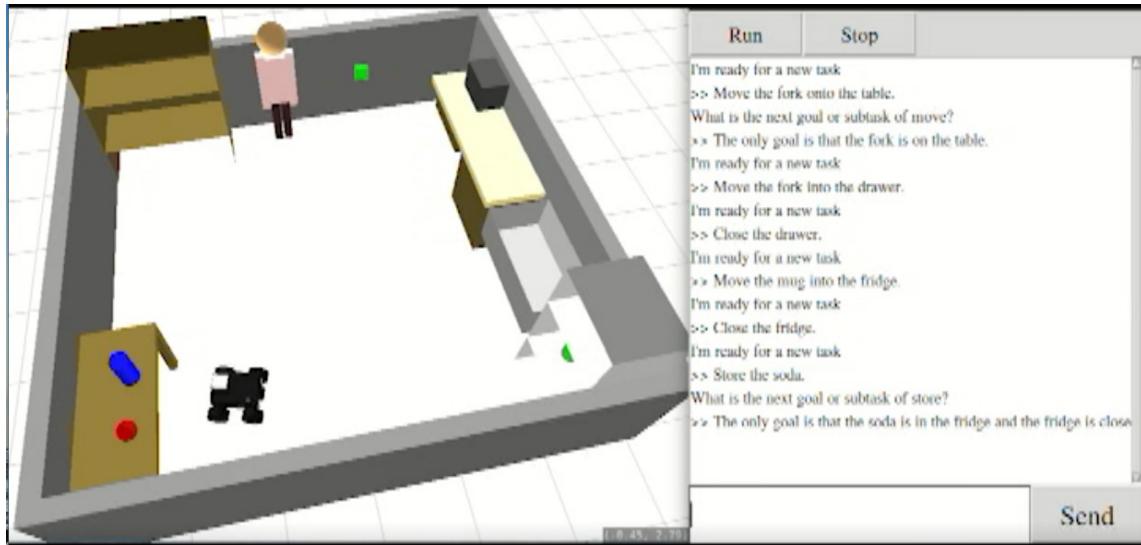


Figure 2-7: Rosie working in the kitchen

Figure 2-8 gives the script for this simple task to test Rosie's ability to move objects around in a kitchen.<sup>4</sup>

```

I: Move the fork onto the table.
Rosie tries to plan to move the fork and realizes it doesn't know how.
R: What is the next goal or subtask of move?
I: The only goal is that the fork is on the table.
Rosie picks up the fork and puts it down on the table.
R: I'm ready for a new task
I: Move the fork into the drawer.
Rosie opens the drawer, picks up the fork, and puts it down in the drawer.
R: I'm ready for a new task
I: Close the drawer.
Rosie closes the drawer.
R: I'm ready for a new task
I: Move the fork onto the table.
Rosie opens the drawer, picks up the fork, and puts it down on the table.
R: I'm ready for a new task
I: Move the mug into the fridge.
Rosie opens the fridge, picks up the mug, and puts it down in the fridge.
R: I'm ready for a new task
I: Close the fridge.
Rosie closes the fridge.
R: I'm ready for a new task
I: Store the soda.
Rosie tries to plan to store the soda and realizes it doesn't know how.
R: What is the next goal or subtask of store?
I: The only goal is that the soda is in the fridge and the fridge is closed.
```

---

<sup>4</sup> Videos of Rosie performing a variety of tasks using ITL are available at:  
<https://www.youtube.com/channel/UCf5R6KZ2JDZkjR2-CjgCnEw>.

<i>Rosie opens the fridge, picks up the soda, puts it down in the fridge, and closes the fridge.</i>
R: I'm ready for a new task
I: Store the apple.
<i>Rosie does all the steps of storing the apple without further instruction.</i>
R: I'm ready for a new task

Figure 2-8: Rosie's *kitchen* task

Rosie uses the knowledge it already has to decide how to act. Once Rosie has learned the goal of the *move* or *store* task, it plans how to execute the task. In these Robot tasks, the agent is moving around in the world during the dialog, so Rosie needs to keep track of its current state in the world. Lucia grounds each sentence to that current state, since objects will move around and doors will open and close, etc.

This is a fairly simple script, but Mininger (2021) has built the ITL capability in Rosie to do diverse tasks, both goal-based and procedural, using diverse actions, physical, mental, and communicative, and diverse control structures. This corpus of sentences greatly expands the language structures and meanings Lucia needs to handle, including language to describe time intervals, conditional or repeated execution, interrupts, naming people to interact with, and questions Rosie should ask or messages it should deliver.

## 2.2 Sentence Corpora

The development of Lucia's knowledge of language and processing has been done incrementally, one sentence at a time. Three different corpora of ITL sentences have been used, as Figure 2-9 shows. The Baseline corpus of 207 sentences that was used for the original Lucia development was selected from a much larger list of sentences used with Rosie to give a broad range of linguistic phenomena. The two other corpora, one for Games and Puzzles (J. R. Kirk, 2019) and one for Robot Tasks (Mininger, 2021), were built by concatenating all the sentences from all the scripts used for those respective domains, and then selecting all the unique sentences from that list.

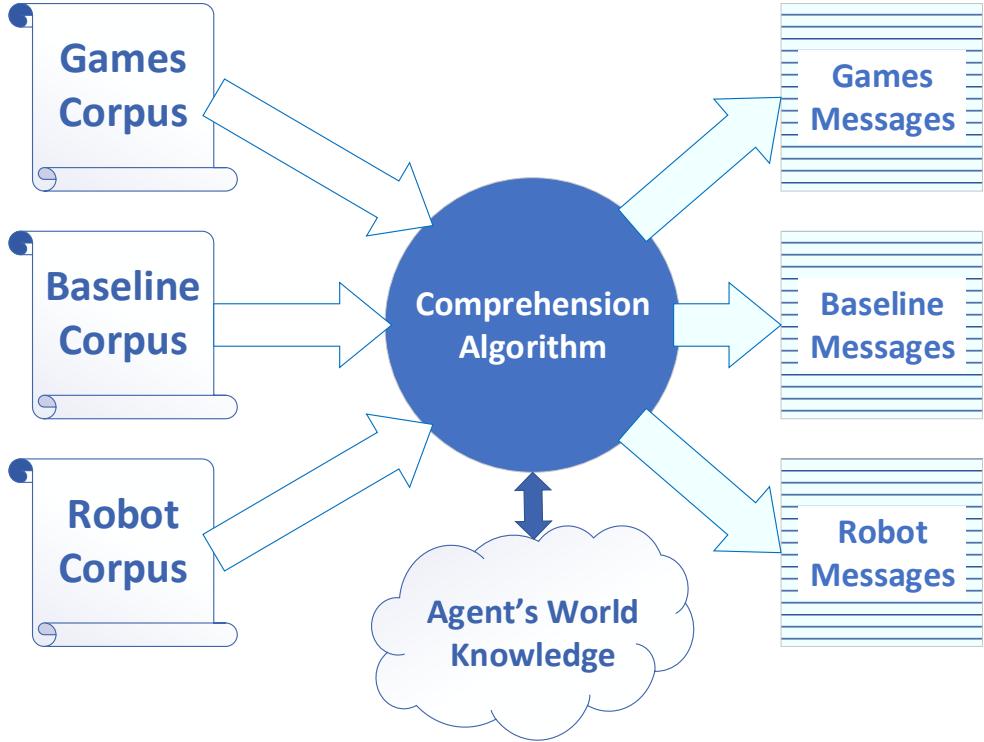


Figure 2-9: A comprehension algorithm for Rosie

Each sentence must be grounded to the current world state, which changes as Rosie executes a script. To test Lucia on each sentence independently, different world states are set up for each sentence when necessary. Appendix 1 lists all the unique sentences for each corpus, along with some additional information on the development needed for each sentence and the world models required.

These sentences were developed by James Kirk and Aaron Mininger, with help from John Laird. They created the sentences that were needed to provide instruction for each of the tasks implemented. They also determined what the output message should be for each sentence. In developing the sentences and messages, there was a preference to reuse existing message and grammatical structures. Thus, some of the sentences have somewhat tortured grammatical structure, and the message structures can be similarly idiosyncratic.

Historically, during the development of the Rosie agent, there have been two different versions of the comprehension algorithm. The first we call the *Laird*

*parser*, a system built by John Laird that was inspired by construction grammars but did not use the ECG formalism and instead used an ad-hoc grammar developed incrementally in parallel with Rosie. Later Lucia was created to take its place. Development of Rosie was done with the Laird parser, and the messages it produces are used as a “gold standard” reference for the development of Lucia, although some of the resulting messages are idiosyncratic due to the ad-hoc nature of their development. These sentence/message pairs were given to the Lucia project as is, and Lucia must turn each input sentence into a correct internal message with the information needed to allow Rosie to do the rest of the reasoning necessary to learn the task. If a message does not convey the right meaning, Rosie will fail in learning its task. As we shall see Chapter 3, Lucia and Rosie can handle a much larger space of sentences than those in these corpora.

### **2.3 Knowledge of the World**

There are two main interfaces between the Lucia Comprehender and Rosie Operations modules of the Rosie agent, called Shared World Knowledge and Action Messages in Figure 2-2. For Rosie to act on messages produced by Lucia, linguistic units such as action verbs and referring expressions need to be represented in terms of the shared knowledge. This *world knowledge* has two major parts: information about the current perceived scene in working memory (WM), and information kept permanently in long-term declarative memory (LTM). The LTM knowledge includes several types: fixed items that define perceptual symbols and concepts, long-term information about the environment such as a map of the building or information about known people and places, and information about objects the agent has seen previously but that are not currently visible.

The language used to refer to items in the world can vary substantially. For example, *the green block* can refer to any object of category *block* that has the color *green*. However, suppose there are three green blocks in the current scene. In that case, a longer phrase such as *the green block on the stove* may be needed to identify a specific block. Therefore, Lucia must incorporate the

flexibility to specify things in the world in a variety of ways. This will be discussed further in Chapters 3, 4, and 5.

### 2.3.1 Perceptual properties

Many lexical items refer to properties of object, such as colors, shapes, and sizes. Information about each of these properties is stored in LTM, with a unique *handle* for each. Handles are arbitrary symbols needed to internally identify uniquely an element of Rosie's world knowledge, independent of language. The strings for some handles contain English words to aid the developers, but these words mean nothing to Lucia or Rosie. Rather, a handle provides a necessary level of indirection between a word and its internal meaning. Lucia's knowledge of language includes handles that are part of the meaning representation of many lexical items. When such a word is grounded, a retrieval is made from LTM using that handle as the cue. Table 2-1 shows some correspondences:

Table 2-1: Handles for perceptual properties

<b>Color Word</b>	<b>Handle</b>	<b>Shape Word</b>	<b>Handle</b>	<b>Size Word</b>	<b>Handle</b>
red	red1	sphere	sphere1	tiny	tiny1
green	green1	triangle	triangle1	small	small1
blue	blue1	rectangle	rectangle1	medium	medium1
orange	orange1	square	square1	large	large1

Internally, these handles are associated in LTM with the agent's knowledge of its perceptual system. For instance, red1 is a handle applied by Rosie's visual system to the color property of an object that is seen as being in the red part of the color spectrum. The form-meaning mapping for the word *red* includes this handle.

### 2.3.2 Objects and relations in the World Model

Rosie has a World Model in working memory that represents the objects that its visual perception system is currently aware of, as well as visible locations and spatial relations between those things. A typical World Model consists of a list of objects and locations, a list of spatial relations, and a special entry for Rosie

itself. A representation of the objects and locations in a world model instance that was used for developing Lucia for the Baseline corpus is given in Table 2-2.

Table 2-2: Objects and locations in a simple World Model

<b>ID</b>	<b>Handle</b>	<b>Predicates</b>
06	large-orange-triangle1	category(block) color(orange1) indicated(pointed) shape(triangle1) size(large1)
07	small-red-triangle1	category(block) color(red1) shape(triangle1) size(small1)
08	medium-green-block1	category(block) color(green1) shape(rectangle1) size(medium1)
09	large-green-block1	category(block) color(green1) shape(rectangle1) size(large1)
010	large-green-sphere1	category(block) color(green1) shape(sphere1) size(large1)
011	small-orange-triangle1	category(block) color(orange1) shape(triangle1) size(small1)
012	medium-purple-triangle1	category(block) color(purple1) shape(triangle1) size(medium1)
013	small-green-box1	category(block) color(green1) shape(box1) size(small1)
014	conference-room-lights1	category(fixture) shape(lights1)
L2	1	category(location) door(closed) name(pantry)
L3	2	category(location) door(closed) heat(off) name(stove)
L4	3	category(location) door(closed) name(garbage)
L5	4	category(location) name(sink)
L6	6	category(location) name(table)
L7	7	category(location) name(waypoint)
L8	8	category(location) name(conference) property(current) spatial-shape(room1)
L9	office1	category(location) name(office) spatial-shape(room1)
L10	building1	category(location) name(building) spatial-shape(building1)
L11	loc-main1	category(location) name(main1) spatial-shape(room1)
L12	loc-soar1	category(location) name(soar1) spatial-shape(room1)
T1	trash1	category(block) shape(trash1)

The IDs are used in Soar as the primary way to reference these items. They are arbitrary and unique, with a single letter followed by one or more digits. The Properties shown here are the ones that Lucia can use to ground referring expressions in the language to these objects in the World Model. The *indicated* property on O6 means that the instructor is currently pointing to this object.

This is used to ground a deictic pronoun such as *this* or *that*. There are a number of additional properties on these objects not shown in the table that indicate things such as spatial position or affordances. These are important to Rosie in its planning of actions, but do not connect to Lucia or language.

Table 2-3: Spatial relations in a simple World Model

Type	Instances	
in1	in1(07, L2)	in1(014, L8)
left-of1	left-of1(08, 09)	
right-of1	right-of1(09, 08)	
on1	on1(08, L3)	on1(06, 07)

Table 2-3 shows the spatial relations in the same example World Model, referencing the objects shown above. For example, it shows that O7, the small red triangle, is in the pantry (L2), that the medium green rectangle (O8), is to the left of the large green rectangle (O9), that O8 is on the stove (L3), and that O6 is on O7.

Given this world model, Lucia can ground referring expressions to the objects in the world. Table 2-4 shows examples of sentences with the referring expressions and the objects in the world that they are grounded to in italics.

Table 2-4: Examples of grounding to the World Model

Pick up <i>the green sphere</i> (010).
Pick up <i>the green block</i> (08) <i>on the stove</i> (L3).
Move <i>the orange triangle</i> (06) <i>on the red triangle</i> (07) <i>to the stove</i> (L3).
The goal is that <i>the box</i> (013) is in <i>the office</i> (L9).
If you see some trash (T1) then throw it (T1) away.

### 2.3.3 Actions, concepts, and the environment

In addition to the perceptual properties, a number of other words and phrases are grounded to items in LTM. The grounding retrieves representations from LTM that provide important information to Rosie. Actions are grounded to their representations in LTM using a handle that is defined in the ECG grammar. For example, for the sentence *Pick up the green sphere*. the *pick up* action is attached by the grammar to the handle pick-up1. The action retrieved from LTM by this handle defines an action procedure labeled internally as op\_pick-up1. Other

details returned with the action are information Rosie needs to perform the action, but those details don't influence language comprehension.

This is a general property of grounding referents in the action messages: the language provides enough information to identify the desired entity, and the representation of that entity has additional information that is needed for the agent to act on. This is part of the difference between comprehension and understanding.

Concepts such as *goal*, *north*, or *meters* are also retrieved from LTM in a similar way. Again, Lucia knows only the handle specified for a word to use as a cue to retrieve the item, and Rosie knows how to use the additional information that is retrieved. Another kind of information in LTM is knowledge about the environment, such as a labeled map of the building, which includes rooms, their location, and sometimes their possible contents. This information is retrieved based on cues for either a name or a handle.

## 2.4 Messages as Sentence Meanings

The input-output relation for Lucia can be thought of as a function that takes two arguments, a new input sentence and the current agent state, and produces an action message. Previous sections have described the input sentences and the agent's world knowledge. This section describes, at a high level, the structure of the action messages that Lucia can send to Rosie.

There is a fixed set of action messages in Rosie. These were developed by James Kirk, Aaron Mininger, and John Laird to be effective in instructing Rosie without any consideration for Lucia. Each message has a type and a list of arguments. The descriptions here are organized by case studies. As shown below, each case study shows an example sentence that produce each type of message, along with the structures produced for that type of message. Each example is labeled by the identifier it has in the corpus listings in Appendix 1, with "B-" sentences from the Baseline corpus, "G-" sentences from the Games corpus, and "R-" sentences from the Robot corpus.

We use an abbreviated notation for messages and other Soar data structures to show only the most important aspects. A message is shown as a type name followed by its arguments in parentheses. Some arguments are themselves complex structures that are abbreviated. Action and object descriptions derived from grounding are shown simply as their handles or internal identifiers, which lead internally to complex data structures.

#### **2.4.1 Simple messages and quoted sentences**

Some sentences are very simple, and those messages need no arguments at all. For example, there are times when the instructor needs to respond to a yes/no question asked by Rosie, or tell Rosie that an instruction is done.

*Case Study 2.1: Simple sentences*

B-043	Yes. yes()
B-060	No. no()
B-103	Done. finished()

*Case Study 2.2: Quoted sentences*

R-087	"Hello Charlie". quoted-sentence( Hello Charlie )
-------	--

In the case of a quoted-sentence message, the argument is the actual string that was quoted in the input. Quoted strings are used by Rosie to represent questions or messages Rosie should say to a person it will interact with.

#### **2.4.2 Declarative sentences**

*Case Study 2.3: Definitions*

Often a sentence will identify or describe some object. An *object-definition* message is used for expressions that refer to some object, usually used as an answer to a question from Rosie. An *adjective-definition* is used to define a new word that denotes a property.

R-123	The pantry. object-definition(pantry1_6)
B-044	Octagon is a shape. adjective-definition(word(octagon), property(shape))

An *object-description* message can describe a property of an object, a relation an object is in, or the definition of a goal state to be achieved.

#### Case Study 2.4: Descriptions of objects

B-001	The sphere is green. object-description(010, predicate(green1))
B-007	The red triangle is on the stove. object-description(07, on1(L3))
R-068	You are done. object-description(rosie, task-completed)
R-072	Alice is in Alice's office. object-description(person-alice1, in1(loc-alice-off1))

#### Case Study 2.5: Goal definitions

B-145	The goal is that the box is in the office. object-description(concept(goal), subclause(013, in1(L9)))
R-009	The only goal is that the fork is on the table. object-description(concept(goal), modifier1(only1), subclause(action(is1), 0530, on1(0118)) )
R-117	The goal is that the fork is in the drawer. object-description(concept(goal), subclause(action(is1), fork1_10, in1(drawer1_4)))
R-015	The only goal is that the soda is in the fridge and the fridge is closed. object-description(concept(goal), modifier1(only1), subclause( subclause(action(is1), soda1_12, in1(fridge1_1)), subclause(action(is1), fridge1_1, not-open1))))
G-006	The goal is that a red block is on a green block and the red block is below an orange block. object-description(concept(goal), subclause( subclause(action(is1), NO-ID1, on1(NO-ID1)), subclause(action(is1), NO-ID1, below1(NO-ID3))))

A subclause is used for a clause that defines a goal. Often a goal is stated as “*The only goal is that ...*” to indicate that Rosie does not have to ask for additional goals. If a sentence defining a goal has a conjunction of two goal conditions, then the subclause of the message points to a structure that in turn

has two subclauses. The messages for the Games and Robot corpora define “is1” as an “action.” This is an anomaly of the message formats used, and was not used in the Baseline corpus.

### **2.4.3 Commands**

Many of the messages used with Rosie have the message type *command*. Each such message specifies an action to be performed at that point in a task, and most also have an object argument to be acted upon. Some cases have a modifier for the action.

#### *Case Study 2.6: Simple commands*

Some commands have just an action with no arguments.

B-082	Go. command(initiate-go1)
B-051	Go forward. command(initiate-go1)
B-036	Stop. command(initiate-stop1)
R-022	Scan. command(op_scan1)

Both B-051 and B-080 produce the same message. Lucia figures out that the word *forward* does not change the meaning in this case.

#### *Case Study 2.7: Communication commands*

In the Robot corpus, some commands tell the robot to say something to a person in the environment.

R-039	Ask "What drink would you like?". command(op_ask1,  What drink would you like? )
R-041	Ask Alice "What drink would you like?". command(op_ask1, agent(person-alice1),  What drink would you like? )
R-034	Say "Hello there!" to Alice. command(op_ask1,  What drink would you like? , to1(person-alice1))
R-086	First, ask "What is the message?". command(op_ask1, modifier(first1),  What is the message? )

Both R-041 and R-034 specify the name of a person to talk to, but the structure of both the sentence and the message is different. In R-086, *first* is treated as a modifier of the action.

### *Case Study 2.8: Commands with objects*

Many commands tell the robot to take some action on a given object. Some of these also specify a target location for the action.

B-009	Pick up the green sphere. command(op_pick-up1, 010)
B-017	Put that in the pantry. command(op_put-down1, 06, in1(L2))
B-020	Pick up the green block on the stove. command(op_pick-up1, 08)
B-028	Pick the green block that is small. command(op_pick-up1, 013)
B-032	Move the green rectangle to the left of the large green rectangle to the pantry. command(move1, 08, to1(L2))
R-001	Find the fork. command(op_find1, NO-ID1(fork1))
R-002	Move the fork onto the table. command(move1, 0530, on1(0118))

B-009 shows a simple case with an object uniquely identified. In B-017 *that* refers to the object the instructor is pointing to. B-020, B-028, and B-032 show different linguistic forms to select a specific object by adding a qualifying expression to a simple noun phrase that is semantically ambiguous. In the messages, only the result of resolving this ambiguity appears. B-032 and R-002 show a more complex action verb that requires a target location, and in B-032 Lucia sorts out the correct interpretations of the two prepositional phrases. For R-001 Lucia creates a new data structure for *the fork* with new-object-id1 as its handle since there is no fork currently in the world model. Here and hereafter the notation NO-IDn is a shorthand for new-object-idn.

### *Case Study 2.9: Commands with multiple objects*

Some commands have multiple objects.

R-084	Tell Charlie a message. command(tell1, person-charlie1, NO-ID34(message))
R-096	Serve Mary a desired drink. command(op_serve157, chef1_16, NO-ID72(drink1, modifier1(desired1)))
R-101	Permanently remember the fridge as the storage location of a juice. command(op_remember1, modifier(permanent1), fridge1_1, NO-ID7(storage1), NO-ID8(juice1))

The representation here for *Charlie* is an entry for a known object in LTM, whereas the representation for *Mary* comes from the world model when Rosie knows that Mary is also the chef in this situation.

#### *Case Study 2.10: Navigation commands*

B-035	Orient north. command(initiate-orient1, cardinal-direction1(north1))
B-040	Turn around. command(initiate-turn1, relative-direction1(around1))
B-052	Turn right. command(initiate-turn1, relative-direction1(right1))
R-026	Turn right twenty-five degrees. command(op_turn1, relative-direction1(right1), number(25), unit(degrees))
B-054	Go to the kitchen. command(op_go-to-location1, to1(L28))
R-008	Go to the starting location. command(op_go-to-location1, to1(NO-ID1(location), modifier1(starting1)))

Most of these command messages are straightforward. R-026 gives quantitative turn angle, which is represented on the message with a number and a unit type. R-008 is more complicated because *the starting location* is not usually something in the immediate vicinity of the robot. Lucia creates a new data structure for a location labeled as new-object-id1 and a modifier starting1 is added. It is up to Rosie to search its memories to find a location that was marked as the starting location for the current task.

#### *Case Study 2.11: Terminated commands*

A number of commands have termination conditions that use clauses starting with *until*. The messages used with the Baseline corpus represented these with an until-clause structure. With the Robot corpus a different message format was used, as in R-045, in order to agree with the format for messages specifying a time period, as in R-159.

B-119	Drive until you sense a wall. command(initiate-go1, until-clause(agent(rosie), action(initiate-sense1), NO-ID3(wall1)))
B-120	Drive down the hall until you reach the end. command(initiate-go1, hall1, until-clause(agent(rosie), action(initiate-sense1), NO-ID10(end1)))

R-045	Explore until you see a stapler. command(explore1, temporal-clause(rosie, op_sense1, until1, stapler1), temporal-predicate(until1))
R-159	Observe Bob for ten minutes. command.observe1, person_29, temporal-clause(number(10), unit(minutes)), temporal-predicate(for1))

### *Case Study 2.12: Enabled commands*

In the Games corpus there are a number of sentences which are command sentences preceded by *You can ...*. The meaning of these within the context of a game or puzzle is that in some future situation the action defined by the command is available.

G-004	You can move a clear block onto a clear object. command(agent(R5), action(move1), action-modifier(can), NO-ID1(clear), on1(NO-ID2(clear)))
-------	--

### **2.4.4 Conditionals**

Many sentences in our corpora are conditional, of the form *If <declarative-condition> then* followed by either a command or declarative clause.

### *Case Study 2.13: Conditional sentences*

B-046	If the green box is large then go forward. conditional(if-subclause(013, predicate(large1)), then-subclause(action(op_go-to-location1)))
G-005	If a location is not below an object then it is clear. conditional( if-subclause(action(is1), modifier(negation), NO-ID8(location) below1(NO-ID9(object))), then-subclause(action(is1), NO-ID8(location), predicate(clear)))
G-024	If the volume of a block is more than the volume of an object then the block is larger than the object. conditional( if-subclause(action(is1), volume-of(NO-ID10(block)), more-than(volume-of(NO-ID11(object)))), then-subclause(action(is1), larger-than(NO-ID10(block), NO-ID11(object))))

G-011	If the number of the locations between a location and a covered location is the number of the blocks that are on the covered location then you can move it onto the former location. <pre>conditional(     if-subclause(action(is1),         number-of(NO-ID9(location, multiple),             between(NO-ID7(location),                 NO-ID8(location, covered))),         number-of(NO-ID10(block, multiple),             on1(NO-ID8(location, covered)))),     then-subclause(agent(R5),         action(move1), action-modifier(can),         NO-ID10(block, multiple), on1(NO-ID7(location))))</pre>
-------	---

In these sentences there are often multiple references to the same object, and these coreferences have the same NO-ID $n$ . If two different sentences have the same NO-ID $n$  number, that is not a coreference but just an artifact of how the data was captured.

#### 2.4.5 Questions

Lucia knows how to process several types of questions that are used in Rosie's ITL scripts, and each type has a separate message type.

##### *Case Study 2.14: Object questions*

B-098	Is this red? object-question(06, color(red1))
B-077	Is this a sphere? object-question(06, shape(sphere1))
B-078	Is the green sphere on the table? object-question(010, on1(L6))

##### *Case Study 2.15: What-is questions*

B-096	What is this? what-is-question(06)
B-062	What is on the red triangle? what-is-question(predicate(on1, 07))

##### *Case Study 2.16: Where-is questions*

B-048	Where is the red triangle? where-is-question(07)
-------	---

##### *Case Study 2.17: Predicate questions*

B-090	What color is the large sphere? predicate-question(06, predicate(color))
-------	---

## 2.5 Evaluation

There are several ways to evaluate a system like Lucia. Some would look only at the external performance as an input-output relation. Other methods would look at the system's internal workings and compare these to appropriate criteria. This chapter is focused on an external view of the input-output relation of Lucia in its Rosie context, so here we only consider this type of evaluation. Other chapters will evaluate Lucia's internal knowledge, processing, and mechanisms.

One way to do an external evaluation is to compare to other systems qualitatively, or quantitatively with respect to a benchmark if possible. Chapter 1 gave a qualitative comparison of several systems to the five comprehension principles we defined there. We have found no relevant published benchmarks other than the one we published for Rosie and Lucia (Lindes et al., 2017).

Since Lucia is embodied in Rosie, another approach is to test Rosie's behavior when using Lucia to do its sentence comprehension. Figure 2-10, which is based on the Figure 2-3, shows this approach, and two possible shortcuts.

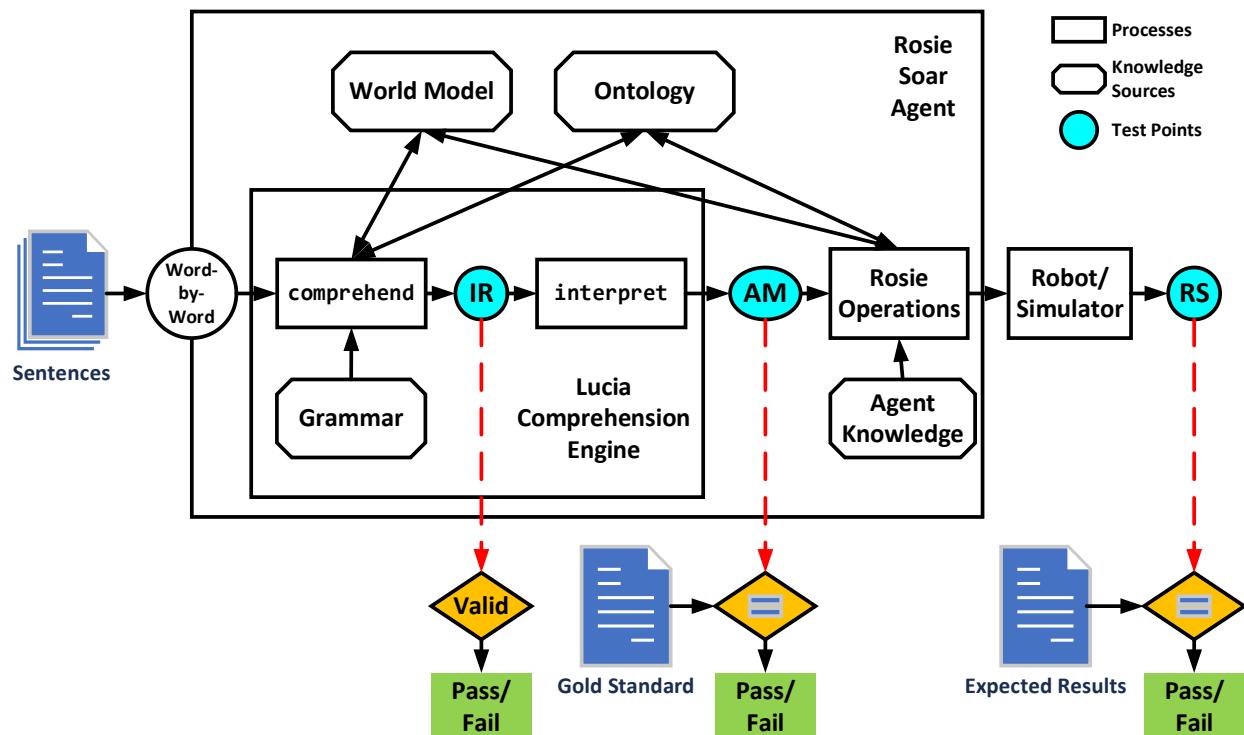


Figure 2-10: Ways to test Lucia's output

At the RS point in this schematic, we have a document that describes the final state of Rosie in the world after processing an entire ITL script, along with the sequence of actions it performed to get there. We also have documentation of *expected results* based on running Rosie with the Laird parser mentioned above. If these two documents are the same, it means Lucia produced a set of messages for that script sufficiently correct so that Rosie's behavior was equivalent to that done with the Laird parser. We have done some testing in this way for Lucia in Rosie.

At the AM point we test Lucia's output for each sentence against a *gold standard* message produced by the Laird parser for the same sentence using the same state of Rosie's world knowledge, giving a more precise measure of the correctness of Lucia's comprehension of each individual sentence. This is the technique used for most of our evaluation of Lucia's embodied, end-to-end comprehension.

At the IR point we can do a validity check on an internal representation of the result which includes syntactic and semantic structure and grounding for a particular sentence. This can verify that the IR for that sentence is a complete and well-formed structure, but does not guarantee that all the semantic analysis and grounding were done correctly. Some of our sentences have been tested only at this level due to lack of sufficient time to develop the logic needed to map these IR structures into the often-idiosyncratic pre-defined message in the gold standard. Our testing at all three of these test points produces only a binary pass/fail result for each sentence, no other quality score is used.

At the RS test point, 9 of the 60 Games scripts have been completed and 2 of the eight Robot scripts have been completed. For all these completed scripts, all the sentences are correct at the IR and AM test points, and the full scripts produce the expected results. The results of testing all the sentences in the three corpora at the IR and AM test point are shown in Table 3-1.

Table 2-5: Results of testing<sup>5</sup>

<b>Corpus</b>	<b>All Sentences</b>	<b>Sentence Forms</b>	<b>Dev Set</b>	<b>Forms Working at IR</b>	<b>Forms Working at AM</b>	<b>Generality Ratio IR/AM</b>
Baseline	207	207	143	207	207	1.45/1.45
Games	1,104	172	48	92	51	1.92/1.06
Robot	228	160	60	115	110	1.92/1.83
Total	1,539	539	251	414	368	1.65/1.47

The generality ratio shown is the ratio of the number of sentence forms working to the number of forms used for development to achieve that. The ratios defined this way are relatively small due to the fact that the sentences were developed to represent unique meanings not encountered before. The large difference between the ratios at IR and at AM is because we decided, due to the fact that the idiosyncrasies of the message formats for Rosie make it very hard to get the sentence interpretation right, to do further development of the grammar without completing the related interpretation. These data show how far we have come to date in covering the Rosie corpora.

---

<sup>5</sup> These data are as of 3 March 2022.

## **Chapter 3 Composable Knowledge of Meaning**

The primary goal of the language comprehension system we are building is to produce an actionable internal message from each input sentence. Each input sentence is a sequence of words that encodes some meaning that the speaker intends to convey. We call this sequence of words the *form* of the sentence, and the resulting internal message the *meaning* of the sentence.

To make this transformation from form to meaning, a comprehension system needs knowledge of how to map form to meaning. Humans do this, and with knowledge that is general enough to be applied to many sentences never heard or seen before. Such generality is important for an artificial agent as well, so that it can respond flexibly and creatively. Achieving this generality of form-meaning mapping requires that the knowledge of how linguistic forms map to their meanings be organized in units at word, phrase, and clause levels that can be composed in many different and novel ways. Therefore, Lucia needs *composable knowledge of the meaning of language* (CKM).

**CKM: Composable Knowledge of the Meaning of Language** – *Knowledge of the meaning of language is in composable units, called constructions, in order to have generality, so that it can be used to comprehend (E3C) many sentences that have never been previously experienced.*

Incremental, immediate interpretation processing (I3P) also requires compositionality so that each element of form, whether a word, a phrase, or a clause, can have its meaning immediately grounded and then combined with the ongoing interpretation of a sentence. It is important that the meaning produced for each sentence be *correct*. For Lucia to succeed in Rosie, it must provide *semantic precision*, the ability to produce a specific, precise, and accurate meaning-in-context for each sentence it processes. This precision must be at every level so that Rosie can understand exactly how it needs to act, internally

or externally, on each sentence it receives. Thus generality, incrementality, and semantic precision all require composable knowledge of meaning.

### **3.1 Defining the Problem**

To achieve a model for transforming the form of each sentence to its meaning, the model must have knowledge of how the elements of language map to elements of meaning, and this knowledge must be compositional. This leads to a computational question:

*What representation of composable knowledge of meaning (CKM) is sufficient for the range of language to be comprehended?*

Form can be mapped to meaning at any level, for words, phrases, or clauses. A representation of knowledge of language as a large inventory of composable elemental pairings of form to meaning enables these elements to be composed in many different ways to provide generality. Constructing meaning piece by piece by composing instances of these composable form-meaning mappings provides a general way to construct the meanings of full sentences, as well as the ability to ground, or interpret, each element of meaning incrementally (I3P).

A possible approach to creating such a system of representation of knowledge of meaning comes from research in cognitive linguistics on an approach called *construction grammar* (CxG; Hoffman & Trousdale, 2013). We have chosen to adopt a specific formalism called *Embodied Construction Grammar* (ECG; Bergen & Chang, 2013) as the foundation for how we represent composable linguistic knowledge in Lucia.

**Commitment 2:** *Lucia uses ECG to represent composable knowledge of the meaning of language (CKM).*

In this chapter, after reviewing related work, we explore the general field of construction grammar theory, and explain the choice of ECG by describing its theory and formalism and how it has been used to represent a grammar that Lucia uses in Rosie. We also describe the process of developing this grammar,

and evaluate it with respect to the requirements for generality, incrementality, and semantic precision from composable knowledge of meaning.

### **3.2 Related Work**

This section briefly reviews linguistic research relevant to this thesis and how it has led to the theory of construction grammar, and ECG in particular.

#### **3.2.1 Generative Linguistics**

Noam Chomsky (1957) began a revolution that launched the modern study of linguistics. Chomsky lays out theoretical principles which are fundamental to this approach. These principles, summarized under the term *generative grammar*, include: a distinction between *competence* and *performance*, that language can be understood in terms of a representation called *deep structure* and rules for *transformation* that produce a *surface structure* that can be communicated and interpreted, and that language is *creative*.

Chomsky (1965, p. 4) defines *competence* as the knowledge an ideal speaker-hearer has of her language, and *performance* as “the actual use of language in concrete situations.” He excludes performance as of interest in his theory, saying: “a generative grammar is not a model for a speaker or a hearer. (p. 9)”

This line of research has produced rich fruit in understanding what Chomsky calls *competence*. However, since we are concerned with modeling a “hearer” and “the actual use of language in concrete situations,” this theory has limited value for our purposes. Our work is clearly focused on *performance*. Nevertheless, this tradition gives us a resource of research on language phenomena, principles of linguistic creativity, and syntactic structure.

#### **3.2.2 Cognitive Linguistics**

Jackendoff (2003) describes how as early as 1963 a group of generative linguists, “pushed very hard on the idea that Deep Structure should directly encode meaning.” This led to a trend separate from mainstream generative linguistics to

seek ways of understanding meaning in terms of how humans represent and reason about concepts.

This movement began to flourish in the 1970's. Roger Schank (1972) described a theory of natural language understanding based on concepts and their relationships. The basic idea that language is built on the human conceptual system is the foundation of the field now called Cognitive Linguistics. Schank and Abelson (1977) describe a theory that humans have "scripts" relating all the meaningful parts of a typical event, such as eating in a restaurant. Charles Fillmore (1976) began the study of an approach called Frame Semantics, which is both more detailed and more flexible than Schank's scripts.

A different thread of research was advanced by Lakoff and Johnson (1980), drawing on work by several others in the 1970's. They argued that abstract concepts are built from metaphorical projections of more concrete physical and social concepts. Johnson (1987) carries this approach further, exploring "ways in which structures of bodily experience work their way up into abstract meanings and patterns of inference." He introduces the concept of an *image schema* as a schematic mental representation of an object or event.

This same idea has been used by a number of others. Mandler & Pagán Cánovas (Mandler & Pagán Cánovas, 2014), for example, describe the relationship between image schemas and the development of concepts in children. Lakoff (1987) takes a different direction by exploring how the mind represents categories, and relates these ideas closely with the idea of image schemas.

This body of work has led to the development of the theory of embodied cognition and its relation to language, and to the concept of embodied comprehension. Mc Nerney (2011) summarizes this idea as "our cognition is influenced, perhaps determined by, our experiences in the physical world." Several papers develop the theme (Gallese & Lakoff, 2005; Johnson & Lakoff, 2002; Lakoff, 1990, 2012). Bergen (2012) summarizes a large body of empirical evidence that perceptual and motor areas of the brain are activated during

language comprehension, which suggests that language comprehension relies on making connections between language and embodiment.

### **3.2.3 Construction Grammar Theory**

Charles Fillmore (1988) presented a new perspective on linguistic structure, which he called “Construction Grammar” (Fillmore et al., 1988). This approach “gives central place to the notion of grammatical construction” and differs “from transformational grammars in not having transformations.” The work of Fillmore and others represents the birth of a radically different approach to analyzing linguistic structure and the relationship between syntax and semantics. The fundamental idea that has emerged from this work is the idea of a *construction* as a concept that maps a certain form to its meaning, and that comprehending a sentence involves composing smaller mappings to get the sentence meaning.

There are many examples of further work in this area since 1988 (Goldberg, 1995, 2003, 2006; Jurafsky, 1992; Kay & Fillmore, 1999; Michaelis, 2006). Hoffman and Trousdale (2013) edited an Oxford Handbook with 27 papers by different authors on different views of construction grammar (CxG) theory. In 2017, AAAI held a Spring Symposium on Computational Construction Grammar and Natural Language Understanding (AAAI, 2017), including a paper about work being done toward this thesis (Lindes & Laird, 2017b). As construction grammar theory in general, and the ECG approach (Bergen & Chang, 2013) in particular, are central to our theory, we describe them in more detail.

### **3.2.4 Goldberg’s Five Principles**

Adele E. Goldberg (2013) laid out five principles which are common to most theories of construction grammar and provide a good approach to defining this field. Here we summarize and paraphrase her definitions of these principles in a way that shows how they apply to Lucia.

1) *Constructions are learned pairings of form and meaning*, at many levels. Lexical items have associated meanings and/or grammatical functions. Similarly, larger structures, such as noun phrases, clauses, and complete

sentences have meaning structures associated with their forms. The meaning of a sentence can be built up by composition of the meanings of its components.

2) Construction grammar approaches are based on *starting with the surface structure*, the sequence of lexical items, and composing their meanings into larger structures. Rather than transformation of deep structure into surface structure as in the generative grammar approach (Adger, 2003), comprehension begins with surface structure and uses the form patterns specified by constructions to compose a syntactic and semantic analysis of a sentence.

3) *Constructions are interrelated in a network.* Constituency relations enable composition, and a hierarchy of types allows some constructions to be very specific while also belonging to a more general class. This type hierarchy enables having semantic precision and syntactic generality at the same time.

4) Goldberg (2013, p. 23) states that “*Constructionist approaches do not rely on innate universal principles.*” Instead, there is wide variation in constructions over the close to 7,000 languages in the world, which are different from each other in many important ways.

5) Knowledge of language is *usage-based*. Goldberg (2013, p. 27) says: “Particular languages are learned by generalizing over utterances that a learner has heard used” and comprehension involves “combining … basic form-function correspondences.” She goes on to say (p. 28) that “Creativity stems from generalizing instances to form more abstract constructions.” These ideas are similar to the process used to develop Lucia’s grammar, and to ideas on language acquisition we discuss in Chapter 6.

### **3.2.5 ECG in Context**

In the early 2000’s, a research group at UC Berkeley, under Professors Jerome Feldman of Computer Science and George Lakoff of Linguistics, developed a constructionist theory of grammar called *Embodied Construction Grammar* (ECG; Bergen & Chang, 2013; J. Feldman et al., 2009). Several PhD dissertations came out of this work (Bryant, 2008; Chang, 2008; Dodge, 2010; Mok, 2009). This theory provides a formal language for defining a constructionist grammar that

includes both lexical and composite constructions, both constituent and type hierarchies, and a model for representing complex meaning structures.

The ECG theory is a fruit of the research on cognitive linguistics and construction grammar since the 1970's. Its approach to usage-based syntax is based on construction grammar theory (Fillmore, 1988; Goldberg, 1995, 2006). Its theory of how to represent meaning is based on the work in cognitive linguistics on scripts, frames, and image-schemas (Fillmore, 1976; Fillmore & Baker, 2009; Johnson, 1987; Lakoff, 1987; Lakoff & Johnson, 1980; Schank, 1972; Schank & Abelson, 1977). This theory, and a well-developed formalism, make ECG a tool that has worked well in the development of Lucia.

The research on ECG at UC Berkeley did not include consideration of cognitive processing as related to our principles of I3P and GCM. Neither did it make a connection to real robots to achieve E3C. Several years after the original development of ECG, additional work was done to connect the ECG parser (Bryant, 2008) to a robot (Eppe, Trott, & Feldman, 2016; Eppe, Trott, Raghuram, et al., 2016). These additions are a step toward E3C, but they are not built into the language processing nor related to models of human cognitive processing.

When considering how the constructionist approach used in this thesis could be compared to a possible generative approach, our claims are very limited. This thesis shows a detailed implementation of how ECG in particular can be used to perform E3C in a way that also does I3P using GCM. This is an existence proof. We make no claims about whether or not similar end results could be achieved using a generative approach, though we have not found any system that does so. When and if such a system is built, only then will it be possible to compare the two approaches.

### **3.3 Embodied Construction Grammar in Lucia**

Lucia uses a pre-existing formal language for Embodied Construction Grammar (ECG; Bryant, 2008) to represent its knowledge of grammar. This is a declarative language that provides a way of describing mappings from form to meaning. A grammar is represented as a large network of *items* called *constructions* and

*schemas*. Each construction describes a pattern of input form and how it maps to a corresponding meaning, and the meanings are represented by schemas. Both constructions and schemas are organized in type hierarchies that make possible both semantic precision and syntactic and semantic generality.

### 3.3.1 An Example Sentence

To give a reference point for discussing the details of ECG, Figure 3-1 shows a simplified diagram of the data structures resulting from processing a simple sentence. At the bottom is the input sentence: *Pick up the green sphere*. At the top is the action message created at the end of comprehension, in this case a *command* with two arguments: an internal identifier for the *pick up* action, and an internal identifier for the object referred to by *the green sphere*. Green arrows represent grounding links to the agent's knowledge on the periphery.

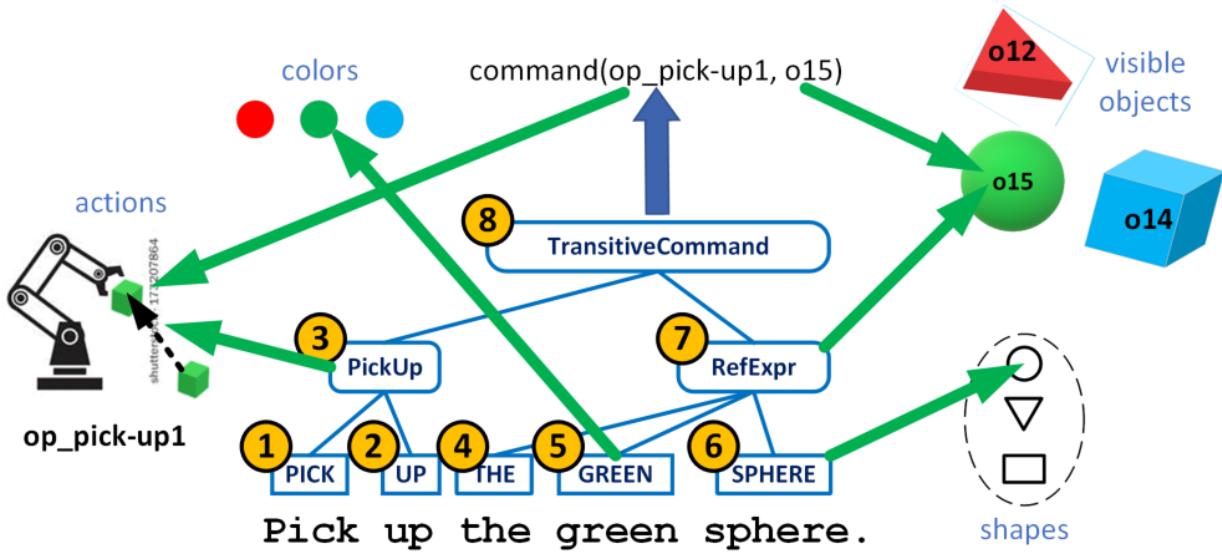


Figure 3-1: Results of processing a simple sentence

In the middle of the figure is a tree structure representing the *comprehension state* at the end of comprehending this sentence, with each blue box representing an instance of a construction. The comprehension algorithm builds these instances one at a time, in the order in which they are numbered.

### 3.3.2 Types of Constructions

There are three types of constructions: *lexical*, *composite*, and *general*. All types have both a form part and a meaning part. The form part of a lexical construction matches a single word or a fixed sequence of words. A composite construction matches a sequence of other constructions that are constituents of a larger linguistic unit. A general construction does not specify its own form pattern, but can be referenced by other constructions as a generalization. The complete network of constructions forms two overlapping hierarchies: a compositional hierarchy defined by the constituents of composite constructions, and a type hierarchy formed by constructions labeled as subcases of general constructions.

### 3.3.3 An ECG Example

In Figure 3-1, there are eight construction instances: five lexical constructions for the five words in the sentence (rectangles), and three composite constructions (rounded rectangles). Each of these instance nodes in the tree is a complex data structure that includes the name of the construction and its generalizations, connections to its constituents for composite constructions, and its meaning schema as populated with the appropriate semantics. To get some idea of the details of the formalism, Figure 3-2 shows the items involved in building the `TransitiveCommand` node. On the left is the full description of the `TransitiveCommand` construction, and on the right are the meaning schemas that it evokes and populates.

<pre>construction TransitiveCommand   subcase of Imperative   constructional     constituents       verb: ActionVerb       object: RefExpr   meaning: ActOnIt   constraints     self.m.action &lt;--&gt; verb.m     self.m.object &lt;--&gt; object.m</pre>	<pre>schema Action   roles     action     direction     location  schema ActOnIt   subcase of Action   roles     object</pre>
---	---

Figure 3-2: An example of ECG items

`TransitiveCommand`, a subcase of `Imperative`, is a composite construction which has two constituents: an `ActionVerb` and a `RefExpr` (short for Referring Expression). It also evokes an instance of `ActOnIt`, a subcase of `Action`, as its meaning. The constraints shown are used to populate the roles of the meaning structure from the meanings of the constituents, thus composing meanings into larger meanings. In Figure 3-1 this construction is instantiated as node 8, with nodes 3 and 7 as its constituents. Not shown in the figures is that `PickUp` is a subcase of `ActionVerb`, to match the verb constituent of `TransitiveCommand`.

### **3.3.4 Hierarchies and Recursion**

Constructions in ECG exist in a network that includes a composition hierarchy and a type hierarchy. In Figure 3-2 we see examples of both kinds of connections. The `TransitiveCommand` construction has two typed constituents for downward links in the composition hierarchy, and it is a subcase of `Imperative`, giving an upward link in the type hierarchy.

Recursion has been proposed as a fundamental principle of human language. Hauser, Chomsky, and Fitch (2002, p. 1569), for example, claim that “the faculty of language ... includes ... the computational mechanisms for recursion, providing the capacity to generate an infinite range of expressions from a finite set of elements.” They even hypothesize that recursion “is the only uniquely human component of the faculty of language.” The ECG formalism includes recursion, and Lucia’s ECG grammar for Rosie uses it. For example, a referring expression can be made up of a noun phrase and a prepositional phrase or relative clause, with these modifiers including another noun phrase or referring expression. We discuss and show examples of recursion below.

### **3.3.5 The ECG Formalism**

Bryant (2008, p. 43) defined a formal language for writing declarative grammars in ECG. We use his exact formal BNF definition of this language, simply changing the format to conform to the syntax used by the ANTLR tool (Parr, 2012) used to build a program to translate ECG grammar into knowledge in Soar. Our version of the formal definition of ECG is shown in Figure 3-3.

```

//      An ANTLR grammar for ECG, adapted from Bryant (2008)
//

grammar ECGL;
@header {package edu.umich.plprelim.ecgantlr;}

ecgl          : (schema | cxn)+ ;

cxnKind       : ('abstract' | 'general')? 'construction' ;
cxn          : cxnKind IDENT parentL? cBlock? fBlock? mBlock? ;
cBlock        : 'constructional' blockType constituentsL? constraintL? ;
fBlock        : 'form' blockType constraintL? ;
mBlock        : 'meaning' blockType evokedL? rolesL? constraintL? ;

schemaKinds   : ('feature' | 'semantic')? 'schema' ;
schema        : schemaKinds IDENT parentL? evokedL? rolesL? constraintL? ;

role          : IDENT (':' typeSpec)? ;
rolesL        : rolesL role | 'roles' ;

evokedElement : 'evokes' typeSpec 'as' IDENT ;
evokedL       : evokedElement evokedL? ;

constitL       : constituentsL constit | 'constituents' ;
constit        : 'optional' IDENT ':' IDENT constituentAnno?
               | 'extraposed' IDENT ':' IDENT constituentAnno?
               | IDENT ':' IDENT constituentAnno? ;
constitAnno    : '[' 'probL ']' ;
probL          : PROB | PROB ',' PROB ;

constraintL    : constraintL constraint | 'constraints' ;
constraint     : 'ignore'? (var chainOperator var | var '<-->' identOrString) ;
chainOperator  : '<-->' | 'before' | 'meets' ;
var            : SLOTCHAIN | IDENT ;
identOrString  : EXTERNALTYPE | IDENT | STRING ;

subcaseOf      : 'subcase' 'of'? ;
parentL        : parentL ',' IDENT | subcaseOf IDENT ;
blockType      : (':' typeSpec)? ;
typeSpec        : IDENT | EXTERNALTYPE ;

IDENT          : [A-Za-z] [A-Za-z0-9_\-]* ;
SLOTCHAIN      : (IDENT '.')+ IDENT ;
EXTERNALTYPE   : '@' [0-9a-zA-Z.\-_]+ ;
STRING          : '\'' .*? '\'' ;
PROB            : '.' [0-9]+ | '1.0' | '1' ;
COMMENT         : ('#' | '//' ) .*? ('\r' | '\n')+ -> skip ;
WS              : [ \t\r\n]+ -> skip ;

```

Figure 3-3: A formal definition of the ECG language

### 3.4 Lucia's ECG Grammar for Rosie

Lucia processes input sentences using a large network of ECG items which constitute what we call its *grammar*. The grammar consists of a number of files containing definitions of constructions and schemas according to the ECG formalism. This is the compositional knowledge of meaning (CKM) Lucia uses, and it is automatically translated by a custom-built compiler into Soar's long-term memories for processing.

The items described in the grammar are *abstract* in the sense that they are like templates that can be applied to many situations. Processing instantiates them to form concrete nodes in the comprehension state. The items in Figure 3-2 are abstract items, while the nodes in Figure 3-1 are concrete instantiations.

The entire Lucia ECG grammar for Rosie consists of 517 constructions and 175 schemas, a total of 692 items. In this section we examine parts of this grammar in both abstract form and with concrete examples. It is impossible here to cover all the details, and Appendix 2 provides more detail of the complete grammar.

#### 3.4.1 Lexical Items

The most specific constructions are ones that define lexical items directly. To illustrate we consider the same sentence shown in Figure 3-1.

*Case Study 3.1: A transitive command*

B-009	Pick up the green sphere. command(op_pick-up1, 010)
-------	--

Figure 3-4 shows all the lexical constructions used for this sentence, along with the general constructions they are subcases of. As a convention, we name lexical constructions in all capital letters. Every lexical construction has a form constraint that sets its orth slot to the string constant that is the spelling, or orthography, of that item. The orth property may include multiple words, in which case that construction will recognize that sequence as a single lexical item, for example for a preposition such as *to the left of*.

Most lexical items also name meaning schemas to be evoked and/or have meaning constraints that set the values of the slots, or *roles*, of those schemas. For instance, THE will evoke a RefDesc schema (short for Referent Descriptor) through its NPSpecifier generalization, GREEN evokes a PropertyDescriptor directly, and SPHERE-noun evokes a specific Sphere schema. The constraints under a meaning clause set values of the schema's roles, as where THE sets its givenness to "definite" and GREEN sets the name of the color to "green1."

<pre> general construction ActionVerb construction PICK   subcase of ActionVerb   form     constraints       self.f.orth &lt;-- "pick"  construction UP   form     constraints       self.f.orth &lt;-- "up"  general construction NPSpecifier   meaning: RefDesc  general construction Determiner   subcase of NPSpecifier  construction THE   subcase of Determiner   form     constraints       self.f.orth &lt;-- "the"   meaning     constraints       self.m.givenness &lt;-- "definite" </pre>	<pre> general construction Property construction GREEN   subcase of Property   form     constraints       self.f.orth &lt;-- "green"   meaning: PropertyDescriptor     constraints       self.m.class &lt;-- @color       self.m.name &lt;-- "green1"  general construction Noun general construction CommonNoun   subcase of Noun  construction SPHERE-noun   subcase of CommonNoun   form     constraints       self.f.orth &lt;-- "sphere"   meaning: Sphere </pre>
---	--

Figure 3-4: Lexical constructions for CS-3.1

### 3.4.2 Referring Expressions

A major part of Lucia's grammar for Rosie is used to comprehend referring expressions. These can be proper names, pronouns, or noun phrases with or without determiners and adjectives. They can also be modified by prepositional phrases or relative clauses.

To continue with Case Study 3.1:, Figure 3-5 shows the meaning structures built for *the green sphere*, including the name of the construction that evoked each one and the data structures they get grounded to in black rectangles. The word *green* is grounded to Rosie's Ontology, giving L28, which is Rosie's internal representation of its perceptual symbol for the color green. When the RefDesc for *the green sphere* is grounded, this color information, along with the shape defined by *sphere*, is used to find an object in Rosie's World Model that has those properties, in this case O10.

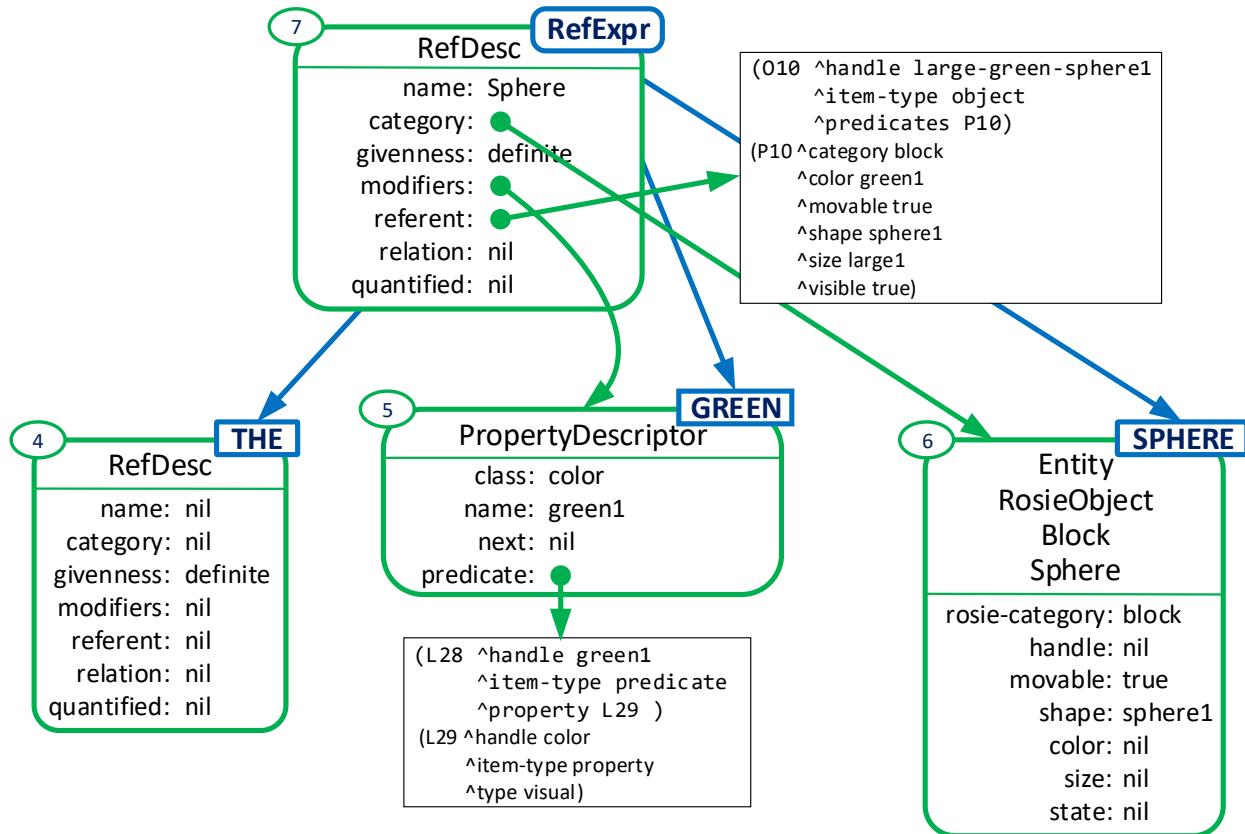


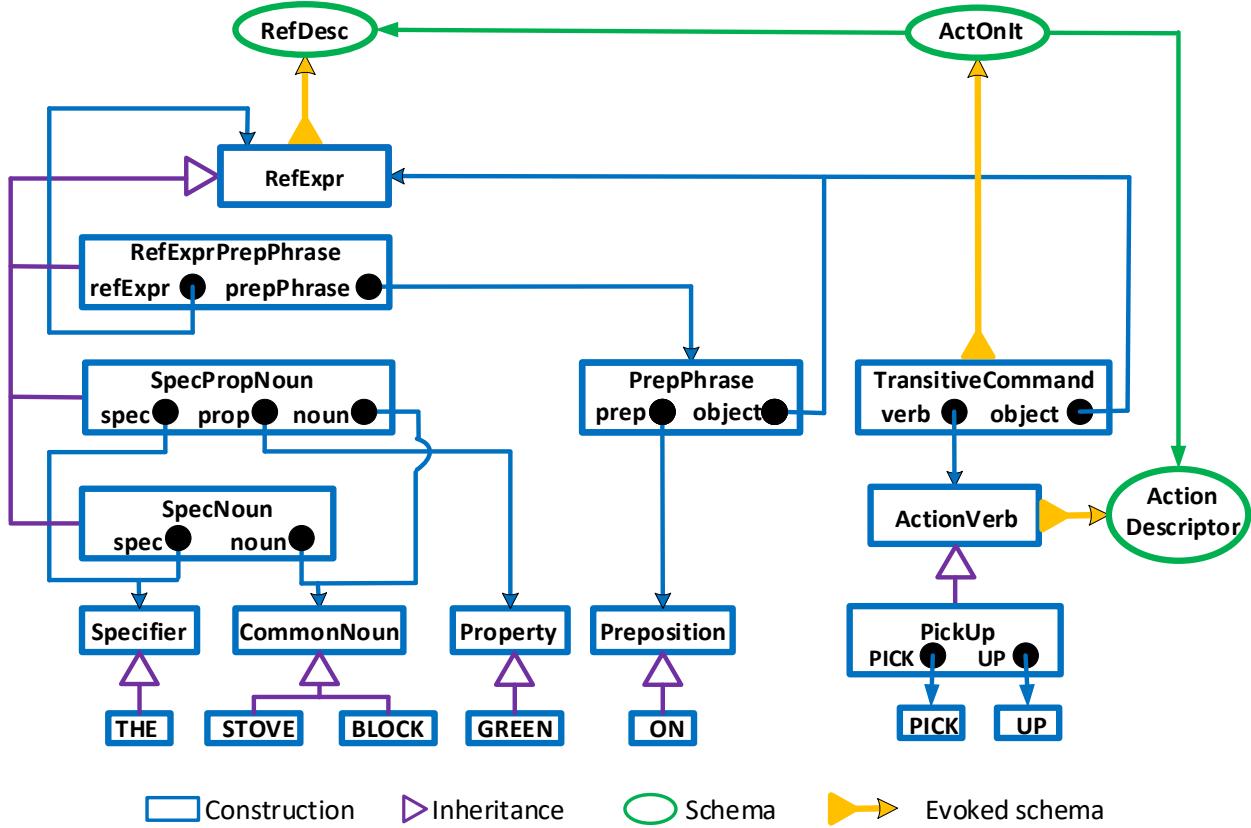
Figure 3-5: Meaning structures for CS-3.1

All the variations of referring expressions require a substantial network of constructions to process them. Below is a case study to illustrate this network.

#### *Case Study 3.2: A complex referring expression*

B-020	Pick up the green block on the stove. command(op_pick-up1, 08)
-------	---

The object of the command here has a noun phrase modified by a prepositional phrase. This is needed since there are several green blocks in the scene, but only one of them is on the stove. Figure 3-6 shows a graphical representation of the network of constructions needed to comprehend this sentence, along with the main meaning schemas evoked.



**Pick up the green block on the stove.**

Figure 3-6: The network of constructions for a complex referring expression

This diagram shows the two types of construction hierarchies. The blue arrows coming from constituent points, which point downward, represent top-down references in the compositional hierarchy. The purple arrows with open triangles as their points show the *subcase of* relations which make up the type hierarchy. This diagram shows that the definition of **RefExpr** is recursive since **RefExprPrepPhrase** has **RefExpr** as both a parent and a constituent, and again as a constituent at the second level through **PrepPhrase**. In Appendix 2 there is a list of all the types of referring expressions and how they are composed.

### 3.4.3 Imperative Sentences

A large proportion of the sentences used with Rosie are imperative sentences, or sentences that give Rosie a command to do something. Consider a simple case.

*Case Study 3.3: A command to go to a known location*

B-054	Go to the kitchen. command(op_go-to-location1, to1(L28))
-------	---

This case, while being simple, shows a number of important properties of the ECG grammar we have developed for Rosie. For this particular case, Figure 3-7 shows the construction instances that are built with their full parent hierarchies.

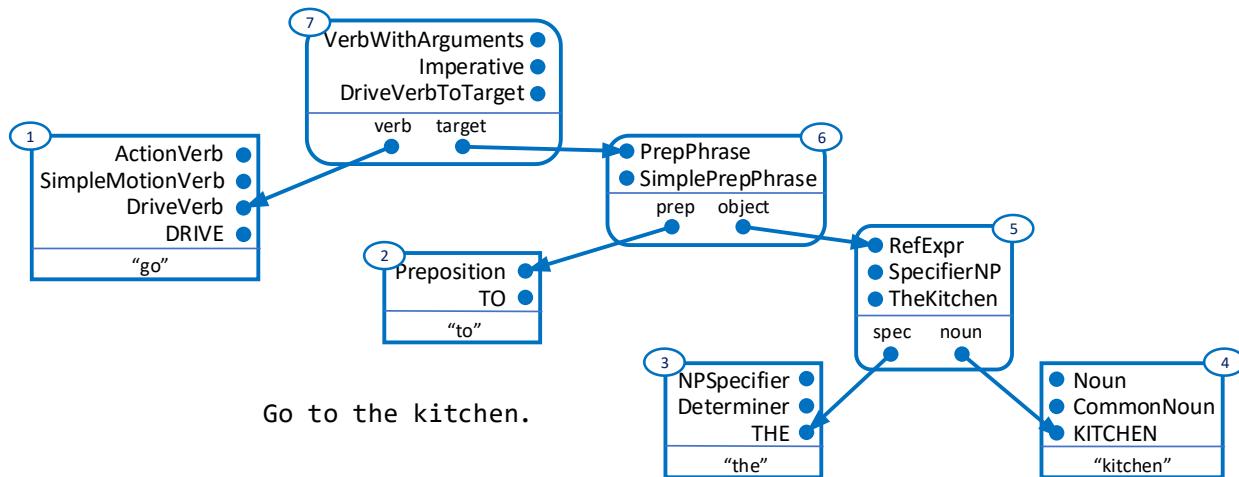


Figure 3-7: Details of syntax with semantic precision

This figure shows how the type hierarchy comes into play to provide semantic precision. Within each construction block, the part below the line shows the composition and above the line is a list of all the types for this construction, with the most specific at the bottom and the most general at the top. Each type name can be a target for composition. Each composition arrow points to the dot corresponding to the type that fits that constituent slot in the parent construction, which can be at any level in the type hierarchy. It turns out that the phrase *the kitchen* represents something that in Rosie is called a *known object*, meaning that it should be grounded to Rosie's map of the building rather than to the World Model. By defining a construction called TheKitchen as a

specific subcase of SpecifierNP, it can have a precise semantic representation that will only match this object in the map.

#### *Case Study 3.4: A command to move an object*

B-022	Put it on the stove. command(op_put-down1, 06, on1(050))
-------	---

Imperative sentences can be recursive. Figure 3-8 shows the network of constructions needed for this case study along with their schemas.

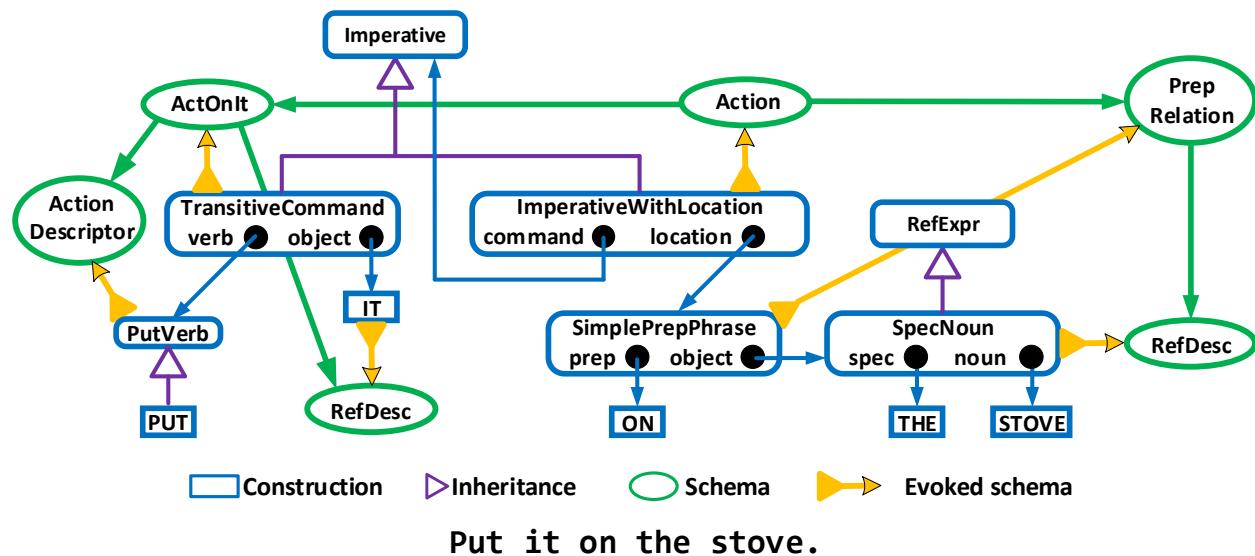


Figure 3-8: Construction network for a simple command

ImperativeWithLocation is the composite construction that spans the whole sentence. It is a subcase of the general Imperative, and also has Imperative as its first constituent, and is thus recursive. For this case, that constituent will be a TransitiveCommand, another Imperative, that spans *Put it*. All together we get an action with both an object to act on and a target location.

#### **3.4.4 Declarative Sentences**

The grammar for declarative sentences is complicated because it includes both conjunctions and recursion. A specific case study below shows some of the complexities.

*Case Study 3.5: A complex declarative sentence*

G-010	The goal is that a red block is on a green block and the red block is below an orange block. object-description(concept(goal), subclause(subclause(action(is1), NO-ID1, on1(NO-ID2)), subclause(action(is1), NO-ID1, below1(NO-ID3))))
-------	--

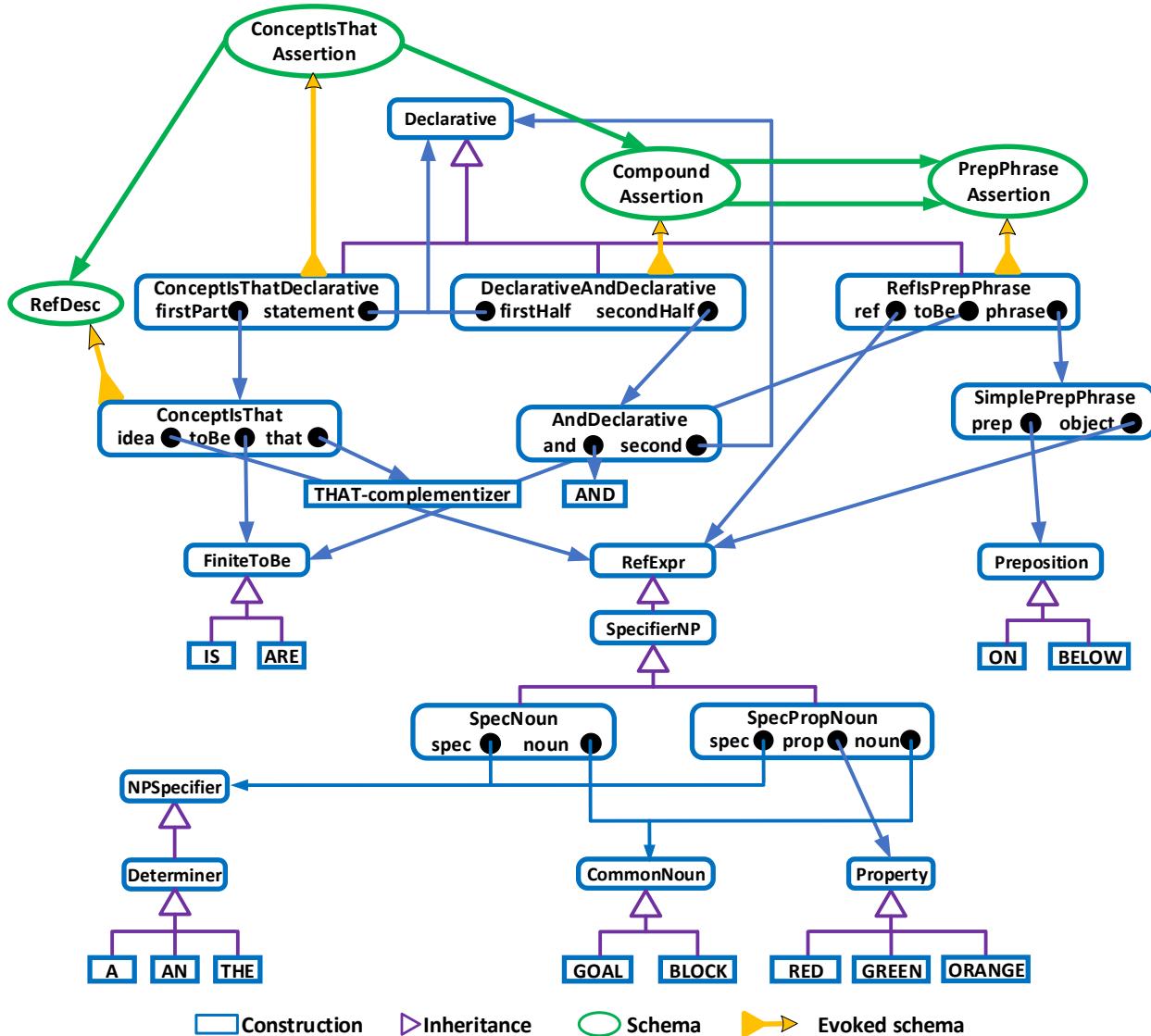
Here NO-ID1 is a new red block object, NO-ID2 is a new green block, and NO-ID3 is a new orange block. (The notation NO-ID $n$  is shorthand for new-object-id $n$ , which is used for indeterminate objects such as these.) The syntactic and semantic structures built from ECG for this sentence look like the following in an abbreviated text form. This example shows the high-level syntactic structure first, followed by the high-level semantic structure after “m:”.

```

ConceptIsThatDeclarative[
  ConceptIsThat[SpecNoun, IS, THAT-complementizer],
  DeclarativeAndDeclarative[
    RefIsPrepPhrase[SpecPropNoun, IS, SimplePrepPhrase],
    AndDeclarative[AND,
      RefIsPrepPhrase[SpecPropNoun, IS, SimplePrepPhrase]]]
]
m:
ConceptIsThatAssertion[RefDesc[concept(goal)],
  CompoundAssertion[
    PrepPhraseAssertion[RefDesc[NO-ID1], PrepRelation[on1, RefDesc[NO-ID2]],
    PrepPhraseAssertion[RefDesc[NO-ID1], PrepRelation[below1, RefDesc[NO-ID3]]],
  ]
]

```

Figure 3-9 shows the subset of the ECG grammar network needed to comprehend this sentence. Here again we see recursion. Both ConceptIsThatDeclarative and DeclarativeAndDeclarative are subcases of the general Declarative, and both also have Declarative as a constituent. A second level recursion goes from DeclarativeAndDeclarative through AndDeclarative and back to Declarative. This example also shows one possible way of handling conjunctions. The AndDeclarative construction looks for *and* followed by some Declarative, and is in turn a constituent of DeclarativeAndDeclarative, giving the syntactic structure shown above. The figure shows the abstract constructions and schemas, while the above text structure shows their instances.



The goal is that a red block is on a green block  
and the red block is below an orange block.

Figure 3-9: Construction network for a complex declarative

### 3.4.5 Questions and Conditionals

In addition to Imperative and Declarative sentences, Lucia's ECG grammar for Rosie handles a number of kinds of questions and conditional sentences. Figure 3-10 shows the top-level grammar network for all the question types along with examples of each.

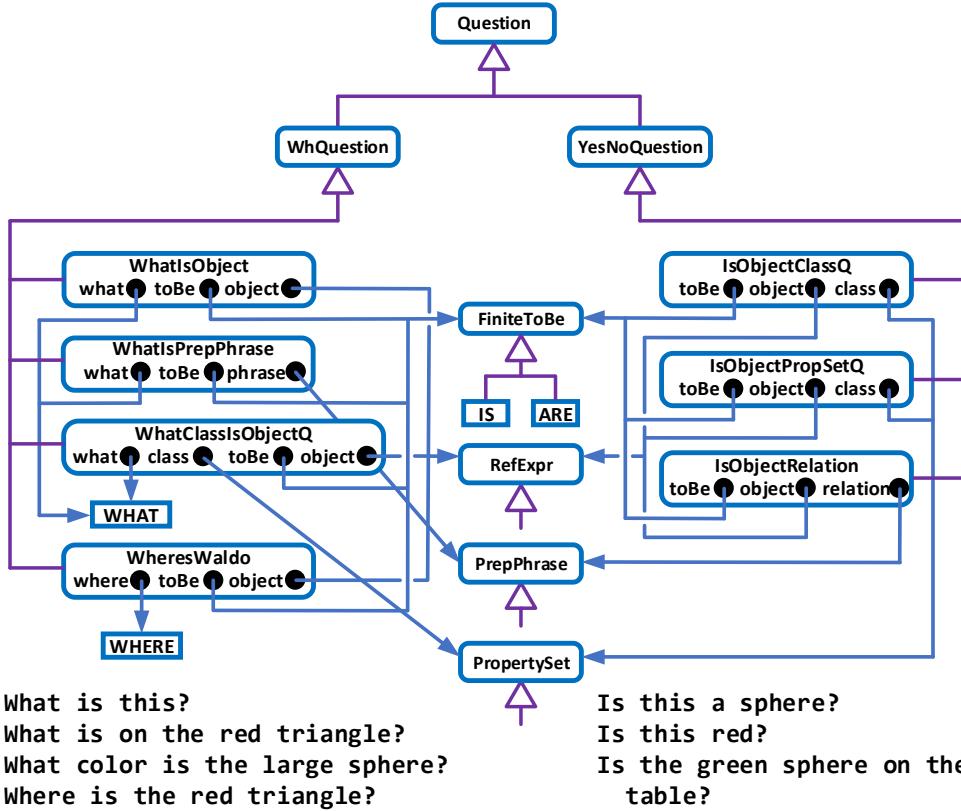


Figure 3-10: Construction network for questions

Traditional generative grammar approaches to analyzing questions in English depend on a transformational operation called *Move* (Adger, 2003, pp. 132, 236–237, 341–375). The constructionist approach, however, is different, simply defining constructions that recognize the surface structure used in a particular class of sentences, as the figure shows. The advantage of this approach is that it is usage-based, to that constructions can be learned from examples of their usage.

A number of sentences in the Rosie corpora have an *if-condition-then-action* or *if-condition-then-statement* structure. Figure 3-11 shows the top-level grammar used for these sentences, along with an example sentence for each of the three main types. IfConditionThenCommand puts a condition on an Imperative sentence, while IfConditionThenStatement does the same for a Declarative. The IfConditionThen construction is an auxiliary that builds up the *if-condition-then* part of these two sentence types. IfConditionCommand is an

intermediate form where the word *then* has been elided or replaced with a comma. The meaning schemas, not shown, simply compose the meanings of the constituents. Since Imperative and Declarative are recursive structures, conditional sentences can be long and complex.

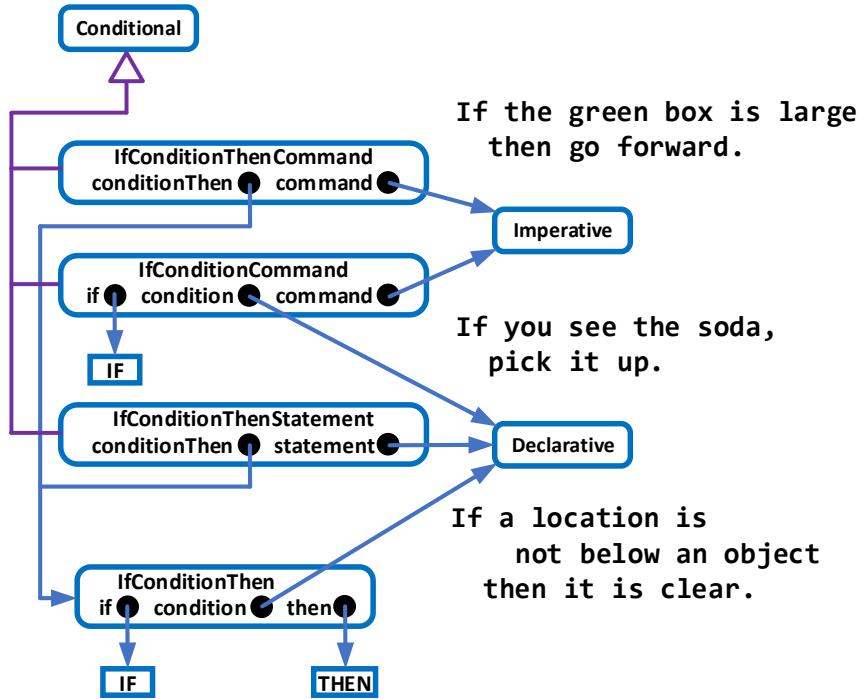


Figure 3-11: Construction network for conditional sentences

### 3.5 The Grammar Development Process

The ECG grammar used in this project was developed incrementally, in a usage-based manner, by manually designing each new construction or schema when it was needed to process a new sentence. A development set of sentences has grown, one new sentence at a time, over the course of the project. Lucia currently has no mechanism for the agent to learn new language knowledge automatically, but the manual development process is consistent with both the usage-based constructionist approach and with our theory of how language acquisition might work in humans, discussed in Chapter 6. Here we show how the accumulation of ECG items progressed as the development process proceeded for the three sentence corpora used to develop Lucia's grammar for Rosie. Importantly, the

Rosie corpora are not “natural” sentences, but were developed deliberately to stretch the system with a high density of new syntactic and semantic structures.

The grammar for each corpus was developed fairly independently from the others, except that the Baseline corpus was developed first and its grammar served as a baseline for the Games and Robot corpora. The development process consisted of repeating three steps: 1) add a sentence that does not work to the development set, 2) add new ECG items and related code to make that sentence work, and 3) identify other sentences that “just work” with no further development. Table 3-1 summarizes the statistics for the three corpora.

Table 3-1: Corpus development statistics

<b>Corpus</b>	<b>Forms</b>	<b>Dev Set</b>	<b>ECG Items</b>	<b>Just Work</b>	<b>Working</b>
Baseline	207	143	461	64	207
Games	172	48	127	44	92
Robot	160	60	104	50	110
Total	539	251	692	158	409

We learned much from developing the Baseline corpus. Figure 3-12 shows the growth of ECG items for the development of the grammar to cover this corpus. Constructions are categorized as lexical (L), composite (C), or general (G).

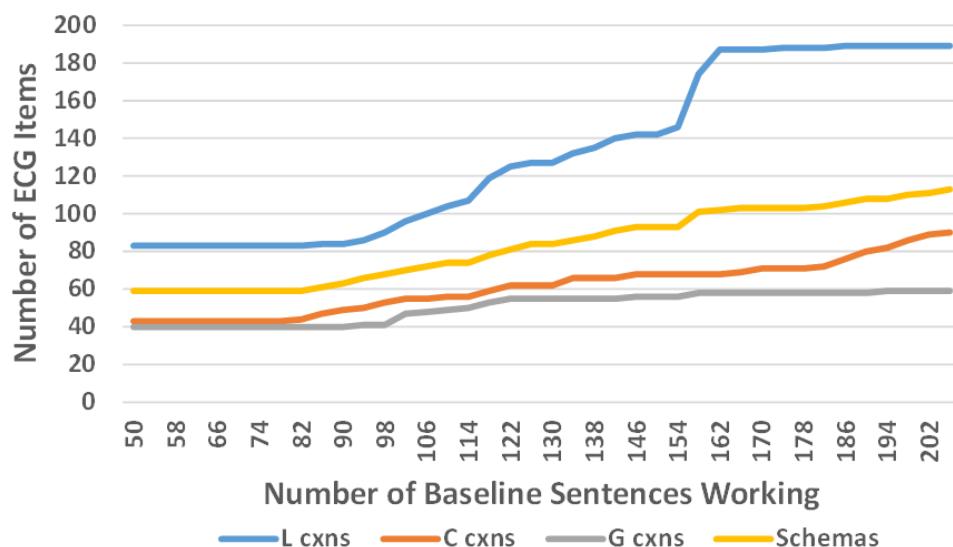


Figure 3-12: Growth of ECG items for the Baseline corpus

Historically, we began collecting this data after 50 sentences were working, and this graph shows the growth through 206 sentences. Data was accumulated in groups of four sentences. The lines are flat between 50 and 82 sentences because once the first 50 sentences were developed and the system was run on the whole corpus, 82 sentences “just worked” without more development. Before 154, lexical constructions were added as needed by specific sentences. There is a big jump in lexicals between 154 and 162 because new vocabulary was added without making any other changes needed to get more sentences working. Details of 200 of these sentences, along with the Rosie data needed to ground them and the messages they produce, are given in Lindes et al. (2017) and its supplemental material.

Beyond these historical artifacts, this graph shows more principled information as well. Lexical constructions grow a lot faster than composites since many different words can fit into the same composite syntactic construction. General constructions grow relatively slowly since often many specific constructions in the same general category are needed to achieve the semantic precision required to do E3C. The need for semantic precision also explains why there is an ongoing need to add more composite constructions. The total number of constructions is about three times the number of schemas, since many schemas are shared by several constructions, and other constructions, such as those for some function words, refer to no schemas at all.

Figure 3-13 and Figure 3-14 show graphs similar to Figure 3-12 for the Games and Robot corpora respectively, where new ECG items were added later to the grammar developed for the Baseline. Note that these two figures show only the ECG items added on top of the Baseline grammar for either the Games or the Robot corpus, for all of the sentence forms shown. A *sentence form* is a sentence with a unique sequence of words. In these corpora many sentences are repeated; for the Games corpus, for instance, there are 1,104 total sentences but only 172 unique sentence forms. Three of these already worked before starting on the Games corpus, and ten already worked for the Robot corpus. These figures show somewhat different trends for the three corpora.

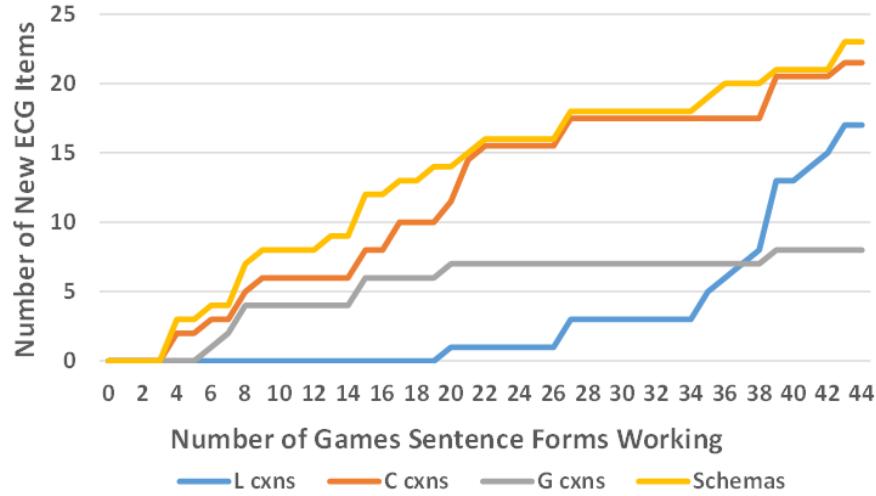


Figure 3-13: Growth of ECG items for the Games corpus

The Games graph shows the need for a lot of new composite constructions early on. This shows that in the first few Games scripts that were considered, the necessary vocabulary already existed in the Baseline, but many new composite constructions were needed to handle new syntactic forms. These new composite constructions also required new schemas to represent their semantics, and often new general constructions were needed as well. Once these items were established, lexical items started contributing most of the growth. More Games sentence forms than these are now working, but we abandoned maintaining these detailed statistics after 44 working sentences.

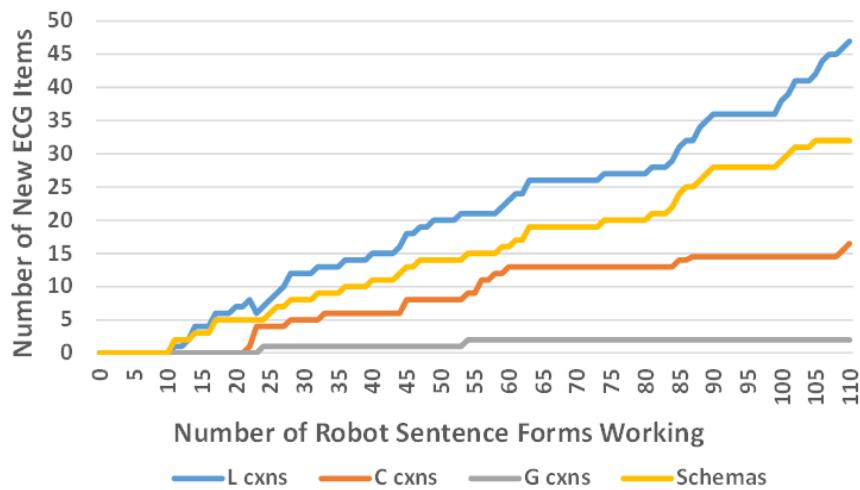


Figure 3-14: Growth of ECG items for the Robot corpus

The Robot graph shows a different pattern. Many new lexical items are introduced, while early on the syntactic forms from the Baseline covered the necessary composite and general constructions. Later more new syntax was introduced, requiring more composites, but new general constructions were needed only rarely. This shows that many of the syntactic structures used in the Robot corpus were already in the Baseline.

It is difficult to draw too many general conclusions from this data since much of the relative variation between the different curves is due to the arbitrary order in which new sentences were chosen for development. However, for the Games and Robot corpora, new sentence forms were chosen for development based on the order in which they occurred in the scripts for various tasks. Thus, the order of new knowledge needed may roughly correspond to what is needed as the domain of semantic and syntactic coverage grows in a task-related way.

### 3.6 Evaluation

In this chapter, our evaluation is mostly qualitative and focuses on ECG's value for representing composable knowledge of meaning (CKM) and Lucia's ECG grammar for Rosie. Here again is our definition of CKM:

**CKM: Composable Knowledge of the Meaning of Language** – *Knowledge of the meaning of language is in composable units, called constructions, in order to have generality, so that it can be used to comprehend (E3C) many sentences that have never been previously experienced.*

This states two essential attributes of our representation of knowledge of meaning: that it must be *composable* and have *generality*. The problem definition given above expands on this with a computational question:

*What representation of composable knowledge of meaning (CKM) is sufficient for the range of language to be comprehended?*

The point of this question is that we must evaluate the *sufficiency* of our knowledge representation by its *language coverage*. We have chosen as a proposed answer to this question a commitment to an approach:

**Commitment 2:** *Lucia uses ECG to represent composable knowledge of meaning (CKM).*

To evaluate Lucia and its ECG grammar for Rosie against these criteria we address three issues in this section: *composability*, *coverage*, and *generality*. This evaluation builds on the prior work on ECG (Bryant, 2008; Dodge, 2010; J. Feldman et al., 2009) by applying it to Lucia’s specific grammar for Rosie.

### **3.6.1 Composability**

Our evaluation of composability is mostly by inspection. The case studies we have presented show cases where the same word, phrase, or clause structure can be composed in different contexts to form larger structures. Appendix 2 expands on this by providing an inventory of all the constructions and how they can be composed. Appendix 3 gives more case studies to show how the composability of semantic structures is used to represent meaning. This is evidence that Lucia’s ECG grammar for Rosie is indeed composable.

### **3.6.2 Coverage**

In Chapter 2 we have shown that Lucia’s ECG grammar does indeed cover the language of the majority of sentences needed for Rosie’s ITL tasks. Other questions arise. What is the scope of language covered? What limits are there to the ECG formalism for representing the necessary knowledge? Here we examine how Lucia and its ECG grammar measure up on these dimensions.

#### *Language phenomena covered*

We list language phenomena that were chosen for the Rosie corpora, by other developers independent of Lucia, where there is demonstrated coverage by the Lucia grammar. Key phenomena covered include:

- *Complex referring expressions* – The Lucia grammar covers many types of referring expressions, including proper names, pronouns, and a variety of noun phrases, including subject relative clauses, both reduced and not reduced, and recursion. Figure 3-6 shows a few options.

- *Imperative sentences* – In the Rosie corpora, imperative sentences show up as command sentences, of which there are many. Figure 3-8 shows a subset of the structures possible.
- *Declarative sentences* – Sentences in the corpora cover a complex structure of declaratives, including conjunctions and recursion, as in Figure 3-9.
- *Yes/No and Wh- questions* – Figure 3-10 gives some examples of questions that Lucia handles, with more in the corpora, and shows the possible structural options. Questions are handled according to their surface structure, in accordance with the construction grammar principles described above, and not with movement transformations as done with generative grammar approaches.
- *Conditional sentences* – The conditional sentences shown in Figure 3-11 have smaller complete sentences embedded in larger sentences with conditions. Negations are also processed correctly.
- *Subordinate clauses* – Some sentences have subordinate clauses, such as goal-defining sentences that use a complementary clause, like the one in Figure 3-9, or in the definition of an enabled command like G-034: *You can move a clear block onto a clear object that is larger than the block.*
- *Recursion* – Recursion is often considered an essential feature of human language, or even the fundamental distinguishing feature of human language (Hauser et al., 2002). The Lucia grammar for Rosie includes recursion in referring expressions (Figure 3-6), in imperatives (Figure 3-8), and in declaratives (Figure 3-9). Some sentences even show recursion a few levels deep, like G-015: *If the number of the locations between a location and a covered location is the number of the blocks that are on the covered location then you can move it onto the former location.*

#### *Language phenomena not covered*

Many language phenomena in common human use of language are *not* covered by this grammar since they don't appear in our corpora. Prior work with ECG has dealt with many of these issues (Bryant, 2008; Dodge, 2010), but Lucia has

not dealt with them so far because they are not needed for any of the ITL tasks taught to Rosie so far. Some of the important ones are described here.

- *Event descriptions* – Much of human language use involves describing events with agents, action verbs, tense, aspect, mood, etc. Given the prior work, the constructions described above suggest that our approach can handle these phenomena, but they are outside the scope of this thesis.
- *Grammatical features* – Much grammatical analysis is concerned with *grammaticality*, confirming whether a particular sentence conforms to certain linguistic conventions. These conventions often involve matching such features as number, case, etc. Humans can judge grammaticality, and a great deal of research in linguistics has been based on these judgements. The ECG formalism can deal with these features, and Lucia keeps track of some of them. However, actual day-to-day language use is often ungrammatical according to these standards. Due to Lucia's focus on producing actionable sentence meanings, we have chosen to ignore many grammaticality issues that had no impact on producing actionable meaning. However, Lucia can detect when a sentence cannot be parsed using its grammar, giving it the ability to judge grammatically relative to its knowledge of language.
- *Tenses* – Lucia does not deal with tenses in any comprehensive way. Almost all the sentences in the Rosie corpora are confined to the present tense, although *was* appears in two sentences, where Lucia treats it as equivalent to *is*.
- *Metaphor and other abstract concepts* – Work in cognitive linguistics, beginning with Lakoff and Johnson (1980), indicates that abstract concepts are derived from more concrete ones through things like metaphoric projections. Lucia and Rosie deal mostly with concrete concepts in the here and now. Some abstract concepts such as *goal* are dealt with, but without reference to how such concepts might be derived from more concrete ones.

- *Discourse dependencies* – The overall Rosie agent does language *understanding*, as we have defined it for this thesis, while Lucia does only language *comprehension*, i.e., transforming input sentences into grounded, actionable messages. Lucia does include a limited ability to resolve anaphoric references, such as when *Pick up the green sphere.* is followed by *Put it on this.* The Rosie agent does complex discourse analysis to understand instructions in their dialog context, but Lucia is not directly involved in that part of the process.

### *Limits of the ECG Formalism*

We have found that ECG’s ability to support compositionality and generality have made it a good tool for developing a grammar for Rosie. However, there are three important aspects of transforming sentences into actionable messages that the ECG formalism does not cover. The three aspects are: making local repairs, grounding to the agent’s knowledge, and formatting the final action messages. Lucia resolves these issues using hand-written Soar code that is distinct from the ECG grammar. Explanations of how they are resolved requires an explanation of the processing algorithm, and are given in Chapter 4. Overcoming these limits is a topic for future work.

#### **3.6.3 Generality**

In the context of this thesis, we use the term *generality* to mean that a comprehension system is capable of comprehending a larger set of sentences than those that were used to develop it. It is important to note that while generality is necessary, semantic precision and accuracy of meaning are also essential, thus creating a tradeoff between the two. Here we present evidence to support the claim that Lucia’s ECG grammar for Rosie has substantial generality. Measuring generality can be challenging, but here we present three strategies: corpus-based generality, syntactic productivity, and semantic breadth. The following explains each of these strategies and documents their results.

### *Corpus-based generality*

A quantitative question to ask is how the coverage grows as the set of sentences used for development grows. As discussed, the development of Lucia's grammar for Rosie proceeds one sentence form at a time. A sentence form that is not currently working is chosen and added to a dev set. Development proceeds by adding or modifying ECG items and hand-built rules until that sentence is processed correctly. Then a test is run on that entire corpus. Any sentence that works that did not work before is labeled as "JUST WORKS" at that point in the process. One way to measure generality is to look at a particular corpus and record how many more sentences worked correctly for each new sentence added to the dev set. Some examples will illustrate.

### *Case Study 3.6: An example of grammar growth for the Games corpus*

G-006	The goal is that a red block is on a green block and the red block is below an orange block.	Add to grounding
G-031	The goal is that a red location is below a red block.	JUST WORKS
G-082	The goal is that a blue block is on a blue location.	JUST WORKS
G-009	The goal is that a <i>blue</i> block is on a purple block and the blue block is below a <i>yellow</i> block.	Add <i>blue</i> and <i>yellow</i> lexicals Add to grounding
G-112	The goal is that a red block is on a red location and a green block is on a green location.	JUST WORKS

Sentence G-006 was added to the dev set. An additional hand-built rule for grounding was added to make this work, and then G-031 and G-082 also worked. G-009 did not work, however, until lexical items for *blue* and *yellow* and another grounding rule were added. With these, both G-009 and G-112 worked. The development for G-006 produced three working sentences for a generality ratio of 3.0, and that for G-009 produced two for a ratio of 2.0.

### *Case Study 3.7: An example of grammar growth for the Robot corpus*

R-047	Put the stapler on the desk.	Add <i>desk</i> lexical and schema
R-046	Pick up the stapler.	JUST WORKS

R-082	Fetch a stapler.	JUST WORKS
R-075	Deliver the stapler to Bob's office.	Add to interpretation
R-077	Deliver the stapler to the copy room.	Add to interpretation
R-079	Fetch a stapler from the copy room.	Add to interpretation
R-080	The only goal is that the stapler is in the starting location.	Add to interpretation
R-122	The only goal is that the plate is in the storage location.	JUST WORKS

For this set of sentences, the only new ECG items needed were the lexical item for *desk* and its associated schema added for R-047. In adding that schema, a problem was discovered in a detail of the schema for *stapler* from the Baseline grammar that needed to be fixed to get *stapler* to work with the Robot grounding system. After this, two more sentences worked, for a ratio of 3.0. For R-075, R-077, R-079, and R-080 no more ECG items were needed, but more interpretation code was needed for each one, giving each but the last a ration of just 1.0. With the interpretation for R-080 working, R-122 worked also, for a ratio of 2.0 for that sentence. If we consider only ECG items, the generality ratio for R-047 would be 8.0, but it is a lot less due to interpretation. This shows some of our challenges for getting all sentences to work fully end-to-end.

Beyond these examples, we measure generality with respect to the Games and Robot corpora, and three subsets of sentences for each. First is the set of *all sentences* in all the scripts for a particular corpus. This includes many sentence forms that are repeated multiple times. Next is the list of *sentence forms*, the subset of all sentences that are distinct, since often the same sentence is used many times in the scripts. Last is the *development set*, the set of sentences used as linguistic experiences for our process to actively develop new grammar.

Section 3.5 gave graphs showing the growth of different kinds of ECG items in the grammar as more and more sentences were developed. Here we look at how each increment of developing the grammar for a new sentence also made additional sentences work that had not worked before.

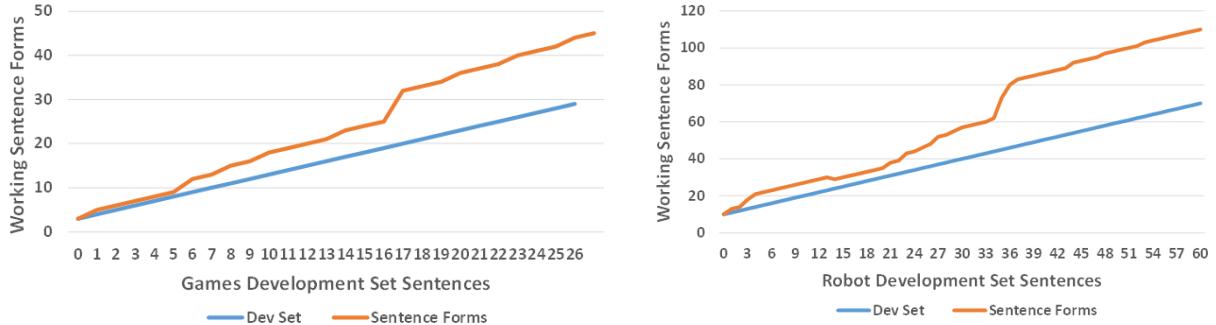


Figure 3-15: Generality for sentence forms

The graphs in Figure 3-15 show the how the number of working sentence forms, out of 172 for Games and 160 for Robot, grows as the development set grows. In these graphs the Dev Set line represents 1-to-1 growth as if no additional sentences ever worked when one new development set sentence was made to work. The Y-intercept of these lines shows the number of sentences that were already working, based on the grammar that was developed for the Baseline corpus, when work on this corpus began. The data show a faster growth for the sentence forms than for the development set, and also that occasionally a new sentence is developed that is similar enough to other sentence forms that several more work without further development.

These data do not show how many more sentences *could* be covered by the grammar as it grows, but only how many more sentences in the pre-defined corpus are covered. Because of the way the sentences in the corpora were designed by the Rosie developers, there are not many sentences that use exactly the same set of constructions. The transfer from one sentence to others is limited because sentences were deliberately designed to be different to stretch the system, so that many sentences have linguistic forms that others don't have.

There are also some anomalies of development in these data. The big jump in the robot data around development sentences 35 and 36 is due to the fact that at that point in the process it was discovered that many of the gold standard messages we were using for testing were obsolete and needed to be brought up to date. A number of sentences were being processed to the IR test point correctly

by Lucia, but the formatted messages did not match the obsolete gold standard. Those changes to the gold standard made these sentences start working without any changes to the grammar.

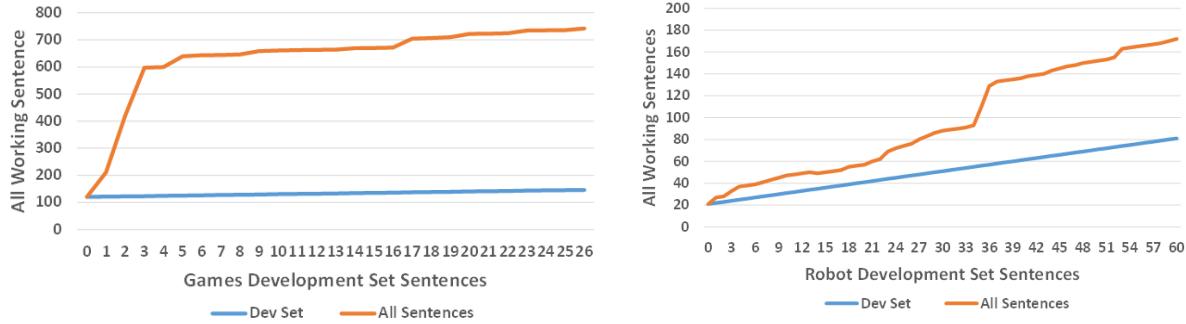


Figure 3-16: Generality for all sentences

Figure 3-16 shows similar trends for the growth of working sentences in the all-sentences sets as the development sets grow, out of 1,104 for Games and 228 for Robot. Due to the repetition of sentence forms in the all sentences set, especially for Games, the number working in this set grows much faster than that for sentence forms alone. For the Games corpus there were already 120 sentences working when development on this corpus began, and the first two sentences developed were ones that were used many times throughout all the scripts. These effects were not as great for the Robot corpus.

These data raise the question: What will happen in the future if we add new sentences or new sentence corpora? That is hard to say. Each new corpus or domain likely will introduce a lot of new vocabulary and, since our coverage is still rather limited, new syntactic and semantic structures as well. Especially since our focus is on constructing the full meaning of a sentence, the range of meanings of all human language is enormous if not unbounded. Learning language for new domains or situations will always require more work. That is why future use of Lucia or some similar system needs to have a system for language acquisition from experience. That is a major area for future work. In Chapter 6 we propose an approach to solving that problem.

### *Syntactic productivity*

By studying the grammar as presented in this chapter and in Appendix 2, it is clear that the composable units can be combined in many more ways than those captured in our corpora. If the system can comprehend and ground a phrase like *the green sphere*, and if the words *red* and *blue* are defined similarly to *green* and can be composed in the same way, then *the red sphere* and *the blue sphere* can also be comprehended. We have developed two quantitative methods to evaluate this intuition, one based solely on syntax and the other more on semantics.

We have written an algorithm that calculates a measure called *productivity* for each construction by calculating the number of all the possible phrases or sentences that it could produce. A lexical construction is assigned a productivity of 1, a composite construction gets the product of the productivities of its constituents, and a general construction gets the sum of the productivities of its children. Since this is infinite when recursion is involved, we limit the expansion to a single level of recursion. Table 3-2 presents the productivity numbers for a number of important constructions.

Table 3-2: Productivity of selected constructions

<b>Construction</b>	<b>Productivity</b>
RefExpr	755,683
CommonNoun	50
SpecifierNP	755,551
SpecNoun	300
SpecPropNoun	13,800
SpecProps2Noun	634,800
RefExprPrepPhrase	1
RefExprRelClause	4
Imperative	4,568,650,852,994
TransitiveCommand	34,005,735
ImperativeWithLocation	15,113,682
Declarative	25,698,594,172,099
DeclarativeAndDeclarative	1
ConceptIsThatDeclarative	1
ConceptIsThat	1,511,366
RefIsPrepPhrase	1,511,366

Question	22,842,552,973,638
YesNoQuestion	22,842,462,291,676
WhQuestion	90,681,962
Conditional	long integer overflow
IfConditionThen	25,698,594,172,099
IfConditionThenCommand	2,157,736,395,692,839,833
IfConditionCommand	7,070,298,445,444,069,771
IfConditionThenStatement	25,698,594,172,099

### Semantic breadth

The above approach produces large productivity numbers, but they are not realistic since many of the sentences generated in this way would not be semantically meaningful, such as *Pick up the green kitchen*. Therefore, we have built a system for testing the meaningful generality of at least parts of Lucia's grammar for Rosie. Using a test version of a single instance of the World Model and knowledge of Rosie's messages, we've generated a set of messages that could be meaningful in that situation. The algorithm maps these messages onto the grammar to produce sentences that are both generated by the grammar and meaningful to Rosie.

Using this method, we have tested the productivity of a part of the grammar involving simple commands, with 12 different verbs and a variety of referring expressions that can be grounded to the particular World Model. The sentence generator produced 1,020 such sentences using the grammar needed for the Baseline corpus, only 8 of which appear in the Baseline. This list of sentences was processed with Lucia, and all 1,020 sentences produced messages conforming to the Rosie message format. This gives a generality ratio of 127.5 generated sentences for each sentence used in development. We expect similar or even greater ratios would result for other parts of the grammar.

These calculations indicate that this grammar, despite its limitations and a small vocabulary, can generate a large number of sentences. If a human were to process one sentence every second, 24 hours a day, 365 days a year, over a lifetime of 100 years that would be a total of a little more than  $3 \times 10^9$  sentences.

The numbers in Table 3-2 go beyond  $7 \times 10^{18}$  for some kinds of sentences. The grammar should grow exponentially with a larger vocabulary, giving much large numbers for, say, 50,000 words instead of a few hundred. Given this difference of many orders of magnitude, and that the productivity numbers were calculated while limiting recursion, it seems clear that Lucia's grammar has a high degree of generality within the scope of what its grammar can cover.

### **3.7 Conclusions and Implications**

This chapter describes a theory of representing composable knowledge of meaning of language (CKM), as well as data on its practical application to the scope of language used by Rosie in learning and performing its ITL tasks. In the Related Work section, we discussed briefly the traditional generative grammar theory of knowledge of language, as well as the radically different construction grammar (CxG) approach. We then presented the ECG theory of CKM and the extensive ECG grammar developed for Lucia in Rosie.

This aspect of our work makes several contributions. First and foremost is a demonstration that a grammar based on CxG theory, and ECG in particular, made up of a network of form-meaning mappings that apply to surface structure and were developed with a usage-based paradigm, can be used to provide embodied, end-to-end comprehension (E3C). We have shown that such a grammar has substantial generality while at the same time providing the semantic precision required by the E3C application in Rosie. A key idea of CxG and ECG that allows us to achieve the semantic precision needed for E3C along with generality is that many specific constructions can provide precise semantics while being subcases of a few general constructions that assemble referring expressions, clauses, and sentences. Constructions are cheap, precision is critical.

The next question is whether an algorithm that does incremental, immediate interpretation processing (I3P) can indeed realize this capability using the ECG grammar we have defined. Chapter 4 dives into that question.

## **Chapter 4 Constructing Meaning, Piece by Piece**

A major contribution of this thesis is the design and implementation of an algorithm that performs incremental, immediate interpretation processing (I3P), using ECG to represent composable knowledge of meaning (CKM) and Soar to model general cognitive mechanisms (GCM), to achieve embodied, end-to-end comprehension (E3C). That algorithm is the subject of this chapter.

Human processing is incremental, word-by-word. Meanings are interpreted immediately in the sense that each word or complete phrase is immediately grounded (when possible) to the person's perception and/or world knowledge to permit the person to act on intermediate results, such as moving the eyes to focus on objects referred to (Tanenhaus et al., 1995). In order to act immediately, a single interpretation must be chosen out of what may be multiple possibilities to provide a single-path process. This immediate commitment is sometimes wrong, so a correction mechanism is also needed. Lewis (1993) summarizes empirical data on incremental processing and presents an approach to modeling that uses local repair to resolve local ambiguities, which is adopted in this work.

Given this requirement, this chapter addresses the following principle of the Lucia theory:

**I3P: Incremental, Immediate Interpretation Processing** *The meaning of a sentence is constructed incrementally in real time, each new construction unit (CKM) added is immediately grounded (as possible) to embodied world knowledge (I3P), and the meaning of each full sentence is interpreted as an action message (E3C).*

As listed in Chapter 1, there are several AI systems that do some form of embodied, end-to-end comprehension (E3C). However, none of these attempts to model the incremental, immediate interpretation processing (I3P) of humans, which is one of the design goals of Lucia. This chapter addresses this problem.

## 4.1 Research Background

Before looking at the details of the problem and Lucia's I3P algorithm, we review briefly some of the research on human incremental processing. Lewis (1993) presents a discussion of the immediacy of syntactic, semantic, and referential processing (what we call grounding), including references to empirical studies that support the idea that humans process incrementally and with immediate interpretation.

A classic paper by Tanenhaus et al. (1995) reports human eye-tracking behavior in an experiment where subjects are looking at a visual scene while listening to sentences. They show that with an ambiguous sentence using a reduced relative clause, the subjects make an eye movement to an object that turns out to be incorrect for the full sentence and return to the correct object when the disambiguating information is received. Their data on the time course of eye tracking shows that the eyes move shortly after a referring expression has been heard and before subsequent words have been processed. This eye movement could not happen unless the referring expression were interpreted immediately. For sentences using an unreduced relative clause there is no ambiguity, and the incorrect eye movement does not happen.

Evidence from neuroscience is that perceptual and motor areas of the brain are activated during comprehension (Kemmerer, 2015; Pylkkänen, 2019). Psychologists often use the term *simulation* to describe this activation (Barsalou, 1999; Bergen, 2012). Although “simulation” may mean something different to computer scientists, this is the term some psychologists and neuroscientists use for this phenomenon. The evidence presented by Bergen (2012) shows that the language interpretation required to produce this simulated activation must be immediate.

A key consequence of this evidence for immediate interpretation is that human processing, in many cases at least, must choose a single path through the space of possible interpretations during incremental processing. Whenever action, or even simulated activation, is required, only one choice is possible. The

eyes cannot move in two directions at once. Lewis (1993) uses this as the basis for his theory of parsing with local repairs when an incorrect choice was made.

Christianson and Chater (2016) present a theory they call “chunk-and-pass,” which states that multiple elements of a comprehension must be combined into larger chunks fairly quickly to avoid being lost due to limits of working memory, which they call the “now-or-never bottleneck.” This implies immediacy of processing, although they do not provide any computational model of how this might be done, nor do they address grounding of references in their theory. The piece-by-piece processing described below implements the theoretical constraints of chunk-and-pass processing, and it was developed concurrently and independently (Lindes & Laird, 2016).

Another aspect of human processing is *semantic priming* (Carroll, 2008, pp. 123–124; Ferrand & New, 2004), where processing one word increases the activation and thus reduces the retrieval time for semantically related words. There is also literature on the idea of *syntactic priming* (Bock et al., 2007; Reitter et al., 2011). The empirical data indicate that both lexical items and syntactic structures spread activation in the brain such that similar or related items are retrieved more easily afterward. These effects have a strong influence on detailed measures of human processing and reading times. This aspect of human processing is not modeled by Lucia at this time. However, in Chapter 6 we discuss experimental explorations of adding priming to Lucia.

## **4.2 Defining the Problem**

Together, this research on human incremental processing implies three constraints that a model of this processing must satisfy. When a sequence of linguistic elements that compose a larger element is encountered, these elements must be combined into larger chunks fairly quickly to avoid loss of information due to limits on human working memory (Christiansen & Chater, 2016). Individual words and phrases must be interpreted immediately, or grounded in our terminology, as soon as they are processed, as indicated by Tanenhaus et al. and the research on simulation cited above. (Tanenhaus et al., 1995). When

a local ambiguity arises, a single choice must be made for further processing, and later corrected if the choice conflicts with subsequent input (Lewis, 1993). We call these the *rapid composition constraint*, the *immediate grounding constraint*, and the *single path constraint*, and we show how Lucia satisfies each of them.

The implication for Lucia of the I3P requirement and these three constraints is that the meaning of a sentence must be composed from small units of form-meaning mapping with an incremental process that performs immediate interpretation of intermediate results. The full sentence meaning must then be interpreted as an action message for E3C. This leads to the computational question of what algorithm can do this I3P processing. This chapter describes Lucia's I3P algorithm at a conceptual, architecture-independent level, while Chapter 5 describes the implementation of that algorithm in the Soar architecture.

The algorithm needs an internal representation of the current state of the comprehension, and a way of updating and grounding this state incrementally as each new unit of form-meaning mapping is added. Lucia's algorithm is built around its *comprehension state*, a central data structure that represents the current state of the comprehension. This state is updated and grounded using the constructions defined in Chapter 3 in a dynamic *construction cycle*, which is the central increment of processing that constructs meaning piece by piece in three phases: a *selection phase* chooses which construction to apply next, an *integration phase* builds a new form-meaning node and adds it to the graph, and a *grounding phase* grounds the node's meaning to the agent's world knowledge.

**Commitment 3:** *Lucia does immediate interpretation processing (I3P) using construction cycles that select, integrate, and ground one construction at a time, often with multiple construction cycles for a single input word.*

Construction cycles perform compositional processing as they build up the comprehension state. For each word cycle, one lexical construction cycle processes the lexical construction for that word, and then as many composite construction cycles as possible combine items in the comprehension state

according to composite constructions, performing *rapid composition*. Each construction cycle has three phases: *select* which construction to apply next, *integrate* that construction with the current state, and *ground* that construction as much as possible. Since each construction is grounded in the same cycle, this process achieves *immediate grounding*. Additional logic performs local repairs when needed to maintain *single path* processing, . This approach satisfies all three of the constraints from related research outlined in the previous section. At the end of a sentence, the final comprehension state is encoded as an action message in a process called *sentence interpretation*.

### 4.3 Overview of the Lucia Processing Algorithm

This section summarizes key points of Lucia's I3P algorithm. Figure 4-1 is a simple sketch of the information flow for a comprehension system, C, in the context of an embodied agent.

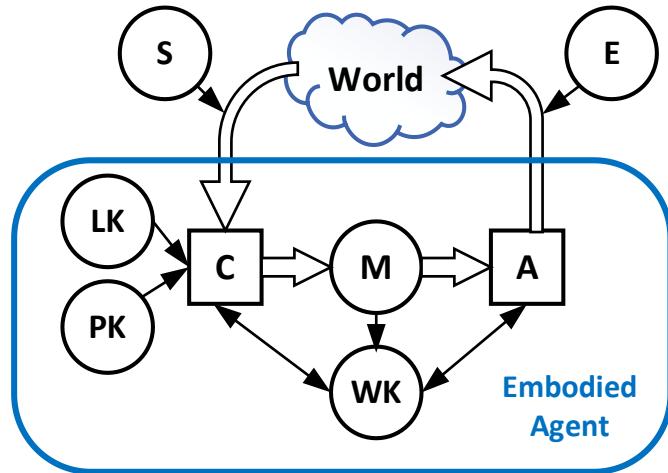


Figure 4-1: Comprehension in context

The World provides situations, S, to the agent, which consist of sentences in natural language along with other perceptual input. The comprehension engine C gathers information from S as well as from its linguistic knowledge, LK, its processing knowledge, PK, and the agent's world knowledge, WK, to process each input sentence to produce an output message M, which is grounded to WK. The operational part of the agent, A, uses each message M to choose one or more

internal or external actions to perform. These actions produce an effect, E, on the external world, and the cycle repeats.

Within this general model, Lucia is the comprehension engine C. Its linguistic knowledge, LK, is the ECG grammar described in Chapter 3. Its processing knowledge, PK, is a set of hand-built Soar production rules to be described in Chapter 5. Lucia produces action messages, M, using the world knowledge, WK, that it shares with Rosie. The Rosie operations module, A, takes actions based on these messages, causing the overall agent to learn and perform new tasks through interactive instruction.

The Lucia comprehension engine, C in Figure 4-1, presented before in Chapter 1, performs a part of the perception, cognition, action cycle shown for each sentence. From Lucia's perspective, the flow of information for each sentence is in the hierarchy of cycles shown in Figure 4-2.

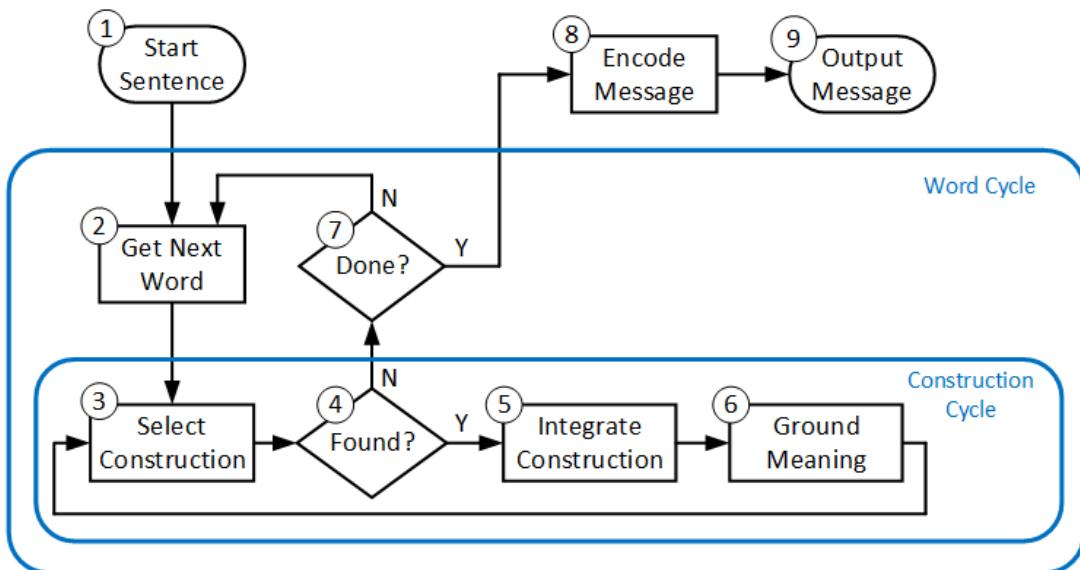


Figure 4-2: Cycles in incremental comprehension processing

Lucia processes one sentence at a time, mostly independently of other sentences. A new input sentence comes in at box 1 and is processed and encoded to produce an action message at box 9. Based on that message, the agent does some reasoning, decides what action to take next, and performs that action. A

variety of actions are possible, including changing the agent's internal knowledge state, outputting a sentence to the instructor, and/or performing a physical action in the world.

Box 1 in the figure shows the processing involved in setting up the algorithm state to start a new sentence. Boxes 2-7 do the comprehension processing, and when the end of the sentence is reached, the complex meaning structure that has been built is interpreted in box 8 to encode the action message that is then sent to the agent. Box 9 posts the message produced in a place for Rosie to reason about it and act on it.

Lucia's processing of an input sentence is incremental on a word-by-word basis in a *word cycle*. Box 2 in the figure represents the logic to bring the next input word into the part of WM where Lucia accesses it. The processing of a word happens in one or more inner cycles, one for each construction to be added to the current state of the comprehension.

The core of Lucia's theory of comprehension processing is this inner cycle we call the *construction cycle*. A single construction is selected, integrated into the comprehension state, and grounded in each construction cycle. Every word cycle has at least one construction cycle where a lexical construction is selected in box 3 to match the current input word. Once this lexical construction is added to the state, in box 3, the algorithm attempts to select a composite construction to add. This can be repeated more than once. In box 4, if no composite construction is found to match the current state, control proceeds to box 7.

Box 7 looks to see if there are more words to process, or if the end of the sentence has arrived. If another word is present, control proceeds back to box 2, where that word is sent to Lucia and processing continues. Otherwise, control proceeds to box 8.

There is level of interpretation for the whole sentence, shown in box 8 in Figure 4-2, that produces an action message. This involves taking the comprehension state that has been built up for the whole sentence and encoding it by projecting it onto the space of possible action messages. Often this involves considerable simplification of the data structures. For example, often the final

message need only contain the internal representations for an action to be performed and an object to act on, rather than all the complexity of the lexical items, syntax, and semantics that were needed to represent the action in natural language. Chapter 5 provides details of how this encoding is done.

## 4.4 Piece-by-Piece Processing

This section describes how construction cycles build the complete structure of the analysis of a sentence, piece-by-piece. While most other approaches to incremental processing consider simply processing one word at a time, our more granular approach processes one construction at a time. This is consistent with our model of representing knowledge of meaning, since the constructions, or pieces, are the units of mapping form to meaning. Several new constructions may be added for each word, and each one has its own meaning that is grounded before going on to the next construction. This section focuses on building syntactic structure, including the selection phase and part of the integration phase of each construction cycle. The following sections elaborate further on achieving immediate interpretation and single path processing along the way.

### 4.4.1 An example sentence comprehension

To make our discussion concrete, consider an example comprehension.

*Case Study 4.1: An example sentence comprehension*

B-020	Pick up the green block on the stove. command(op_pick-up1, 015)
-------	--

Figure 4-3 shows the comprehension state and its grounding after this sentence has been fully processed. In the middle of the figure is a sketch of the tree of construction nodes built up over fifteen construction cycles, along with the input sentence underneath and the output message at the top. The message only needs three pieces of information: the fact that it is a command, and internal identifiers for the action and its object.

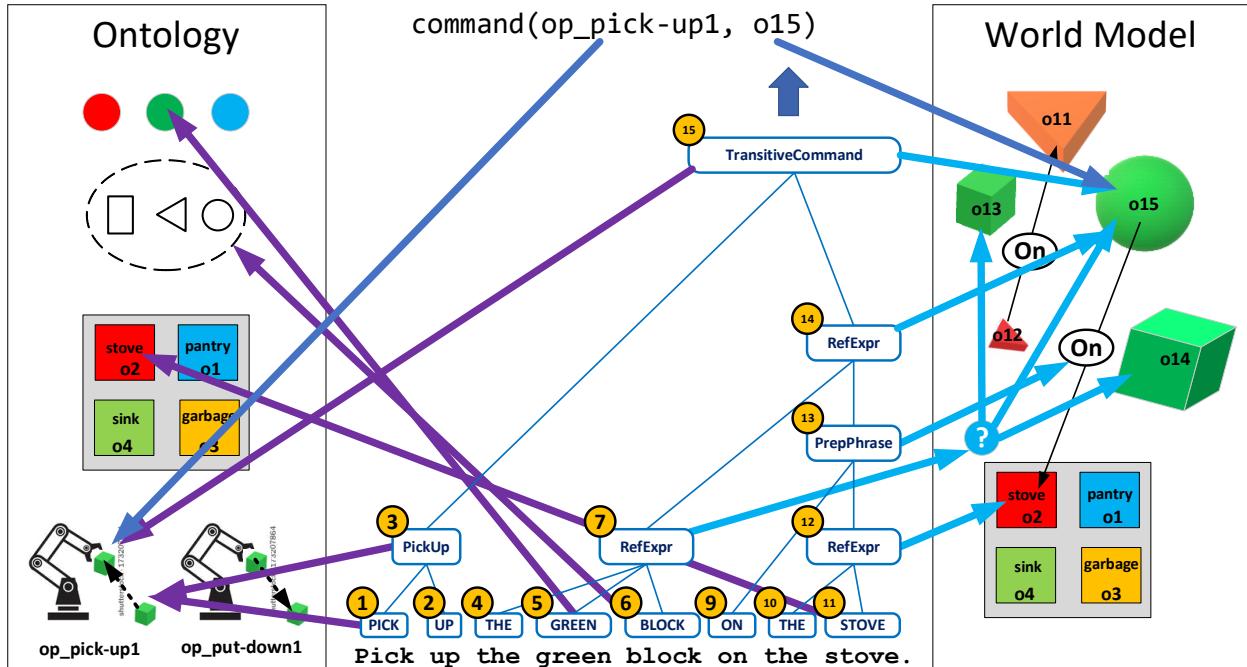


Figure 4-3: The grounded meaning of a sentence

Nodes in the comprehension state represent instantiated constructions with their meanings, and are numbered in the order they were created. Node number 8 is missing because, as we explain below, a node 8 was created but later discarded as part of a local repair.

The block on the left represents the *Ontology*, knowledge in Rosie's LTM, with purple arrows showing how lexical items are grounded to elements there. The block on the right is a sketch of the *World Model*, or Rosie's internal representation of what it currently sees in its visual perception. Blue arrows show grounding to elements there.

The RefExpr (short for referring expression) at node 7, which represents *the green block*, can ground to three green blocks in the world. The phrase *on the stove* allows further grounding for node 14 to select from these three possibilities and ground only to O15, thus resolving the semantic ambiguity.

#### 4.4.2 A tree of chunks with a root

Figure 4-4 shows a different view of the final comprehension state for this example sentence that shows only the syntactic structure. The nodes are much

abbreviated as shown in the key, and the grounding is not shown here. Several important points are illustrated in this diagram.

The structure of the tree in the diagram has a root node, the C in this case. Each composite node has parent-child links to its children, while lexical nodes are leaves of the tree. There are sibling links between the children of a composite node, which appear in the same left-to-right order as their addition to the tree.

The structure shows what we mean by a *chunk*<sup>6</sup> in the context of chunk-and-pass processing. Each of the circular nodes is an instance of a composite construction, and each of these is the root of a subtree made up of its constituents. For our purposes, we consider each such subtree as a chunk. For instance, the phrase *Pick up* is represented as the chunk made up of the A action verb node and the p and u lexical nodes.

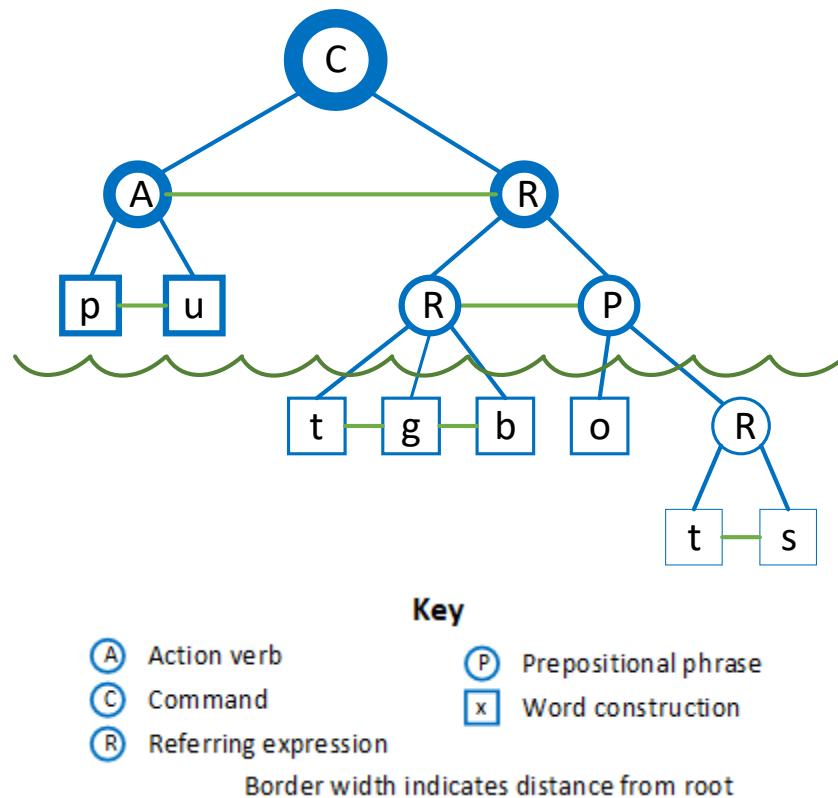


Figure 4-4: The final structure of the example sentence

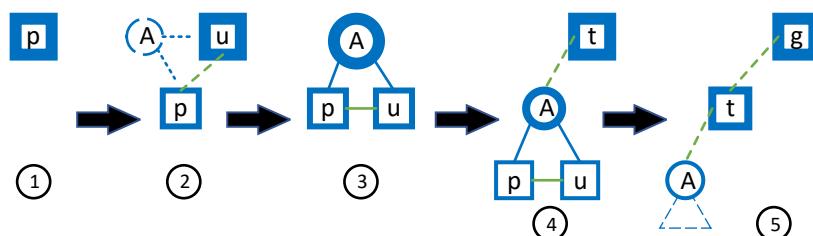
<sup>6</sup> This use of the word *chunk* is very different from its use in Soar and ACT-R. See Chapter 5.

This structure can appear to be similar to a standard parse tree diagram. However, it has a feature that is unique to how Lucia implements its processing. We have found through our experience with designing and implementing the grammar for Rosie that only the top three levels of the tree need to be accessible for selecting the next construction to add during incremental processing. Nodes at level 4 or below are not considered in processing, but they are not forgotten and may later become accessible again as the shape of the tree changes through composition of nodes covering larger parts of the sentence. The figure represents this with the waterline that divides the upper nodes that are actively engaged in processing from the lower ones that are still remembered but not actively engaged. This idea is implemented by convention, since Soar has no architectural mechanism for it. In Chapter 6 we explore experimentally ways to implement this constraint using architectural mechanisms.

#### 4.4.3 Evolution of the tree with rapid composition

The final state shown in Figure 4-4 is built by a sequence of fifteen construction cycles, as shown in Figure 4-5. The following analysis of this figure reveals many of the features of Lucia's incremental processing.

Examination of the fifteen frames from the figure explains how the computational process in Lucia works. The discussion here is limited to structure building, which involves the selection phase and the first part of the integration phase of each construction cycle. Consider the first row of Figure 4-5.



In frame (1) the lexical node for *Pick* is built and becomes the initial root of the tree. In frame (2) the node for *up* is built and replaces the p node as the root. The u node is connected to the p node by a *previous link*, and an A node is

considered to compose these two lexical items. Construction cycle (3) creates a new chunk for A with its two constituents.

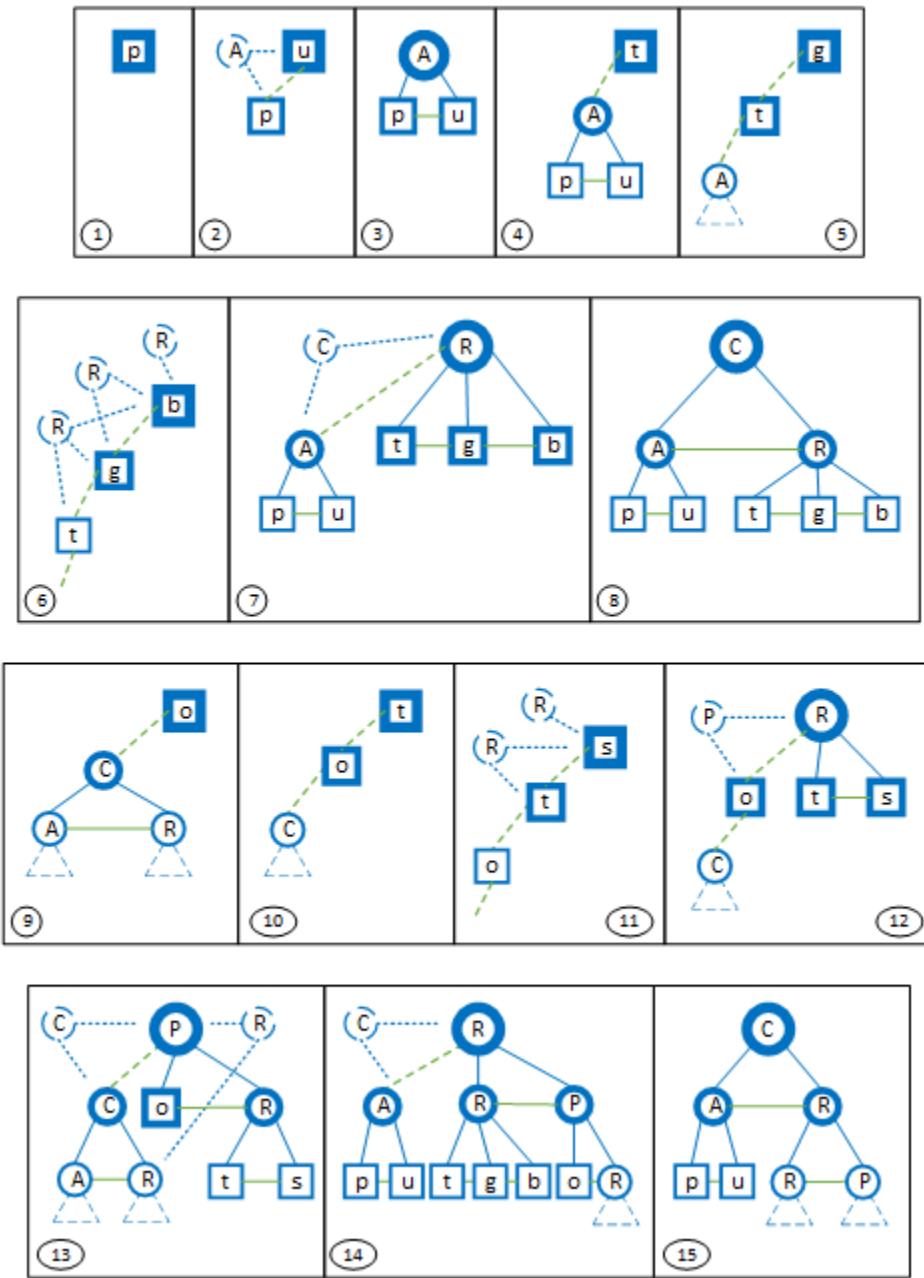
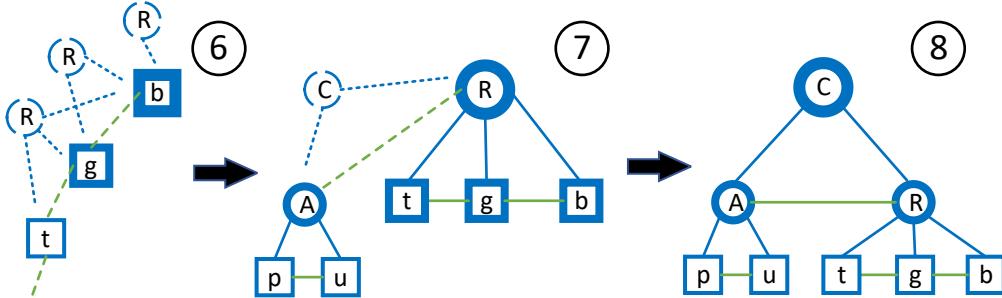


Figure 4-5: Fifteen construction cycles for the example sentence

Frames (4) and (5) show two more lexical nodes for *the* and *green* being added. This gives a sequence of three nodes connected by previous links. This sequence of three nodes illustrates that the root node with a sequence of previous

nodes emanating from it forms a subset of the comprehension state tree, which we call *the stack*. As shown in frame (2), the stack is the primary set of nodes that are candidates for forming a new composite chunk.



The second row of the diagram shows further growth of the stack and further composition. In frame (6) three different RefExpr constructions are candidates for composition, with one, two, or three constituents, respectively. Lucia's processing knowledge, PK, includes a preference heuristic so that when multiple composite constructions all match the stack, one that spans more nodes is preferred. In this case, the R with three constituents is selected.

The above shows three examples of creating a new chunk, in frames (3), (7), and (8). Each new chunk is composed of constituents that are removed from the stack, and the new chunk is "passed" on to the next cycle as the new root of the tree. Hence *chunk-and-pass*.

Each frame in Figure 4-5 represents the result of a single comprehension cycle. In each cycle, the *selection phase* chooses a new construction to be added and the *integration phase* restructures the tree with that new construction added. At this point we need to explain further the selection phase, which operates differently for lexical and composite constructions.

Although the details of architectural mechanisms must wait until Chapter 5, at the conceptual level of this chapter, we can think of each construction contained in the linguistic knowledge, LK, of Lucia as an independent active pattern recognizer. When the flow of control enters the selection phase, all these recognizers are matched in parallel at the state of the comprehension. Any of them that matches the state is a candidate to be selected. When there is more than one candidate, as mentioned above for going from (6) to (7), Lucia's PK uses

a set of heuristics to select one candidate to process on its single path. The section on single path processing below explains how this works for many more cases.

One more detail about the selection phase. The multitude of parallel pattern matchers, or abstract constructions, is divided into two classes: lexical and composite. When the algorithm is in the first construction cycle of a new word cycle, only the lexical constructions are active. Each looks for a match of its known orthography to the current input word, which is not shown in the diagrams. After a lexical construction has been processed for that word, subsequent construction cycles only activate the composite constructions. This process continues until no composite construction is found to match the state, at which point a new word cycle is initiated, as shown in Figure 4-2.

## 4.5 Immediate Interpretation

The previous section described how construction cycles build the comprehension state one piece at a time, covering the selection phase of construction cycles and the part of the integration phase. This section discusses the whole integration phase, including that described above in more detail, and the grounding phase, which together provide immediate interpretation.

### 4.5.1 Integration

Once a construction has been selected, a new node must be constructed and integrated into the comprehension state. There are several parts of this process.

1. The selected construction is *instantiated* by building a data structure representing it, with the construction name as the bottom level of its type hierarchy.
2. Based on *subcase* of specifications, the construction instance is *generalized* by labeling it with the names of all the general constructions in the sequence of subcases.
3. This new node is *attached* into the tree by making it the new root node.
4. A new lexical node gets the previous root node as its *previous* node.

5. For a composite construction, its constituent nodes are *linked* as its children, and whatever was on the stack before its first child is linked as its *previous* node.
6. Whatever meaning schemas the construction specifies are *evoked*, meaning they are instantiated and the node is linked to them.
7. The meaning schemas are *populated* by using the constraints specified in the ECG for the construction and the schemas involved.

#### 4.5.2 Details of nodes in the tree

The diagrams above represent the syntactic structure of the sentence, but what is needed is a process to construct the meaning of the sentence. To see how this is done, we examine in more detail the inside of a node. For this purpose, consider a simpler sentence.

*Case Study 4.2: A simple command*

B-054	Go to the kitchen. command(op_go-to-location1, to1(L28))
-------	---

An overview of the final comprehension state for this sentence is shown in Figure 4-6.

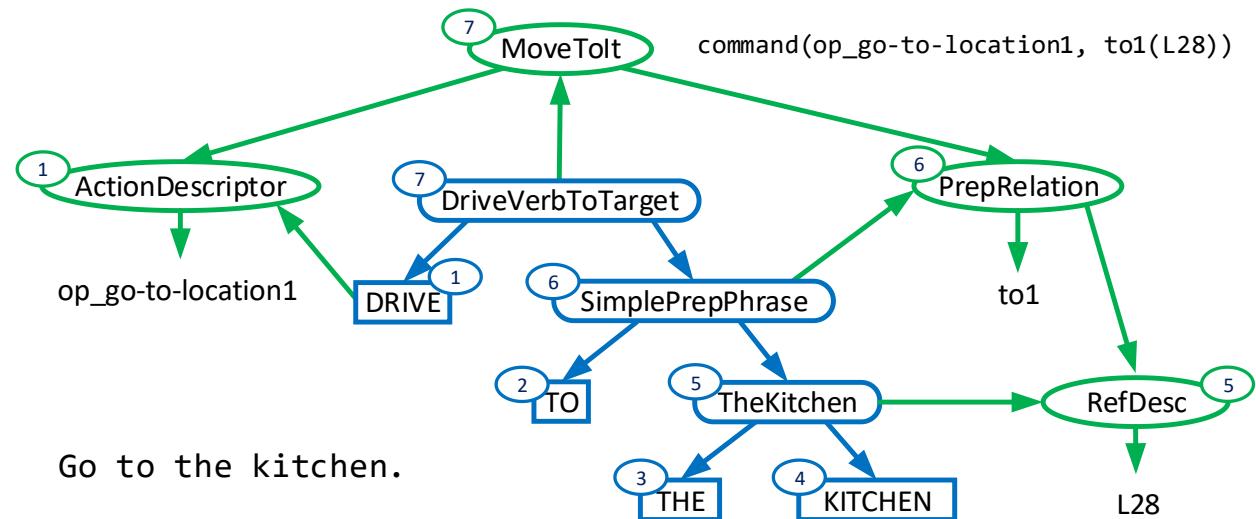


Figure 4-6: Full tree for a simple command

Here there are instances of constructions in blue and instances of schemas in green. The schemas show pointers to items they have been grounded to. To fully understand the algorithm, we need a more detailed view. Figure 4-7 shows just the construction instances with their detail, while Figure 4-8 shows the schema instances in detail, including details of the items they are grounded to. Not explicitly shown in these two diagrams is that each construction instance also has a link to the corresponding schema that was evoked for that construction, as seen in Figure 4-6. What we have been calling a node in our tree consists of a construction instance along with zero, one, or more schema instances that it evokes.

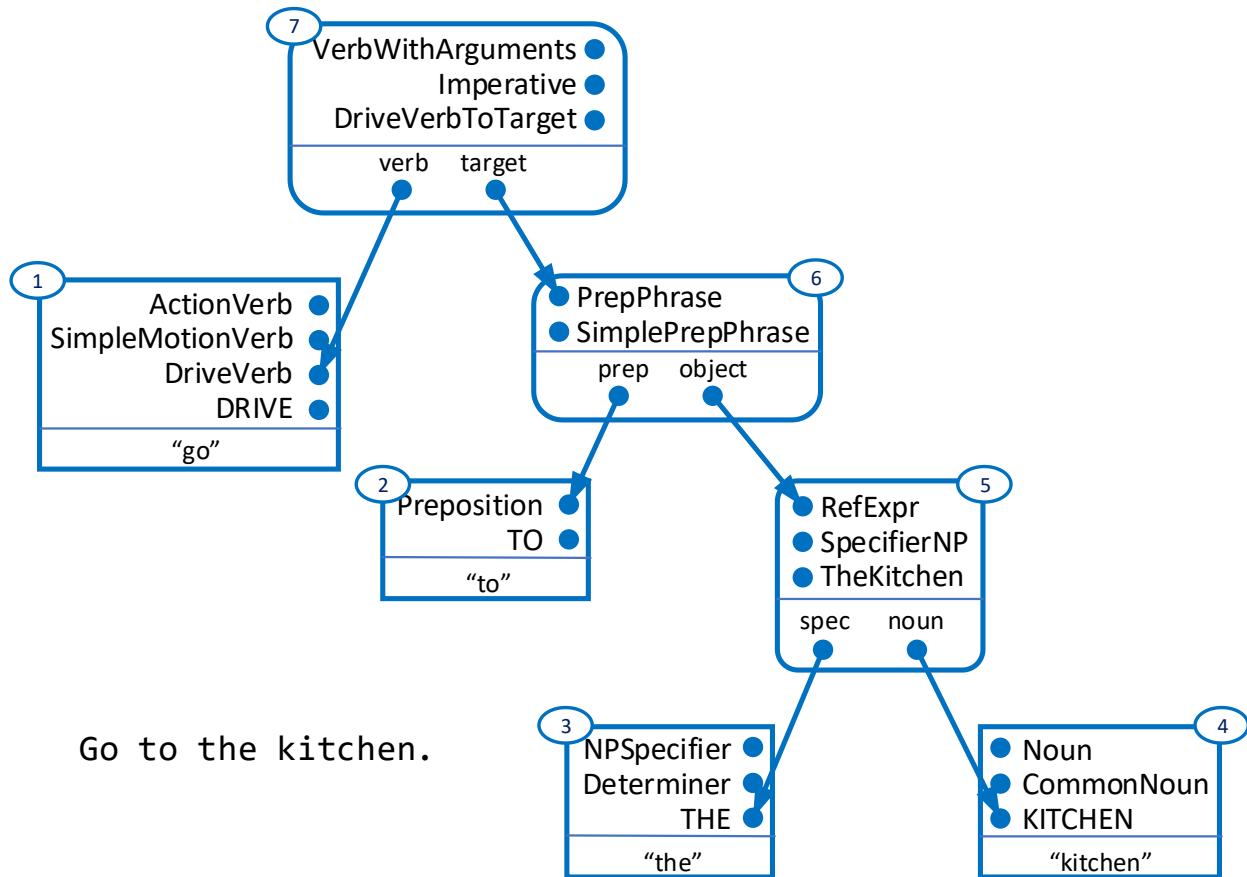


Figure 4-7: Details of construction instances

This sentence also serves as a good example of how semantic precision is implemented. The TheKitchen construction shown as node (5) in Figure 4-6 is chosen over a more general SpecNoun construction that would also match the

current stack because TheKitchen specifies specific lexical items to match to. This is an instance of a preference heuristic for a more specific versus a more general construction.

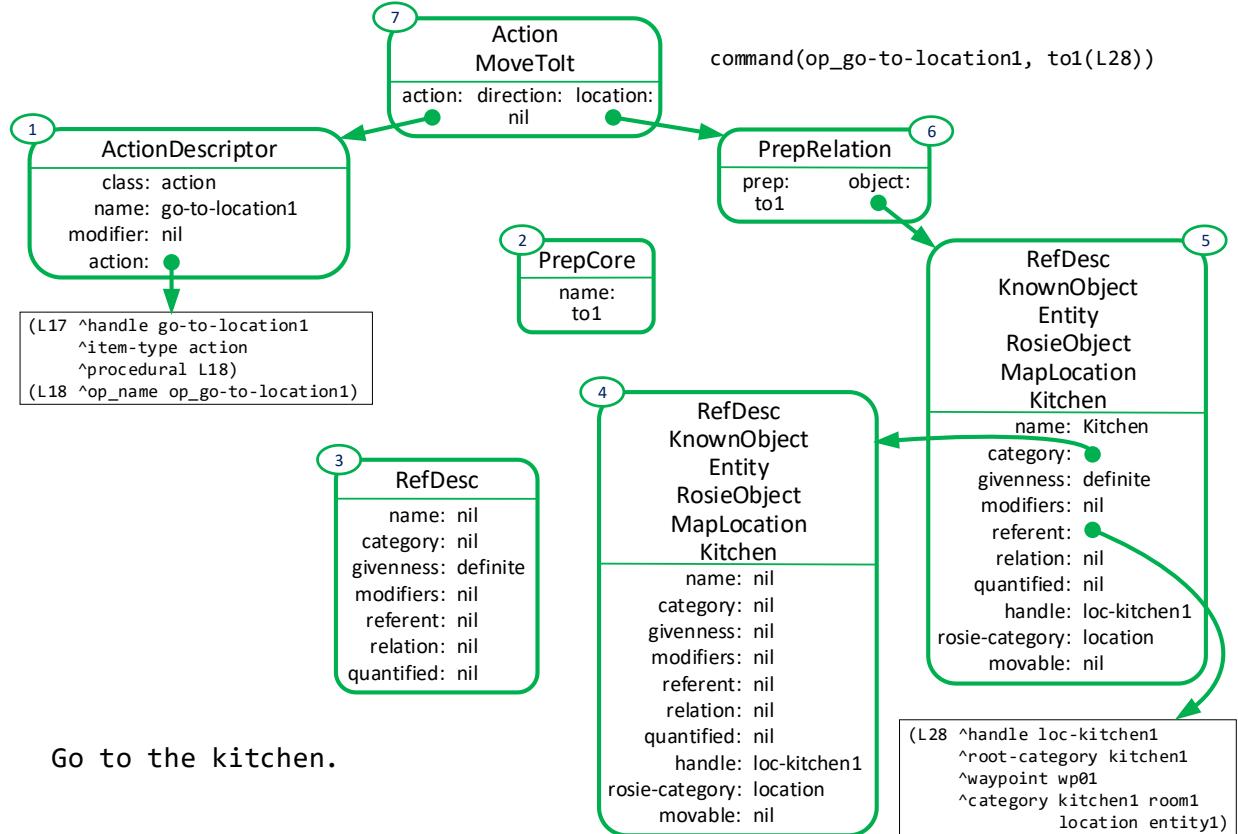


Figure 4-8: Details of schema instances

Each construction or schema in these diagrams has two parts separated vertically. The part above the line is a list of all the labels from the type hierarchy that apply to this item, from most specific, which is also the construction or schema name, at the bottom to the most general at the top. For instance, the construction for *the* is named THE, which is a subcase of Determiner, which is a subcase of NPSpecifier. Depending on the amount of semantic precision needed for a particular construction, a link from parent to child goes to the child label at the proper level of specificity. For example, the TheKitchen construction needs to be specific to be grounded properly, so it refers to its constituents at the lowest level, THE and KITCHEN. The SimplePrepPhrase, on the other hand, can accept any Preposition and any RefExpr as its children. In the design of the grammar,

the choice of specificity for the constituents of each construction is important to achieve the amount of semantic precision desired. Below the line in each instance are the slots for detailed data, namely the orthography (spelling) of a lexical construction and pointers to the constituents of a composite one.

The schemas work a little differently. Their slots are filled with either constant values or pointers to other schemas. They also have their levels from the type hierarchy, but this is not used to connect them. Instead, the ECG for a construction defines what type of schema to evoke, and the schemas themselves define the type hierarchy, which is important for their grounding. ECG constraints in both the evoking construction and the schema itself specify how to populate the values of a schema's slots. When a node is built, the construction and its evoked schemas are instantiated, the type hierarchy is filled out, and the various slots are populated. The piece-by-piece construction of the tree one node at a time results in the meaning composition needed to comprehend each individual sentence, as well as a large space of possible sentences.

### 4.5.3 Grounding

Figure 4-8 shows the network of grounded schemas for our example sentence B-054. The action slot of schema (1) points to a data structure in Rosie's LTM that represents the action for going to a location. The referent slot of schema (5) points to a representation of the kitchen in the current world model. Figure 4-3 shows a more complicated example of how the different nodes in a tree are grounded to Rosie's world knowledge in both LTM on the left and WM on the right. Chapter 2 provided details of the kinds of world knowledge to be grounded to, along with descriptions of hypothetical and dynamic grounding. Chapter 5 will discuss the details of the algorithm for making this happen.

## 4.6 Achieving a Single Path

Natural human language is full of a variety of ambiguities. Some sentences in some contexts may be ambiguous at the full sentence level. These are *global ambiguities* and can only be resolved by additional dialog or exploration in the world, so they are beyond the scope of Lucia's focus on sentence comprehension.

There are also *local ambiguities*, cases where at some intermediate point in incremental processing of a sentence there are two or more possible choices of what to do next, and the information needed to make the choice correctly comes later on in the sentence. Due to our single path constraint, immediate interpretation requires making a single choice in these cases.

In Lucia's I3P model using ECG, these choices for local ambiguities come down to which construction to select next, so the details of the selection phase are key to resolving these ambiguities. Lucia's general strategy for these situations is to make a single, sometimes called *greedy*, choice of a construction based on heuristics, and then make a correction later if necessary. This section examines many cases and how Lucia handles them (see also Lindes & Laird, 2017a). There are also cases where local repair is not sufficient, and more complex deliberative processing would be needed. This is discussed further in Chapters 5 and 6.

#### **4.6.1 Lexical ambiguities**

At the beginning of each word cycle, Lucia selects a specific lexical construction to integrate with the comprehension state. However, many words have multiple meanings or uses. Words can be categorized as function words or content words. Function words, like determiners and prepositions, come from limited sets, and each is often used in many ways with the meaning determined by the context. Content words, like nouns, adjectives, and verbs, carry meaning with the word itself, but may have multiple senses. Lucia has four ways to choose between the multiple options available: resolution by design of the grammar, resolution by syntactic context, and immediate or delayed resolution of multiple senses.

##### *Case Study 4.3: Resolution by design of the grammar*

Many function words have meanings that vary depending on the syntactic context. For example, *up* can be a particle together with a verb as in *pick up*, or it can be a preposition. Various forms of *to be*, such as *is*, have many possible uses. When possible, Lucia has a single construction for a word defined in the grammar and resolves the meaning from the syntactic context of what phrasal

construction uses that word. This follows the principle in construction grammar theory that both words and larger constructions contribute to meaning (Goldberg, 1995). Consider some of the many uses of *is*:

B-001	The sphere <i>is</i> green. object-description(010, predicate(green1))
B-007	The red triangle <i>is</i> on the stove. object-description(07, on1(L3))
B-042	Go until there <i>is</i> a doorway. command(op_go-to-location1, until(object(shape(door))))
B-044	Octagon <i>is</i> a shape. adjective-definition(word(octagon), property(shape))
B-078	<i>Is</i> the green sphere on the table? object-question(010, on1(L6))

*Is* can declare an object property (B-001) or a relation (B-007). With *there*, *is* can declare the existence of something (B-042). *Is* can also define an adjective (B-044) or introduce a question (B-078). None of this information is derived from the lexical construction, but is added as phrasal constructions are recognized.

#### *Case Study 4.4: Resolution by syntactic context*

Another function word that has multiple senses is *that*. In this case the grammar defines three distinct lexical constructions that all use the same spelling but are of different types. These examples show abbreviated syntactic structures.

B-017	Put <i>that</i> in the pantry. ImperativeWithLocation[ TransitiveCommand[PutVerb[PUT], THAT-deictic], SimplePrepPhrase[in the pantry.] ] command(op_put-down1, 06, in1(L2))
B-028	Pick the green block <i>that</i> <i>is</i> small. TransitiveCommand[PickVerb[PICK], RefExprRelClause[SpecPropNoun[the green block], RelativeClauseProperty[ HeadRelativeClause[THAT-relative, IS], SMALL] ]] object-description(07, on1(L3))
B-145	The goal <i>is</i> <i>that</i> the box <i>is</i> in the office. ConceptIsThatDeclarative[ ConceptIsThat[SpecNoun[The goal], IS[is], THAT-complementizer], RefIsPrepPhrase[the box <i>is</i> in the office.] ] object-description(concept(goal), subclause(action(is1), 013, in1(L9)))

In B-017, *that* follows a verb as its object, causing THAT-deictic to be selected and grounded to the object currently pointed to. In B-028 *that* follows a

RefExpr, and THAT-relative is selected to head a relative clause. B-145 uses a ConceptIsThat construction, forcing the selection of THAT-complementizer to connect *The goal* is to the complement clause.

#### *Case Study 4.5: Immediate resolution of multiple senses*

Content words often have multiple senses, with context needed to select from among them. In these cases, the grammar defines two or more alternative lexical constructions. A phrasal construction that matches one of them chooses that one and deletes the others, as in these cases:

B-058	The <i>sphere</i> is <i>red</i> . RefIsProperty[SpecNoun[THE, SPHERE-noun], IS, RED] object-description(035, predicate(color(red1)))
B-048	Where is the <i>red triangle</i> ? WheresWaldo[WHERE, IS, SpecPropNoun[THE, RED, TRIANGLE-noun] ] where-is-question(051)
B-077	Is this a <i>sphere</i> ? IsObjectPropSetQ[IS, THIS, Property1Set[A, SPHERE-class] ] object-question(06, predicate(shape(sphere1)))

These three sentences show different senses for both *sphere* and *red*. *Sphere* has two senses, a noun and a class name. The noun sense is recognized by a SpecNoun construction in B-058, while *a sphere* in B-077 is matched by a Property1Set construction that uses the class sense, discarding the noun. In both B-058 and B-048, *red* is matched as a property, but in B-058 it is applied to the sphere by *is*, while in B-048 it is used as an adjective to modify triangle.

#### *Case Study 4.6: Delayed resolution of multiple senses*

The word *square* can be a property to be applied, a noun, or an adjective, as these sentences (not in our corpora) illustrate.

4.6a	This is a <i>square</i> . ThisIsAThat[THIS, IS, Property1Set[A, SQUARE-class] ] object-description(06, predicate(shape(square1)))
4.6b	Put the <i>square</i> in the <i>square</i> box. ImperativeWithLocation[ TransitiveCommand[PutVerb[PUT], SpecNoun[THE, SQUARE-noun]], SimplePrepPhrase[IN, SpecPropNoun[THE, SQUARE-adjective, BOX] ] command(op_put-down1, 046, in1(084))

All three senses are candidates each time. For a property application as in 4.6a and the noun in 4.6b, the property or noun sense is chosen as above. The second case in 4.6b is more complicated: during incremental processing of this instance of *square*, the noun will be chosen as before. When *box* is being processed, the system recognizes that the chosen sense is wrong, and a *local repair* is needed. The SpecNoun construction that recognized the second *the square* phrase is discarded. Next, the previously ignored adjective sense of *square* replaces the noun sense. Now the whole phrase *the square box* can be recognized, with the adjective sense of *square*. Many nouns can be used as adjectives like this.

The case of *square* as an adjective illustrates the delayed resolution strategy. In immediate resolution, other senses are not completely forgotten; they are linked to the chosen sense and can be brought back and selected in a later context. This is one kind of repair process that makes incremental parsing possible by allowing the comprehender to maintain only a single path in its comprehension state, yet still have enough information available to make a local repair when necessary. However, we shall see that there is a limit to how much delay is possible.

Some lexical ambiguities must be resolved by semantic rather than syntactic context. The meaning of *bank*, for example, depends on whether the semantic context is related to rivers, finances, airplanes, or billiards. Chapter 6 will show how we address this issue using an experimental version of Lucia.

### **4.6.2 Grammatical ambiguities**

Lucia uses one of several strategies when multiple composite constructions match a given comprehension state.

#### *Case Study 4.7: Selecting among competing composite constructions*

B-001	<i>The sphere is green.</i> object-description(010, predicate(color(green1)))
B-094	<i>This is a big triangle.</i> object-description(06,predicate(size(large1)),predicate(shape(triangle1)))

Both of these sentences have noun phrases. However, to produce grounded meaning Lucia must treat *The sphere* in B-001 as a reference to an object, while *a big triangle* in B-094 represents simply some predicates to describe the object referred to by *This*. Lucia has preference heuristics that select SpecNoun over Property1Set for *The sphere* and Property1Set over SpecPropNoun for *a big triangle* based on whether the determiner is definite or indefinite.

In a similar way, a heuristic selects from among BareNoun, SpecNoun, and SpecPropNoun for the phrase *the green block* in cycle 6 of Figure 4-5 by preferring a construction that spans more constituents. In the discussion of Figure 4-7, we described how a construction like *TheKitchen* that is specific to certain lexical items is preferred over SpecNoun that is much more general.

#### **4.6.3 Structural ambiguities**

Often the immediate context suggests one way of integrating a word into the ongoing parse, but later on that decision turns out to be wrong, as in *the square box* where the word *square* should be used as an adjective and not a noun. At the phrasal level, of particular importance are the attachment of prepositional phrases and relative or subordinate clauses. Lucia implements a strategy of *local repair*, modeled after that used by Lewis (1993), to resolve these ambiguities, as the following examples show.

##### *Case Study 4.8: Local repairs for phrase and clause attachment*

B-020	Pick up the green block <i>on the stove</i> . command(op_pick-up1, 015)
B-030	Pick the green block <i>that is on the stove</i> . command(op_pick-up1, 015)
B-015	Put the green sphere <i>in the pantry</i> . command(op_put-down1, 050, in1(068))

We have discussed previously how a local repair is needed for B-020 to attach *on the stove* to *the green block* in order to get an unambiguous grounding. Figure 4-9 sketches the steps for performing this repair, derived from Figure 4-5.

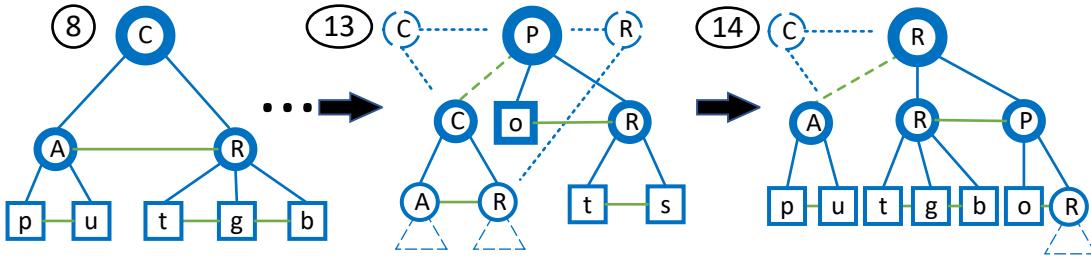


Figure 4-9: A local repair

At frame 8 the incremental processing has built what appears to be a complete TransitiveCommand sentence for *Pick up the green block*. However, by frame 13 the phrase *on the stove* has been added, and the previous command is below it on the stack. At this point there are two candidate composite constructions: an ImperativeWithLocation that would compose the C and the P into a new command sentence, and a RefExprPrepPhrase that would compose the R for *the green block*, which is at level 3 in the tree, with the new P. A heuristic based on the type of verb in the A node selects the R alternative. When this R is composed and integrated with the tree, the C node built in frame 8 is discarded, and the A node moves up to be right under the new R in the stack. Now the A and the R can be composed into a new C, which completes the sentence.

This form of local repair is used by Lucia for many sentences in the Rosie corpora. It shows two important aspects of the Lucia model. First, during the selection phase leading to cycle 14, the R candidate, a RefExprPrepPhrase construction, “reaches down” in the tree to find the RefExpr for *the green block*, even though this node is not on the stack. This is something not considered possible in many symbolic parsing algorithms. Second, this selection of the new R only requires reaching down to level 3 of the tree to consider the R node at that level. When a local repair would have to reach down to level 4, that repair fails, consistent with our theory that three and only three levels of the tree need to be accessible for selection of composite constructions.

Continuing with our case study, B-030 shows that when *that is* added to convert the reduced relative clause into an unreduced one, a similar repair is still needed to build a RefExprRelClause after the relative clause has been

constructed, since *Pick the green block* is first processed as a complete sentence. The logic is different for B-015, since the verb *Put* requires the prepositional phrase to give it a target location for the action. In this case, *the green sphere* is not semantically ambiguous, and a different heuristic selects the C alternative instead of the R in a situation analogous to frame 13 in Figure 4-9.

#### 4.6.4 Semantic ambiguities

We have mentioned the importance of semantic precision for achieving E3C, and that the type and compositional hierarchies of ECG give us tools for resolving semantic ambiguities. Prepositions give an example of this effect.

##### *Case Study 4.9: Resolving the semantics of prepositions*

B-054	Go <i>to</i> the kitchen. command(op_go-to-location1, to1(L13))
B-099	Go <i>down</i> the hall. command(op_go-to-location1, 046)

Most generative grammar approaches produce the same exact grammatical structure for both of these sentences. Such an approach fails in an incremental semantic parse that must produce actionable meanings. The final messages that are to be sent to the robot for these two sentences are different. For B-054, the message specifies a specific waypoint as the goal of the *go* action, whereas for B-099 no specific goal is given, just an object representing the hall to guide the motion. Consider a number of other possible prepositions that could have appeared in one of these sentences: *across*, *along*, *around*, *behind*, *in*, *into*, *out of*, *past*, *through*, *to the left of*, and so on. Some of these would work perfectly well in one of the sentences while making the other infelicitous<sup>7</sup>.

In B-054, *to* is treated as an ordinary preposition. For *down* in B-099, we created a construction that can only be a constituent of a corresponding special subcase of a prepositional phrase. This type of construction provides an

---

<sup>7</sup> Linguists use the term *infelicitous* to describe a sentence which is syntactically correct but does not make sense semantically.

alternative parse, which depends on the particular preposition involved, ultimately producing a different meaning structure. This is an example of how grammatical constructions, not just lexical items, carry meaning in construction grammar theory, as Goldberg (1995) insists.

#### **4.6.5 Grounding ambiguities**

In Chapter 2 we discussed how referring expressions can be grounded directly to the World Model, grounded hypothetically, or grounded dynamically. For grounding to the World Model, the language used determines whether a unique result is possible. In general, Lucia depends on Rosie's instructor to use the language necessary to produce unique grounding, as illustrated by the sentence B-020 we discussed earlier. When the language given leaves remaining ambiguity in the grounding for a whole sentence, Lucia does not have a mechanism to resolve this. The best it can do is report all the options to Rosie, and let Rosie either use task and environmental knowledge to figure out the resolution or ask the instructor to clarify. Hypothetical and dynamic grounding do not have the same ambiguity issues, but they do present other issues that are discussed in Chapter 5.

#### **4.7 Evaluation**

Chapter 2 provided evidence that Lucia can in fact produce correct messages for sentences in the Rosie corpora, thus performing E3C. Chapter 3 presented the grammar used to do this, thus implementing CKM. This chapter has described the dynamic, incremental algorithm that is used to apply the grammar to the sentences to produce correct output messages. Together, these chapters show that Lucia's processing algorithm works correctly in this sense. The other thing to evaluate here is how well it does I3P.

As Figure 4-2 shows, Lucia's processing is incremental, word-by-word. It also shows construction cycles within the processing of each word, each one of which grounds, when possible, an individual word or phrase to Rosie's world knowledge. This incremental grounding works, since the messages to Rosie produced by Lucia are correct in that they enable Rosie to learn and perform

tasks as intended by the instructor. Grounding is an essential part of this correctness, and as this chapter shows it is done incrementally and with immediate interpretation as needed in every construction cycle.

This shows that the algorithm does incremental, immediate interpretation processing (I3P). However, at this stage of development it does not provide detailed predictions of timings in human processing or the relation of the model's workings to human brain data. In Chapter 7 we examine research on detailed measures of human comprehension and how the Lucia model might serve as a basis of future research on modeling those more detailed aspects of human processing.

## **Chapter 5 Lucia in a Cognitive Architecture**

We have seen conceptually how Lucia performs embodied, end-to-end comprehension (E3C) in Rosie using a compositional knowledge of meaning (CKM) in the form of a large ECG grammar with incremental, immediate interpretation processing (I3P). The next phase of our journey is to see how all this is implemented using general cognitive mechanisms in a cognitive architecture. Underlying these architectures is the hypothesized *cognitive architecture principle* (Lehman et al., 1996):

$$\text{ARCHITECTURE} + \text{KNOWLEDGE} = \text{BEHAVIOR}$$

which implies that humans achieve complex behaviors such as language comprehension by applying general cognitive mechanisms to the knowledge needed for a particular task. Applying this principle to sentence comprehension, this complex behavior should arise from an architecture applied to knowledge relevant to that behavior. For Lucia, the behavior desired is E3C in Rosie, and knowledge involved is the ECG grammar described in Chapter 3 and the I3P algorithm described in Chapter 4 (Lindes, 2018).

Based on this idea, this chapter addresses the following principle of the Lucia theory:

**GCM: General Cognitive Mechanisms** – *Comprehension processing is done using domain-general mechanisms of cognition. In practice, this implies that our computational cognitive model will be built using a cognitive architecture.*

This project has chosen to adopt the Soar cognitive architecture:

**Commitment 4:** *Lucia is built within the Soar cognitive architecture.*

The Soar architecture is described briefly below. Its key features are an unbounded working memory, a long-term procedural memory that stores

production rules, long-term declarative memories for storing facts (smem) and previous experiences (epmem), and a decision cycle that applies procedures to the memories in a way that implements the problem-space computational model (PSCM Laird, 2012). Newell (1990) provides extensive arguments on how these mechanisms can produce a variety of human behaviors documented in the empirical psychological literature. For our work we simply assume that Soar is an adequate representation of the general cognitive mechanisms we need.

Given that assumption, our problem here is to show how Lucia's linguistic and processing knowledge can be represented and applied in Soar to produce the desired E3C behavior. The following section describes how we have solved this problem to achieve the performance already documented. In the evaluation section we describe what is working well and what are some of the limitations of the current implementation. Chapter 6 examines a further analysis of the structure and use of working memory, an analysis of how this model might explain certain limitations of human comprehension ability, and experiments how limitations of the model might be overcome.

This chapter describes how to use the mechanisms of an architecture that models cognitive mechanisms that are both general and plausible to exist in the human mind and brain to represent and apply the knowledge described to produce the desired sentence comprehension behavior. We first review the relevant literature on cognitive architectures, and related work on applying these architectures to language comprehension. Then we give a detailed explanation of how Lucia uses the Soar cognitive architecture to achieve E3C in Rosie using the ECG grammar from Chapter 3 and the I3P algorithm from Chapter 4. Since ambiguity is such a challenge for incremental language processing, we then examine how Lucia models resolving ambiguities and processing sentences that are difficult for humans. Finally, we evaluate this use of the architecture.

## **5.1 Cognitive Architectures**

An important line of AI research for several decades has been cognitive architectures. A cognitive architecture defines a set of memories, processing

mechanisms, and learning mechanisms designed to model general human cognitive abilities. The Human Associative Memory (HAM; Anderson & Bower, 1973) was an early attempt to model human memory. A series of successors (Ritter et al., 2019) adding procedural memory and other capabilities has culminated in the Adaptive Control of Thought-Rational architecture (ACT-R; Anderson, 2007) that has been widely used for modeling human behavior. In a similar approach, inspired by earlier work by Newell and Simon (Laird & Rosenbloom, 1996), an architecture called Soar (Laird, 2012; Newell, 1990) was developed with different initial goals and some variations in structure. Kotseruba and Tsotsos (2020) survey 84 cognitive architectures, 49 still being developed actively. We have chosen to use Soar for our implementation of Lucia and Rosie, so we concentrate on Soar with some comparisons to ACT-R (Laird, 2021).

### **5.1.1 The Common Model of Cognition**

Newell (1990) and Anderson (2007) develop theoretical arguments for how their architectures, Soar and ACT-R respectively, model general human cognition. Laird, Labiere, and Rosenbloom (2017) merged and abstracted key features of these two architectures, as well as a new one called Sigma (Rosenbloom et al., 2016), to form the Common Model of Cognition (CMC; Laird, Lebiere, et al., 2017).

The CMC<sup>8</sup> is a fairly simple model that serves as a good entry into this field. Figure 5-1, adapted from Laird et al. (2017), shows the principal components of the model. At the center is Working Memory (WM), which holds short-term knowledge of the current state of the agent and its interactions with the world. Knowledge in long-term procedural memory consists of production rules, which are IF-THEN rules. When a rule's left-hand-side, the IF part, matches the current state of WM, that rule fires, and the actions defined by the right-hand-side, the THEN side, make changes to WM. The architecture brings

---

<sup>8</sup> This model was originally called the Standard Model of the Mind by Laird et al. (2017), but subsequently a community consensus has renamed it the Common Model of Cognition.

perceptual information into WM automatically, and the firing of rules puts instructions into WM that initiate retrievals from long-term declarative memory or actions to be performed by the motor component.

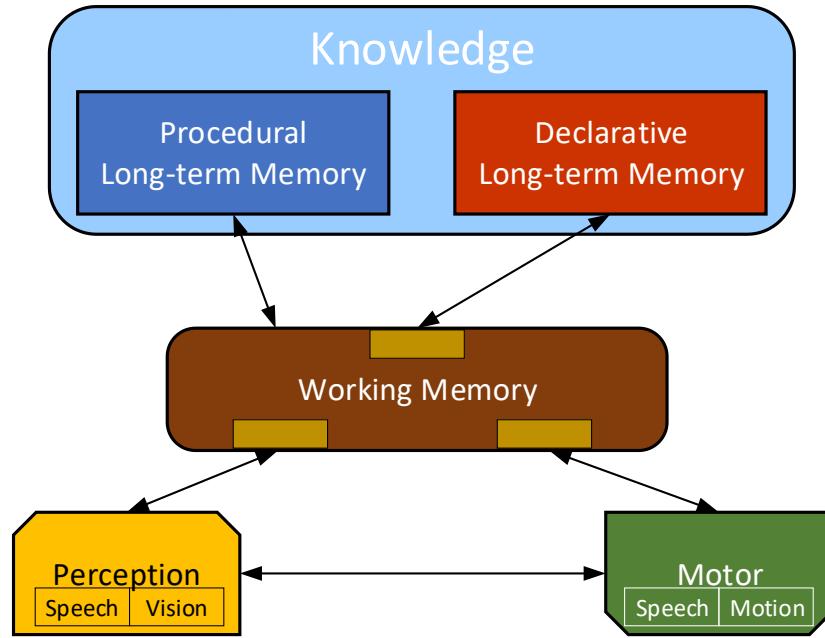


Figure 5-1: General cognitive mechanisms in the CMC (from Laird, Lebiere, et al., 2017)

The figure shows a static diagram of the model. The dynamics come from the firing of rules, changes in perception, and retrievals from long-term memory, which take place in what is called the *cognitive cycle*. Research shows (Anderson, 2007; Newell, 1990) that, in humans, the duration of this cycle is approximately 50ms. Complex behavior arises from applying knowledge over many cognitive cycles.

### 5.1.2 The Soar Cognitive Architecture

Soar has a more complex structure as shown in Figure 5-2 (Laird, 2012). In Soar, the symbolic Working Memory (WM) is made up of working memory elements (WMEs), each of which has an identifier, a named attribute and a value, which may be a constant or another identifier. Typically, a single identifier will have a number of WMEs attached to it, thus forming a graph structure. In Soar the

number of identifiers and WMEs is unbounded. WM is also organized according to a hierarchy of states which can change dynamically.

Soar's procedural long-term memory contains production rules. A rule fires when its left-hand-side (LHS) matches the state of WM, and then it makes changes to WM according to its right-hand-side (RHS). Soar's declarative long-term memory is divided into two parts: semantic memory (smem), which holds facts, and episodic memory (epmem), which holds a record of the history of episodes of processing.

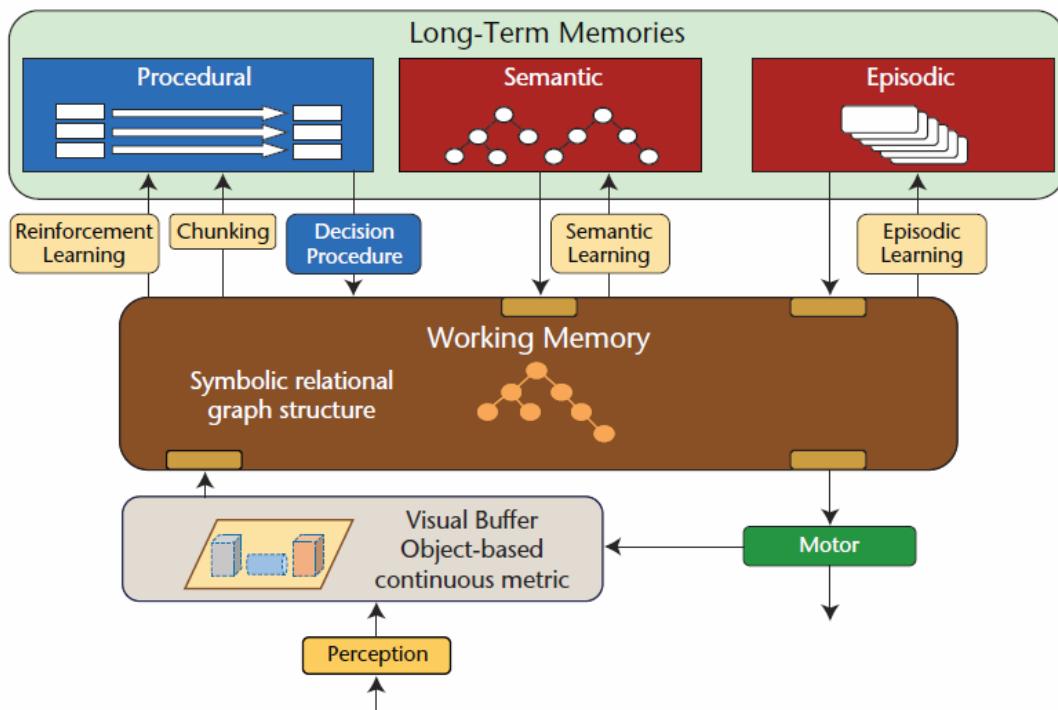


Figure 5-2: The Soar Cognitive Architecture (from Laird et al., 2017)

Soar is designed around an idea called the *problem space computational model* (PSCM; Laird, 2012), which grew out of earlier work the General Problem Solver (GPS; Newell & Simon, 1961). Computation is organized around *problem spaces* that “provide a framework for organizing knowledge of operators, states, and control knowledge.” (Laird & Rosenbloom, 1996, p. 5). In each state in Soar, an operator can be selected and applied, changing the system to a new state.

The PSCM leads to a cognitive cycle, which in Soar is called a *decision cycle* (DC). During each cycle an *operator* is selected and applied, often involving many rule firings. If the knowledge encoded in rules is insufficient to select or apply an operator, an impasse is reached and a new substate is created. This cycle has a number of phases, as shown in Figure 5-3.



Figure 5-3: The Soar Decision Cycle

In the Input (I) phase, external input is added to a special location in WM called the *input link*. In the Proposal (P) phase, rules fire to propose operators. There may be several operators proposed, and in the Decision (D) phase, preference rules and architectural mechanisms select a single operator to be applied in that cycle. This is an important parallel-to-serial decision point. In the Apply (A) phase, rules that have knowledge of how to apply the selected operator fire, possibly in multiple parallel waves of multiple rules per wave. In the Output (O) phase, special locations in WM send commands to the long-term declarative memories to initiate retrievals and/or to the Action module to produce motor actions. The cycle gathers input from the world, uses parallel processing to both propose operators and apply the operator selected, and may send commands to long-term memories or output to the world. If operator selection or application fails for whatever reason, an impasse is detected and substate is created.

### 5.1.3 ACT-R

The Lucia theory has not been implemented in ACT-R (Anderson, 2007; Bothell, 2021), but we present an overview of that architecture because we will compare comprehension theories and use of working memory between ACT-R and Soar. Figure 5-4 (Laird, Lebiere, et al., 2017) shows the general structure of ACT-R, including theories of how its various modules map onto regions of the brain. Rather than having working memory in the middle surrounded by other modules, this view has the procedural module, which includes long-term

procedural memory, in the center surrounded by buffers that connect it with various other modules. In effect, the buffers are the working memory of ACT-R, since production rules can only match against and change the contents of these buffers. This differs from Soar's unbounded working memory.

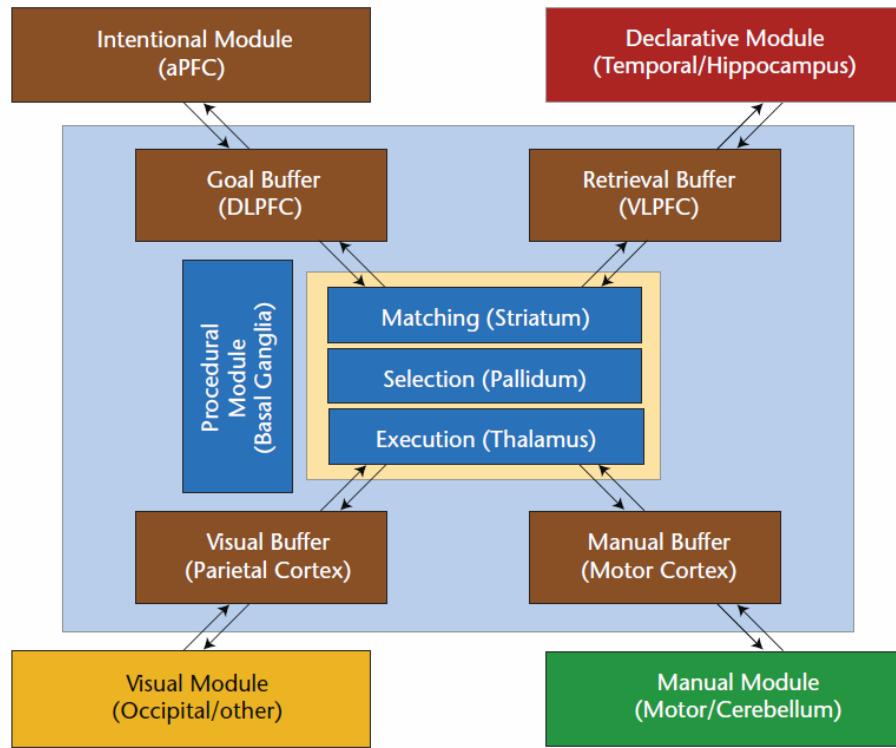


Figure 5-4: The ACT-R Cognitive Architecture (from Laird, Lebiere, et al., 2017)

Another key difference between ACT-R and Soar is how the cognitive cycle works. There are no operators in ACT-R. The Procedural Module simply selects a single production rule from its long-term procedural memory that matches the contents of the buffers at each cycle, and the selected rule modifies the contents of the buffers. Goals are represented in the Goal Buffer, which can affect the flow of control like any buffer, but there is nothing that corresponds to Soar's impasses and state hierarchy. Even though the diagrams for Soar and ACT-R differ in how they organize the modules, the CMC makes visible the important commonalities between these architectures.

## **5.2 Language Comprehension in Cognitive Architectures**

In Chapter 1 we examined a number of AI systems and compared them to Lucia on each of the main dimensions of our theory. Of these, there are four that use cognitive architectures comparable to the CMC, one in Soar and three in ACT-R. (We ignore here the LEIA system (McShane & Nirenburg, 2021) since it does not have a comparable architecture.) We look briefly here at how each of these uses general cognitive mechanisms for language processing.

Allen Newell (1990) suggested an approach to language comprehension using his theory of unified theories of cognition in Chapter 8 of his book. Others developed this theory further, calling it NL-Soar. Richard L. Lewis (1993) describes using Soar for parsing, including an extensive study of modeling garden-path and parsing breakdown effects. His algorithm is based on generative grammar and X-bar theory, uses Soar's chunking mechanism to learn procedural knowledge for parsing in real time, and gives a theoretical account of immediate interpretation (I3P) without fully implementing it. Shortcomings of NL-Soar include that it is not embodied and does not do true end-to-end comprehension (E3C) in that it does not create a representation used to produce internal or external actions by an agent. Although X-bar theory is a composable theory for syntax, NL-Soar does not have a composable model of knowledge of meaning (CKM) similar to ECG, nor does it include a theory of language acquisition from experience (LAE).

Lewis and Vasishth (2005; henceforth LV05) pursued a different approach for GCM by using ACT-R. As before, their new approach does not do E3C, use CKM, or consider LAE. It does employ incremental processing, but without grounding to the knowledge of an embodied agent (I3P). At the center of this model is the use of cue-based retrievals from ACT-R's declarative memory, and how these are affected by activation levels and interference. ACT-R's theory includes formulas for estimating retrieval times from long-term memory, and the LV05 model uses these to make predictions about human reading times. The paper reports success in comparing model results to empirical data on five

experiments, with an  $R^2$  ranging from 0.76 to 0.91 for different data sets of human reading times.

Jones (2020) points out what he sees as several deficiencies in LV05 and similar models. ACT-R only allows production rules to access its small number of buffers, and claims that Lewis and Vasishth “assume extra working memory capacity in the form of overt or hidden buffers … linked to … a set of assumed phrasal categories.” He also criticizes previous models for using only binary branching in their tree structures, a weak association between syntactic structure and meaning, and unrealistic approaches to resolving ambiguities. He then presents his model, which itself adds three additional buffers and a “multibuffer” to ACT-R. Some strengths of his model are that it uses basically the same algorithm to process sentences in English and Korean, and its approach to disambiguation, which includes using prosodics. Jones’s (2020) approach does not include an approach to achieving E3C, I3P, or LAE.

Ball’s system (J. T. Ball, 2012) is implemented within a “synthetic teammate” (J. Ball et al., 2009; Demir et al., 2015), an agent that coordinates with two human operators to perform a simulated UAV reconnaissance mission. It does a form of E3C as it communicates via text with the human operators using a domain-specific subset of English, but does not actually ground to knowledge of the physical world. It is based on Ball’s (2004) Double-R theory of grammar and comprehension. Its grammar is similar to construction grammar, but the way in which it maps form to meaning is not grounded to the physical world. It does a form of incremental processing using retrievals from ACT-R’s declarative memory, but ACT-R is extended with several additional language-specific buffers (J. T. Ball, 2013). This is the most complete language comprehension system in ACT-R, since it is part of a larger agent that acts in its virtual world using language to communicate with humans.

NL-Soar and the three ACT-R systems all do sentence processing using a cognitive architecture; however, only Ball’s Double-R system (J. T. Ball, 2013; Demir et al., 2015) does a form of E3C. They all do incremental processing, but without immediate interpretation. All of the ACT-R models require significant

extensions to ACT-R's working memory. In Chapter 6 we analyze in some detail Lucia's use of working memory in comparison to these other systems in cognitive architectures.

### 5.3 Lucia's Implementation in Soar

This section describes the details of the representations and algorithms used to implement Lucia in Soar. Lucia is embedded in Rosie's Soar agent and uses its CKM, I3P algorithm knowledge, and Rosie's world knowledge to transform each input sentence into a grounded action message (E3C), as shown in Figure 5-5.

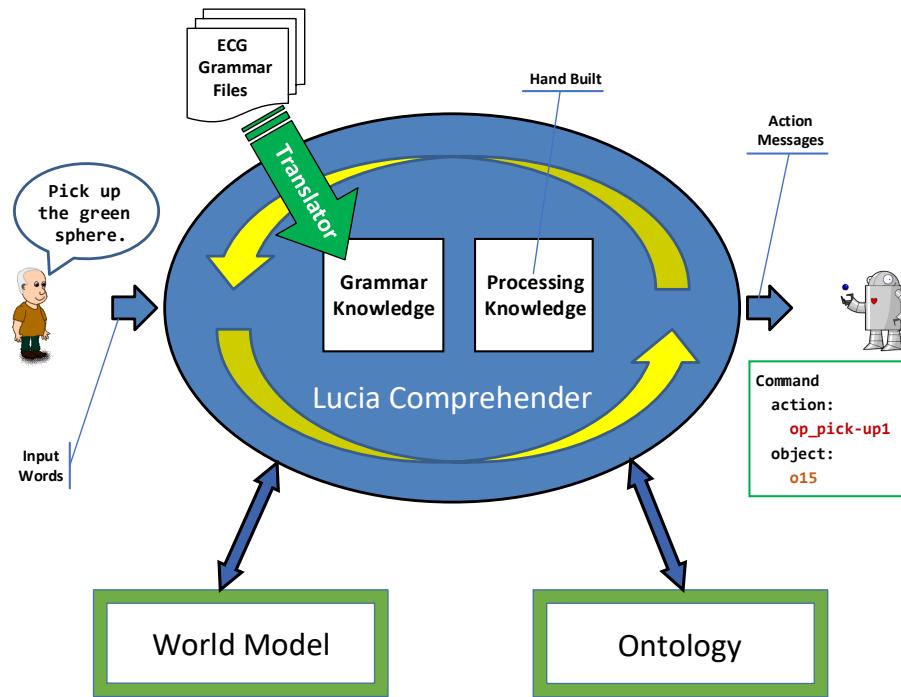


Figure 5-5: Lucia as embedded within Rosie

The comprehension state, not shown, resides in working memory. Grounding is done to Rosie's World Model in working memory and its Ontology in long-term declarative memory. Each sentence is processed incrementally, with immediate grounding. An action message for each sentence is added to Soar's WM for the rest of Rosie to act on. The grammar from Chapter 2 (CKM) is translated into Soar production rules. Each construction cycle uses operators to select, integrate, and ground a new construction. Many rules fire, often in parallel, during each decision cycle. Using this approach, Lucia simulates adult

skilled comprehension in simulated real time. Chapter 6 discusses an alternative version of Lucia, called System B, that uses a declarative representation of CKM that can facilitate LAE.

### 5.3.1 Knowledge of the meaning of language in Soar

Lucia's composable knowledge of meaning (CKM) originates in files containing constructions and schemas written in ECG's formal language. This "grammar" is translated into Soar knowledge by ECGtoSoar, a program shown in Figure 5-6. ECG ANTLR is the file defining the ECG language for the ANTLR parser (Parr, 2013), and Soar ST defines the syntax of Soar data.

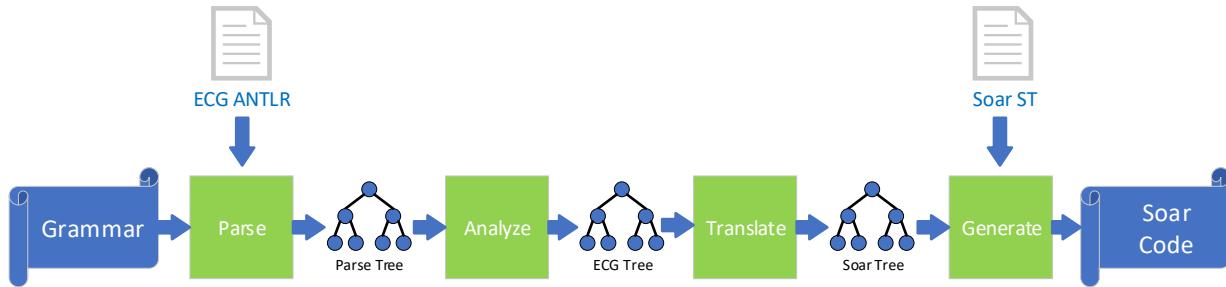


Figure 5-6: The ECGtoSoar translator

The Soar Code produced by ECGtoSoar for the standard version of Lucia is a set of Soar production rules. Lucia consists of these rules generated from ECG as well as hand-built rules that are, for the most part, not specific to any ECG item. The following subsections give detailed examples of the different kinds of rules generated by the translator from ECG, as well as representative hand-built rules.

### 5.3.2 Overall control structure

This subsection gives a high-level view of the flow of control in Lucia's I3P algorithm in Soar, with more detail in the following subsections. The control structure uses knowledge both from ECG and hand coding, and follows the pattern of Soar's PSCM theory defined for NL-Soar by Newell (1990) and used by Lewis (1993). Discussion of how ambiguity and local repairs follows below.

There is a top level **comprehend** operator that has no apply rules and thus creates an impasse and a substate in which all the I3P processing for a sentence is done. Each word cycle is made up of a **next-word** operator and a **comprehend-word** operator that also generates a substate. The construction cycles for processing an individual word all happen within the **comprehend-word** state for that word. An example sentence illustrates this control flow.

### *Case Study 5.1: Control flow for a simple sentence*

B-009	Pick up the green sphere. command(op_pick-up1, 010)
-------	--

Figure 5-7 shows parts of an abbreviated Soar trace for processing this sentence.

1: 0: 01 (init-agent) 2: 0: 02 (init-comprehender) 3: 0: 018 (comprehend) ... 6: 0: 020 (next-word) #1: Pick 7: 0: 021 (comprehend-word) ... 14: 0: 027 (next-word) #2: up 15: 0: 028 (comprehend-word) ... 22: 0: 034 (next-word) #3: the 23: 0: 035 (comprehend-word) ... 27: 0: 038 (next-word) #4: green 28: 0: 039 (comprehend-word) ... 34: 0: 044 (next-word) #5: sphere 35: 0: 045 (comprehend-word) ... 44: 0: 056 (comprehend-done) Sentence #1: "Pick up the green sphere." 45: 0: 057 (interpret) ... 49: 0: 060 (act) ... 53: 0: 063 (repeat)	Received word #1: Pick 7: 0: 021 (comprehend-word) 8: ==>S: S3 (operator no-change) 9: 0: 022 (lexical-access) 10: 0: 023 (match-construction) Matched a PickVerb construction. 11: 0: 025 (lookup-action) 12: 0: 026 (retrieve-item) 13: 0: 024 (comprehend-word-done)
	Received word #5: sphere 35: 0: 045 (comprehend-word) 36: ==>S: S7 (operator no-change) 37: 0: 046 (lexical-access) 38: 0: 047 (lookup-property) 39: 0: 053 (retrieve-item) Prefer SpecPropNoun(3) > BareNoun(1). 40: 0: 050 (match-construction) Deleting the lexical option C30. Matched a SpecPropNoun construction. 41: 0: 054 (ground-reference) 42: 0: 055 (match-construction) Matched a TransitiveCommand construction. 43: 0: 052 (comprehend-word-done)

Figure 5-7: Soar trace for a simple sentence

The left-hand side shows the top-level processing of the complete sentence. The numbers at the left of each processing line are sequence numbers of the decision cycles, and the ellipses mark gaps where details have been elided. The “O:” marks indicate the operator selected in each decision cycle (DC), and indentation of these indicates the level of substates involved. Operators O18 and O56, at DC’s 3 and 44, bracket the comprehend state for this sentence.

After DC 44, there are three more top-level operators, `interpret`, `act`, and `repeat`. `Interpret` and `act` implement the sentence interpretation process to produce the message structure created by Lucia, and `repeat` resets the system to be ready to process the next sentence.

The right-hand side of the figure shows details of the `comprehend-word` substates for the words *Pick* and *sphere*. Here we describe the basic function of each operator, and the following subsections give details of the rules used to propose and apply these operators for each phase of the construction cycle.

Each lexical-access operator uses ECG knowledge to perform the selection and integration phases of the construction cycle for a new input word. The match-construction operators perform these two phases of each composite construction cycle. The `lookup-xx` and `retrieve-item` operators perform grounding for concepts that are grounded to Rosie’s Ontology. A ground-reference operator grounds a referring expression to an object in Rosie’s World Model. If none are found, a `create-new-object` operator builds a new object structure and stores it in the World Model. This is not needed in this particular example. The remaining decision cycles for `comprehend-word`, `operator-no-change`, and `comprehend-word-done` perform the overhead functions needed to create and resolve a Soar impasse to implement PSCM processing.

The functioning of Lucia involves parallel processing punctuated with serial selection decisions at several levels. The Soar architecture provides for multiple parallel operator proposals and a selection procedure involving preference rules to select a single operator for each decision cycle. Lucia uses this mechanism to select among multiple match-construction candidates

proposed for composite constructions. Multiple lexical items can be proposed in parallel on the Soar state with apply rules for lexical-access, multiple operators are proposed to process them, and preference rules choose among them based on the syntactic context.

During the apply phase of a typical decision cycle for lexical-access or match-construction many rules fire. The ECGtoSoar translator builds the knowledge from ECG into construction-specific apply rules for lexical-access, proposal and apply rules for match-construction, and rules to construct ECG schemas and populate them. Other hand-built rules that are not construction-specific perform generalization, grounding, and other house-keeping functions. Rules fire in a series of parallel waves, as shown in Figure 5-8 for DC 42 in Figure 5-7 where the TransitiveCommand is built.

```

42:      0: 055 (match-construction)
--- apply phase ---
--- Firing Productions (PE) For State At Depth 3 ---
Firing comprehend-word*apply*match-construction*TransitiveCommand
Matched a TransitiveCommand construction.
--- Firing Productions (PE) For State At Depth 3 ---
Firing comprehend-word*apply*match-construction*add-source-and-text*2
Firing comprehend-word*generalize-cxn*Imperative*simple
Firing comprehend-word*evoked-schema*ActOnIt*create
--- Firing Productions (PE) For State At Depth 3 ---
Firing comprehend-word*evoked-schema*Action*exists
Firing comprehend-word*TransitiveCommand-constraint*self-m-object*UNIFY*object-m
Firing comprehend-word*generalize-cxn*VerbWithArguments*simple
--- Firing Productions (PE) For State At Depth 3 ---
Firing comprehend-word*apply*match-construction*all-done*clear-ref-resolved
Firing comprehend-word*TransitiveCommand-constraint*self-m-action*UNIFY*verb-m
--- Firing Productions (IE) For State At Depth 3 ---
Retracting comprehend-word*propose*match-construction*TransitiveCommand
--- output phase ---
--- input phase ---
--- propose phase ---
--- decision phase ---
43:      0: 052 (comprehend-word-done)

```

Figure 5-8: Inside a match-construction decision cycle

The trace in this figure is abbreviated, but it shows multiple parallel waves of rule firings, all in the apply phase. Every line starting with “--- Firing

“Productions” marks the beginning of a new wave, and the rules under it fire before the next wave and make changes to working memory in parallel. The `*apply*match-construction*TransitiveCommand` rule instantiates the construction and sets it as the new root of the comprehension state in the first wave. The next wave generalizes it to an Imperative, creates an evoked `ActOnIt` schema, and annotates the node with text information. The next wave generalizes again to `VerbWithArguments`, sets the value of the object role of `ActOnIt`, and generalizes this schema to an Action. Then another wave sets the value of the action role of Action and completes the integration process.

*Case Study 5.2: Control flow for a sentence with local repair*

B-020	Pick up the green block on the stove. <code>command(op_pick-up1, 015)</code>
-------	---

In Chapter 4 we analyzed the processing of this sentence, including that when *block* is processed, a complete sentence construction is built, which must be discarded later after *stove* by a local repair to build a `RefExpr` for the combined phrase *the green block on the stove*. Figure 5-9 shows the sequence of constructions selected and their grounding to comprehend this example sentence. Operators that initiate and terminate the word cycle are not shown.

Each numbered rectangle in the figure represents the operation of a single Soar operator. The white rectangles on the left are new input words being acquired, and each is processed to form a lexical construction. These are then combined into composite constructions, with each horizontal line representing all the processing done for a single word.

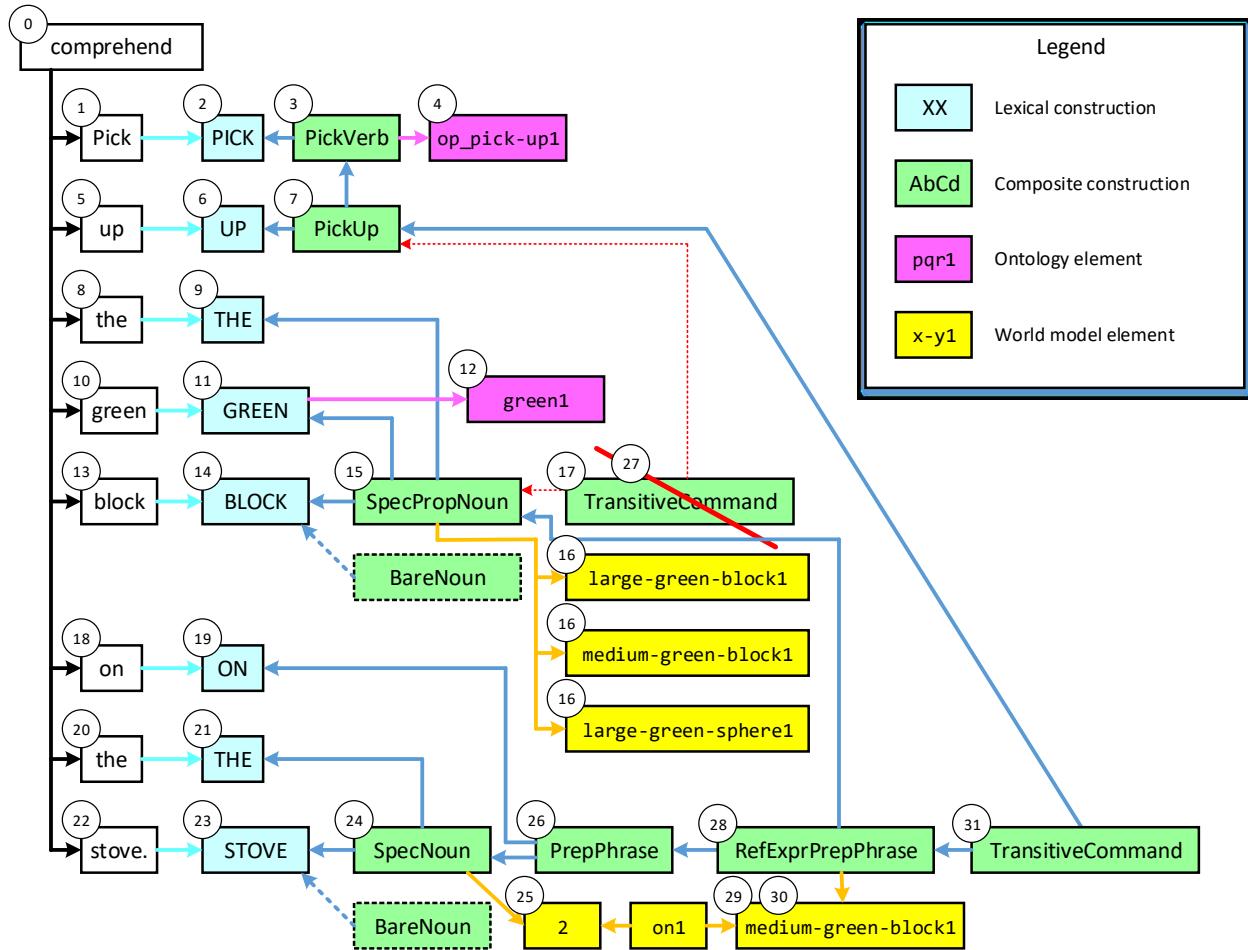


Figure 5-9: Construction cycles for a sentence with local repair

At 15 and 24 we see examples of competing constructions, with the dotted one being rejected, as described below for the selection phase. At 27 a local repair deletes an already built construction to allow the prepositional phrase to be attached to a referring expression, with a new TransitiveCommand finally being built at 31. Operator 16 finds three objects in the world model that fit the green block, but 25, 29, and 30 use *on the stove* to select the one that fits the whole sentence. The constructions are shown with their most specific identity. Their generalizations, for instance that both SpecNoun and RefExprPrePhrase are subcases of RefExpr, are not shown here.

### 5.3.3 The selection phase

This subsection and the following two, respectively, provide detail on how the Soar rules generated by ECGtoSoar, as well as the related hand-built rules, perform the processing for the three phases of the construction cycle: selection, integration, and grounding. The details given are with respect to this sentence:

*Case Study 5.3: A simple command*

B-054	Go to the kitchen. command(op_go-to-location1, to1(L28))
-------	---

Each construction cycle adds a new node to the comprehension state as described in Chapter 4, so a decision must be made about which of the many available constructions should be selected for integration. The selection phase, which differs for lexical and composite constructions, selects a single construction to add next and creates a skeleton instance of it on the Soar state. The integration phase connects this node with the comprehension state and fills it out.

*Lexical selection*

A lexical-access operator is proposed and selected at the beginning of each word cycle. Lexical constructions are selected in two steps. First, each lexical construction in the grammar has an apply rule for lexical-access created by ECGtoSoar. Each of these fires to propose a candidate lexical construction if the current input word has the orthography (spelling) compiled into that rule, and puts a skeleton of a new comprehension node on the Soar state. In some cases, multiple candidates with the same orthography but different meanings are created, such as the multiple senses of *that* discussed in Chapter 4. The lexical construction for *go* and the apply rule generated from ECG for it look like this:

<pre>construction GO   subcase of DriveVerb   form   constraints     self.f.orth &lt;- "go"     self.m.name &lt;-       "go-to-location1"</pre>	<pre>IF:   problem-space=comprehend-word   ∧ operator=lexical-access ∧ word("go") THEN:   +cxn(name(GO), is.a(GO),         +m(&lt;meaning&gt;),         subcase-of(DriveVerb))</pre>
---	--

The ECG is on the left and the rule is on the right. Under IF is a list of tests that must all be satisfied for the rule to fire. Under THEN is a list of items to be added to or deleted from working memory. A “+” indicates that a WME is added to the Soar state with the additional information shown. Every lexical construction produces an apply rule like the one shown for GO. For a word with multiple senses, multiple cxn nodes are built in parallel.

### *Composite selection*

ECGtoSoar generates a `match-construction propose` rule for each composite construction where the constituent types of that construction match the items in the top levels of the comprehension state. For example:

<pre> construction TheKitchen   subcase of SpecifierNP     constructional     constituents       spec : THE       noun : KITCHEN   meaning: Kitchen   constraints     self.m.name &lt;--&gt;       noun.m.schema-name     self.m.category &lt;--&gt;       noun.m     self.m.givenness &lt;--&gt;       spec.m.givenness </pre>	<p>IF:</p> <pre> problem-space=comprehend-word ^ lexical-access(done) ^ cxn:&lt;noun&gt;(is.a(KITCHEN)   ^ previous:&lt;spec&gt;(is.a(THE)     ^ previous:&lt;previous&gt;)) </pre> <p>THEN:</p> <pre> propose match-construction(   type(construction),   cxn-name(TheKitchen),   noun(&lt;noun&gt;), spec(&lt;spec&gt;),   span(2), lexical(2),   previous(&lt;previous&gt;)) </pre>
---	--

The LHS of the rule looks for a `cxn` WME on the Soar state that is a KITCHEN construction, preceded by a THE construction. The notation `:<x>` indicates the attachment of a variable name  $x$  to this Soar node so that it can be referred to later in the same rule, usually on the RHS. In this rule, the RHS proposes a `match-construction` operator with parameters for the operator type, the construction name, its noun and spec constituents, and its previous link.

The `propose` rule for TheKitchen also adds two other important parameters, called `span` and `lexical`, to the operator it proposes. These are used by two general preference rules that determine the selection. The rule for `span` looks like this:

```

IF:
  match-construction( $c_1$ , span( $s_1$ )) ∧ match-construction( $c_2$ , span( $s_2$ ))
  ∧  $s_1 > s_2$ 
THEN:
  prefer match-construction( $c_1$ , span( $s_1$ ))

```

This rule says that if two `match-construction` operators have been proposed for two different constructions, and the `span` for the first is greater than the `span` for the second, then prefer the first over the second. This prefers the construction with the greater `span`, meaning the one that has more constituents and thus covers more of the sentence. This is a heuristic that forms an important part of Lucia's selection of composite constructions. A similar rule tests the `lexical` parameter of constructions and prefers one with more lexical constituents. This is a simple heuristic to prefer a construction that is more semantically precise.

Examples of the `span` preference are shown at DC 40 in Figure 5-7 and at items 15 and 24 in Figure 5-9. `SpecPropNoun` is preferred because it spans three items while `BareNoun` spans only 1. An example of the `lexical` preference is preferring `TheKitchen` over `SpecNoun` in the sentence `Go to the kitchen.`, as discussed for this example in Chapter 4.

### 5.3.4 The integration phase

Once a construction has been selected, a new node is constructed and integrated into the comprehension state. Chapter 4 gave a list of the steps involved in the integration phase of the construction cycle, which is essentially the same for both lexical and composite constructions. This subsection gives some details of how each of these steps is carried out, using some ECG-generated and some hand-built rules.

8. The selected construction is *instantiated* by building a data structure representing it, with the construction name as the bottom level of its type hierarchy.

This step is done by an ECG apply rule for each construction. For a lexical construction, the same apply rule for `lexical-access` that selected a

specific construction also builds a skeleton instance on the Soar state. For a composite construction, an apply rule for the specific `match-construction` operator that was selected builds a skeleton instance. A *skeleton* instance means a simple data structure with the name, base type, and constituents of the construction along with additional flags for triggering the later generalization and evocation steps of the integration phase. The `match-construction` apply rule for `TheKitchen` looks like this:

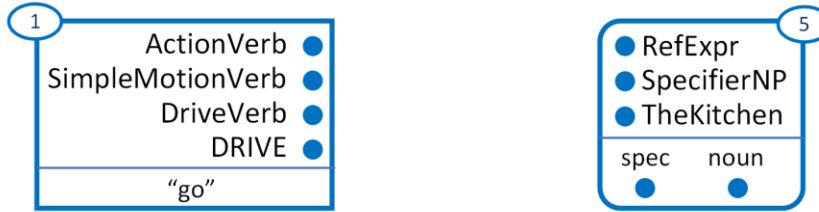
```
IF:
  problem-space=comprehend-word
  ^ operator=match-construction(
    cxn-name(TheKitchen), is.a(TheKitchen),
    noun:<noun>, spec:<spec>, previous:<previous>)
THEN:
  +cxn(name(TheKitchen), is.a(TheKitchen),
    noun(<noun>), spec(<spec>), subcase-of(SpecifierNP),
    +m:<meaning>, +evokes(schema(Kitchen), target(<meaning>)))
```

9. Based on subcase of specifications, the construction instance is *generalized* by labeling it with the names of all the general constructions in the sequence of subcases.

The following ECG rule applies to any new construction labeled as a subcase of `DriveVerb`, and generalizes the `GO` construction to a `DriveVerb`. It removes the `subcase-of(DriveVerb)` WME and, since `DriveVerb` is a subcase of `SimpleMotionVerb`, adds a new `subcase-of` label for that. The generalization rule for `SimpleMotionVerb` then fires to add that label. This rule also adds an `evokes` WME, which enables the firing of the rule for the `ActionDescriptor` schema.

<pre>general construction DriveVerb   subcase of SimpleMotionVerb   meaning: ActionDescriptor</pre>	<pre>IF:   problem-space=comprehend-word   ^ operator=lexical-access OR match-construction   ^ cxn(subcase-of(DriveVerb), m:&lt;meaning&gt;) THEN:   cxn(-subcase-of(DriveVerb),     +subcase-of(SimpleMotionVerb,     +evokes(schema(ActionDescriptor),       target(&lt;meaning&gt;)))</pre>
---	--

This and similar generalization rules are applied sequentially to generalize the nodes for the GO and TheKitchen constructions to look like this:



10. This new node is *attached* into the tree by making it the new root node.

This happens automatically in `comprehend-word` when the new node is built on the Soar state and then its previous link is connected as described in the following two steps. At the end of every word cycle the `comprehend-word-done` operator attaches the root of the tree to the Soar superstate for `comprehend`.

11. A new lexical node gets the previous root node as its *previous* node. The apply rule shown for the GO construction makes this *previous* link.
12. For a composite construction, its constituent nodes are *linked* as its children, and whatever was on the stack before its first child is linked as its *previous* node.  
The apply rule shown for TheKitchen makes these links.
13. Whatever meaning schemas specified by the construction are *evoked*, meaning that they are *instantiated* and the node is *linked* to them. The generalization rule for DriveVerb shown above creates an evokes link on the Soar state with a new node specifying that an ActionDescriptor schema should be created and attached to the `m:<meaning>` node. An ECG rule specific to this schema creates an instance of the schema, attaches it to the target WME, and removes the evokes link. The rule looks like this:

<pre> schema ActionDescriptor   roles     class     name     modifier </pre>	<p>IF:</p> <pre> problem-space=comprehend-word   ^ cxn(evokes:&lt;evokes&gt;(     schema(ActionDescriptor),     target(&lt;meaning&gt;))) </pre> <p>THEN:</p> <pre> cxn(-evokes(&lt;evokes&gt;),   +m:&lt;meaning&gt;(     schema-name(ActionDescriptor),     is.a(ActionDescriptor),     class(nil), name(nil), modifier(nil))) </pre>
--	---

14. The meaning schemas are *populated* by using the constraints specified in the ECG for the construction and the schemas involved.

The GO construction has two constraints:

```

self.f.orth <-- "go"
self.m.name <-- "go-to-location1"

```

These are called ASSIGN constraints since they assign a constant value to a slot in a schema. The ECGtoSoar translator creates a rule for each constraint, including this rule for the name constraint if GO:

<p>IF:</p> <pre> problem-space=comprehend-word   ^ cxn(is.a(GO),     m:&lt;meaning&gt;(name:&lt;target&gt;)) </pre>	<p>THEN:</p> <pre> cxn(-m.name(&lt;target&gt;),   +m.name(go-to-location1)) </pre>
---	--

The ECG for TheKitchen includes the following constraints:

```

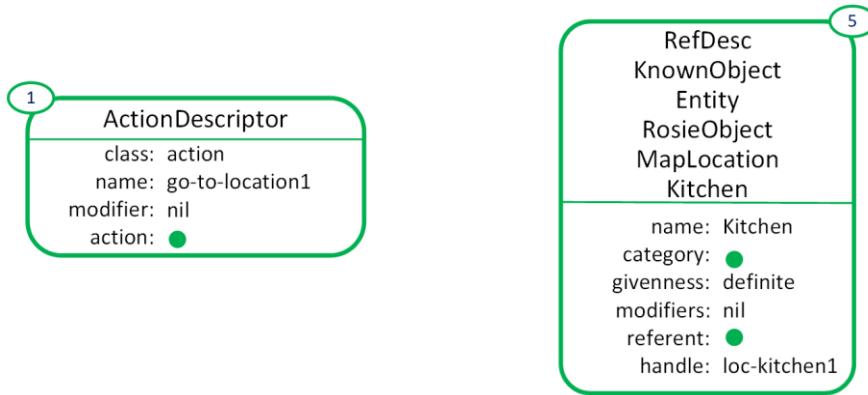
self.m.name <--> noun.m.schema-name
self.m.category <--> noun.m
self.m.givenness <--> spec.m.givenness

```

These are called UNIFY constraints since they populate a slot in one schema with a value or a pointer in an already existing schema. The ECG rule for the givenness constraint looks like this:

<p>IF:</p> <pre> problem-space=comprehend-word   ^ cxn(is.a(TheKitchen),     m:&lt;meaning&gt;(       givenness:&lt;target&gt;,       spec(m(givenness(&lt;value&gt;)))) </pre>	<p>THEN:</p> <pre> cxn(-m.givenness(&lt;target&gt;),   +m.givenness(&lt;value&gt;)) </pre>
---	--

After applying generalizations and constraints the schemas look like this:



### 5.3.5 The grounding phase

Performing immediate interpretation requires grounding each lexical or composite construction, except for some function words that help construct the syntactic structure without contributing separate semantics. Three basic types of grounding are possible: grounding to Rosie's Ontology, grounding to the World Model, or creating a new object. Different operators are used for each type. Grounding is always completed before going on to select another construction in a new construction cycle.

Most content words, such as nouns, adjectives, and verbs are grounded to Rosie's Ontology. These are grounded by one of several different `lookup-xx` operators, where `-xx` indicates the type of item being grounded. These operators initiate cue-based queries to semantic memory and are always followed by a `retrieve-item` operator to gather the result in WM. In Figure 5-7, the operators applied in DC's 11-12 find Rosie's representation of the action called *pick* or *pick up* in English, and operators applied in 38-39 find Rosie's definition of the shape named by *sphere*. Items 4 and 12 in Figure 5-9 also are created by retrievals.

Complete referring expressions, such as *the green sphere*, are grounded with a `ground-reference` operator, as shown at DC 41 in Figure 5-7. This operator gathers all the semantic information stored in the RefDesc schema built for that RefExpr construction, including knowledge obtained by retrieving the meanings of words like *green* and *sphere*, and tries to match that information against all the objects in Rosie's World Model. Any matching objects are posted as referents on the RefDesc. If there are multiple matches, usually the sentence

will have an additional prepositional phrase or relative clause after this RefExpr that will allow Lucia to resolve to a single referent. If this is not possible, as in Case Study 5.2:, further language input creates a larger RefExpr and the additional information resolves the problem. When this ambiguity is not resolved, Rosie can ask the instructor for clarification.

A third possibility is that none of the objects in the World Model match the semantics in the RefDesc. At this point, Lucia looks at Rosie's knowledge in smem about its environment, which includes things like a building map and long-term knowledge about the location of people and objects. If this fails, a `create-new-object` operator builds an object representation structure from the RefDesc and posts it into the World Model. Such an object is marked with a `^dialog-object true` flag and has a handle of `new-object-idn` where  $n$  is a unique integer. In some of our examples we abbreviate these id's to `NO-IDn`. These are hypothetical objects that Rosie uses, especially with Games, to ground later during task execution. See below.

So far, we have looked at grounding to Rosie's Ontology and World Model. Sentences in the Games corpus require different grounding. When learning games and puzzles, Rosie is not operating directly in a physical or simulated world but gathering information to develop the knowledge needed to solve the puzzle or play the game later. Thus, referring expressions have no World Model to ground to. These sentences are processed in what is called *hypothetical* mode.

#### *Case Study 5.4: Hypothetical grounding for Games*

G-034	If the volume of a block is more than the volume of an object then the block is larger than the object. conditional( if-subclause(action(is1), volume-of(NO-ID1), more-than(volume-of(NO-ID2))), then-subclause(action(is1), larger-than(NO-ID1, NO-ID2)))
-------	---

In hypothetical mode, Rosie learns a model of how to play the game or solve the puzzle based on goals, actions, failure states, etc. Then after the instruction is finished Rosie can proceed to solve the puzzle or play the game. When operating in Games mode, Rosie has a *hypothetical* flag set in its top level

which tells Lucia to create a new hypothetical object for any indefinite referring expression that would otherwise be grounded to the World Model, and these objects are stored in the World Model by `create-new-object` operators. A definite RefExpr later can be grounded to one of these. At the end of processing a sentence, these objects are deleted from the World Model.

In this example there are two indefinite referring expressions, *a block* and *an object*. In hypothetical mode each of these creates a new object, designated here by `N0-ID1` and `N0-ID2`. Later in the sentence the definite phrases *the block* and *the object* are grounded by ground-reference to these objects. In some conditional sentences for Games, such new objects are referred to later by *it*, and Lucia keeps track of the subject of a previous subordinate clause to successfully resolve these anaphoric references.

#### *Case Study 5.5: Dynamic grounding for Robot tasks*

The grounding process for sentences in the Robot corpus is complex because, as the agent moves around in the environment, objects move in and out of its visual perception and may or may not be remembered from prior experience. As part of Rosie's operation, it maintains a world model of not only what it currently senses, but also what it has recently perceived, as well as having access to episodic and semantic memory to provide information about objects and location outside the current environment. Here we look at a few examples that illustrate the sort of issues involved.

R-030	Find the fork. <code>command(op_find1, N0-ID1)</code>
R-011	Move the fork onto the table. <code>command(move1, 0530, on1(0118))</code>

If Rosie is in the kitchen and the fork is there but not currently visible nor remembered, Lucia has nothing to ground *the fork* to in R-030. In this case Lucia creates a new object id for the fork. Rosie acts on R-030 by scanning around the kitchen until it sees the fork on the counter and enters it into the World Model. When Lucia is then given R-011, it grounds *the fork* to that visible object, 0530.

Another form of grounding for Robot tasks is grounding to locations in the agent’s map of a building or other environment. For *Go to the kitchen*, for example, *kitchen* is called a KnownObject. When the agent is in the kitchen, a representation of it is in the World Model and Lucia easily grounds to it. If it’s not there, Lucia retrieves it from Rosie’s map in semantic memory. If it’s not there either, then a new object is created.

To summarize, the grounding process consists of three different kinds of operations: retrieving knowledge from Rosie’s Ontology that defines a word in terms of Rosie’s internal perceptual properties or action representations, searching in Rosie’s world knowledge for an object to satisfy a referring expression, and creating new object representations when no referent is found. Referring expressions are grounded in three different modes: in direct mode, objects are found in the World Model; in hypothetical mode for Games temporary new objects are created for indefinite RefExprs; and in Robot mode objects not in the World Model are searched for in Rosie’s environment map in semantic memory.

## **5.4 Ambiguity and Local Repairs**

Human language in general, and the sentences in Rosie’s corpora in particular, have many sentences that are locally ambiguous, require local repairs to do I3P processing, or are difficult even for humans to parse. Lucia has several strategies for dealing with these issues, many described at a conceptual level in Chapter 4. This section gives additional detail of how the Soar architecture and Lucia’s production rules solve problems of this sort (Lindes & Laird, 2017a).

### **5.4.1 Lexical selection**

Some function words have multiple uses that must be resolved. The word *that* in particular has three senses defined in the grammar, so three lexical construction candidates are built in lexical-access: THAT-deictic, THAT-relative, and THAT-complementizer. A combination of ECG rules and hand-built rules selects one of these candidates, and remembers the others as options for possible future local repairs. Here are some examples to illustrate.

*Case Study 5.6: Selection by syntactic context*

B-017	Put <i>that</i> in the pantry. command(op_put-down1, 06, in1(L2))
B-028	Pick the green block <i>that</i> is small. object-description(07, on1(L3))
B-145	The goal is <i>that</i> the box is in the office. object-description(concept(goal), subclause(action(is1), 013, in1(L9)))

During a lexical-access decision cycle, all the integration rules for all candidate senses fire, so that each candidate has been generalized and its meaning schema has been evoked and populated. At the end of this decision cycle, new operators can be proposed that use each of the candidates, possibly giving multiple operator proposals.

Three types of operators are possible: a grounding operator to ground that lexical item, a repair operator to initiate a local repair as discussed further below, or a match-construction operator to build a new composite construction including this lexical item. General preference rules determine that a repair operator has top priority, followed by a grounding operator, followed by a match-construction operator. Based on the proposals and the preferences, Soar selects an operator to be applied in the next decision cycle.

The selected operator always uses one of the lexical candidates, which becomes the selected sense of the word. With an operator selected, a lexical-selection rule fires for each other lexical candidate, removing it from the Soar state and remembering it under the selected candidate for possible future use in a local repair. There are two lexical-selection rules that can fire here, one if the selected operator is a grounding or repair operator, and the other for a match-construction operator.

For the word *that*, one, two, or three operators can be proposed, depending on the syntactic context. After *Put that* in B-017 three operators are proposed: a grounding operator called *ground-this-n-that* for THAT-deictic, and two match-construction operators to build a TransitiveCommand using each of the pronoun senses. Grounding is always preferred, so THAT-deictic is grounded to the object the instructor is pointing to and the other two senses are removed.

After the grounding, the `match-construction` for THAT-deictic is selected to build a `TransitiveCommand` with that grounded `RefExpr`.

After *Pick the green block that* in B-028, a repair operator called `attach-relative-pronoun`, is proposed using THAT-relative, along with two `match-construction` operators for `DiTransitiveCommand` using the two pronouns. The repair operator seeks to uncover the `RefExpr` for *the green block* that has already been built into a `TransitiveCommand`. Since repair operators are always the most preferred, that operator is selected. As part of its repair, it discards the `TransitiveCommand` that was built for *Pick the green block*, the other two lexicals are also discarded, and both `match-constructions` for `DiTransitiveCommand` are retracted. Next a `resolve-relative-pronoun` operator grounds the THAT-relative to the preceding `RefExpr` to serve as the subject of the subject-relative clause that is beginning. (Lucia's grammar does not handle object-relative clauses, since there are none in our corpora. Chapter 6 describes extensions for those.)

For B-145, THAT-complementizer is the desired sense after *The goal is that*. It is preceded on the stack by a `FiniteToBe` for *is*, in turn preceded by a `RefExpr` for *The goal*. Three `match-construction` operators are proposed at this point, one for a `ConceptIsThat` that uses THAT-complementizer and two for `RefIsRefs` that use the two pronoun senses. `ConceptIsThat` specifies THAT-complementizer specifically as its third constituent, so the preference rule that depends on the number of lexical constituents prefers it over the other two, and it is selected. Then `lexical-selection` rules remove the other senses, and the `ConceptIsThat` construction is integrated into the comprehension state.

In a similar way in DC 40 on the right of Figure 5-7, once the `match-construction` operator for `SpecPropNoun` has been selected, the `lexical-selection` rule for `match-construction` selects the `CommonNoun` sense of *sphere* and rejects the `PropertyName` sense. The same strategy resolves many other lexical ambiguities.

This processing is rather complicated for a simple word like *that*, but it shows how multiple candidate composite constructions are proposed by ECG

rules that consider the syntactic context, along with help from a few hand-built rules that apply to a number of situations, results in correct selection among several senses of the same word. All the grounding in Lucia is done with hand-built rules and not those generated from ECG. The ECG formalism works well for representing syntax mapping it to semantics, but it does not cover grounding.

### 5.4.2 Grounding pronouns

Pronouns such as *this*, *that*, *me*, *you*, *it*, and *one* do not provide properties of an object to look for in the world, but rather are referring expressions which require looking in the current situation and dialog context to find their referent. Lucia has several specific strategies for grounding these words that are explained in these examples.

#### *Case Study 5.7: Sentences with pronouns to be grounded*

B-017	Put <i>that</i> in the pantry. command(op_put-down1, 06, in1(L2))
B-018	Put <i>it</i> on <i>this</i> . command(op_put-down1, 07, on1(06))
B-028	Pick the green block <i>that</i> is small. command(op_pick-up1, 013)
B-065	This <i>one</i> is orange. object-description(06, color(orange1))
B-165	Tell <i>me</i> the answer. command(initiate-tell1, me, concept(answer1))
B-178	If <i>you</i> see some trash then throw <i>it</i> away. conditional(if-subclause(agent(R5), action(op_sense1), trash1), then-subclause(action(op_throw1), trash1))
G-009	If a location is not below an object then <i>it</i> is clear. conditional(if-subclause(action(is1), NO-ID1, below1(NO-ID2)), then-subclause(action(is1), NO-ID1, predicate(clear)))

As we saw, *that* as a pronoun can be either THAT-deictic or THAT-relative. The deictic form, as in B-017, is grounded by ground-this-n-that to the object in the World Model marked as being pointed to by the instructor. The relative from, as in B-028, becomes the subject of a subject-relative clause, and as such is grounded by resolve-relative-pronoun to the referent of the RefExpr that precedes it. The pronoun *this* is always treated as deictic, as in B-018, and grounded in the same way as the deictic form of *that*. *This* can also be treated as

a specifier in a phrase like *this one*. Other uses of *this* are not included in our corpora and therefore are not handled by the current Lucia.

The pronoun *you*, as used in our corpora, is always grounded to a special object in Rosie’s World Model that represents Rosie itself. The pronoun *me* is rarely used, and it is represented in the final message as simply an object whose handle is `me`. Rosie looks in its situation model to find who `me` is, based on who it is talking to.

The pronouns *it* and *one* are treated basically the same. A number of clausal constructions, including `ImperativeWithLocation`, `RefIsPrepPhrase`, `PropertySetIsPrepPhrase`, and `PropertySetIsNotPrepPhrase` evoke an instance of a schema called `Salient` in addition to their main meaning schema. Each of these constructions uses a `UNIFY` constraint to populate the reference role of this schema with the referent from its meaning structure that seems most salient to be referred to outside this clause. An apply rule for `match-construction` puts an indication on Soar’s top state of what this salient referent is. Then when the `resolve-pronoun` operator tries to ground *it* or *one*, it grounds it to what it finds there. A sentence like G-009 is processed in hypothetical mode for Games, and in this mode any salient item is deleted at the end of the sentence so that it is only used locally within a sentence. This strategy cannot handle many other anaphoric or cataphoric references that can appear in English, but it is adequate for the limited number of cases in our corpora.

### 5.4.3 Local repairs

Fulfilling the single-path principle requires making choices that may later turn out to be incorrect. This subsection examines two such cases. Both sentences have an initial part that is syntactically complete but semantically ambiguous, followed by either a prepositional phrase or relative clause that resolves the ambiguity. However, to resolve the ambiguity, the additional structure must be composed with a referring expression that has already been composed into the initial syntactically complete sentence. A local repair is required in each case. We consider these cases at the level of Soar operators and rules.

*Case Study 5.8: A prepositional phrase requiring a local repair*

B-020	Pick up the green block <i>on the stove</i> . command(op_pick-up1, 015)
-------	--

Syntactically, *Pick up the green block* looks like a complete sentence, and Lucia builds its structure that way after *block*. However, there are three green blocks in the scene, so this part of the whole sentence is semantically ambiguous. The instructor, knowing that, adds the phrase *on the stove* to resolve the ambiguity. Lucia now has the challenge of how to integrate the sentence properly. Figure 5-10 shows the three construction cycles involved in addressing this challenge, with a “?” over the step that appears problematic. There are three possible solutions: the one Lucia uses, a more principled alternative explained below, and one requiring the experimental approach described in Chapter 6.

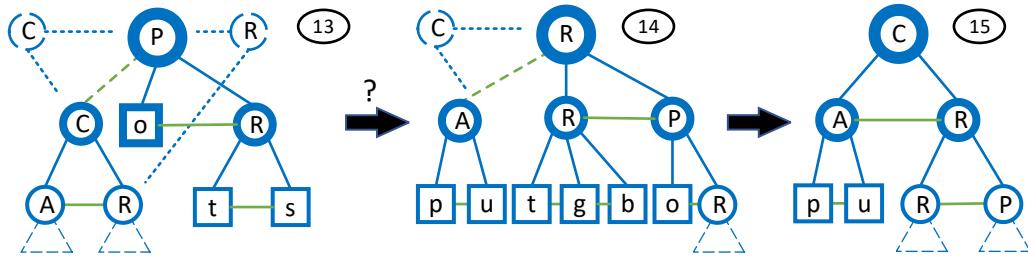


Figure 5-10: A prepositional phrase requiring a local repair

At cycle 13, the partial sentence has been built up as the C node, which precedes the P node for the prepositional phrase on the stack. At this point the figure shows two possible new nodes that could be built. A new C node could be made up from the C <- P pattern, or a new R node for RefExprPrepPhrase could compose the P node with the R node which has already been built into the existing C node. A **match-construction** operator for ImperativeWithLocation is proposed for the possible new C node. However, the ECGtoSoar translator does not produce **match-construction** operators that can reach down to match nodes that are children of previously composed nodes. To solve this problem, Lucia uses a technique modeled after the local repair strategy introduced by Lewis (1993) in NL-Soar. Figure 5-11 shows how this is done.

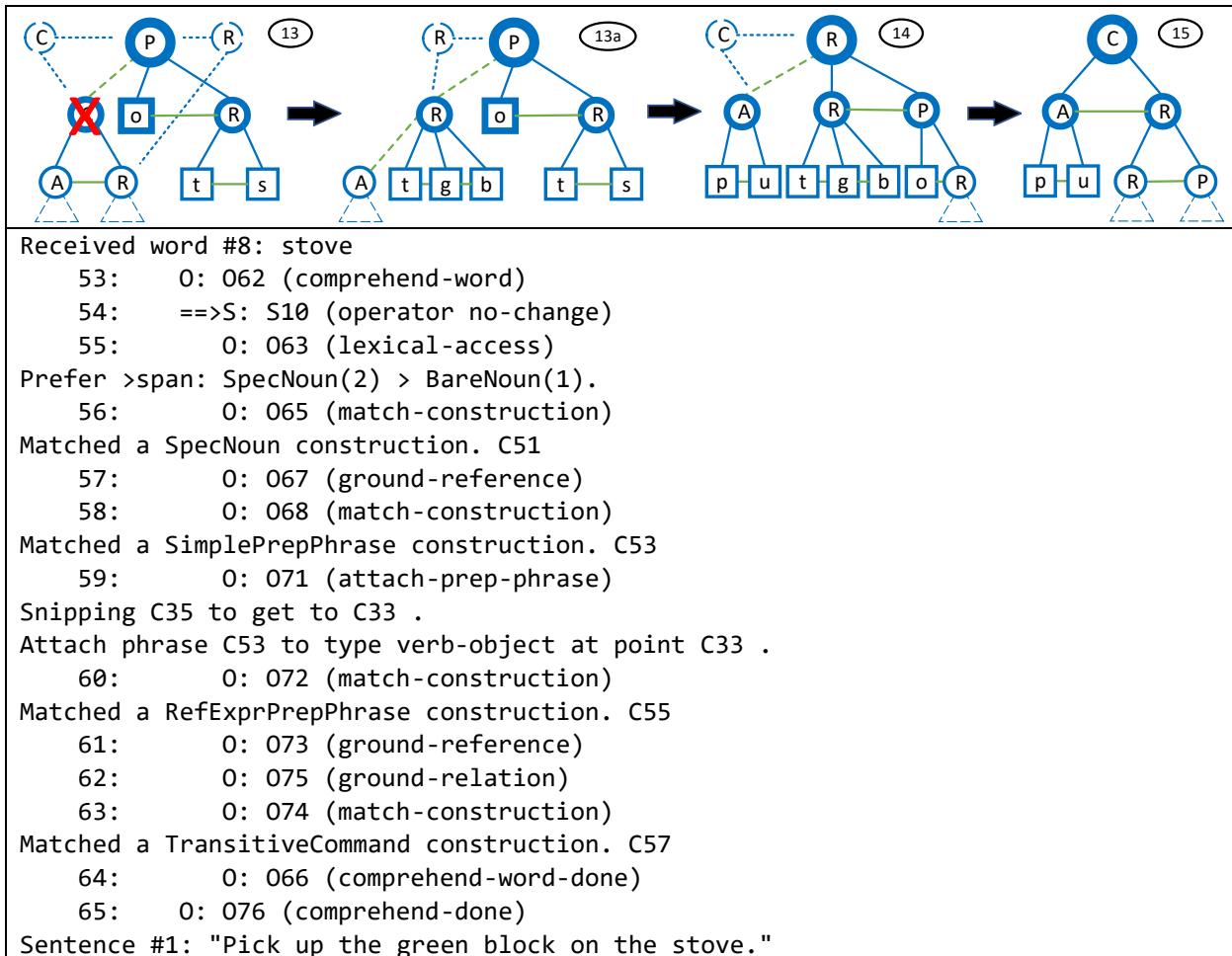


Figure 5-11: Using a snip for a local repair

The key to this solution is an operation called *snip*. Since no match-construction operator can, in this case, compose a new R node using the existing one on level 3 as a constituent, the tree must be reconfigured. This involves deleting the existing C node and restoring its constituents to the stack as predecessors of the P node, as shown in 13a. With this done, the regular proposal rule for RefExprPrepPhrase fires and we get frame 14. In Lucia this is implemented with operator in an additional decision cycle between the construction cycles 13 and 14 shown above. Several different operators are used to perform snips in different situations.

For this particular sentence and some similar ones, the snip is performed by an operator called *attach-prep-phrase*. It has several different propose, preference, and apply rules that handle a variety of prepositional phrase

attachment situations, some requiring a snip and some not. Several other operators perform snips for many syntactic contexts not enumerated here, and quite a few of the sentences in the Rosie corpora use one or more snips in Lucia.

The snip strategy works to perform many local repairs, but it is somewhat dissatisfying since it requires these special-purpose operators. Even though each such operator has enough generality to apply to a number of specific sentence forms, it would be desirable to have a more principled approach. There is such an approach: simply generate two or more `match-construction` operators for any composite construction that might need a snip. An example would be an operator that can reach down in the tree to match the possible R construction in frame 13 above. If such an operator existed, two `match-construction` operators would be proposed in that frame, and a simple preference rule would resolve the problem. We call this an *implicit snip*.

We have experimented with additional `match-construction` propose rules for some constructions, and that approach works without the need for an explicit snip. If the possible R in frame 13 is selected, the tree is transformed directly to the form in frame 14. The erroneous C node is discarded implicitly. Although this approach works as a proof of concept, its full implementation would require a way to determine which constructions need the extra propose rules, and which do not, and a thorough test of the effect over all the sentences in the corpora. This remains as future work. Chapter 6 explores a third approach that is architecturally based, involving constructions in semantic memory and changes to the architecture.

With all these solutions, the snip strategy shows the value of making all three top levels of the tree, not just the stack, available to participate in selection of composite constructions. Section 5.5 examines what happens for sentences that would need more than three levels to resolve.

#### *Case Study 5.9: A relative clause requiring a local repair*

B-028	Pick the green block <i>that is small</i> . object-description(07, on1(L3))
-------	--

A sentence with a relative clause often requires a similar local repair. This sentence has the same semantic ambiguity as the previous one, and a similar strategy is used to do a local repair. However, in this case the need for a repair comes when the word *that* is processed rather than after processing the complete relative clause. The word *that* has three lexical senses that must be selected from. An operator called `attach-relative-pronoun` that is selected after `lexical-access` in this syntactic context uses the THAT-relative option. Figure 5-12 shows the sequence of operators for this case and a diagram of how the `attach-relative-pronoun` operator changes the comprehension state.

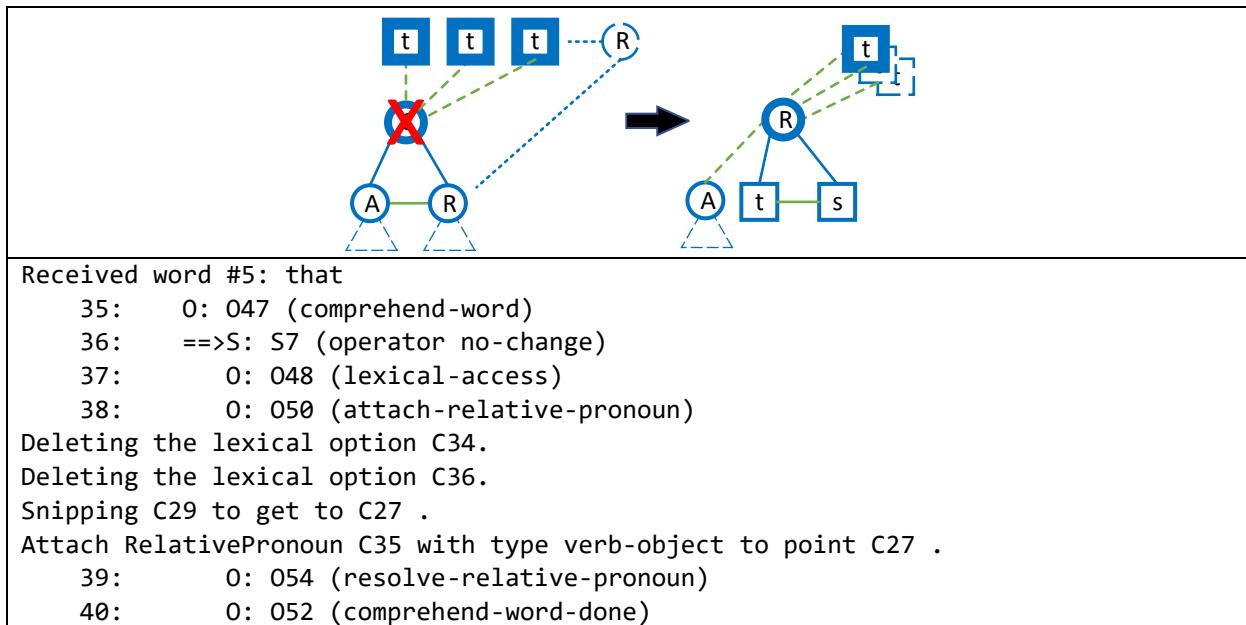


Figure 5-12: A repair combined with lexical selection

The selection of `attach-relative-pronoun` causes a `lexical-selection` rule to fire twice, removing the other lexical items. The attachment operator matches to the node on level 3 that *that* should be attached to, the R node that was previously composed into a C node similar to the one in frame 13 of in previous example. This results in a snip that leaves the THAT-relative at the root, preceded by the previous R and A nodes. When *is* is processed, a `HeadRelativeClause` construction is built, beginning the processing of *that is small*.

All-in-all, several processes are combined to fully comprehend this sentence after *that* has been processed by lexical-access after THAT-relative is selected. The snip is performed by an attach-relative-pronoun operator. A resolve-relative-pronoun operator grounds *that* to the referent of *the green block*. The rest of the relative clause is built up. A RefExprRelClause construction is built for *the green block that is small*. That whole phrase is grounded to a unique object. A TransitiveCommand for the complete sentence is built and grounded. Finally, the sentence is interpreted to form a message to Rosie.

## 5.5 Difficult Sentences

There are sentences that appear to be grammatical based on a careful syntactic analysis, but which humans find difficult or impossible to comprehend. *Garden-path sentences* are difficult because the first part of a sentence leads the comprehender in a direction that later turns out to be incorrect, and for which local repairs don't seem to be possible. *Parsing breakdown* happens when human processing fails completely, often due to a structure called *center embedding*.

Human comprehension processing is usually automatic, or without conscious deliberation. When standard syntactic and/or semantic analysis fails, as with garden-path or parsing-breakdown effects, conscious deliberation strategies are sometimes used to find a meaning for such sentences.

The Rosie corpora have several sentences that have difficulties of these kinds. In the development of Lucia, we have used some hand-built, ad-hoc techniques to find correct interpretations of these sentences. These techniques may roughly model human post-hoc deliberative processing. Lewis (1993) presents 100 examples of different types of difficult sentences, along with similar ones that are not difficult. This section examines how Lucia's algorithm deals with some difficult sentences from the Rosie corpora, and in Chapter 6 we explore examples from Lewis that require grammatical constructions that have not been included in Rosie. The original ECG parser (Bryant, 2008) may handle these more easily since it does a best-fit global analysis, but this prevents it from doing incremental processing.

### *Case Study 5.10: A repair too deep to be local*

B-032	Move the green rectangle to the left of the large green rectangle to the pantry. command(op_move1, 047, to1(068))
-------	--

B-032 does not fit the pattern of local repairs explained above. After *to the left of the large green rectangle* it appears this phrase is providing a target for *Move*, so a complete sentence is formed. But after *to the pantry*, it turns out the *to the left of* phrase was intended to modify *the green rectangle* and that *the pantry* is the actual target for *Move*.

In this case Lucia uses a deliberative strategy that involves setting part of the tree aside temporarily, restructuring the remnant, and rebuilding differently. Figure 5-13 shows graphs of several steps, along with a Soar trace showing the sequence of decision cycles. Because of the complexity of this example, decision cycle numbers and the internal Soar identifiers of many nodes have been added to the graphs to help correlate them with the Soar trace. The first frame in the top row shows the state of the tree after finishing the processing of *Move the green rectangle to the left of the large green rectangle*. (Note that *to the left of* is treated as a single lexical item labeled ttlo.) C74 is an ImperativeWithLocation construction that covers what appears to be a whole sentence. More construction cycles are needed to process *to the pantry*, culminating in DC102 where a SimplePrepPhrase, C92, is built by match-construction.

At this point no new constructions are proposed, because there is none that can compose the C74 <- C92 pattern. What should happen next is that C28 and C72 should be composed into a new RefExprPrepPhrase, but by now C28 is inaccessible down at level 4. This situation is not a local repair according to our theory of processing operating only on the top three levels of the tree. Since a local repair is not possible, the normal Lucia algorithm fails at this point, conforming to our intuition that this is a garden-path sentence that is difficult for a human to understand. To solve this problem, we have implemented a solution that violates our normal rules of locality and represents one possibility of how humans might do a deliberative repair.

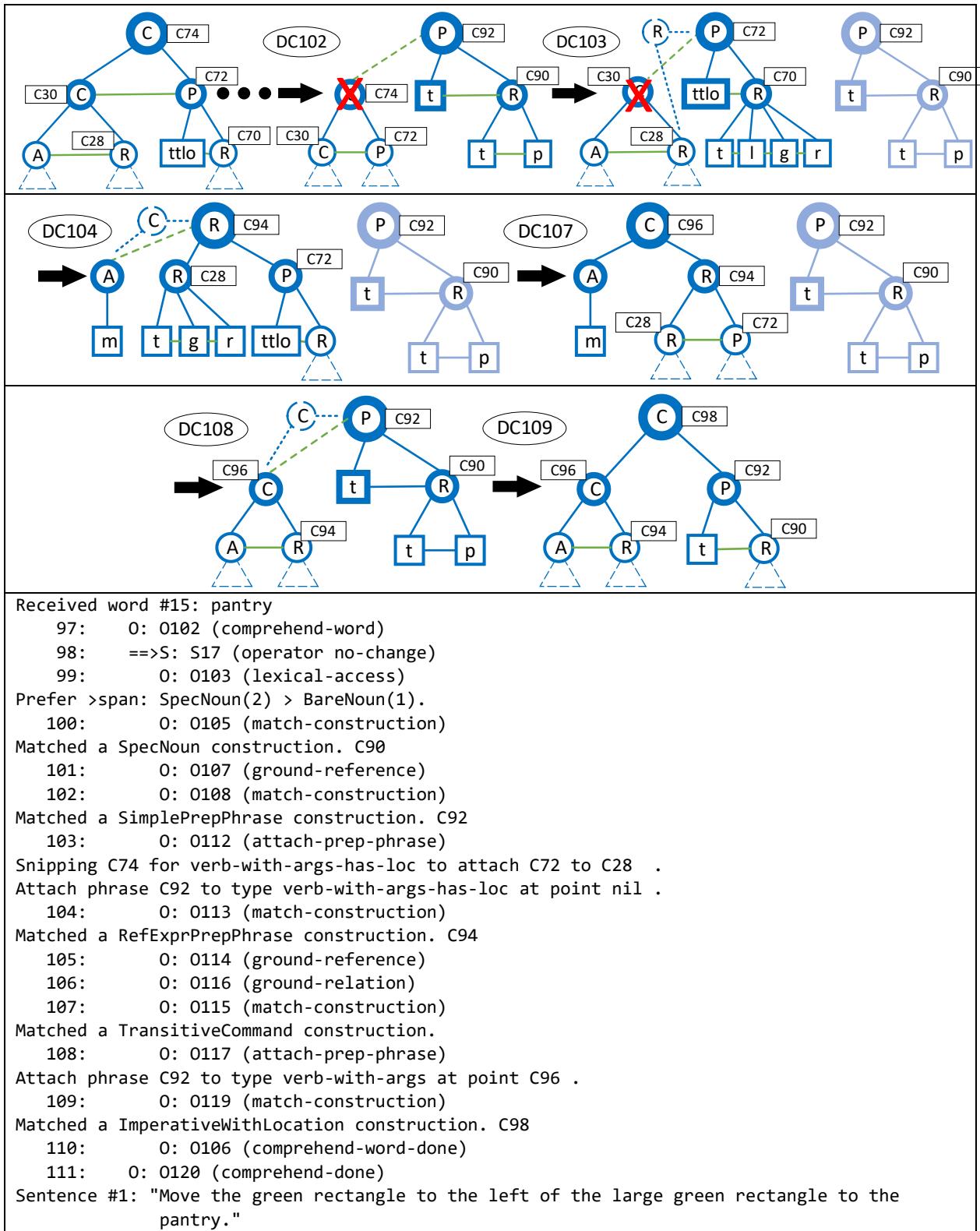


Figure 5-13: A deliberative repair

The attach-prep-phrase operator in DC103 does several things that violate Lucia's normal rules, but could reasonably be part of a deliberative process. The first step, since nothing matches the stack as it is, is to remove the top node, along with its subtree, from the stack and set it aside for future use. This is shown in the graph as the C92 node and its subtree disconnected from the main tree and grayed out. C74 is discarded, leaving C72 as the new root of the main tree.

At this point a normal snip operation, or its equivalent, is possible since a RefExprPrepPhrase can match C72 with the C28 RefExpr at level three. Now DC104 snips C30 and composes C94, still leaving C92 on the side. After C94 is grounded in DC105 and DC106 (not shown in the graph), DC107 builds a new TransitiveCommand, C96, for *Move the green rectangle to the left of the large green rectangle*, but this time with the object of *Move* being the entire phrase *the green rectangle to the left of the large green rectangle*, represented by C94, which has been fully grounded. C92 is still on the side.

At this point an attach-prep-phrase operator at DC108 restores C92 to the top of the stack and the root of the tree, allowing a new ImperativeWithLocation construction, C98, to compose the entire sentence with the correct structure.

Two important steps in this process are clear violations of Lucia's normal algorithm, but reasonable things to do as part of a conscious, deliberative search for a complete grammatical comprehension of the sentence. The first is setting a node and its subtree aside for several decision cycles. The second is in DC102 where C74 is discarded, or snipped, without any clear goal for what construction could match afterward. At this point the algorithm is just experimenting to see if it can find a way out of its dead end. Once C74 has been removed, a second, more normal snip is done to allow composing C94. Finally, the deliberation restores C92 to its rightful place on the stack in DC108, and normal processing takes over from there. This example demonstrates how deliberative processing can work in Lucia, but a general deliberation system remains as future work.

## **5.6 Evaluation**

We have shown in the previous chapters that a model of sentence comprehension using the ECG form of construction grammar as composable knowledge of meaning (CKM) and Lucia's construction-cycle algorithm for incremental, immediate interpretation processing (I3P) can indeed do embodied, end-to end comprehension (E3C), at least in a limited domain. This chapter describes how the implementation of the Lucia theory using the Soar cognitive architecture to represent general cognitive mechanisms (GCM) satisfies E3C, CKM, and I3P.

To evaluate the Soar implementation, we review the evidence that it does indeed represent CKM, perform I3P, and achieve E3C. We evaluate a metric of real-time performance against measures of human speech and reading rates. We review limitations of Lucia in Soar, and how future work might alleviate or eliminate some of these.

### **5.6.1 Basic E3C-CKM-I3P performance**

Chapter 2 and Appendix 1 discuss the three corpora of sentences used for teaching ITL tasks to Rosie. The Evaluation section of Chapter 2 describes details of how we tested that Lucia does in fact perform correct E3C on these sentences. Table 3-1 repeats the summary of test data given there, but showing only the data for the sentences that perform the entire E3C function, including grounding and sentence interpretation. The generality ratio shown is the ratio of the number of sentence forms working to the number used for development to achieve that. The ratios defined this way are relatively small due to the fact that the sentences were developed to represent unique meanings not encountered before. When a new lexical, syntactic, or semantic element is added to the grammar, it may be very general and capable of matching a large number of sentences. However, few if any such sentences may be found in our corpora. The generality ration shown here represents only sentences in our corpora, not the full range of generality that is possible.

Table 5-1: Results of testing

<b>Corpus</b>	<b>All Sentences</b>	<b>Sentence Forms</b>	<b>Development Set</b>	<b>Forms Working</b>	<b>Generality Ratio</b>
Baseline	207	207	143	207	1.45
Games	1,104	172	48	51	1.06
Robot	228	160	60	110	1.83
Total	1,539	539	251	368	1.47

These data show how far we have come to date in covering the Rosie corpora, with a total of 368 out of 539 distinct sentences producing messages that are correct to achieve Rosie's goals. Given that the model is implemented using Soar's architectural mechanisms as described in this chapter, these data show that Lucia in Soar performs E3C for the scope of these sentences.

The above sections described in detail how Soar production rules generated by the ECGtoSoar translator program represent CKM in the form of proposal rules for `match-construction` operators and apply rules for `match-construction` and `lexical-access` operators. These rules, along with the hand-built processing rules described, perform Lucia's construction-cycle algorithm for I3P, including resolution of various kinds of ambiguity and making local repairs, as shown in the various case studies given above. Taken together, the corpus test data and the detailed explanations of how Soar's mechanisms are used to represent CKM and process I3P show that Lucia in Soar implements E3C within these constraints of knowledge and processing.

### 5.6.2 Real-time performance

Human adults comprehend sentences at a rate in the range of 150 words/minute for speech and up to 250-300 wpm in reading. On a modern computer, Lucia in Soar can run at a much higher rate than this; however, our aim is to model human performance. To compare Lucia's processing rate to human performance, we use a *simulated real time* metric. There is a reasonable consensus among cognitive architecture researchers that a human cognitive cycle is on the order of 50ms (Anderson, 2007; Laird, Lebiere, et al., 2017; Newell, 1990). To calculate

the simulated real time for Soar to do certain processing, we count the number of decision cycles taken and multiply that by 50ms.

A run of Lucia on our Baseline corpus takes 10,683 decision cycles, or a simulated real time of 534.15 seconds. In that run 1,150 words were processed, yielding an average of 0.464 seconds/word, giving a simulated processing rate of 129.2 wpm. This rate indicates that Lucia is slower than human comprehension of speech, about 86% of human speech rate. It is around half of the human rate for skilled reading.

During Lucia's development, the primary emphasis was on achieving E3C performance with a system using CKM to do I3P with GCM. Simulated real time performance comparable to humans was a concern, but not the main focus. Therefore, once that becomes a criterion for evaluation, more work is needed on Lucia's processing to decrease the number of decisions required for each word, which in turn would speed up its simulated wpm. Soar provides a learning mechanism called *chunking* which could possibly replace some decision cycles by learning more skilled processing of the **comprehend-word** operator. Using this approach, chunks to speed up processing would be learned gradually over time as more and more combinations of words and contexts are encountered.

For example, it seems reasonable that the entire word cycle for the word *Pick* shown in Figure 5-7 could be reduced to a single decision cycle from the seven it takes now. The word cycle for *sphere* later on, however, could not be fully reduced because the grounding to the World Model must remain flexible to allow for changes in the external situation and resulting perception. It is not at all obvious how to learn new rules in some cases, but not collapse those operators that must be sensitive to the immediate unique situation. To discover the full impact of chunking would require implementing it fully, measuring its effect on a large corpus, and then finding ways to avoid problems caused by learned rules that are too general. Unfortunately, this is likely to take a lot of work to make sure the learned rules correctly handle the large number of cases involved, so this is left for future work.

At this time, we can say that Lucia in Soar comprehends language at a simulated real-time rate that is close to the rate of human speech, and its simulated processing rate could probably be improved to equal human processing rates.

### **5.6.3 Limitations**

Many other measures of human processing have been made that the current Lucia implementation cannot reproduce. The current Lucia does not have a mechanism for measuring variable retrieval times for constructions, and therefore does not make predictions about word-by-word details of human reading times as Lewis and Vasishth (2005) have done. To model this sort of thing would require a future version of Lucia with a more nuanced relationship between construction cycles and simulated real time.

Neither can Lucia model the brain measurements Brennan and Hale (2019) discuss. They measure EEG data of humans that “listen passively to an audiobook story,” and look for correlations with other computational models. Their EEG data shows significant events at 400ms and 600ms after word onset, which implies there may be overlap between processing one word and the next since words come in as often as every 200ms. Little is currently understood about mapping brain events onto cognitive architecture cycles, but reproducing this sort of overlap would be a challenge for the current Soar architecture (Lindes, 2019).

## **Chapter 6 Exploring Grammar, Architecture, and Acquisition**

The previous chapters have explained in detail the theory and implementation of Lucia, how it satisfies our E3C, CKM, I3P, and GCM requirements, and a number of its limitations. This chapter describes three explorations of ways to extend Lucia beyond its current scope and limitations. The first looks at ways to extend the ECG grammar to cover event descriptions, object-relative clauses, and sentences that are difficult for humans to parse. The second explores what we call System B, a version of Lucia that explores a quite different way of using and extending the Soar architecture by storing the ECG grammar in Soar's semantic memory rather than in production rules in order to better model human comprehension and lay the groundwork for the third exploration. Given the Lucia model of comprehension, our third exploration is to develop a theory of how the agent could automatically acquire more knowledge of the meaning of language through experience. These explorations are experimental, without the full implementation that we have done for the version of Lucia described in previous chapters, which we call System A.

### **6.1 Events and Difficult Sentences**

Chapter 3 lists language phenomena not covered by Lucia's grammar for Rosie, including event descriptions, object-relative clauses, which are English forms not used in the corpora with which Rosie was developed. To better model human comprehension, Lucia should exhibit coverage and limits of its comprehension abilities similar to those of humans. This section explores additions to Lucia's ECG grammar to cover events and object-relative clauses, then how Lucia, with this additional grammar, can model the problems humans have in processing certain kinds of sentences. As examples we use sentences with the garden-path and parsing-breakdown effects analyzed by Lewis (1993) in NL-Soar.

### 6.1.1 Grammar for event descriptions and relative clauses

Additional grammar is needed for two important grammatical phenomena that don't appear in the Rosie corpora and therefore have not been covered in Lucia's grammar for Rosie: event descriptions and object-relative clauses. Adding these to the grammar allows us to examine some sentence structures that are difficult for humans to process, which are addressed in the next two subsections.

Event descriptions are complete declarative sentences that have a subject, a verb indicating an action the subject takes, possibly an object, and possibly a prepositional phrase indicating a target or location. The general form is shown in (1), along with examples. Angle brackets indicate a class of items and square brackets indicate something that is optional. A subscript indicates a particular sense of a word with multiple senses, in this case the past tense of *raced*.

- (1) a. <subject> <action> [<object>] [<modifier-phrase>]  
b. The cat ran away.  
c. The horse raced<sub>1</sub>.  
d. The mouse chased the bird.  
e. The horse raced<sub>1</sub> past the barn.

With the additional constructions summarized in Table 6-1, along with some new lexical items, all the sentences in (1) are parsed correctly. (Full semantics for these forms have not yet been added.)

Table 6-1: Grammar for event descriptions

Construction	Constituents
ActionClause subcase of Declarative	ref : RefExpr verb : ActionVerb
EventWithLocation subcase of VerbWithArguments	event: ActionClause location: PrepPhrase
TransitiveClause subcase of Declarative	subject: RefExpr verb: ActionVerbTransitive object: RefExpr
ActionVerbTransitive	general construction

A relative clause is a clause structure that is used to modify a noun phrase. Often, they are introduced with a relative pronoun, such as *that*, *which*, or *who*.

A relative clause without a pronoun is called *reduced*. There are two general classes of relative clauses: *subject-relative* clauses where the preceding noun phrase serves as the subject of the verb, and *object-relative* clauses where the preceding noun phrase serves as the object of the verb. Lucia's grammar for Rosie covers subject-relative but not object-relative clauses. Here we examine both and show the extensions needed to the grammar for the object-relative case. General forms with examples of subject-relative clauses are given in (2), with the relative clauses italicized.

- (2) a. <subject> <relative-pronoun> <finite-to-be> <property>  
b. the green block *that is small*  
c. the green block *that is on the stove*  
d. <subject> <relative-pronoun> <action-verb> [<object>]  
e. the cat *that ran away*  
f. the mouse *that chased the bird*

These are not complete sentences. The entire structure with an initial noun phrase followed by a relative clause makes up a form of referring expression. The pronouns seem to be necessary in the subject-relative forms, and the verb can just be a <finite-to-be> like *is* or *was* with a <property> to be attributed to the subject, or an <action-verb> with an optional <object> describing something the <subject> is doing or has done. Examples of object-relative clauses are given in (3), with the past participle sense of *raced*.

- (3) a. <object> [<relative-pronoun>] <subject> <action-verb>  
b. the bird *that the mouse chased*  
c. the cat *the bird scared*  
d. the horse *that was raced<sub>2</sub>*  
e. the horse *raced<sub>2</sub>*

Object-relative clauses, none of which are in the Rosie corpora, violate the normal English form of subject-verb-object by reversing it to object-subject-verb, and, with the pronoun left out, they give the unusual form of two noun phrases in a row. This presents challenges for both grammar representation and incremental processing.

The Lucia grammar for Rosie has grammar for the subject-relative form using <finite-to-be>, and additional constructions were added to cover the object-relative possibilities. The grammar additions for both event descriptions and relative clauses make it possible to address the garden-path and local-repair issues addressed in the following subsections. Table 6-2 shows all the constructions for relative clauses.

Table 6-2: Grammar for relative clauses

<b>Construction</b>	<b>Constituents</b>
HeadRelativeClause	pro : RelativePronoun <u>tobe</u> : FiniteToBe
RelativeClauseProperty subcase of RelativeClause	head : HeadRelativeClause prop : Property
RelativeClausePrepPhrase subcase of RelativeClause	head : HeadRelativeClause prepPhrase: PrepPhrase
RelativeClause	general construction
RefExprRelClause subcase of RefExpr	subject : RefExpr clause : RelativeClause
IntransitiveRelativeClause subcase of RelativeClause	pro : THAT-relative verb: ActionVerb
TransitiveHeadRelativeClause	pro : THAT-relative verb: ActionVerbTransitive
TransitiveRelativeClause subcase of RelativeClause	head : TransitiveHeadRelativeClause object: RefExpr
ObjectRelativeClause	that : THAT-relative subject: SpecifierNP verb: ActionVerbTransitive
RefExprObjectRelClause subcase of RefExpr	object: RefExpr clause : ObjectRelativeClause
RefExprReducedORC subcase of RefExpr	object: RefExpr subject: SpecifierNP verb: ActionVerbTransitive
RefExprReducedRelative subcase of RefExpr	ref : RefExpr verb : PassiveVerb
PassiveVerb	general construction

The top constructions through RefExprRelClause are in the Rosie grammar, and the rest were added to handle object-relative clauses. The RefExprRelClause construction combines a RefExpr subject with a subject-

relative clause to form a larger RefExpr. RefExprObjectRelClause and RefExprReducedORC represent RefExprs with object-relative clauses, such as (3b) and (3c). RefExprReducedRelative handles a case like (3e).

With this grammar in place, we are ready to look at how Lucia can process sentences with event descriptions and object-relative clauses, as well as those that are often difficult or impossible for humans to comprehend. Lewis (1993) discusses garden-path sentences and human parsing breakdown. His examples include both event descriptions and object-relative clauses, as well as both of these effects that are difficult for humans. The next two subsections explore how Lucia, with the extended grammar just given, can process some of his examples.

### 6.1.2 Garden-path sentences

Humans have trouble with sentences where an early choice leads down a path that cannot be easily corrected. These are called garden-path sentences.

#### *Case Study 6.1: A garden path sentence*

- (4) The horse raced past the barn fell.

This is a garden-path sentence discussed by Lewis (1993, p. 28, GP-14). The word *raced* can have two senses, as a simple past tense of *race* or as a past participle. While doing incremental comprehension, humans tend to choose the past tense form, since this is more common. However, this produces the structure of a complete sentence after *barn*, and leaves no way to integrate *fell*. Figure 6-1 shows the problem.

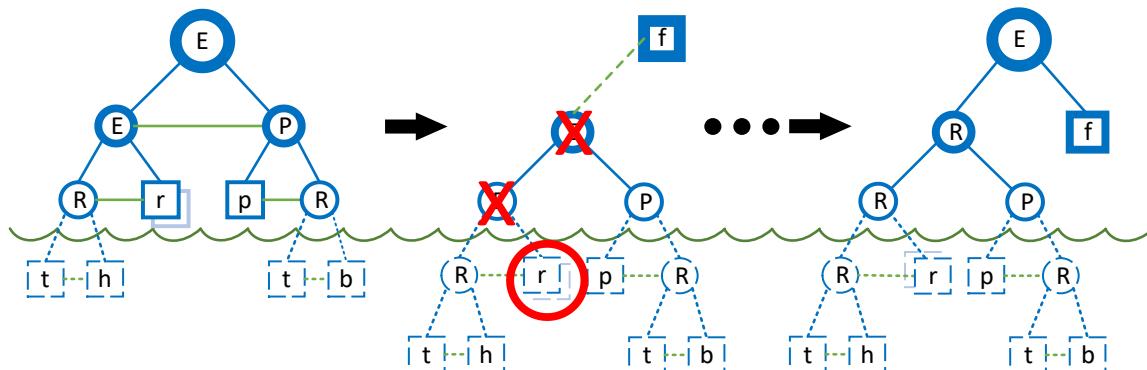


Figure 6-1: Garden path failure and recovery

This diagram shows the whole comprehension tree, including the nodes below the accessibility threshold. E stands for an event description. In the left panel we see the apparently complete sentence after processing *barn*. Notice that *The horse raced<sub>1</sub>* forms one complete event, but that there is a second meaning of *raced* hidden behind the one that was chosen. The middle panel shows that when *fell* is added to the tree the previous tree is pushed down a level, and the ambiguity for senses of *raced* is now below the water line. As we saw in a previous example, a local repair is not possible here. A deliberative repair would have to set the node for *fell* aside, discard both the E nodes marked in red, and reverse the choice of senses for *raced*. With all these changes made, the sentence structure could be rebuilt in the new form shown in the right panel.

A deliberate repair strategy to handle this case has not been implemented in Lucia. However, with the added grammar described above, Lucia can parse this sentence in two ways, depending on which sense of *raced* is chosen when it is processed. At this point, one preference rule chooses a `match-construction` operator for `ActionClause` over `RefExprReducedRelative`, making the lexical-selection rules choose the normal past-tense form, which we have called *raced<sub>1</sub>*. With this rule enabled Lucia produces the garden-path effect shown in the middle panel. An alternative preference rule makes the opposite choice between the two `match-construction` operators, so that the past-participle, called *raced<sub>2</sub>*, is selected. With that rule enabled, Lucia produces the full parse shown in the right panel.

Is there a principled way of choosing between these alternative word senses? One possibility would be to enable one of the choices based on the relative probability of the two different word senses based on statistics from some corpus. This has not been implemented in Lucia, but it could be done. The ECG formalism allows for probabilities to be assigned to constructions, and Bryant (2008) used this for his best-fit analysis parser, without incremental processing. These probabilities could be used in Soar by assigning numeric preferences to the preference rules. This example, and the demonstration done with these two

preference rules, shows that such an approach could be added to Lucia without sacrificing I3P.

There is evidence that humans do use statistics of this sort to address word-sense ambiguities like the one in this example. Merlo and Stevenson (2000, p. 162) cite the very example sentence we are using here, along with another of sentence of the same general form:

- (5) The butter melted in the microwave was lumpy.

They argue that both these sentences have exactly the same “main verb/reduced relative (MV/RR) ambiguity,” but that (5) is “easily interpreted,” and that this supports an “account in which verb-specific frequency information is largely responsible for the variability of processing difficulty in this construction.” Adding statistical information to Lucia’s I3P processing opens a door to a possible direction for future work.

Lewis (1993, pp. 35–49) discusses garden-path and “unproblematic ambiguity” phenomena and theories to explain them. He summarizes with the hypothesis that: “Garden path effects are purely a function of differences between the syntactic structure of the preferred interpretation, and the syntactic structure of the globally correct interpretation.” In a later chapter (pp. 161–193), he discusses how NL-Soar deals with these phenomena. We have not attempted to apply Lucia to his whole range of examples, but the way the Lucia theory deals with these phenomena fits well with both Lewis’s theory and his practice. Lucia covers much of this space by using the various techniques we have discussed for resolving local ambiguities, performing local repairs, and failing when needed repairs are not local and require deliberation, or what Lewis calls “reprocessing.”

### **6.1.3 Parsing breakdown**

Lewis (1993) presents other sentences which cause *parsing breakdown* in humans. One type of breakdown arises in sentences that appear grammatically correct but are difficult or impossible to parse, such as ones that have *center embedding*, like the two versions this one:

### *Case Study 6.2: A sentence that causes parsing breakdown*

- (6) a. The cat that the bird that the mouse chased scared ran away.  
b. The cat the bird the mouse chased scared ran away.

Sentence (6a) from Lewis (1993, p. 49) has nested relative clauses, in a form called *center-embedded* or *self-embedded*. Sentence (6b) has the two instances of *that* removed, making the relative clauses reduced. Both sentences are virtually impossible for a human to understand, despite the fact that they are grammatically correct according to a careful syntactic analysis. We consider two possible explanations consistent with Lucia theory for what might cause this parsing breakdown.

Before doing a detailed analysis, it is useful to compare the problematic sentences in (6) with other sentences that express the same meaning in a way that is easy to understand, along with some shorter sentences that show important pieces of these sentences. Several variations are given in (7).

- (7) a. The mouse chased the bird.  
b. The bird scared the cat.  
c. The cat ran away.  
d. The mouse chased the bird and the bird scared the cat.  
e. The mouse chased the bird and the bird scared the cat and the cat ran away.  
f. The bird that the mouse chased scared the cat.  
g. The cat the bird scared ran away.  
h. The mouse chased the bird that scared the cat that ran away.

All of the sentences in (7) are easy for humans to parse, and Lucia, with the additional grammar outlined above, can parse them correctly. (7a) and (7b) are simple uses of the *TransitiveClause* construction, while (7c) uses the simpler *ActionClause* that has no object. (7d) and (7e) use *AndDeclarative* and *DeclarativeAndDeclarative* constructions to assemble complex clauses with conjunctions. (7f) and (7g) both use single relative clauses with constructions *RefExprObjectRelClause* and *RefExprReducedORC*, respectively. Sentence (7h) has two relative clauses, but they are not nested within each other.

The problem comes in (6a) and (6b) because one relative clause is nested, or embedded, within another one. It is easy to draw a hierarchical syntax diagram for these sentences, so why are they so problematic for humans? We suggest two possible explanations, one syntactic and one semantic.

We start by examining syntactic processing for (6b), since it is a simpler sentence, and the argument is the same for (6a). Figure 6-2 shows the sentence at the top and two possible parsing attempts below it. On the left is a parse that works with a RefExprReducedORC having another instance of the same construction as a constituent. This is an example of the recursion that is possible with Lucia's ECG formalism. Using this recursion, the Lucia parses the sentence with no problem.

The cat the bird the mouse chased scared ran away.

### A parse with recursion

```
ActionClause[
    The cat the bird the mouse
    chased scared ran away.]
RefExprReducedORC[
    The cat the bird the mouse
    chased scared]
SpecNoun[The cat]
    THE[The]
    CAT[cat]
RefExprReducedORC[
    the bird the mouse chased]
SpecNoun[the bird]
    THE[the]
    BIRD[bird]
RefExprReducedORC[
    the mouse chased]
SpecNoun[the mouse]
    THE[the]
    MOUSE[mouse]
CHASE-past-tense[chased]
SCARE-past-tense[scared]
RUN-AWAY-past-tense[ran away.]
```

### A parse without recursion

```
SpecNoun[The cat]
    THE[The]
    CAT[cat]
RefExprReducedORC[
    the bird the mouse chased]
SpecNoun[the bird]
    THE[the]
    BIRD[bird]
SpecNoun[the mouse]
    THE[the]
    MOUSE[mouse]
CHASE-past-tense[chased]
SCARE-past-tense[scared]
RUN-AWAY-past-tense[ran away.]
```

Figure 6-2: A parse that succeeds and one that fails

The parse on the right, however, fails. It succeeds in constructing the inner instance of RefExprReducedORC, but fails to parse the outer one because recursion has been eliminated. Table 6-3 shows the grammar change:

Table 6-3: A construction with and without recursion

<b>Construction</b>	<b>Constituents</b>
RefExprReducedORC subcase of RefExpr	object: RefExpr subject: RefExpr verb: ActionVerbTransitive
RefExprReducedORC subcase of RefExpr	object: RefExpr subject: SpecifierNP verb: ActionVerbTransitive

RefExprReducedORC is a subcase of RefExpr, so in the top version both its object and subject constituent links are recursive. This version parses (6b) correctly as shown on the left in Figure 6-2. In the bottom version, the subject link has been changed to SpecifierNP, breaking the recursion by allowing only a noun phrase rather than a general referring expression in the subject link. With this version of the construction the parse fails, as shown on the right in the figure. Making a similar change to the object link makes no difference in the results, and it is the same for (6a).

Why should eliminating recursion in this particular point of the grammar cause the parse to fail? As mentioned earlier, there can be some doubt about whether human processing truly uses full recursion or not, since certainly working memory limits will result in limited depth. Perhaps this is a case where humans cannot do recursion. But why? In (6b) the inner relative clause in *the bird the mouse chased* is parsed just fine. However, in the outer relative clause in *The cat the bird the mouse chased scared*, the subject position in the <object> <subject> <verb> sequence is *the bird the mouse chased*, which is a complex expression and not a simple noun phrase. For some reason, the human processing system fails when trying to do recursion at this point. Perhaps there is an explanation in the semantics of these forms.

Figure 6-3 is a diagram of the semantic processing required for (6b) at two stages of processing, at *chased* on the left and at *scared* on the right. In each noun phrase, the noun specifies a class of objects and *the* requires a particular one by picking from perception or imagining one. Before grounding the relative clause on the left, none of the three NPs have sufficient information to be grounded. The grounding must be left in some sense “pending.”

When processing *chased*, there are three such pending noun phrases plus the verb on the stack. To ground the whole phrase *the bird the mouse chased* requires finding something in the current perceived or imagined scene a particular mouse chasing a particular bird. An event description with a particular mouse chasing a particular bird is formed. The result of that whole phrase then is a RefExpr whose meaning is grounded to the bird. Sentence (7g) is a complete sentence with a similar phrase that parses correctly.

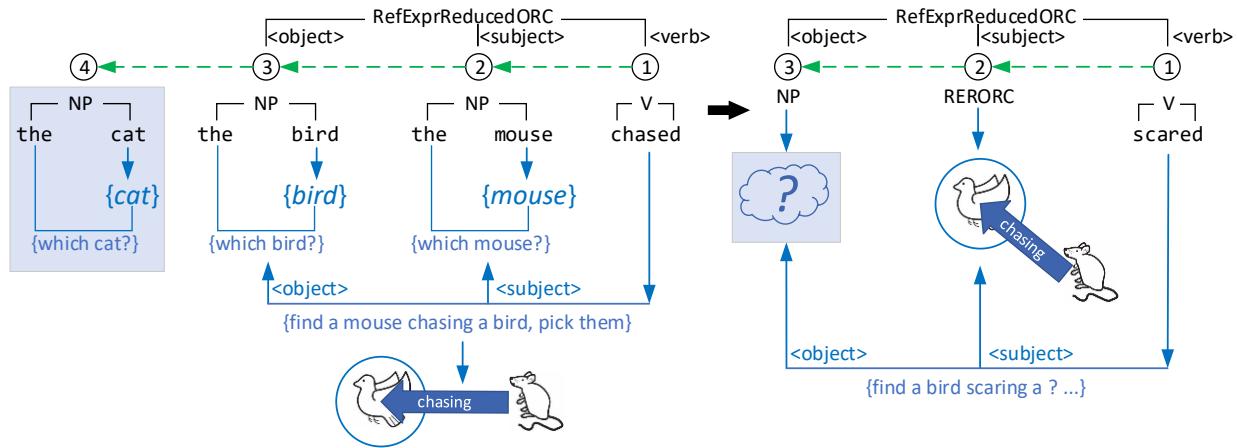


Figure 6-3: Semantics of a parsing breakdown

On the right in Figure 6-3 is the next step with the word *scared*. The stack has the verb at the top, preceded by the RefExpr for *the bird the mouse chased* that was just found, preceded by the pending NP for *the cat*. The figure assumes we allow recursion for the subject of the relative clause. As shown by the shading, however, most information about the pending NP appears to have been lost.

Introspective intuition suggests that, by this time, we have forgotten about the poor cat. We conjecture that there is a mechanism in the human mind that

can perform grounding for a phrase with two pending noun phrases and a verb, but in doing so it forgets any other pending phrases that came before. This forgetting makes a sentence with center embedding like this impossible to comprehend. Future work will be needed to fully model such a mechanism and explore whether it can be supported by human data.

Lewis (1993, pp. 49–67) provides lists of sentences that cause parsing breakdown, including our (6a) and 6(b), and those that show what he calls “acceptable embeddings,” similar to those in (7). He describes syntactic theories and processing architectures that attempt to explain these phenomena. The best explanations are similar to our technique for limiting recursion. He explains (pp. 139–159) how NL-Soar addresses this issue, again providing a syntactic argument that appears equivalent to our limitation of recursion. None of these theories look at limits on semantic processing as suggested here.

These ideas on how to handle garden-path and parsing breakdown effects are conjectures. Much future work remains to validate them in comparison to other theories in the literature and actual human data.

## **6.2 System B: Language Knowledge in Semantic Memory**

The Lucia system described in detail in Chapters 2 through 5, called System A here, performs E3C with I3P using GCM, but has limitations with respect to both functionality and cognitive plausibility in humans: 1) It does not allow for situational context to bias lexical selection. 2) It cannot predict word-by-word reading times. 3) There is no support in the Soar architecture for the limit that only three levels of the tree are accessible for processing, and explicit hand-built operators are required for local repairs. 4) There is no support for a theory of language acquisition from experience since all its CKM comes as already built skilled procedural knowledge from an external source.

This section describes an experimental System B version of Lucia that attempts to address these limitations. System B uses the memories of Soar in a very different way, and makes extensions to it. It implements only part of the Lucia theory, but shows the viability of an approach toward solving them.

### **6.2.1 Overview of System B**

System B is an experimental system designed to test approaches to improving on these limitations, including supporting the theory of language acquisition described in the following section by representing knowledge of language as declarative knowledge that is gradually compiled into procedural skill. System B uses the same ECG grammar as System A, the same structure of the comprehension state, and the same conceptual construction-cycle-based algorithm for doing incremental comprehension.

For System B, the ECG grammar is translated into declarative representations that are stored in Soar's semantic memory, rather than procedural knowledge represented as production rules. Hand-built rules that know nothing of the specific content of ECG items use the mechanisms of semantic memory (smem), including spreading activation, to retrieve this data and use it according to the same phases of the construction cycle to build the same structure of the comprehension state described in Chapters 4 and 5. A new experimental architectural mechanism for attention in working memory (WM) is used to modulate the spreading activation, and multiple parallel retrievals from semantic memory are allowed. Soar's chunking mechanism gradually converts the interpretation of declarative knowledge of language retrieved from smem into procedural knowledge in learned production rules, thus transitioning over time from declarative to procedural access of constructions.

As developed so far for this experiment, System B does not yet implement the full algorithm. Some of the hand-built logic in System A, including the grounding phase, sentence interpretation, and parts of the logic for local repairs are not yet implemented. Because the comprehension state data structures are the same as System A, these features can use the same logic. However, the control structure, described below, is quite different. As a result, the amount of work needed to make this logic work in System B is beyond the scope of this exploratory project. These remain for future work.

### 6.2.2 System B control structure

System A uses a `comprehend-word` operator that creates an impasse in which the word cycle of processing happens, interspersed with `next-word` operators. Both the selection and integration phases of each construction cycle are subsumed within either a `lexical-access` or a `match-construction` operator, with grounding done separately. When this control structure was implemented, the theory of the construction cycle had not yet emerged. The simulated processing rate of System A is too slow, and one cause is that every word cycle uses three decision cycles that are needed to handle the Soar impasse for `comprehend-word`.

For System B, it was decided to eliminate `comprehend-word` and its impasse overhead and instead use an iterative approach based on the construction cycle. The construction cycle has three operators: `cxn-query` that performs the selection phase, `build-cxn` which performs the integration phase, and `ground-cxn` which does the grounding phase. Processing in a substate for these operators can be proceduralized by Soar's chunking mechanism to eventually eliminate the impasse, as shown above for `build-cxn`. So far `ground-cxn` does nothing.

Figure 6-4 shows the sequence of decision cycles needed to process the word cycle for `Go` in `Go to the kitchen`. Those in blue are essential for implementing the construction cycles, those in yellow are the overhead for each word cycle, and those in gray are operators that could be optimized away in System B. Those not highlighted are for retrieving each new input word.

<b>System A</b>	<b>System B</b>
6: 020 ( <code>next-word</code> ) #1: Go	111: 0147 ( <code>next-word</code> ) #5: Go
7: 021 ( <code>comprehend-word</code> )	112: 0148 ( <code>cxn-query</code> ) Go
8: ==>S3 ( <code>operator no-change</code> )	113: 0150 ( <code>build-cxn</code> )
9: 022 ( <code>lexical-access</code> ) Go	114: 0151 ( <code>ground-cxn</code> )
10: 023 ( <code>lookup-action</code> )	115: 0152 ( <code>cxn-done</code> )
11: 026 ( <code>retrieve-item</code> )	116: 0153 ( <code>cxn-query</code> )
12: 024 ( <code>match-construction</code> ) SimpleAction	117: 0155 ( <code>build-cxn</code> ) SimpleAction
13: 025 ( <code>comprehend-word-done</code> )	118: 0161 ( <code>ground-cxn</code> )
14: 027 ( <code>next-word</code> ) #2: to	119: 0162 ( <code>cxn-done</code> )
	120: 0163 ( <code>cxn-query</code> )
	121: 0168 ( <code>partial-match</code> )
	122: 0169 ( <code>next-word</code> ) #6: to

Figure 6-4: Control sequences for System A and System B

A careful analysis of the experimental code shows that many of the operators used could be eliminated with some fairly straightforward optimization of the code. The optimization shows an 11.8% advantage for System B on this entire sentence. This does not seem to justify the change in the control structure. In fact, restoring comprehend-word to System B and then working out how to use chunking for its substate would probably result in a greater speedup.

### **6.2.3 Using semantic memory and spreading activation**

System A does construction selection with rules generated from ECG, along with hand-built preference rules for categories of operators. System B uses a cxn-query operator to retrieve a selected construction from smem using spreading activation (SA). For this experiment, the SA algorithm for smem has been extended in two ways. An attention mechanism in WM modulates the source of spread from WM to smem. Another extension allows for multiple parallel retrievals for the same query. The details and benefits of these two extensions are explained below.

A query to smem is made using a cue, and all smem nodes that match that cue become candidates. Then SA begins using nodes in WM that were previously retrieved from smem as sources of SA to the items in smem that they came from. From those nodes, activation spreads along edges in the smem graph to other items. The strength of the source of spread from a node in WM in System B is based on the new attention mechanism. Of the candidate nodes, the ones with the highest activation are retrieved, based on a parameter to determine how many are retrieved. System B uses this SA mechanism for the selection phase of each construction cycle to retrieve the representation of a construction from smem. Some other features of smem and SA in Soar's base architecture, including base-level activation and allowing spread beyond one step, have been turned off for this exploration in order to make the experiment more tractable.

#### *Resolving word senses from context*

Lexical items are retrieved using a cue with the orthography, or spelling, of the item, which match several senses of a word in smem. If nothing in WM sources

spread to these candidates, they all have the same activation, and an arbitrary, fixed selection is made. SA can favor one or another of these candidates based on connections from items in working memory. System B has a mechanism for using WM nodes that represent semantic context to spread activation that biases these retrievals. We have tested this with the following sentences.

*Case Study 6.3: Sentences for selecting senses of the word bank*

Bank-1	Take the <i>canoe</i> down to the <i>bank</i> .
Bank-2	Take the <i>check</i> down to the <i>bank</i> .
Bank-3	Put the <i>plane</i> in a <i>bank</i> .
Bank-4	Bounce your <i>shot</i> off the <i>bank</i> .

In these examples there are four different senses of *bank* (of at least fourteen that we have identified), and in each case there is a previous word in the sentence that indicates a situational context. In the ECG formalism, the term *evoke* is used for the ECG schemas that are created and populated when a new construction is integrated into the comprehension state. In System B, lexical items can use the *evoke* feature in ECG to define their semantic context. During the integration phase of a construction cycle, the *evoke* feature causes an instance of an ECG schema to be created as a context. That item in WM can be a source of SA that biases the retrieval of subsequent lexical items.

Figure 6-5 shows the ECG items in smem relevant to this experiment. On the left are the lexical constructions for the senses that we care about of the context nouns *shot*, *canoe*, *check*, and *plane*, while on the right are lexical constructions for four noun senses of *bank*. Each noun has a schema, not shown, that defines its basic meaning. There are also a set of schemas called *contexts*, and the lexical constructions for the context nouns in our example sentences use the *evoke* feature to cause retrieval of these context schemas. Each context schema also has an edge in the smem graph to the related sense of *bank*.

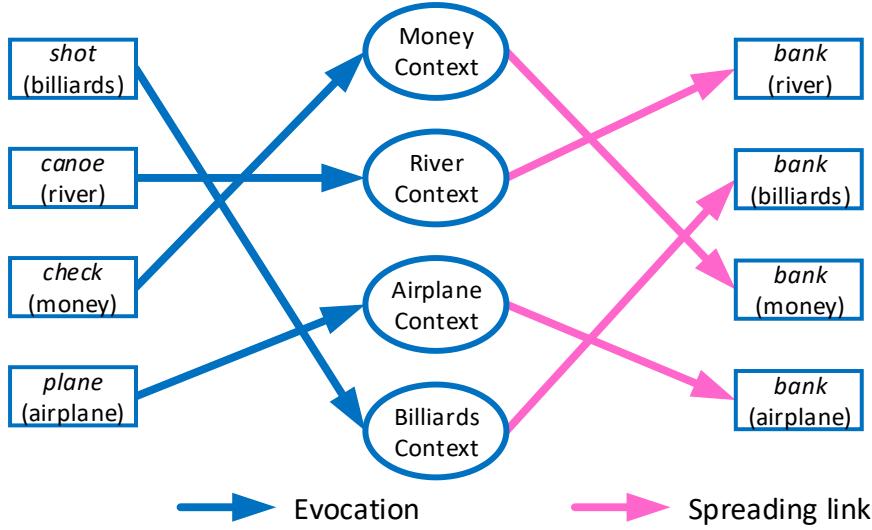


Figure 6-5: ECG lexical constructions and context concepts

System B's processing of our Bank-1 sentence proceeds as follows. When the selection phase performs a retrieval for *canoe*, it does a cue-based retrieval from smem using the spelling of that word. This retrieves the lexical item for *canoe* shown in the figure. In the integration phase, the retrieved item evokes a RiverContext schema, using a cue-based retrieval with the name of that schema as the cue. Spreading does not influence these two retrievals, since the cues are unique. During retrieval for *bank*, however, all four senses become candidates because they all match the cue. Simultaneously, the instance of the RiverContext schema in WM sources spread to its base node in smem, which in turn spreads activation to the river sense of *bank*, which then has the highest activation among the candidates and is thus retrieved, giving the correct answer.

#### *Selecting composite constructions*

System A uses a proposal rule generated from ECG for each composite construction, along with hand-built preference rules based on properties of the constructions to select among competing proposals. For System B, the Rosie grammar instead generates composite constructions in smem. The cue used by cxn-query for composite constructions makes all of them initial candidates. The

nodes in the comprehension state in WM can spread activation to a large number of them, including ones that don't match the current stack.

Our experimentation with a few example sentences showed that selecting the right composite construction requires extensions to the architecture. All the nodes in the comprehension state are sources of spread to smem, whereas the correct composition at any moment depends only on the few near the root. If all these nodes source an equal amount of spread, many spurious retrievals result. The new attention mechanism makes the amount of spread sourced from each WM node depend on its distance from the root, so that only the nodes most relevant to the current selection produce significant spread in smem. Even then, many activated constructions in smem do not match the stack. A mechanism is needed to select the one that best matches. Multiple parallel retrievals provide an opportunity to do this with rules that fire in parallel.

Therefore, selection of the right composite construction involves retrieving several candidates that have significant activation and then using rules to see which of these actually match the stack. For multiple matches, preference rules similar to those in System A make the final selection. The `cxn-query` operator for composite constructions uses several steps, all in a single decision cycle:

1. Retrieval from smem in Soar requires placing a command on a buffer called the *smem link* to initiate the retrieval. An apply rule places such a retrieval command, with an *attention* pointer to the root node of the comprehension state, and a *breadth* number, N, to specify how many items to retrieve.
2. The smem retrieval algorithm assigns *attention* to nodes in WM that can source spread to smem. These are the nodes of the comprehension state as described in Chapter 4. A value of 1.0 is assigned to the root node, moving out from there to other nodes in the WM tree, reducing the value by a factor of 0.5 for each step away from the root.
3. The spreading activation algorithm computes the activation of nodes in smem, with the attention value of each node in WM as the source of spread coming from that node.

4. Based on the activation levels computed, the top N items in smem are retrieved to the smem results link in WM. Retrieving N items is called *breadth retrieval*, and is another new extension to the architecture.
5. Hand-built proposal rules with general knowledge of composite constructions in smem propose a build-cxn operator for each retrieved item that fully matches the current comprehension state, and a partial-match operator for each one that partially matches.
6. Preference rules rank all the build-cxn operators as in System A for match-cxn, preferring a larger span and a larger number of lexical constituents.
7. There are partial-match operators to allow for a future system for prediction of subsequent input. Currently, these operators do nothing.

At the end of this decision cycle, Soar selects the most preferred build-cxn operator, if any. If none was found, the word cycle is terminated. Without the parallelism provided by the breadth retrieval and the proposal and preference rules that fire in parallel, it would be necessary to retrieve and vet one candidate at a time. This would take many decision cycles, seriously degrading the system's real-time performance. Consider an example we have seen before.

#### *Case Study 6.4: Spreading activation for a local repair*

B-020	Pick up the green block on the stove. command(op_pick-up1, 015)
-------	--

Figure 6-6 shows the last three construction cycles for this sentence. In cycle 13, both a C construction and an R construction match the state. Choosing the R option to get to cycle 14 involves discarding the previously-built C node.

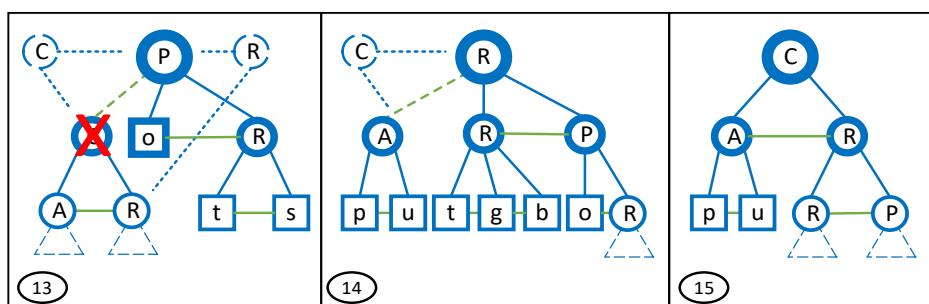


Figure 6-6: Composite selection for a local repair

Figure 6-7 shows cycle 13 in more detail, with the nodes annotated with the attention values they will get in step 2. An important feature of this figure is the way the R construction option reaches down to the other R in the third level of the tree. If a build-cxn is selected in this cycle, node 8 with the previous C is automatically discarded without any explicit *snip* operation.

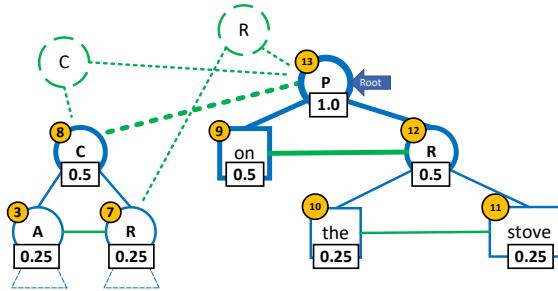


Figure 6-7: Spread of attention in working memory

Given the attention values shown in Figure 6-7, activation spreads to many composite constructions in smem in step 3. Table 6-4 shows the results for the 10 smem items with the most activation. The Items Match column shows how well each one matches in step 5, with green highlighting for the two full matches. A build-cxn operator will be proposed for each of these two.

Table 6-4: Composite selection based on spreading activation

	WM nodes that source spread							Total Spread	Items Match
	1	3	7	8	9	12	13		
RefIsPrepPhrase	0.125	0.250	0.250	0.500	0.500	0.500	1.000	1.750	1/3
RefExprPrepPhrase			0.250			0.500	1.000	1.750	2/2
RefIsNotPrepPhrase			0.250			0.500	1.000	1.750	1/3
IsObjectRelation			0.250			0.500	1.000	1.750	1/3
ImperativeWithLocation				0.500			1.000	1.500	2/2
DiTransitiveCommand			0.250	0.500		0.500		1.250	1/2
SimplePrepPhrase			0.250		0.500	0.500		1.250	0/2
TurnObjectOn			0.250		0.500	0.500		1.250	0/3
SubjectVerb	0.125	0.250	0.250			0.500		1.125	0/2
TransitiveCommand	0.125	0.250	0.250			0.500		1.125	0/2

Step 6 must select between the build-cxn for RefExprPrepPhrase, the R option, and ImperativeWithLocation, the C option. Although no specific snip is

now required, at this point a preference rule is needed that has the knowledge that, given the semantics of this sentence, the R option is preferred. Thus System B has eliminated the explicit snip, but some knowledge beyond ECG is still required in the form of a hand-built preference rules.

The evaluation subsection below shows how well the new attention mechanism and breadth retrieval feature added to the architecture serve for comprehending sentences, and what is still missing in this experimental version. These mechanisms seem general enough to be useful for any task that involves building meaningful hierarchies, such as analyzing any structured sequence of events, like music, or even constructing a scene graph from visual input.

#### **6.2.4 Learning procedural skill**

The next section discusses a theory of language acquisition from experience that fits with Lucia. In this theory, the meanings of language forms are derived from individual experiences, and these form-meaning mappings are generalized over time into declarative knowledge. In processing this declarative knowledge, new procedural knowledge is learned, developing an automatic skill to replace the declarative knowledge. System B models using declarative knowledge for comprehension and converting that knowledge into procedural skill.

In System A, multiple production rules fire during the decision cycles for lexical-access and match-construction to perform the integration phase. The integration of each construction cycle is implemented in System B with a build-cxn operator for both lexical and composite constructions. It creates a substate where a number of operators perform specific pieces of the integration. Soar's chunking mechanism then learns production rules to replace these substates.

Figure 6-8 illustrates the results of this process, showing parts of a trace of the Soar processing for the sentence B-020 discussed above. Some instances of build-cxn take many decision cycles, while others take only one. The create-cxn operators in the substates take the data retrieved from smem and use it to create a node in the comprehension state. A subcase-of-query followed by a generalize-cxn together perform the generalization of a construction based on

subcase of statements in the ECG. A `compose-cxn` operator removes the constituents of a new composite construction from the stack and connects them as children of the new composition.

Soar's chunking mechanism is turned on in the `build-cxn` substate, and it learns new production rules that replace entire operators. As more processing is done, fewer operators are needed for each `build-cxn`, until there is no substate at all. The figure only shows a few examples. In the case of STOVE, for example, all the substate operators that were needed for BLOCK have already been chunked, and these chunks to all the work for its `build-cxn` in the apply phase for that operator with now need for a substate.

Construction selected: THE	Build a SpecPropNoun construction.
48: 0: 073 ( <code>build-cxn</code> )	84: 0: 0125 ( <code>build-cxn</code> )
49: ==>S: S7 (operator no-change)	85: ==>S: S10 (operator no-change)
50: 0: 074 ( <code>create-cxn</code> )	86: 0: 0128 ( <code>create-cxn</code> )
51: 0: 075 ( <code>subcase-of-query</code> )	87: 0: 0129 ( <code>subcase-of-query</code> )
52: 0: 076 ( <code>generalize-cxn</code> )	88: 0: 0131 ( <code>generalize-cxn</code> )
53: 0: 077 ( <code>subcase-of-query</code> )	89: 0: 0132 ( <code>subcase-of-query</code> )
54: 0: 078 ( <code>generalize-cxn</code> )	90: 0: 0133 ( <code>generalize-cxn</code> )
55: 0: 079 ( <code>subcase-of-query</code> )	91: 0: 0130 ( <code>compose-cxn</code> )
56: 0: 080 ( <code>generalize-cxn</code> )	92: 0: 0134 ( <code>build-cxn-done</code> )
57: 0: 081 ( <code>ground-cxn</code> )	93: 0: 0135 ( <code>ground-cxn</code> )
...	...
Construction selected: BLOCK	Construction selected: THE
74: 0: 0114 ( <code>build-cxn</code> )	120: 0: 0174 ( <code>build-cxn</code> )
75: ==>S: S9 (operator no-change)	121: 0: 0175 ( <code>ground-cxn</code> )
76: 0: 0115 ( <code>create-cxn</code> )	...
77: 0: 0116 ( <code>subcase-of-query</code> )	Construction selected: STOVE
78: 0: 0117 ( <code>generalize-cxn</code> )	127: 0: 0191 ( <code>build-cxn</code> )
79: 0: 0118 ( <code>subcase-of-query</code> )	128: 0: 0192 ( <code>ground-cxn</code> )
80: 0: 0119 ( <code>generalize-cxn</code> )	...
81: 0: 0121 ( <code>ground-cxn</code> )	Build a SpecNoun construction.
...	131: 0: 0196 ( <code>build-cxn</code> )
	132: 0: 0198 ( <code>ground-cxn</code> )

Figure 6-8: An example of skill learning

This process turns relatively slow deliberation into fast, skilled processing. It may be possible to apply this approach to the selection and grounding phases as well, but so far it has only been implemented for integration.

### **6.2.5 Evaluation of System B**

The implementation of System B to date is limited. It uses the same ECG grammar as System A and implements the same conceptual I3P algorithm, but with no grounding or sentence interpretation yet. Adding these parts would not change the processing described here, and would just involve modifying the hand-built code from System A to work in the System B control structure. The coding and debugging of System B for the Rosie corpora was based entirely on three sentences:

Go to the kitchen.  
Pick up the green sphere.  
Pick up the green block on the stove.

These three sentences served to test lexical and composite selection, construction integration including skill learning, and the overall control structure. Following development with only these three sentences, System B was tested on the complete Baseline corpus, with the results shown in Table 6-5. A surprising number of sentences were processed correctly, suggesting that the approach developed from those three sentences was pretty general.

Table 6-5: System B parsing statistics

Total Baseline sentences	207	100%
Baseline sentences correct	152	73.4%
Total lexical failures	40	19.3%
Total composite failures	15	7.2%

Analysis shows that all the failures are due to features of System A not yet transferred to System B. Lexical failures are due to the following three missing features: an operator to handle an unknown word not in the grammar; lexical-selection rules to select among word senses based on syntactic context; and a mechanism for handling multi-word lexical items such as *to the left of*. These features could be added to System B by modifying the System A rules to work within the System B control structure. Table 6-6 summarizes their effects.

Table 6-6: System B lexical failures

Total Baseline sentences	207	100%
Lexically correct sentence	167	80.7%
Total lexical failures	40	19.3%
No handling of unknown words	10	4.8%
No lexical-selection rules	18	8.7%
No multi-word lexicals	12	5.8%

When a lexical construction failed to be processed correctly, that whole sentence failed. The table explains how many lexical failures were caused by each of the System A features not yet implemented in System B.

For the 167 sentences that are lexically correct, successful parsing depends on all the composite constructions working properly. Three System A features not yet in System B caused failures at this level: a lack of implementation of the ECG option to have optional constituents, a pair of constructions that violate the convention of having no more than three constituents, and the lack of preference rules to handle multiple matching constructions. Resolving these three problems will require modifying the ECGtoSoar translator and composite selection code to handle optional constituents, replacing the two constructions that have four constituents with three-constituent alternatives, and modifying the preference rules to work in System B. Table 6-7 shows the statistics of these failures.

Table 6-7: System B composite failures

Baseline sentences with no lexical failures	167	100%
Sentences with no composite failures	152	91.0%
Total composite failures	15	9.0%
No handling of optional constituents	4	2.4%
Violation of three-level limit	5	3.0%
No specific preference rules	6	3.6%

These results were surprisingly good. To understand them further, we did some ablation studies. Since the two architectural extensions affect only the

selection of composite constructions, we tested these 167 sentences again without one of the extensions, as Table 6-8 shows.

Table 6-8: Results of ablation studies of System B

Configuration	Number correct	% of 167
Full System B	152	91.0
Without attention	11	6.6
Without breadth retrieval	75	44.9

With attention turned off, only 11 of the sentences were parsed correctly. Of these, seven are single-word sentences, and the other four have only two words. This shows that without focusing attention on the nodes near the root, when the tree gets larger than a few nodes SA activates many constructions based on nodes throughout the tree. This causes activation and retrieval of many candidates that cannot possibly match the state. This interference overwhelms the retrieval of the correct composite constructions and the entire algorithm fails.

With breadth retrieval turned off, the results are better, but worse than with it on. Of the 75 sentences parsed correctly, 63 were from four words long or less. This shows that attention by itself makes a significant contribution, but still many constructions that do not match the state are retrieved, as Table 6-4 shows. Without breadth retrieval, it is rare that the correct construction is retrieved once a large tree activates many candidates.

The above results show that System B functions well for syntactic analysis of the Baseline corpus. In addition, System B addresses the limitations of System A listed at the beginning of this section in several ways: 1) It demonstrates contextual bias for lexical selection. 2) A small speedup was achieved, which it turns out may not justify the change in the control structure. 3) The new attention mechanism provides architectural support for the principle that only three levels of the tree need to be accessible for processing, since beyond this WM nodes source very little spreading activation. 4) Language knowledge in semantic memory, and skill learning based on that knowledge, demonstrate

mechanisms that could be useful for parts of a system for language acquisition like that described in the following section.

System B serves as a proof of concept of a model of a general-purpose attention mechanism for implementing retrievals from semantic memory based on the current state of working memory, and also for a model of transforming declarative knowledge of language into a procedural skill. These ideas suggest possible future work on both human language comprehension and human cognition in general.

### **6.3 Language Acquisition from Experience**

We have presented the Lucia theory of language comprehension and some of its implications. However, humans' acquisition of language is ongoing, and ongoing learning of new language would be important for how artificial agents can use language to collaborate with humans. What is missing is a theory of how the knowledge of language can be acquired from experience.

Humans learn language gradually over time through individual experiences of its use. Any human-like model of language acquisition must include an incremental process whereby individual, situated experiences draw on prior knowledge, as well as reasoning about the current situation, to produce new increments of knowledge of the meaning of language, and modify these increments of knowledge from additional experience.

Given these requirements, this section addresses the following principle of the Lucia theory:

**LAE: Language is Acquired from Experience** – *Knowledge of how to map form to meaning is learned incrementally from individual experiences where situational knowledge and reasoning are used to learn increments of form-meaning mapping called constructions, and these constructions are modified as indicated by further experiences.*

This section is devoted to understanding how this learning might be done computationally, as a basis for future work toward a computational model of LAE. Our treatment here is limited, focusing on learning ECG-like constructions that Lucia can use to map syntactic forms to their schematic meanings. Learning

new lexical items, grounding, sentence interpretation, and knowledge of processing are beyond our scope here. The section is divided into three parts: a definition of the problem based on how Lucia's ECG grammar was developed, an analysis of related theoretical work that could contribute to developing a computational model of acquisition, and the presentation of a theory of how a computational model of LAE might be added to Lucia in Rosie.

### **6.3.1 Defining the problem**

Lucia's ECG grammar for Rosie has been developed by hand by a human developer. Nevertheless, it shows a model of how an agent might automatically learn new constructions. The development process consisted of many specific cases of the reasoning needed to add an increment of new knowledge to the grammar. Each case involved a particular sentence that failed to parse along with its intended meaning, and the developer reasoned over this information to discover a new construction that would solve the problem. The overall development involved several hundred individual cases, showing that reasoning based on individual language experiences in a situational context can be used to learn a complex grammar. This experience sheds light on how to address the problem of building LAE into an autonomous agent.

We would like the agent itself to do this learning, with the help of a human instructor but not of a code developer. For each case the agent must do reasoning based on the situation, perhaps including further interaction with the instructor, to make a reasonable hypothesis about the intended meaning. Then further reasoning is needed to derive a construction that maps the given form to the hypothesized meaning. Over time, multiple similar learning episodes can be generalized into more entrenched declarative knowledge of the construction, and processing of this declarative knowledge can convert it into procedural skill. What is missing is a computational model of how this reasoning and learning can be built.

### **6.3.2 Related theoretical work**

A long tradition in generative linguistics is the idea of a *Universal Grammar (UG)* (Chomsky, 1965, 2017), postulating that children are born with an innate language faculty, and that a child learns the details of their particular language within the constraints of that universal framework. Other researchers strongly disagree with this approach (Dąbrowska, 2015).

In accordance with the computational cognitive modeling approach of this thesis, we take a practical approach to developing a model of how a cognitive agent could acquire composable knowledge of meaning (CKM) within an incremental, immediate interpretation processing (I3P) algorithm capable of using general cognitive mechanisms (GCM) to achieve embodied, end-to-end comprehension (E3C). This approach is based on acquiring from many individual experiences the knowledge that, when added to the mechanisms of general cognition, can produce comprehension behavior. We want to acquire, piece by piece, increments of knowledge of how to map linguistic forms to meanings, represented as constructions and schemas, that are grounded to the world knowledge of an agent that can act in the world. This subsection reviews several lines of research on language acquisition from a cognitive linguistics viewpoint that can help develop such a theory for a computational cognitive model of LAE.

#### *Acquisition from experience*

The generative approach to acquisition attempts to explain how to learn a set of formal rules capable of generating all possible grammatical sentences in a language. Chomsky (1980) coined the term “poverty of the stimulus,” arguing that the data available to children is insufficient to learn the formal generative grammar he assumes we have, since the data includes few negative examples.

Constructionist researchers take a usage-based approach. Krashen (1985, 2003) bases his theory of acquisition on the “input hypothesis,” saying that acquisition only happens when the learner receives and processes “understandable input.” In such an experience, a person hears an input utterance that contains some part of its linguistic form that the person does not

yet understand. Given the situational context, and reasoning using non-linguistic information, the person can infer a likely meaning for the input and make a mapping between the previously not understood form and the inferred meaning. This supports a theory that increments of knowledge of language are acquired from individual usage-based experiences of understandable input.

### *Embodied word learning*

Acquisition of mapping form to meaning often begins with the meanings of words. Bloom (2000, p. 1) describes empirical studies of a phenomenon in children called *fast mapping* in which “Young children can grasp aspects of the meaning of a new word on the basis of a few incidental exposures.” He cites experiments that require simultaneous exposure to a situational context that includes sensory input from vision, touch, or other modalities, making embodiment important.

Trueswell et al. (2013) report on eye-tracking experiments with adults in a laboratory setting trying to learn to map novel words to visual images. They propose a refinement to the theory of fast mapping they call *propose-but-verify*. They argue that, rather than accumulating statistics over many trials, “successful learning in this setting is instead the product of a one-trial procedure in which a single hypothesized word-referent pairing is retained across learning instances, abandoned only if the subsequent instance fails to confirm the pairing.” This work supports the idea of learning from individual experiences.

Fincher-Kieffer (2019) presents the idea that “language comprehension is grounded in bodily action” and “involves sensorimotor simulations.” Barsalou (2008) argues that cognition includes simulation, situated action, and bodily states. Bergen (2012) presents data on simulation in the brain during language comprehension, and these ideas were an important part of the development of the ECG theory of grammar we use in this thesis.

### *Acquiring knowledge of structure from language use*

Tomasello (2003) summarizes decades of empirical research on understanding the mechanisms children use to learn language. He describes psycholinguistic

mechanisms such as shared attention, analogy, and distributional analysis that enable children to learn words and constructions for referring expressions and verb argument structures. Kuhl (2000) reviews empirical studies of language acquisition in young children and concludes in part “infants initially parse the basic units of speech allowing them to acquire higher-order units created by their combinations.”

In addition to this empirical research, work on learning complex constructions has been done in the ECG community. Feldman (2006) describes a general theory of how complex constructions can be learned and represented. Chang (2008) explores “the view that grammar learning is driven by meaningful language use in context” by building a “cognitively motivated and computationally precise” model using the ECG formalism that integrates language learning and use. Mok (2009) “describes a model of child grammar learning” using ECG “that demonstrates how the problem of impoverished input is alleviated through bootstrapping from the situational context.” Their work does not include incremental processing or cognitive modeling, but nevertheless can be instructive in developing a theory of LAE in Lucia. Figure 6-9, taken from Mok (2009), illustrates schematically the process of learning from a particular language use episode as proposed by Feldman, Chang, and Mok.

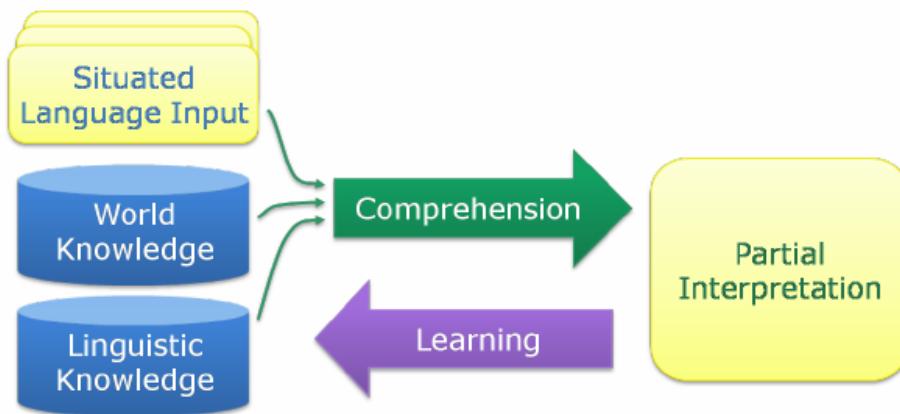


Figure 6-9: Learning from situated language episodes (from Mok, 2009, p. 8)

### *Generalization*

Tomasello (2003) discusses several psycholinguistic processes that are essential to language acquisition, including: *schematization* and *analogy* to create abstract constructions from specific instances of language use, *distributional analysis* to learn categories of linguistic elements, and *entrenchment* and *competition* to “constrain their abstractions to those that are conventional in their linguistic community.” Goldberg (2019) carries these ideas further and describes in some detail how they can be applied to learning word meanings, abstract categories, and linguistic conventions. She describes a theory in which instances can “cluster together along certain dimensions within our hyper-dimensional associative memory.” She describes in some detail how this clustering could work computationally, and how entrenchment, competition, and distributional analysis work together to learn not only abstract constructions but also arbitrary linguistic conventions. However, as far as we know, no actual computational models of these theories have been built, yet.

### *Acquiring skill*

Suppose we have a computational model that can learn abstract grammatical constructions through generalizing over individual language experiences, and that this process yields a declarative representation of these constructions. Real-time performance requires a process for turning this declarative knowledge into procedural knowledge, which can be processed much faster.

Anderson (2007, p. 94) describes skill acquisition, saying that declarative knowledge, accessible to consciousness, has been replaced by “the acquisition of new procedures (production rules in ACT-R).” He describes the process of *production compilation* by which procedures are learned, and a series of fMRI experiments that confirm his theory of how production compilation works in the brain. Laird (2012) describes learning in Soar, centered on a mechanism called *chunking*, which converts a result found in a substate into a production rule. He also mentions how Soar’s episodic memory can provide a memory of previous experiences to serve as instances for learning mechanisms, thus supporting

incremental learning. Young and Lewis (1999) provide a detailed discussion of how Soar's chunking can be used to learn new long-term procedural knowledge. Stearns (2021) develops a theory in Soar for combining primitive operators to incrementally refine a cognitive model, and relates this to "the three-phase learning theory from psychology," referring to Fitts and Posner (1967).

### **6.3.3 Toward a computational cognitive model of LAE**

Given this theoretical background, including much empirical data from psychological experiments, linguistic theory, and experience with applying cognitive architectures to complex learning tasks, our challenge is to develop a computational cognitive theory of how language can be acquired. We propose a three-phase theory of how humans might acquire language. This theory is hypothetical, it has not been implemented, and we describe it only at a conceptual level. However, it is grounded in both our experience with Lucia and the related research described above. The three phases of the theory are shown schematically in Figure 6-10.

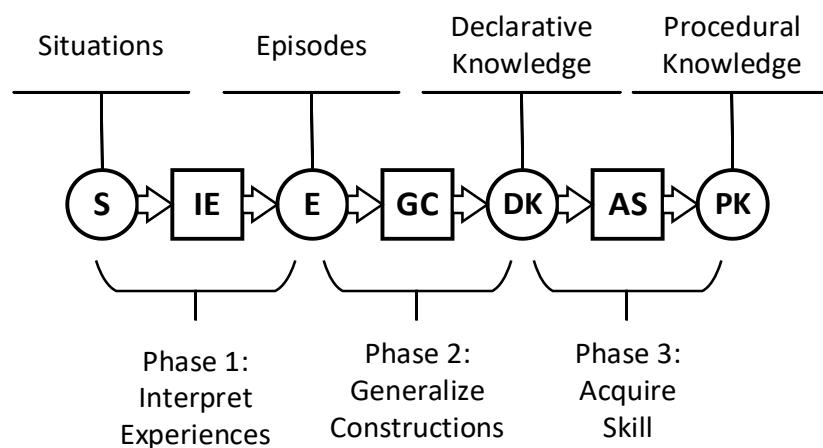


Figure 6-10: The three-phase theory of language acquisition

Phase 1 deals with processing individual episodes of language experience where a learner does not yet have constructions that fit all or part of an input utterance. Without sufficient knowledge the agent cannot comprehend that utterance using its automatic comprehension process, which corresponds to the normal Lucia processing of a sentence. However, typically a person has other

knowledge about the current situation, and can use deliberate reasoning to infer a likely meaning for that sentence. Of course, this deliberate reasoning will take more time than automatic comprehension.

Now the agent has both an input utterance and a likely meaning for it. This form-meaning pair is saved in episodic memory, and then further processing can hypothesize one or more new constructions that would make it possible to use normal processing to compute the meaning of the sentence. This can be thought of like the propose-but-verify heuristic Trueswell et al. (2013) propose, except that it can be used for both lexical and composite constructions.

Phase 2 applies once a new construction has been hypothesized. The new construction is compared to others already saved as declarative knowledge, which, if we follow Goldberg's (2019) theory, would be stored as clusters of previous episodes along with an abstraction over that cluster stored as declarative knowledge. If the new construction is similar to an existing cluster, it is added to that cluster. Otherwise, a new cluster is created for it. Using the processes of entrenchment and competition (Goldberg, 2019; Tomasello, 2003), over many episodes a commonly used construction becomes more and more solidly entrenched, becoming declarative knowledge in our model. Tomasello (2003, p. 139) further describes a sequence of more and more abstract stages of children's understanding of specific linguistic forms.

Phase 3 is the process of creating skills which are automatic and unconscious through repeated processing using declarative knowledge of constructions. Declarative processing is deliberative and slow, involving both many operations over time and retrievals from long-term declarative memory. As processing of a construction is repeated, the learning mechanisms in a cognitive architecture, such as Soar, create procedural knowledge for that construction, thus greatly speeding up the process.

Overall, this theory provides an incremental process whereby individual, situated experiences can draw on prior knowledge, as well as reasoning about the current situation, to produce new increments of knowledge of language, and modify these increments of knowledge from additional experience. Over time and

much experience these three phases of processing can lead to adult language comprehension ability. Chapter 7 suggests some future work that might be done to implement this theory.

## **Chapter 7 Conclusions**

This chapter is a brief summary of what has been accomplished in this thesis, the how the Lucia model can be a basis for future research, and some lessons learned about how to do research in Artificial Intelligence and Cognitive Science, even helping tie these two fields together more.

### **7.1 What has been accomplished**

At the beginning of this project, ECG looked like a promising way to represent knowledge of the meaning of language, but no theories existed for how to use it with single-path incremental processing, for how to ground its meanings to agent world knowledge for end-to-end comprehension, or for how to implement its comprehension in a cognitive architecture. The preceding chapters have shown that, in developing Lucia according to the qualitative goals outlined in Chapter 1, much has been accomplished to overcome these limitations.

#### **7.1.1 E3C: Embodied End-to-End Comprehension**

The Lucia comprehension system has been built within the Rosie embodied agent, as explained in Chapter 2. It produces internal meaning messages that are grounded and actionable for several hundred sentences used for teaching Rosie its ITL tasks. These messages have been tested and shown to match a pre-defined gold standard, and Rosie has in fact performed many of its learned tasks using the messages produced by Lucia. From this we have demonstrated that a system built around the constraints of CKM, I3P, and GCM can produce grounded, actionable sentence meanings in an embodied agent.

The original work on ECG (Bryant, 2008; Feldman et al., 2009) was not in an embodied agent. Later ECG work (Eppe, Trott, & Feldman, 2016; Eppe, Trott, Raghuram, et al., 2016) added a back end to Bryant's ECG parser to drive a robot, but the ECG system itself was not embodied and did not do grounding or

produce actionable messages. The FCG system (Steels, 2013) does work within a robot, but it does not interpret human natural language to produce messages that the robot can use to learn and perform new tasks. As far as we know, Lucia is the only system that uses some form of construction grammar (CxG) to do full E3C as we have defined it here.

### **7.1.2 CKM: Composable Knowledge of the Meaning of Language**

Using the Embodied Construction Grammar (ECG) formalism, a grammar has been developed to cover all the lexical items, composite constructions, and meaning schemas needed to process correctly sentences in the three Rosie corpora presented in Appendix 1. This grammar provides mapping of form to meaning in increments called constructions. Meanings are defined for constructions at all levels of language structure, from words to phrases to clauses to complete sentences. Lucia uses these constructions to perform its comprehension, using small packages of form-meaning mapping that can be composed in many ways. Measures of the generality of the grammar show that it can process orders of magnitude more sentences than were used to develop it.

Previous work on ECG (Bryant, 2008; Dodge, 2010; J. Feldman et al., 2009) focused on complex verb-argument structures. Instructing Rosie does not involve sentences of this sort, but does require complex grammar for referring expressions and conditional sentences. Lucia demonstrates that construction grammar theory, and ECG in particular, can represent form-meaning mappings for the language needed to enable Rosie to learn the tasks that it has been applied to. An area for future research would be to determine whether the pre-existing grammars developed in ECG can transfer to the approach described in this thesis.

### **7.1.3 I3P: Incremental Immediate Interpretation Processing**

As Tanenhaus et al. (1995) and others have shown, human language processing is not only incremental, word-by-word, but also does immediate interpretation by immediately grounding references in situated language to observed objects in the subject's visual field or imagined objects in simulation. Other empirical

evidence suggests that the mind simulates language meanings during processing with activation of perceptual and motor regions of the brain (Bergen, 2012). This research implies that humans interpret, or ground, individual words and phrases immediately upon processing them, requiring that the comprehension process must commit to a single interpretation at each choice point, implying a single-path process and the need to make local repairs when choices are later shown to be incorrect.

Neither the ECG, FCG, or other CxG systems we have studied do incremental processing with immediate interpretation. Lucia's processing algorithm described in Chapter 4 does do I3P processing, demonstrating that this is possible with CxG. This algorithm is built on two important principles that seem significant for future research: a comprehension state in working memory in the form of a tree structure, and a construction cycle that selects one construction at a time, integrates it with the developing comprehension state, and performs grounding. This fits well with the independently developed "chunk-and-pass" theory of Christiansen and Chater (2016).

The Lucia processing algorithm includes several mechanisms for maintaining a single path at points of ambiguity and for correcting mistaken choices when disambiguating information is available soon afterward, whereas the previous implementation of ECG (Bryant, 2008) used a global, non-incremental analysis. Parts of Lucia's algorithm are modeled after the local repair strategy Lewis (1993) used in NL-Soar. This demonstrates that ECG and I3P can work together, in an algorithm based on construction cycles and using local repair, to produce E3C.

#### **7.1.4 GCM: General Cognitive Mechanisms**

Human sentence processing is performed by the human mind/brain system. We assume that human language processing uses domain-general cognitive mechanisms, leading to a model where the mechanisms of a general-purpose cognitive architecture, along with knowledge of language, CKM, and how to process it, I3P, result in human language comprehension behavior. Lucia is

implemented in the Soar cognitive architecture (Laird, 2012), an architecture based on Newell's (1990) Soar theory and several decades of implementation, development, and application to a variety of complex tasks. Lewis's (1993) NL-Soar model is the main work on sentence processing in Soar published prior to the work for this thesis (Lindes, 2018, 2020; Lindes et al., 2017; Lindes & Laird, 2016, 2017a, 2017b). Lucia has used this architecture to implement grounded, actionable sentence comprehension using these two types of knowledge, without employing or adding any language-specific mechanisms in the underlying architecture. From this we demonstrate that knowledge of language and processing in a domain-general architecture can perform grounded, actionable language comprehension.

### **7.1.5 LAE: Language is Acquired from Experience**

Research on human language acquisition shows a complex process that is based on using situational knowledge to derive meanings for linguistic forms not yet understood. Lucia does not include an automatic algorithm for doing this acquisition. However, as described in Chapter 3, the process of development of Lucia's grammar for Rosie by a human has been very much an incremental process based on individual form-meaning examples. Chapter 6 also describes a three-phase, research-based theory of acquisition that does reasoning to deduce hypotheses from individual experiences, generalizes over these to build declarative knowledge of constructions, then learns automatic skill by comprehension processing using the declarative knowledge to learn new procedural knowledge. An experimental model illustrates the third phase of this theory. From this we suggest that Lucia as a comprehension system can form a useful basis for future research on human-like language acquisition.

## **7.2 The Model as a Basis for Future Research**

Our analysis of the design and performance of the Lucia sentence comprehension system, when compared to other related research, leads us to suggest that this thesis makes the following contributions to AI and CogSci.

### **7.2.1 A model to build AI systems on**

AI systems designed to interact with humans in real time could benefit greatly from a flexible, robust, and general-purpose language comprehension system. Lucia can be considered as an early prototype of such a system. Several challenging problems need to be addressed to meet this objective. First, a more general and consistent system for representing the grounded meanings of sentences is needed. Second, a grammar with a much larger vocabulary and broader syntactic and semantic coverage is needed. Improvement in this direction could come from incorporating linguistic knowledge from systems like WordNet (Fellbaum, 1998) or FrameNet (Ruppenhofer et al., 2006). Third, and perhaps most important, a robust system for learning new compositional knowledge of the meaning of language (CKM) from experience interacting with human interlocutors (LAE) is essential.

Work along these lines could begin with extending the explorations described in Chapter 6. Adding deliberate reasoning triggered by parsing failures to the Lucia model could make it possible to learn new constructions, resolve garden-path sentences, and learn procedures for performing local repairs. This would contribute a great deal to making a Lucia-based system more general, robust, and capable of learning additional language from experience.

### **7.2.2 A basis for research on human comprehension**

The Lucia model, including System B, postulates several novel mechanisms as part of human-like language comprehension: a tree-like structure in working memory for representing the comprehension state, limits on how much of this tree is accessible to processing and on maximum useful depth of the tree, the construction cycle model of I3P processing, the use of spreading activation biased by situational context for selection of constructions, and an attention mechanism in working memory that modulates spreading activation for retrievals from declarative memory.

Given these elements of the Lucia model, it could be used in a number of ways as a basis for further experimentation and research, particularly to explore

whether or not human experiments confirm or disconfirm the conjectures that this thesis proposes. Could a model of memory retrieval times be added to allow prediction of human reading times along the lines of Lewis and Vasishth (2005)? Could this model shed light on analysis of fMRI data on human comprehension of the sort done by Bhattsali et al. (2018)? Could the addition of the prediction model suggested in Chapter 6 provide a model that would help explain the brain data analyzed by Brennan and Hale (Brennan & Hale, 2019)? Can the work on garden-path and parsing-breakdown effects described in Chapter 6 be extended to show more clearly how working memory limits and the grounding of semantics can predict Lewis's (1993) results on human parsing limitations? Can Lucia's model of working memory and its usage contribute to unifying the many theories about human working memory (Baddeley, 2012; Brady et al., 2016; Cowan, 2017; Isbilen & Christiansen, 2020; Oberauer, 2019; Young & Lewis, 1999)? The Lucia model has potential to contribute to research in all these areas.

### **7.2.3 A basis for future architecture research**

The System B experiment was surprisingly effective in demonstrating the value of using Lucia's structured model of working memory along with an attention mechanism applied to that structure to modulate retrieval from long-term declarative memory. It also demonstrated that parallel retrievals combined with parallel processing of the results provides a way of sifting through many false positives to find the best match without extended sequential processing. Much work remains to make System B capable of doing all that System A does, let alone expanding to wider coverage and language acquisition.

System B is interesting in terms of cognitive architecture research because it incorporates a novel way of using a deliberate attention mechanism with working memory to bias the access to semantic memory, and this novel way involves extensive subsymbolic computation. The selection of constructions overlaps the boundary between symbolic and subsymbolic computation, offering an opportunity to study this boundary further. In addition to completing System B to perform Lucia's full E3C functions, there is much to do to perfect the details

of this model of memory use. Greatly expanding language coverage will certainly challenge this approach. Exploring a system for language acquisition from experience requires a model of how knowledge of the meaning of language is represented, and how that knowledge is processed. Lucia in a perfected version of System B provides such a model.

### **7.3 Lessons Learned about Doing Research**

Most of this thesis has discussed in great deal the specific system of Lucia running as part of the Rosie agent. Now we step back to consider what we have learned from a broader and more abstract perspective. The development of Lucia has taught us three important lessons about how to go about building AI systems that address problems in Cognitive Science.

#### **7.3.1 A holistic approach to development**

An embodied agent receives input from situations in the world and produces output effects to the world. Inside the agent are subsystems, not necessarily distinct modules, that process input from sensors to produce internal representations, perform reasoning on those representations relative to the agent's goals and knowledge, and perform actions both in the external world and internally to update that knowledge.

The Rosie project, as represented by Kirk (2019), Mininger (2021), this thesis, and other published papers,<sup>9</sup> has developed an integrated agent that learns to perform tasks in the world based on situated interactive instruction from a human instructor (Kirk & Laird, 2019; Mininger & Laird, 2018)(Kirk & Laird, 2019; Mininger & Laird, 2018). Lucia (Lindes et al., 2017) has been developed within the context of this embodied agent. The great strength of the Rosie project has been its holistic approach. Rosie is an embodied agent that in fact acts in the physical world, possibly in simulation. The necessity to produce meaning that Rosie can act on has been essential to the development of Lucia.

---

<sup>9</sup> See <https://soar.eecs.umich.edu/Soar-RelatedResearch>

### **7.3.2 The centrality of meaning**

Assume that an embodied, autonomous agent must have subsystems for perception, cognition, and action. The cognition subsystem incorporates three kinds of information: the agent's goals, G, the agent's knowledge and beliefs about the world, itself, and the current situation, K, and representations of meaning, M, of the current situation in relation to G and K. This representation of meaning, M, is central to the cognitive reasoning the agent uses to make decisions about what action to take next to move toward its goals based on its knowledge and beliefs.

Many computational approaches to language processing produce impressive statistical results while ignoring the grounded, actionable meaning of each sentence. For a language comprehension system to be useful within an embodied agent that must act in the world based on language input, meaning is central. We can think of Rosie as the entire agent and Lucia as the part of the perception subsystem that produces meaning from language. Other parts of perception produce meaning from vision or other inputs, and Lucia must ground the meanings of sentences to the meanings of visual or other inputs. One of the great weaknesses of the Rosie/Lucia project has been a lack of sufficient attention early on to designing representations of meaning that can be shared between Lucia and the reasoning used by the rest of Rosie. This has caused a great deal of extra development work and computational overhead for the sentence interpretation part of Lucia.

### **7.3.3 Modeling human behavior qualitatively**

Much research on cognitive modeling of language processing focuses on matching quantitative measures, such as reading times (Lewis & Vasishth, 2005) and brain measurements (Brennan & Hale, 2019). The five goals for this thesis set out in Chapter 1, E3C, CKM, I3P, GCM, and LAE, are all *qualitative* measures that are characteristics of human processing. Within these qualitative constraints, we have implemented a working system that produces concrete, quantitative results. The advantage of this approach is that, if the qualitative

constraints are chosen well, the results give a more complete system that supports both understanding human processing and building AI systems.

Newell and Simon (1976) argue that “Laws of qualitative structure are seen everywhere in science” and “they often set the terms on which a whole science operates.” The five qualitative principles that form the basis of this thesis have made it possible to solve hundreds of specific problems of language comprehension. With the model we now have based on this approach, we can move forward toward refining the model to give better quantitative results while still meeting the high-level qualitative goals.

## **Appendices**

## **Appendix 1 The Rosie Corpora**

The sentences used to develop the Lucia comprehension system were taken from ITL scripts used to develop the Rosie agent. These scripts and sentences were designed by other developers for their work on Rosie, and were not influenced by the needs of Lucia. Three corpora are listed here, a Baseline used for initial development of Rosie, a Games corpus developed by James Kirk (2019) for teaching games and puzzles to Rosie, and a Robot corpus developed by Aaron Mininger (2021) teaching navigation and delivery tasks.

### **A1.1 The Baseline Corpus**

This section contains a list of the 207 sentences that were used as the Baseline corpus for developing Lucia in Rosie. These sentences were not written by the Lucia developer, but were taken from a list prepared in 2015 by other researchers working on teaching tasks to Rosie. They were chosen from that larger list to use with Lucia simply to reflect as wide a range of language phenomena as possible.

The following list is ordered in a way that reflects the development of Lucia for this corpus. One sentence at a time was added to the Baseline development set, and then all changes to the ECG grammar and Soar operators needed to make that sentence work in System A were made before going on to the next sentence in order. In addition to listing the sentences, the following table gives some rough numbers of things that needed to be added or changed to make a given sentence work in this development order. The column headings for the following table are defined as follows:

- ID** Identifier: A unique identifier for this sentence, made up of B for the Baseline corpus and the sequence number in this list. This sequence number also indicates the order in which these sentences were added to the Baseline development set.

- W** Works: A “1” in this column indicates that this sentence simply worked without any changes to Lucia or the ECG grammar when this sentence was added in order to the development set.
- L** Lexical: The number of new lexical constructions added to the ECG grammar for this sentence.
- C** Composite: The number of new composite, or in some cases general, constructions added to the ECG grammar for this sentence.
- HB** Hand Built: The number of hand-built Soar code items that had to be added or modified to make this sentence work. These items are usually the operators for doing grounding and sentence interpretation. Some modifications did not add to the number of rules, and in other cases one modification might have added several rules.

ID	W	L	C	HB	Sentence
B-001		4	4	5	The sphere is green.
B-002		2	1	4	The medium block is green.
B-003		3	0	2	The red triangle is clear.
B-004		1	0	2	The sphere is cooked.
B-005		4	2	1	The red triangles are on the pantry.
B-006		2	0	1	The stove is off.
B-007		0	0	2	The red triangle is on the stove.
B-008		1	2	1	The red triangle is on the big green block.
B-009		2	3	3	Pick up the green sphere.
B-010		1	1	2	Pick this up.
B-011		2	0	1	Pick up the purple object.
B-012		1	0	2	Pick it up.
B-013		1	0	1	Pick the unknown.
B-014		0	0	1	Pick the red box.
B-015		2	2	4	Put the green sphere in the pantry.
B-016		1	0	3	Put the green one in the pantry.
B-017		1	0	3	Put that in the pantry.
B-018		0	0	4	Put it on this.
B-019		1	1	3	Put down the green sphere in the pantry.
B-020		0	1	5	Pick up the green block on the stove.
B-021		2	1	6	Store the large green sphere on the red triangle.
B-022		0	0	3	Put this on the stove.
B-023		0	0	3	Put it on the stove.
B-024		0	0	4	Put the sphere in the red triangle.
B-025		3	2	7	Move the orange triangle on the red triangle to the stove.
B-026		1	0	5	Put the green sphere in front of the pantry.
B-027		3	0	6	Move the green sphere to the right of the garbage.

B-028		2	2	7	Pick the green block that is small.
B-029		1	0	5	Pick a green block that is small.
B-030		0	1	2	Pick the green block that is on the stove.
B-031		1	0	9	Pick a green block that is larger than the green box.
B-032		3	0	5	Move the green rectangle to the left of the large green rectangle to the pantry.
B-033		2	1	5	Drive to the wall.
B-034		2	0	5	Go to the waypoint.
B-035		2	1	3	Orient north.
B-036		1	0	2	Stop.
B-037		1	1	5	Follow the right wall.
B-038		1	0	4	Forward.
B-039		0	1	5	Drive forward.
B-040		2	0	3	Turn around.
B-041		0	0	3	Turn left.
B-042		3	2	4	Go until there is a doorway.
B-043		1	0	1	Yes.
B-044		2	1	2	Octagon is a shape.
B-045		2	0	2	Study is a location.
B-046		2	5	7	If the green box is large then go forward.
B-047		2	2	2	What is inside the pantry?
B-048		1	1	5	Where is the red triangle?
B-049		0	1	6	Is the large orange block a sphere?
B-050		1	1	3	Is the small orange triangle behind the green sphere?
B-051	1	0	0	0	Go forward.
B-052	1	0	0	0	Turn right.
B-053	1	0	0	0	Follow the left wall.
B-054	1	0	0	0	Go to the kitchen.
B-055	1	0	0	0	Drive to the kitchen.
B-056	1	0	0	0	Pick up the box.
B-057	1	0	0	0	Put down the box.
B-058	1	0	0	0	The sphere is red.
B-059	1	0	0	0	Go to the garbage.
B-060	1	0	0	0	No.
B-061	1	0	0	0	What is inside the pantry
B-062	1	0	0	0	What is on the red triangle?
B-063		0	0	4	The large one is red.
B-064	1	0	0	0	This is orange.
B-065		0	0	2	This one is orange.
B-066	1	0	0	0	The big triangle is red.
B-067	1	0	0	0	The red triangle is behind the stove.
B-068	1	0	0	0	The red triangle is to the left of the stove.
B-069	1	0	0	0	Store the sphere.
B-070	1	0	0	0	Put this in the pantry.

B-071	1	0	0	0	Put the green sphere next to the pantry.
B-072	1	0	0	0	Move the green sphere to the left of the garbage.
B-073	1	0	0	0	This is larger than the green sphere.
B-074	1	0	0	0	Pick up this.
B-075	1	0	0	0	Pick up that.
B-076	1	0	0	0	Store the pantry.
B-077	1	0	0	0	Is this a sphere?
B-078	1	0	0	0	Is the green sphere on the table?
B-079	1	0	0	0	Is the large triangle to the right of the green sphere?
B-080	1	0	0	0	Pick a green block that is on the stove.
B-081	1	0	0	0	Pick a green block that is on a stove.
B-082		0	1	2	Go.
B-083		1	0	3	Go to the location.
B-084		0	1	1	Go to waypoint.
B-085		0	1	3	Follow the right wall until there is a doorway.
B-086		0	0	1	Store the green block.
B-087		1	2	2	Green is a color.
B-088		1	1	1	Mauve is color.
B-089		1	0	0	Taupe is a color.
B-090		0	1	1	What color is the large sphere?
B-091		0	0	3	What shape is this?
B-092		0	0	2	What size is the red triangle?
B-093		0	0	1	The medium block is smaller than the large block.
B-094		0	2	3	This is a big triangle.
B-095		1	1	3	Turn on the stove.
B-096		0	1	2	What is this?
B-097		0	1	2	Is the large sphere green?
B-098		0	0	2	Is this red?
B-099		1	3	2	Go down the hall.
B-100		1	4	1	Orient s.
B-101		1	1	1	Face east.
B-102		0	0	1	Face west.
B-103		1	0	2	Done.
B-104		1	0	2	The task is done.
B-105		1	1	1	The task is finished.
B-106		1	0	1	The task is over.
B-107		10	0	1	Pick up the stapler.
B-108	1	0	0	0	Put down the stapler.
B-109		1	0	1	Find a stapler.
B-110		3	1	1	Explore until a stapler is visible.
B-111		1	0	1	You are done.
B-112	1	0	0	0	You are in the kitchen.
B-113		1	0	1	Go to the starting location.

B-114		1	1	1	Go to your starting location.
B-115		2	4	8	Drive forward one meter.
B-116		1	0	2	Deliver the box to the kitchen.
B-117		1	0	2	Take the box to the kitchen.
B-118		1	2	6	Drive slowly to the kitchen.
B-119		1	3	5	Drive until you sense a wall.
B-120		1	1	2	Drive down the hall until you reach the end.
B-121		0	0	1	Follow the right wall until you reach the end.
B-122		3	2	4	Follow the right wall until you see two doors.
B-123		1	0	2	Pick up the soda.
B-124	1	0	0	0	Put down the soda.
B-125		1	0	2	Fetch a soda.
B-126		0	0	5	The soda is in the kitchen.
B-127	1	0	0	0	Explore until you see a soda.
B-128	1	0	0	0	Explore until you see the soda.
B-129	1	0	0	0	Deliver the box.
B-130	1	0	0	0	Find the soda.
B-131		1	1	2	Fetch a soda from the kitchen.
B-132		1	2	1	Recall a soda in a location.
B-133		2	2	2	Go forward until you see an intersection.
B-134		2	0	2	Go to the conference room.
B-135		1	0	2	Pick up the trash.
B-136	1	0	0	0	Put down the trash.
B-137		1	0	1	Patrol.
B-138		1	0	2	Patrol the building.
B-139		0	0	1	Fetch the box from the kitchen.
B-140		1	0	2	Go to the office.
B-141		1	0	2	Go to the main office.
B-142		1	0	2	Go to the soar office.
B-143	1	0	0	0	Deliver the box to the office.
B-144	1	0	0	0	Put the trash in the garbage.
B-145		2	2	4	The goal is that the box is in the office.
B-146	1	0	0	0	The goal is that the soda is in the starting location.
B-147	1	0	0	0	The goal is that the stapler is in the starting location.
B-148	1	0	0	0	The goal is that the box is in the kitchen.
B-149	1	0	0	0	The goal is that the trash is in the garbage.
B-150	1	0	0	0	The goal is that the soda is in the office.
B-151		1	0	0	Clean the pantry.
B-152		1	0	0	Close the pantry.
B-153		1	0	0	Cook the red triangle.
B-154		1	0	0	Discard the large green block.
B-155	1	0	0	0	Clean pantry.
B-156		1	0	0	Open the pantry.

B-157		1	0	0	Organize the pantry.
B-158		1	0	0	Set the table.
B-159		1	0	0	Serve the red triangle.
B-160		2	0	1	Ask the question.
B-161	1	0	0	0	Say the answer.
B-162	1	0	0	0	Say the response.
B-163	1	0	0	0	Say the message.
B-164		2	1	1	Interrogate Bob.
B-165		2	1	2	Tell me the answer.
B-166	1	0	0	0	Tell Bob a message.
B-167	1	0	0	0	Recall Bob in a location.
B-168		1	1	0	All the red triangles are in the pantry.
B-169	1	0	0	0	All of the red triangles are in the pantry.
B-170		1	1	1	Remember the answer as the question.
B-171	1	0	0	0	Remember the answer as the message.
B-172	1	0	0	0	Remember the answer as the response.
B-173		1	1	1	Turn off the lights.
B-174	1	0	0	0	If the lights in an empty room are lit then turn off the lights.
B-175	1	0	0	0	Turn off the lights in the conference room.
B-176		2	0	2	Remember the current location as the starting location.
B-177		2	2	0	Throw away the trash.
B-178		1	4	2	If you see some trash then throw it away.
B-179	1	0	0	0	If you see the soda, pick it up.
B-180	1	0	0	0	If you see the soda then pick it up.
B-181		2	1	2	Wait for one minute.
B-182	1	0	0	0	Wait for two minutes.
B-183	1	0	0	0	Wait until the sphere is cooked.
B-184		1	1	0	Rosie, clean the pantry.
B-185	1	0	0	0	Rosie clean the pantry.
B-186		1	3	3	The goal is that Bob heard the message.
B-187		1	1	4	The goal is that you said the response in the starting location.
B-188		0	0	1	The goal is that the lights in the conference room are off.
B-189		0	2	3	The goal is that the box is in the kitchen and the sphere is purple.
B-190		2	2	3	The goal is that you are not holding the box.
B-191	1	0	0	0	The goal is that the box is in the kitchen and you are not holding the box.
B-192		0	2	1	An orange block is on a small location.
B-193		1	2	0	A location is not below a red block.
B-194	1	0	0	0	The purple block is not on a red location.
B-195		2	1	0	Follow these steps.
B-196		2	2	2	Check if the conference room is empty.

B-197	1	0	0	0	Check if the lights in the conference room are lit.
B-198		0	4	2	Remember if the lights in the conference room are lit as the response.
B-199		0	1	0	Clean small red triangle.
B-200		0	1	0	All red triangles are in the pantry.
B-201		0	1	2	Ask What is the message.
B-202	1	0	0	0	Ask What is the question?
B-203		1	0	1	Transport is an action.
B-204		1	0	1	In-a-row is a relation.
B-205		0	1	3	Newverb the sphere.
B-206		0	1	2	Newverb2 the sphere in the pantry.
B-207		0	1	2	Newverb3 in the pantry.

## A1.2 The Games Corpus

This section contains a list of the 172 sentences that were used as the corpus of Games scripts for developing Lucia in Rosie. These sentences were not written by the Lucia developer. The list was compiled by concatenating all the scripts for the 60 games and puzzles, removing duplicates, and then ordering them according to the ordering in which they were used for Lucia development.

The following list is ordered in a way that reflects the development of Lucia for this corpus. One sentence at a time was added to the Games development set, and then all changes to the ECG grammar and Soar operators needed to make that sentence work in System A were made before going on to the next sentence in order. Starting with G-033 an abbreviated, development process was used such that most of the sentences from then on parsed correctly but did not have the sentence interpretation working correctly. In addition to listing the sentences, the following table gives some rough numbers of things that needed to be added or changed to make a given sentence work in this development order. The column headings for the following table are defined as follows:

- ID** Identifier: A unique identifier for this sentence, made up of B for the Baseline corpus and the sequence number in this list. This sequence number also indicates the order in which these sentences were added to the Baseline development set.
  - D** Dev: A “1” used for development for Games, and it is working.
  - W** Works: A “1” in this column indicates that this sentence simply worked without any changes to Lucia or the ECG grammar when this sentence was added in order to the development set.  
If a row has nothing in either the **D** or **W** columns, it is not working yet.
  - L** Lexical: The number of new lexical constructions added to the ECG grammar for this sentence.
  - C** Composite: The number of new composite, or in some cases general, constructions added to the ECG grammar for this sentence.
- Hand-built rules have not been tabulated for this corpus.

ID	D	W	L	C	Sentence
G-001	1		3	2	The <i>name of the puzzle</i> is blocks-world.
G-002	1		1	2	<i>Load init-blocksworld.</i>
G-003	1		1	1	<i>Ok.</i>
G-004	1		2	4	You <i>can move</i> a clear block <i>onto</i> a clear object.
G-005	1			1	If a location is not below an object then it is clear.
G-006	1				The goal is that a red block is on a green block and the red block is below an orange block.
G-007		1			Done.
G-008		1			Yes.
G-009	1		2		The goal is that a <i>blue</i> block is on a purple block and the blue block is below a <i>yellow</i> block.
G-010		1			No.
G-011	1		6	4	If <i>the number of the locations between</i> a location and a <i>covered</i> location is the number of the <i>blocks</i> that are on the covered location then you can move it onto the <i>former</i> location.
G-012	1				If a location is below an object then it is covered.
G-013	1		$9^{10}$	1	There is <i>six</i> .
G-014	1				The goal is that all the blocks are on a red location.
G-015	1		1	3	If the number of the locations between a location and an <i>accessible</i> covered location is the number of the blocks that are on the covered location then you can move it onto the former location.
G-016	1			3	A location that is not below a red block is accessible.
G-017		1			There is <i>five</i> .
G-018		1			The name of the goal is three-clear.
G-019	1			1	The goal is that there are <i>three</i> clear locations.
G-020	1				You can move a block onto a location.
G-021	1				The goal is that all the red blocks are on a red location and all the blue blocks are on a blue location and all the green blocks are on a green location.
G-022	1		3	2	The <i>solution has three steps</i> .
G-023	1				You can move a clear block onto a clear object that is larger than the block.
G-024	1		2		If <i>the volume of</i> a block is <i>more than</i> the volume of an object then the block is larger than the object.
G-025		1			The goal is that there are two clear green locations.
G-026	1		1		The name of the <i>game</i> is break-through.
G-027	1		1		If a red block is <i>under</i> a clear location then you can move the block onto the location.

<sup>10</sup> For this sentence, lexical items for three through twelve were all added at once.

G-028	1		3	3	If an occupied location is <i>attackable</i> by a red block then you can <i>remove</i> a block on the occupied location and move the red block onto the <i>occupied</i> location.
G-029		1			If a location is below a blue block then it is occupied.
G-030	1		1		If a location is under an object and the location is <i>diagonal with</i> the object then the object is attackable by the location.
G-031		1			The goal is that a red location is below a red block.
G-032	1		1		If a blue location is occupied then you <i>lose</i> .
G-033	1		2	1	If a clear location is not <i>above</i> a location then you can move an <i>available</i> clear red block onto the clear location.
G-034		1			If a block is not on a location then it is available.
G-035		1			If a clear location is above a covered location then you can move an <i>available</i> clear red block onto the clear location.
G-036		1			If four of the occupied locations are in a line then you lose.
G-037	1		3		If the locations are <i>linear</i> then <i>they</i> are in a <i>line</i> .
G-038	1		1	1	The goal is that four of <i>captured</i> locations are in a line.
G-039		1			If a location is below a red block then it is captured.
G-040		1			You can move an available clear red block onto a clear location.
G-041		1			The goal is that three of the captured locations are in a line.
G-042		1			If three of the occupied locations are in a line then you lose.
G-043		1			If the locations between a clear location and a captured location are occupied then you can move an available red block onto the clear location.
G-044		1			The goal is that all locations are covered and the number of captured locations is more than the number of occupied locations.
G-045		1			If all the locations are covered and the number of occupied locations is more than the number of captured locations then you lose.
G-046	1				The name of an action is capture-location.
G-047		1			You can move a clear available red block onto a clear location.
G-048	1		1		If all the red blocks are on <i>their</i> locations then you can move a red block onto a clear location.
G-049	1		1		The name of the <i>failure</i> is opponent-three.
G-050		1			If all the red blocks are on their locations and a red block is next to a clear location then you can move the block onto the clear location.
G-051	1		1		If all the red blocks are on their locations and a red block is <i>adjacent to</i> a clear location then you can move the block onto the clear location.
G-052	1		1	2	If a location is next to an object <i>but</i> it is not diagonal with the object then it is adjacent to the object.
G-053	1		1		If a clear location is adjacent to a red block then you can move the red block onto the clear location and move a clear available blue block onto a location that <i>was</i> below the red block.
G-054		1			The solution has eleven steps.

G-055		1			The goal is that all locations are covered.
G-056	1		1		If a clear location is <i>movable from</i> a captured location then you can move a red block onto the clear location and move an available clear blue block onto the captured location.
G-057	1		1		If a location that is <i>alongside</i> a captured location is diagonal with an object that is not next to the captured location then the object is movable from the captured location.
G-058		1			You can move a clear available block onto a clear location.
G-059		1			The solution has six steps.
G-060		1			The name of a failure is place-same-diag.
G-061	1		2		If two of the <i>placed</i> blocks are <i>cross-diagonal</i> then you lose.
G-062		1			If a block is on a location then the block is placed.
G-063	1		4	1	If <i>the difference of the rows of the blocks is equal to the difference of the columns of they</i> then they are cross-diagonal.
G-064		1			The goal is that all blocks are placed.
G-065		1			The solution has four steps.
G-066	1		1		If a block is next to <i>another</i> block then you lose.
G-067		1			The solution has eight steps.
G-068	1		1		If a block is <i>jumpable</i> by another block then you lose.
G-069	1				If an object that is alongside a block is diagonal with a location that is not next to the block then the location is jumpable by the block.
G-070	1		3		If two of the blocks are placed and they <i>have the same row</i> then you lose.
G-071	1		1		If two of the blocks are placed and they have the same <i>column</i> then you lose.
G-072		1			The name of the action is slide-block.
G-073		1			If a block is adjacent to a clear location then you can move the block onto the clear location.
G-074	1				The goal is that there are eight <i>matched</i> locations.
G-075	1			2	If the color of a location is the color of the block that is on the location then the location is matched.
G-076	1		1		The goal is that there are eight <i>matching</i> locations.
G-077	1		1	1	If <i>the value of</i> a location is the value of the block that is on the location then the location is matching.
G-078	1		1		The goal is that there are <i>fifteen</i> matching locations.
G-079		1			The goal is that there are five matched locations.
G-080		1			If a blue block is adjacent to a clear location then you can move the block onto the location.
G-081	1		1	1	If a covered location is between a clear location and an occupied location then you can move a block on the covered location onto the clear location <i>plus</i> move a block on the occupied location onto the covered location.
G-082		1			The goal is that a blue block is on a blue location.

G-083		1		The goal is that all locations are covered and a red block is adjacent to a green location.
G-084				You can write a number that is between one and four onto an empty location.
G-085				If the value of a location is absent then the location is empty.
G-086		1		If the value of a location is the value of an object that is next to the location then you lose.
G-087				The goal is that all locations are filled.
G-088				If the value of a location is more than zero then the location is filled.
G-089				If the row of a location is the row of a location and the value of the former location is the value of the latter location then you lose.
G-090				If the column of a location is the column of a location and the value of the former location is the value of the latter location then you lose.
G-091				If the section of a location is the section of a location and the value of the former location is the value of the latter location then you lose.
G-092				You can write a number that is between one and nine onto an empty location.
G-093				If two of the green locations are identical and they have the same row then you lose.
G-094				If the locations have the same value then they are identical.
G-095				If two of the green locations are identical and they have the same column then you lose.
G-096				If the green locations in a file are filled and the sum of the values of the file is not the value of a location that is atop the file then you lose.
G-097				If the blocks have the same column then they are in a file.
G-098				If a block is blue and the column of the block is the column of a location then the block is atop the location.
G-099				If the green locations in a rank are filled and the sum of the values of the rank is not the value of a location that is aside the rank then you lose.
G-100				If the blocks have the same row then they are in a rank.
G-101				If a block is yellow and the row of the block is the row of a location then the block is aside the location.
G-102				If the locations in a group are filled and the sum of the values of the group is not ten then you lose.
G-103				If the blocks have the same color then they are in a group.
G-104				If the locations in a region are filled and the sum of the values of the region is not ten then you lose.
G-105				If the blocks have the same section then they are in a region.
G-106				If the color of a location is the color of a location and the value of the former location is the value of the latter location then you lose.
G-107				If the green locations in a file are filled and the sum of the values of the file is not the value of a location that is beneath the file then you lose.

G-108				If a block is blue and the column of the block is the column of a location then the block is beneath the location.
G-109				If the green locations in a rank are filled and the sum of the values of the rank is not the value of a location that is beside the rank then you lose.
G-110				If a block is blue and the row of the block is the row of a location then the block is beside the location.
G-111				If a covered location is between a clear location and a blue block then you can move a block on the covered location onto the clear location plus move the blue block onto the covered location.
G-112	1			The goal is that a red block is on a red location and a green block is on a green location.
G-113	1			If the value of a location is the value of another location then you lose.
G-114				If the locations in a grouping are filled and the sum of the values of the grouping is not the value of a block that is above the grouping then you lose.
G-115				If a block is filled and their locations are under the block then they are in a grouping.
G-116				If the blue locations are filled and the sum of the values of them is not ten then you lose.
G-117				If the red locations are filled and the sum of the values of them is not nine then you lose.
G-118				If the green locations are filled and the sum of the values of them is not twenty-six then you lose.
G-119				You can write a number that is between one and twelve onto an empty location.
G-120				If the value of a green location is the value of another green location then you lose.
G-121				You can move a child that is on the current bank and another child that is on the current bank and the boat onto the opposite bank.
G-122				If an object is below a boat then it is current.
G-123				If a location is not below a boat then it is opposite.
G-124				You can move the boat and a block on the current bank onto the opposite bank.
G-125	1			The solution has nine steps.
G-126				The goal is that all the blocks are on a destination bank.
G-127				You can move a boat onto the opposite bank.
G-128				If a red block is on an opposite bank and a yellow block is on the opposite bank then you lose.
G-129				If a green block is on an opposite bank and a yellow block is on the opposite bank then you lose.

G-130				You can move a block that is on the current bank and another block that is on the current bank and the boat onto the opposite bank.
G-131		1		The solution has five steps.
G-132				If a woman is on a bank and the husband of the woman is not on the bank and a man is on the bank then you lose.
G-133				If the last-name of a woman is the last-name of a man then the man is the husband of the woman.
G-134				If a man is on a bank and the wife of the man is not on the bank and another woman is on the bank then you lose.
G-135				If the last-name of a man is the last-name of a woman then the woman is the wife of the man.
G-136				You can move the boat and a person on the current bank onto the opposite bank.
G-137				If an object is a block and it is not a boat then it is a person.
G-138				You can move a person that is on the current bank and another person that is on the current bank and the boat onto the opposite bank.
G-139				If a stranded actor is accompanied by another manager then you lose.
G-140				If a person is a star then they are an actor.
G-141				If a person is a cylinder then they are a manager.
G-142				If an actor is on a bank and the agent of the actor is not on the bank then the actor is stranded.
G-143				If the color of a manager is the color of an actor then the manager is the agent of the actor.
G-144				If a person is on a bank and a manager is on the bank then the person is accompanied by the manager.
G-145				If the number of cannibals on a bank is more than the number of missionaries on the bank then you lose.
G-146		1		If a clear location is jumpable by a block then you can move the block onto the clear location.
G-147		1		The goal is that all the red blocks are on the red locations and all the blue blocks are on the blue locations.
G-148				If a clear location is to the right of a blue peg then you can move the peg onto the location.
G-149				If an object is a block then it is a peg.
G-150				If a clear location is to the left of a red peg then you can move the peg onto the location.
G-151				If a red peg is to the right of a peg and the peg is to the right of a clear location then you can move the red peg onto the clear location.
G-152				If a blue peg is to the left of a peg and the peg is to the left of a clear location then you can move the blue peg onto the clear location.
G-153	1			The solution has <i>twenty-four</i> steps.

G-154				If a clear location is to the right of a toad then you can move the toad onto the location.
G-155				If an object is blue and the object is a block then it is a toad.
G-156				If a clear location is to the left of a frog then you can move the frog onto the location.
G-157				If an object is red and the object is a block then it is a frog.
G-158				If a frog is to the right of a toad and the toad is to the right of a clear location then you can move the frog onto the clear location.
G-159				If a toad is to the left of a frog and the frog is to the left of a clear location then you can move the toad onto the clear location.
G-160				You can move a free block onto a clear garbage.
G-161				If a block is clear and the position of the block is absent then it is free.
G-162				If the value of a free block is one less than the value of a top block then you can move the free block onto the top block.
G-163				If a block is clear and the position of the block is one then it is top.
G-164				If the value of a free block is one more than the value of a top block then you can move the free block onto the top block.
G-165				The goal is that all blocks are trashed.
G-166				If the position of a block is one then it is trashed.
G-167				If the color of a clear available block is the color of another clear available block then you can move the blocks onto a garbage.
G-168	1			The goal is that all blocks are on a garbage.
G-169				If a peg is between a clear location and a block then you can remove the peg plus move the block onto the clear location.
G-170	1			The goal is that the number of covered locations is one.
G-171				If a peg is between a clear location and a block then you can move the peg onto a garbage plus move the block onto the clear location.
G-172				If the sum of the values of two of the clear available blocks is thirteen then you can move it onto a garbage.

### A1.3 The Robot Corpus

This section contains a list of the 160 sentences that make up the corpus of sentences from the Robot scripts for developing Rosie. These sentences were not written by the Lucia developer. The list was compiled by concatenating all the scripts for the eight Robot tasks, removing duplicates, and then ordering them according to the ordering in which they were used for Lucia development.

The following list is ordered in a way that reflects the development of Lucia for this corpus. One sentence at a time was added to the Robot development set, and then all changes to the ECG grammar and Soar operators needed to make that sentence work in System A were made before going on to the next sentence in order. In addition to listing the sentences, the following table gives some rough numbers of things that needed to be added or changed to make a given sentence work in this development order. The column headings for the following table are defined as follows:

**ID** Identifier: A unique identifier for this sentence, made up of B for the Baseline corpus and the sequence number in this list. This sequence number also indicates the order in which these sentences were added to the Baseline development set.

At this time, unfortunately, the additional data on the Robot sentences has not yet been compiled.

ID	Sentences
R-001	Find the fork.
R-002	Move the fork onto the table.
R-003	Find Bob.
R-004	Find Alice.
R-005	Go to the kitchen.
R-006	Go to the main office.
R-007	Go to the current location.
R-008	Go to the starting location.
R-009	The only goal is that the fork is on the table.
R-010	Move the fork into the drawer.
R-011	Close the drawer.
R-012	Move the mug into the fridge.
R-013	Close the fridge.

R-014	Store the soda.
R-015	The only goal is that the soda is in the fridge and the fridge is closed.
R-016	Store the apple.
R-017	Go to the copy room.
R-018	Go to the conference room.
R-019	Go to Alice's office.
R-020	Go to Bob's office.
R-021	Go to Charlie's office.
R-022	Scan.
R-023	Turn right.
R-024	Turn left.
R-025	Turn around.
R-026	Turn right twenty-five degrees.
R-027	Orient north.
R-028	Orient south.
R-029	Orient east.
R-030	Orient west.
R-031	Drive forward one meter.
R-032	Drive through the door.
R-033	Say "What time is the meeting?".
R-034	Say "Hello there!" to Alice.
R-035	Pick up the mug.
R-036	Describe the held object.
R-037	Describe the grabbed object.
R-038	Put down the mug.
R-039	Ask "What drink would you like?".
R-040	A water.
R-041	Ask Alice "What drink would you like?".
R-042	The soda.
R-043	Ask Bob "Would you like a soda?".
R-044	Yes.
R-045	Explore until you see a stapler.
R-046	Pick up the stapler.
R-047	Put the stapler on the desk.
R-048	Open the drawer.
R-049	Close the pantry.
R-050	Turn on the lightswitch.
R-051	Turn off the lightswitch.
R-052	Permanently remember tea as the preferred drink of Alice.
R-053	Permanently remember the fridge as the storage location of a drink.
R-054	Permanently remember the current location as the office of Mary.

R-055	Permanently remember the main office as the office of Alice.
R-056	Permanently remember the bookshelf as the storage location of a book.
R-057	Permanently remember soda as the desired drink of Mary.
R-058	Recall the preferred drink of Alice.
R-059	Recall the storage location of a drink.
R-060	Recall the office of Mary.
R-061	Recall the office of Alice.
R-062	Recall the storage location of a book.
R-063	Recall the desired drink of Mary.
R-064	Recall the storage location of a computer.
R-065	Recall the office of Bob.
R-066	Pick up the fork.
R-067	Put the fork on the table.
R-068	You are done.
R-069	Move the fork onto the counter.
R-070	Deliver the apple to Alice.
R-071	If Alice is a person then the only goal is that Alice is holding the apple.
R-072	Alice is in Alice's office.
R-073	Pick up the apple.
R-074	Put down the apple.
R-075	Deliver the stapler to Bob's office.
R-076	If Bob's office is a location then the only goal is that the stapler is in Bob's office.
R-077	Deliver the stapler to the copy room.
R-078	Deliver the papers to Mary.
R-079	Fetch a stapler from the copy room.
R-080	The only goal is that the stapler is in the starting location.
R-081	First, remember the current location as the starting location.
R-082	Fetch a stapler.
R-083	Fetch a mug.
R-084	Tell Charlie a message.
R-085	The only goal is that Charlie heard the message.
R-086	First, ask "What is the message?".
R-087	Hello Charlie.
R-088	Remember the answer as the message.
R-089	Charlie is in Charlie's office.
R-090	Tell Bob a message.
R-091	Serve Mary.
R-092	The goal is that Mary is holding the desired drink.
R-093	Ask Mary "What drink would you like?".
R-094	A soda.
R-095	Remember the answer as the desired drink.

R-096	Serve Mary a desired drink.
R-097	Permanently remember Alice's office as the office of Alice.
R-098	Permanently remember Bob's office as the office of Bob.
R-099	Permanently remember Charlie's office as the office of Charlie.
R-100	Permanently remember the main office as the storage location of a water.
R-101	Permanently remember the fridge as the storage location of a juice.
R-102	Permanently remember water as the preferred drink of Bob.
R-103	Permanently remember soda as the preferred drink of Mary.
R-104	Recall the preferred drink of Mary.
R-105	If there is a preferred drink then ask Mary the preferred drink.
R-106	If the answer is yes then remember the preferred drink as the desired drink.
R-107	Permanently remember the desired drink as the preferred drink of Mary.
R-108	Serve Alice.
R-109	If nothing was recalled then ask Alice "What drink would you like?".
R-110	Remember the answered drink as the desired drink.
R-111	Serve Bob.
R-112	No.
R-113	If the answer is nope then ask Bob "What drink would you like?".
R-114	Serve Charlie.
R-115	A juice.
R-116	If the drawer is closed then the goal is that the drawer is opened.
R-117	The goal is that the fork is in the drawer.
R-118	Store the fork.
R-119	If the fork is a utensil then the only goal is that the fork is in the drawer and the drawer is closed.
R-120	If the soda is a drink then the only goal is that the soda is in the fridge and the fridge is closed.
R-121	Store the plate.
R-122	The only goal is that the plate is in the storage location.
R-123	The pantry.
R-124	Remember the answer as the storage location.
R-125	Store the spoon.
R-126	Store the juice.
R-127	Move the plate onto the counter.
R-128	Fill the green mug with water.
R-129	If the drink is water then move the green mug into the watercooler.
R-130	If the drink is water then press the blue button.
R-131	If the drink is milk then pick up the carton.
R-132	If the drink is milk then pour the carton into the green mug.
R-133	Pick up the green mug.
R-134	Pour the green mug into the sink.
R-135	Put the green mug onto the counter.

R-136	Fill the green mug with milk.
R-137	Put the carton onto the counter.
R-138	Fill the blue mug with water.
R-139	Pick up the blue mug.
R-140	Pour the blue mug into the sink.
R-141	Put the blue mug onto the counter.
R-142	Fill the blue mug with milk.
R-143	Fill2 the green mug with water.
R-144	If the drink is water then first pick up the green mug.
R-145	Put the green mug into the watercooler.
R-146	Press the blue button.
R-147	Fill2 the blue mug with milk.
R-148	If the drink is milk then first pick up the carton.
R-149	Pour the carton into the blue mug.
R-150	Fill2 the blue mug with water.
R-151	Fill2 the green mug with milk.
R-152	Keep the soda in the fridge.
R-153	The only goal is that the soda is always in the fridge.
R-154	Unknown.
R-155	Monitor the door until the meeting is finished.
R-156	The only goal is that the door is always closed.
R-157	The meeting is finished.
R-158	Go to the Bob's office.
R-159	Observe Bob for ten minutes.
R-160	The only goal is that Bob is always visible.

## **Appendix 2 The Lucia ECG Grammar for Rosie**

This appendix is dedicated to display in some detail the entire ECG grammar that has been developed for use by Lucia in Rosie. It includes lists and other descriptive material for each of the four types of ECG items in the grammar, as well as a more structural description of the parts of the grammar that define referring expressions, clauses, and sentences.

Table A2-9 summarizes the types of ECG items and their quantities in the current version<sup>11</sup> of the Lucia grammar for Rosie.

Table A2-9: Numbers of ECG Items

Type of ECG Item	Number in Grammar
Lexical Constructions	237
Composite Constructions	118
General Constructions	69
Meaning Schemas	146
<b>Total</b>	<b>570</b>

### **A2.1 Lexical Constructions**

Lexical constructions map words and multi-word items to their meanings. Each as a form element called *orth* which defines its orthography, or spelling. Multi-word items such as *to the left of* are treated as single lexical items, almost as if they were single words. Each lexical construction can also specify a meaning schema to be evoked and/or a set of constraints used to populate that schema.

Almost all lexical items are subcases of one or more general constructions. This has the effect of putting words into more general categories. These categories are somewhat analogous to traditional parts of speech, but due to the need for semantic precision this Lucia grammar for Rosie has many more

---

<sup>11</sup> The data for this Appendix was captured on 10/26/2021 and does not include later additions.

categories than are usually used for parts of speech. The categories allow words to fit into syntactic roles, but with a good deal of semantic precision.

Table A2-10 lists all the categories for lexical items along with the words or multi-word items that are members of each category. The individual items are listed in alphabetical order for each category by their orthography. Multi-word items have their constituent words listed in order separated by hyphens. Each entry in the table has the name of the general construction for the category, the number of members it has in parentheses, and the list of items in square brackets.

Table A2-10: Lexical Constructions by Category

ActionVerb(27) [ask, clean, close, cook, discard, explore, fetch, find, has, heard, interrogate, move, open, organize, pick, put, recall, said, say, serve, set, store, tell, throw, turn-off, turn-on, wait]
ActionVerbNeedsTarget(2) [deliver, take]
Adverb(2) [not, slowly]
AdverbialPreposition(2) [for, from]
Agent(5) [ <u>alice</u> , bob, <u>rosie</u> , we, you]
CardinalNumber(13) [eight, eleven, five, four, nine, one, seven, six, ten, three, twelve, twenty-five, two]
CommonNoun(49) [answer, apple, block, blocks, box, building, doors, doorway, drawer, end, fork, <u>fridge</u> , game, game, garbage, goal, hall, intersection, kitchen, lights, location, locations, message, mug, name, name, object, office, pantry, puzzle, puzzle, question, rectangle, response, room, soda, solution, sphere, square, stapler, steps, stove, table, task, trash, triangle, triangles, wall, <u>waypoint</u> ]
Complementizer(1) [that]
ConditionalPreposition(1) [until]
Conjunction(2) [and, as]
DeicticPronoun(2) [that, this]
Determiner(3) [a, an, the]
DiPreposition(1) [between]
Direction(4) [around, forward, left, right]
DirectionalPreposition(1) [down]
Done(3) [done, finished, over]
DriveVerb(2) [drive, go]
East(2) [e, east]
EnablingVerb(1) [can]
FiniteToBe(2) [are, is]
FunctionName(2) [the-number-of, the-volume-of]
InterrogativePronoun(4) [what, when, where, who]
LoadVerb(1) [load]
MotionVerb(1) [follow]
NONE(3) [check, remember, there]
NPSpecifier(1) [these]
North(2) [n, north]

```

OnePronoun(1) [one]
Orient(2) [face, orient]
Particle(3) [away, down, up]
PossesivePronoun(1) [your]
Possessive(3) [alice's, bob's, charlie's]
Preposition(18) [behind, below, holding, in, in-front-of, inside, into, larger-
    than, more-than, next-to, of, of, on, onto, smaller-than, to, to-the-left-of,
    to-the-right-of]
Pronoun(4) [it, me, we, you]
ProperName(2) [alice, bob]
Property(28) [accessible, big, blue, clear, closed, conference, cooked, copy,
    covered, current, empty, former, green, large, lit, main, medium, off, only,
    orange, purple, red, small, soar, square, starting, visible, yellow]
PropertyClassName(9) [action, color, location, relation, shape, size, sphere,
    square, triangle]
Quantifier(3) [all, all-of, some]
RelativePronoun(1) [that]
SimpleMotionVerb(4) [forward, patrol, scan, stop]
South(2) [s, south]
TransitiveSituationVerb(3) [reach, see, sense]
TurnVerb(1) [turn]
UnitsName(11) [centimeter, centimeters, degrees, feet, foot, inch, inches, meter,
    meters, minute, minutes]
West(2) [w, west]
Word(2) [if, then]
YesNo(1) [no]
YesWord(2) [ok, yes]

```

Of all the lexical items listed, there are five that have multiple types and three that have no types, as shown in TX.

Table A2-11: Lexical Items with Multiple or No Types

<b>Multiple Types</b>	<b>No Type</b>
alice [ProperName, Agent] bob [ProperName, Agent] down [Particle, DirectionalPreposition] we [Pronoun, Agent] you [Pronoun, Agent]	check remember there

There are also three pairs of words that the grammar considers synonyms. For each pair there is a separate construction for each orthography but a single name for the base type, as follows:

DRIVE(go) *synonym of* DRIVE(drive)  
 ORIENT(face) *synonym of* ORIENT(orient)  
 YES(ok) *synonym of* YES(yes)

## A2.2 Composite Constructions

Composite constructions are those that compose larger syntactic structures out of one or more constituents. Each has a name and a list of constituents. Each constituent has a slot name and the name of the type of construction that can fill that slot. As with lexical constructions, each composite construction also can optionally specify one or more general constructions it is a subcase of, one or more meaning schemas to evoke, and one or more constraints used to populate the meaning schema(s).

Table A2-12: Composite Constructions and the Composition Hierarchy lists all the composite constructions. The line for each construction starts with its name followed by its *productivity* in angle brackets, its parent types in curly brackets, and the types of its constituents in square brackets. The meaning and calculation of productivity is explained in Chapter 3. (Note: these numbers need to be updated!)

Table A2-12: Composite Constructions and the Composition Hierarchy

```
# Composites processed Wed Jul 14 17:27:21 MDT 2021
ActAlongDirection<3524264> {Imperative} [SimpleAction, DirectionalPrepPhrase]
ActAlongDirectionUntil<881067> {Imperative} [ActAlongDirection, UntilClause]
ActInDirection<8> {Imperative} [SimpleAction, Direction]
ActInDirectionForDistance<143> {Imperative} [ActInDirection, NumberOfUnits]
ActInDirectionUntil<881067> {Imperative} [ActInDirection, UntilClause]
ActionForTimePeriod<6435> {Imperative} [ActionVerb, FOR, NumberOfUnits]
AlicesOffice<1> {SpecifierNP} [ALICE-S, OFFICE]
AndDeclarative<1> {} [AND, Declarative]
AskQuestion<1> {Imperative} [ASK, Question]
BareNoun<50> {RefExpr} [CommonNoun]
BetweenPropertySets<1> {PrepPhrase} [DiPreposition, PropertySetAnd, PropertySet]
BobsOffice<1> {SpecifierNP} [BOB-S, OFFICE]
CharliesOffice<1> {SpecifierNP} [CHARLIE-S, OFFICE]
CheckCondition<1804840131> {} [CHECK, IF, Declarative]
ConceptIsThat<1511366> {} [RefExpr, FiniteToBe, THAT-complementizer]
ConceptIsThatDeclarative<1> {Declarative} [ConceptIsThat, Declarative]
DeclarativeAndDeclarative<1> {Declarative} [Declarative, AndDeclarative]
DiTransitiveCommand<755683> {Imperative} [TransitiveCommand, RefExpr]
DirectionalPrepPhrase<440533> {} [DirectionalPreposition, RefExpr]
DoUntil<68011515> {Imperative} [ActionVerb, UntilClause]
DriveInDirection<4> {Imperative} [DriveVerb, Direction]
DriveVerbToTarget<15113682> {Imperative} [DriveVerb, PrepPhrase]
EnabledCommandSentence<-1676782454> {Declarative} [EnablerPhrase, Imperative]
EnablerPhrase<5> {} [Agent, EnablingVerb]
```

```

FunctionWithArgument<2> {RefExpr} [FunctionName, RefExpr]
HeadRelativeClause<2> {} [RelativePronoun, FiniteToBe]
IfConditionAs<1> {} [IF, Declarative, AS]
IfConditionCommand<-525289077> {Conditional} [IF, Declarative, Imperative]
IfConditionThen<1804840131> {} [IF, Declarative, THEN]
IfConditionThenCommand<-667675751> {Conditional} [IfConditionThen, Imperative]
IfConditionThenStatement<1804840131> {Conditional} [IfConditionThen, Declarative]
ImperativeWithLocation<15113682> {Imperative} [Imperative, PrepPhrase]
ImperativeWithLocationSet<20> {Imperative} [Imperative, PropertySetPrepPhrase]
IntransitiveVerbDefinitionSentence<15113682> {Imperative} [UNKNOWN-WORD,
    PrepPhrase]
IsObjectClassQ<69522836> {YesNoQuestion} [FiniteToBe, RefExpr, Property]
IsObjectPropSetQ<87659228> {YesNoQuestion} [FiniteToBe, RefExpr, PropertySet]
IsObjectRelation<1669029484> {YesNoQuestion} [FiniteToBe, RefExpr, PrepPhrase]
LoadWorldFile<1> {Imperative} [LoadVerb, UNKNOWN-WORD]
ModifiedDriveVerb<2> {DriveVerb} [DriveVerb, Adverb]
ModifierList<1> {} [ModifierList, Modifier]
MotionOnObject<3022732> {Imperative} [MotionVerb, RefExpr]
MoveOnObjectUntil<-1387808300> {Imperative} [MotionOnObject, UntilClause]
MoveVerb<1> {ActionVerbNeedsTarget} [MOVE]
NameDefinitionSentence<30227320> {} [TheName, SimplePrepPhrase, FiniteToBe,
    UNKNOWN-WORD]
NegatedPrepPhrase<1> {PrepPhrase} [NOT, PrepPhrase]
Negation<2> {} [FiniteToBe, NOT]
NumberOfThings<650> {SpecifierNP} [CardinalNumber, CommonNoun]
NumberOfUnits<143> {SpecifierNP} [CardinalNumber, UnitsName]
PickUp<1> {ActionVerb} [PickVerb, UP]
PickVerb<1> {ActionVerb} [PICK]
Properties2<2116> {} [Property, Property]
Properties2Set<27> {PropertySet} [Determiner, Properties2, PropertyClassName]
Properties3<97336> {} [Properties2, Property]
Property1Set<27> {PropertySet} [Determiner, Property, PropertyClassName]
PropertyDefinitionSentence<116> {} [UNKNOWN-WORD, FiniteToBe, PropertySet]
PropertyNoun<50> {SpecifierNP} [Property, CommonNoun]
PropertyRedefinitionSentence<116> {} [Property, FiniteToBe, PropertySet]
PropertySetAnd<1> {} [PropertySet, AND]
PropertySetIsNotPrepPhrase<2> {Declarative} [PropertySet, Negation, PrepPhrase]
PropertySetIsPrepPhrase<2> {Declarative} [PropertySet, FiniteToBe, PrepPhrase]
PropertySetIsProperty<2> {Declarative} [PropertySet, FiniteToBe, Property]
PropertySetPrepPhrase<20> {PrepPhrase} [Preposition, PropertySet]
PropertySetRelClause<4> {PropertySet} [PropertySet, RelativeClause]
Props2Noun<105800> {SpecifierNP} [Properties2, CommonNoun]
PutDown<1> {ActionVerbNeedsTarget} [PutVerb, DOWN]
PutVerb<1> {ActionVerbNeedsTarget} [PUT]
QuantifiedRefExpr<16> {RefExpr} [Quantifier, RefExpr]
RefExprPrepPhrase<1> {RefExpr} [RefExpr, PrepPhrase]
RefExprRelClause<4> {RefExpr} [RefExpr, RelativeClause]
Ref IsNotPrepPhrase<1511366> {Declarative} [RefExpr, Negation, PrepPhrase]
Ref IsPrepPhrase<1511366> {Declarative} [RefExpr, FiniteToBe, PrepPhrase]
Ref IsProperty<3866> {Declarative} [RefExpr, FiniteToBe, Property]
Ref IsRef<-347707758> {Declarative} [RefExpr, FiniteToBe, RefExpr]

```

```

RelativeClausePrepPhrase<2> {RelativeClause} [HeadRelativeClause, PrepPhrase]
RelativeClauseProperty<2> {RelativeClause} [HeadRelativeClause, Property]
RememberConditionasB<755683> {RememberAasB} [REMEMBER, IfConditionAs, RefExpr]
RememberRefExprasB<1> {RememberAasB} [REMEMBER, RefExpr, AS, RefExpr]
RosieCommand<34005735> {Imperative} [ROSIE, ActionVerb, RefExpr]
SimpleAction<8> {Imperative} [SimpleMotionVerb]
SimplePrepPhrase<15113660> {PrepPhrase} [Preposition, RefExpr]
SpecNoun<300> {SpecifierNP} [NPSpecifier, CommonNoun]
SpecPropNoun<13800> {SpecifierNP} [NPSpecifier, Property, CommonNoun]
SpecProps2Noun<634800> {SpecifierNP} [NPSpecifier, Properties2, CommonNoun]
StoreVerb<1> {ActionVerbNeedsTarget} [STORE]
SubjectHas<755683> {} [RefExpr, HAS]
SubjectHasObject<755683> {Imperative} [SubjectHas, RefExpr]
SubjectVerb<34005735> {Declarative} [RefExpr, ActionVerb]
SubjectVerbObject<755683> {Declarative} [SubjectVerb, RefExpr]
SubjectVerbObjectPP<15113682> {Declarative} [SubjectVerbObject, PrepPhrase]
THE-Mods-Noun<50> {DefiniteNP} [THE, ModifierList, CommonNoun]
TheConferenceRoom<1> {SpecifierNP} [THE, CONFERENCE, ROOM]
TheCopyRoom<1> {SpecifierNP} [THE, COPY, ROOM]
TheKitchen<1> {SpecifierNP} [THE, KITCHEN]
TheMainOffice<1> {SpecifierNP} [THE, MAIN, OFFICE]
TheName<1> {} [THE, NAME]
TheSoarOffice<1> {SpecifierNP} [THE, SOAR, OFFICE]
ThereAre<2> {} [THERE, FiniteToBe]
ThereAreNumber<13> {Declarative} [ThereAre, CardinalNumber]
ThereAreRefExpr<1> {Declarative} [ThereAre, RefExpr]
ThisIsAThat<232> {Declarative} [DeicticPronoun, FiniteToBe, PropertySet]
ThisIsTheThat<3022732> {Declarative} [DeicticPronoun, FiniteToBe, RefExpr]
ThrowAway<1> {ActionVerb} [THROW, AWAY]
TransitiveCommand<34005735> {Imperative} [ActionVerb, RefExpr]
TransitiveCommandParticle<1> {Imperative} [TransitiveCommand, Particle]
TransitiveSituationClause<-521561637> {SituationClause} [RefExpr,
    TransitiveSituationVerb, RefExpr]
TransitiveVerbDefinitionSentence<755683> {Imperative} [UNKNOWN-WORD, RefExpr]
TurnDirection<3> {Imperative} [TurnVerb, Direction]
TurnDirectionByAngle<143> {Imperative} [TurnDirection, NumberOfUnits]
TurnObjectOn<755683> {Imperative} [TURN, RefExpr, ON]
TurnToDirection<3> {Imperative} [TurnVerb, TO, Direction]
TurnToTheDirection<3> {Imperative} [TurnVerb, TO, THE, Direction]
UntilDeclarativeClause<1> {UntilClause} [UNTIL, Declarative]
UntilThereIsClause<1511366> {UntilClause} [UNTIL, ThereAre, RefExpr]
WhatClassIsObjectQ<87659228> {WhQuestion} [WHAT, PropertySet, FiniteToBe, RefExpr]
WhatIsObject<1511366> {WhQuestion} [WHAT, FiniteToBe, RefExpr]
WhatIsPrepPhrase<2> {WhQuestion} [WHAT, FiniteToBe, PrepPhrase]
WheresWaldo<1511366> {WhQuestion} [WHERE, FiniteToBe, RefExpr]
WordDefinitionSentence<1511366> {} [RefExpr, FiniteToBe, UNKNOWN-WORD]
# Total composite constructions: 118.

```

### A2.3 General Constructions

General constructions add additional type information to any construction that references a general construction through a “subcase of” reference. Many lexical or composite constructions have several levels of generalization. During processing, each level provides a label to each instance of the base construction that can be referenced for composition by higher-level compositions. This provides the combination of a type hierarchy and a compositional hierarchy that are necessary to provide semantic precision and syntactic generality simultaneously. Table A2-13 lists all the general constructions, showing the whole type hierarchy. The format uses “<<{...}” to list the subcases of each general construction.

Table A2-13: General Constructions and the Type Hierarchy

```
# Generals processed Fri Jul 16 11:15:45 CDT 2021
ActionVerb<45> {} <<[ASK, ActionVerbNeedsTarget, CLEAN, CLOSE, COOK, DISCARD,
    EXPLORE, FETCH, FIND, HAS, HEARD, INTERROGATE, LoadVerb, MOVE, OPEN, ORGANIZE,
    PICK, PUT, PickUp, PickVerb, RECALL, SAID, SAY, SERVE, SET, STORE,
    SimpleMotionVerb, TELL, THROW, TURN-OFF, TURN-ON, ThrowAway, WAIT]
ActionVerbNeedsTarget<6> {ActionVerb} <<[DELIVER, MoveVerb, PutDown, PutVerb,
    StoreVerb, TAKE]
Adverb<2> {} <<[NOT, SLOWLY]
AdverbialPreposition<2> {Preposition} <<[FOR, FROM]
Agent<5> {} <<[ALICE, BOB, ROSIE, WE, YOU]
Aux<0> {Word, HasVerbFeatures}
BareSpecifier<2> {RefExpr} <<[DeicticPronoun]
CardinalNumber<13> {Quantifier} <<[EIGHT, ELEVEN, FIVE, FOUR, NINE, ONE-number,
    SEVEN, SIX, TEN, THREE, TWELVE, TWENTY-FIVE, TWO]
CommonNoun<50> {Noun} <<[ANSWER, APPLE, BLOCK, BLOCKS, BOX, BUILDING, DOORS,
    DOORWAY, DRAWER, END, FORK, FRIDGE, GAME, GAME-game, GARBAGE, GOAL, HALL,
    INTERSECTION, KITCHEN, LIGHTS, LOCATION-noun, LOCATIONS, MESSAGE, MUG, NAME,
    NAME-name, OBJECT, OFFICE, OnePronoun, PANTRY, PUZZLE, PUZZLE-puzzle, QUESTION,
    RECTANGLE, RESPONSE, ROOM, SODA, SOLUTION, SPHERE-noun, SQUARE-noun, STAPLER,
    STEPS, STOVE, TABLE, TASK, TRASH, TRIANGLE-noun, TRIANGLES, WALL, WAYPOINT]
Complementizer<1> {} <<[THAT-complementizer]
Conditional<611875303> {} <<[IfConditionCommand, IfConditionThenCommand,
    IfConditionThenStatement]
ConditionalPreposition<1> {Word} <<[UNTIL]
Conjunction<2> {Word} <<[AND, AS]
Declarative<1804840131> {VerbWithArguments} <<[ConceptIsThatDeclarative,
    DeclarativeAndDeclarative, EnabledCommandSentence, PropertySetIsNotPrepPhrase,
    PropertySetIsPrepPhrase, PropertySetIsProperty, RefIsNotPrepPhrase,
    RefIsPrepPhrase, RefIsProperty, RefIsRef, SituationClause, SubjectVerb,
```

```

SubjectVerbObject, SubjectVerbObjectPP, ThereAreNumber, ThereAreRefExpr,
    ThisIsAThat, ThisIsTheThat]
DefiniteNP<50> {RefExpr} <<[THE-Mods-Noun]
DeicticPronoun<2> {Specifier, BareSpecifier} <<[THAT-deictic, THIS]
Determiner<3> {NPSpecifier} <<[A, AN, THE]
DiPreposition<1> {Word} <<[BETWEEN]
Direction<12> {Property} <<[AROUND, East, FORWARD-direction, LEFT, North, RIGHT,
    South, West]
DirectionalPreposition<1> {Word} <<[DOWN]
Done<3> {Property} <<[DONE, FINISHED, OVER]
DriveVerb<4> {SimpleMotionVerb} <<[DRIVE, DRIVE-go, ModifiedDriveVerb]
East<2> {Direction} <<[E, EAST]
EnablingVerb<1> {} <<[CAN-verb]
FiniteToBe<2> {Word, HasVerbFeatures} <<[ARE, IS]
FunctionName<2> {} <<[THE-NUMBER-OF, THE-VOLUME-OF]
HasVerbFeatures<3> {} <<[Aux, FiniteToBe, Verb]
Imperative<-1194349950> {VerbWithArguments} <<[ActAlongDirection,
    ActAlongDirectionUntil, ActInDirection, ActInDirectionForDistance,
    ActInDirectionUntil, ActionForTimePeriod, AskQuestion, DiTransitiveCommand,
    DoUntil, DriveInDirection, DriveVerbToTarget, ImperativeWithLocation,
    ImperativeWithLocationSet, IntransitiveVerbDefinitionSentence, LoadWorldFile,
    MotionOnObject, MoveOnObjectUntil, RememberAasB, RosieCommand, SimpleAction,
    SubjectHasObject, TransitiveCommand, TransitiveCommandParticle,
    TransitiveVerbDefinitionSentence, TurnDirection, TurnDirectionByAngle,
    TurnObjectOn, TurnToDirection, TurnToTheDirection]
InterrogativePronoun<4> {} <<[WHAT, WHEN, WHERE, WHO]
LoadVerb<1> {ActionVerb} <<[LOAD]
MotionVerb<4> {} <<[FOLLOW, TurnVerb]
NPSpecifier<6> {Word} <<[Determiner, PossessivePronoun, Specifier, THESE]
North<2> {Direction} <<[N, NORTH]
Noun<59> {Word} <<[CommonNoun, PropertyClassName]
OnePronoun<1> {CommonNoun} <<[ONE-pronoun]
Orient<2> {TurnVerb} <<[ORIENT, ORIENT-face]
Particle<3> {} <<[AWAY, DOWN, UP]
PossessivePronoun<1> {NPSpecifier} <<[YOUR]
Possessive<3> {Property} <<[ALICE-S, BOB-S, CHARLIE-S]
PrepPhrase<15113682> {} <<[BetweenPropertySets, NegatedPrepPhrase,
    PropertySetPrepPhrase, SimplePrepPhrase]
Preposition<20> {Word} <<[AdverbialPreposition, BEHIND, BELOW, HOLDING, IN, IN-
    FRONT-OF, INSIDE, INTO, LARGER-THAN, MORE-THAN, NEXT-TO, OF, OF-of, ON, ONTO,
    SMALLER-THAN, TO, TO-THE-LEFT-OF, TO-THE-RIGHT-OF]
Pronoun<4> {RefExpr, Word} <<[IT, ME, WE, YOU]
ProperName<2> {RefExpr} <<[ALICE, BOB]
Property<46> {Word} <<[ACCESSIBLE, BIG, BLUE, CLEAR, CLOSED, CONFERENCE, COOKED,
    COPY, COVERED, CURRENT, Direction, Done, EMPTY, FORMER, GREEN, LARGE, LIT,
    MAIN, MEDIUM, OFF, ONLY, ORANGE, PURPLE, Possessive, RED, SMALL, SOAR, SQUARE-
    adjective, STARTING, VISIBLE, YELLOW]
PropertyClassName<9> {Noun} <<[ACTION, COLOR, LOCATION-class, RELATION, SHAPE,
    SIZE, SPHERE-class, SQUARE-class, TRIANGLE-class]
PropertySet<58> {} <<[Properties2Set, Property1Set, PropertySetRelClause]
Quantifier<16> {} <<[ALL, ALL-OF, CardinalNumber, SOME]

```

```

Question<1916893510> {} <<[WhQuestion, YesNoQuestion]
RefExpr<755683> {} <<[BareNoun, BareSpecifier, DefiniteNP, FunctionWithArgument,
Pronoun, ProperName, QuantifiedRefExpr, RefExprPrepPhrase, RefExprRelClause,
RelativePronoun, SpecifierNP]
RelativeClause<4> {} <<[RelativeClausePrepPhrase, RelativeClauseProperty]
RelativePronoun<1> {RefExpr, Word} <<[THAT-relative]
RememberAasB<755684> {Imperative} <<[RememberConditionasB, RememberRefExrasB]
SimpleMotionVerb<8> {ActionVerb} <<[DriveVerb, FORWARD-verb, PATROL, SCAN, STOP]
SituationClause<-521561637> {Declarative} <<[TransitiveSituationClause]
South<2> {Direction} <<[S, SOUTH]
Specifier<0> {NPSpecifier} <<[DeicticPronoun]
SpecifierNP<755551> {RefExpr} <<[AlicesOffice, BobsOffice, CharliesOffice,
NumberOfThings, NumberOfUnits, PropertyNoun, Props2Noun, SpecNoun,
SpecPropNoun, SpecProps2Noun, TheConferenceRoom, TheCopyRoom, TheKitchen,
TheMainOffice, TheSoarOffice]
TransitiveSituationVerb<3> {} <<[REACH, SEE, SENSE]
TurnVerb<3> {MotionVerb} <<[Orient, TURN]
UnitsName<11> {} <<[CENTIMETER, CENTIMETERS, DEGREES, FEET, FOOT, INCH, INCHES,
METER, METERS, MINUTE, MINUTES]
UntilClause<1511367> {} <<[UntilDeclarativeClause, UntilThereIsClause]
Verb<0> {Word, HasVerbFeatures}
VerbWithArguments<1137164380> {} <<[Declarative, Imperative]
West<2> {Direction} <<[W, WEST]
WhQuestion<90681962> {Question} <<[WhatClassIsObjectQ, WhatIsObject,
WhatIsPrepPhrase, WheresWaldo]
Word<148> {} <<[Aux, ConditionalPreposition, Conjunction, DiPreposition,
DirectionalPreposition, FiniteToBe, IF, NPSpecifier, Noun, Preposition,
Pronoun, Property, RelativePronoun, THEN, Verb, YesNo]
YesNo<3> {Word} <<[NO, YesWord]
YesNoQuestion<1826211548> {Question} <<[IsObjectClassQ, IsObjectPropSetQ,
IsObjectRelation]
YesWord<2> {YesNo} <<[YES, YES-ok]
# Total general constructions: 69.

```

## A2.4 Meaning Schemas

The meaning of a construction is represented by a meaning schema, which is a structure with roles, or slots, that hold related information. Slots are filled by ECG constraints in either the definition of the schema itself or in the construction that evoked it. Table A2-14 lists all the meaning schemas.

Table A2-14: Meaning Schemas

ActOnIt {Action} [object] Action {} [action, direction, location] ActionDescriptor {} [class, name, modifier]	Pantry {RosieLocation} [] PastParticiple {NonFinite} [] PossessiveProperty {PropertyDescriptor} [possessive]
---	--

ActionForTime {Action} [time]	PrepCore {} [name]
AgreementFeatures {} [number, person]	PrepPhraseAssertion {Assertion}
Alice {RosieObject, KnownObject} [name]	[prerel: PrepRelation, target: RefDesc, modifier]
Answer {Concept} []	PrepRelation {} [prep: PrepCore, object: RefDesc]
Apple {ObjectNotBlock} []	PrepRelation2 {PrepRelation} [object2: RefDesc]
AskQuestion {} []	PropertyApplication {Assertion}
AskQuestionCommand {ActOnIt} []	[property: PropertyDescriptor, target: RefDesc]
Assertion {} []	PropertyClass {} [name]
AuxiliaryFeatures {} [type]	PropertyClassDescriptor {} [name]
Block {RosieObject} [shape, color, size, state]	PropertyDefinition {} [word, class]
Bob {RosieObject, KnownObject} [name]	PropertyDescriptor {} [class, name, next]
Box {RosieObject} [shape, color, size, state]	PropertyRedefinition {} [word, old, class]
Building {MapLocation} []	PropertySetDescriptor {} [predicate, givenness, property, property2]
CheckIt {Action} [condition]	Puzzle {Concept} []
CompoundAssertion {Assertion}	Puzzle {Concept} []
[assertion1, assertion2]	Quantification {} [type]
Concept {Entity, PropertyDescriptor}	Question {Concept} []
[concept-type, concept-handle]	Rectangle {Block} []
ConceptIsThatAssertion {Assertion}	RefDesc {} [name, category, givenness, modifiers, referent, relation, quantified]
[concept, assertion]	RefExprAssertion {Assertion} [reference]
Condition {} [statement]	Response {Concept} []
ConferenceRoom {MapLocation} []	Room {SpatialShape} []
DoItInDirection {Action} [object]	Rosie {RosieObject} [name]
DoTransfer {ActOnIt} [object2]	RosieLocation {} [rosie-category, name, category, root-category]
Doorway {SpatialShape} []	RosieObject {Entity} [handle, movable]
Drawer {ObjectNotBlock} []	Salient {} [reference]
EnabledCommand {} [enabler, command]	SituationDescriptor {} [subject, action, object]
Enabler {} [agent, ability]	SoarOffice {MapLocation} []
End {SpatialShape} []	Soda {ObjectNotBlock} []
Entity {} [rosie-category]	Solution {RosieObject} []
EqualComparison {} [value1: RefDesc, value2: RefDesc]	SpatialShape {RosieObject} [spatial-shape]
EventDescriptor {} [eventType: Process, profiledProcess: Process, profiledParticipant, profiledState, spatialSetting, temporalSetting, speechAct]	Sphere {Block} []
Finite {FiniteOrNonFinite, FiniteOrGerund} []	Square {Block} []
FiniteOrGerund {FiniteOrNonFinite} []	Stapler {RosieObject} [name]
FiniteOrNonFinite {} []	Steps {RosieObject} [shape]
Fork {ObjectNotBlock} []	Stove {RosieLocation} []
Fridge {ObjectNotBlock} []	SubjectActOnIt {ActOnIt} [subject]
Function {} [fn-handle]	Table {RosieLocation} []
FunctionApplication {RefDesc} [argument: RefDesc, predicate]	Task {Concept} []
Game {Concept} []	
Game {Concept} []	
Garbage {RosieLocation} []	

<pre> Gerund {NonFinite, FiniteOrGerund} [] Goal {Concept} [] Hall {SpatialShape} [] IfThenCommand {} [condition, command] IfThenStatement {} [condition,     statement] Infinitive {NonFinite} [] Intersection {SpatialShape} [] IntransitiveAssertion {Assertion}     [subject, verb] IntransitiveVerbDefinition     {VerbDefinition} [location] Kitchen {MapLocation} [] KnownObject {RefDesc} [handle] Lights {KnownObject} [name] LoadWorldCommand {} [action, word] Location {RosieObject} [] MainOffice {MapLocation} [] MapLocation {KnownObject, RosieObject}     [] Measurement {} [units, number] Message {Concept} [] MoveDistance {Action} [distance] MoveToIt {Action} [] Mug {ObjectNotBlock} [] Name {Concept} [] Name {Concept} [] NameDefinition {} [word, relation, name] NegatedPrepRelation {PrepRelation}     [negation] NominalAgreementFeatures     {AgreementFeatures} [case] NominalFeatures {} [features:     NominalAgreementFeatures] NonFinite {FiniteOrNonFinite} [] NumberAssertion {Assertion} [number] Object {RosieObject} [shape, color,     size, state] ObjectIsClassQuestion {AskQuestion}     [object, class] ObjectIsRelationQuestion {AskQuestion}     [object, relation] ObjectNotBlock {RosieObject} [shape,     color, size, category] Office {MapLocation} [] </pre>	<pre> TerminatedAction {Action} [terminator] TerminatedActionOnObject {Action}     [object, terminator] ThisIsAThatAssertion {Assertion} [this:     RefDesc, that:         PropertyClassDescriptor] ThisIsTheThatAssertion {Assertion}     [this: RefDesc, that: RefDesc] TransitiveAssertion {Assertion}     [subject, verb, object] TransitiveAssertionPrepPhrase     {Assertion} [subject, verb, object,         relation] TransitiveVerbDefinition     {VerbDefinition} [object] Trash {Block} [] Triangle {Block} [] TurnByAngle {Action} [angle] TurnToIt {Action} [direction] Units {Concept} [type] UntilDeclarative {} [assertion] UntilSituation {} [situation] UntilThereIs {} [object] VerbAgreementFeatures     {AgreementFeatures} [verbform] VerbDefinition {} [word] VerbFeatures {} [features:     VerbAgreementFeatures] VerbModifier {} [id] Volume {Concept} [] Wall {SpatialShape} [] Waypoint {SpatialShape} [] WhatClassIsObjectQuestion {AskQuestion}     [class, object] WhatIsObjectQuestion {AskQuestion}     [object] WhatIsQuestion {AskQuestion} [] WhatIsRelation {WhatIsQuestion}     [predicate] WhereIsObject {AskQuestion} [object] WordDefinition {} [word, object] WordForm {} [orth] YesNoAnswer {Assertion} [answer] </pre>
--	--

## A2.5 Referring Expressions

Since referring expressions are such an important part of the Lucia grammar for Rosie, and this part is so complex, it's worth looking in detail at the grammar hierarchies for this part of the grammar. Table A2-15 shows a hierarchy tree beginning at the RefExpr construction and going three levels deep. Under each general construction its children in the type hierarchy are shown indented one more level. Under each composite construction its constituent constructions are shown indented. Lexical constructions are shown by their construction name, which is in capitals, and they are listed in the constituent lists but not as children.

Table A2-15: The RefExpr Hierarchy

```
# RefExprs processed Wed Jul 14 15:41:01 MDT 2021
RefExpr<755683> {}
  BareNoun<50> {RefExpr} [CommonNoun]
    CommonNoun<50> {Noun}
      OnePronoun<1> {CommonNoun}
  BareSpecifier<2> {RefExpr}
    DeicticPronoun<2> {Specifier, BareSpecifier}
  DefiniteNP<50> {RefExpr}
    THE-Mods-Noun<50> {DefiniteNP} [THE, ModifierList, CommonNoun]
      ModifierList<1> {} [ModifierList, Modifier]
        CommonNoun<50> {Noun}
  FunctionWithArgument<2> {RefExpr} [FunctionName, RefExpr]
    FunctionName<2> {}
    *RefExpr<755683> {}
  Pronoun<4> {RefExpr, Word}
  ProperName<2> {RefExpr}
  QuantifiedRefExpr<16> {RefExpr} [Quantifier, RefExpr]
    Quantifier<16> {}
      CardinalNumber<13> {Quantifier}
    *RefExpr<755683> {}
  RefExprPrepPhrase<1> {RefExpr} [RefExpr, PrepPhrase]
    *RefExpr<755683> {}
    PrepPhrase<15113682> {}
      BetweenPropertySets<1> {PrepPhrase} [DiPreposition, PropertySetAnd,
        PropertySet]
        NegatedPrepPhrase<1> {PrepPhrase} [NOT, PrepPhrase]
        PropertySetPrepPhrase<20> {PrepPhrase} [Preposition, PropertySet]
        SimplePrepPhrase<15113660> {PrepPhrase} [Preposition, RefExpr]
  RefExprRelClause<4> {RefExpr} [RefExpr, RelativeClause]
    *RefExpr<755683> {}
    RelativeClause<4> {}
      RelativeClausePrepPhrase<2> {RelativeClause} [HeadRelativeClause, PrepPhrase]
      RelativeClauseProperty<2> {RelativeClause} [HeadRelativeClause, Property]
```

```

RelativePronoun<1> {RefExpr,Word}
SpecifierNP<755551> {RefExpr}
  AlicesOffice<1> {SpecifierNP} [ALICE-S, OFFICE]
  BobsOffice<1> {SpecifierNP} [BOB-S, OFFICE]
  CharliesOffice<1> {SpecifierNP} [CHARLIE-S, OFFICE]
  NumberOfThings<650> {SpecifierNP} [CardinalNumber, CommonNoun]
    CardinalNumber<13> {Quantifier}
    CommonNoun<50> {Noun}
  NumberOfUnits<143> {SpecifierNP} [CardinalNumber, UnitsName]
    CardinalNumber<13> {Quantifier}
    UnitsName<11> {}
  PropertyNoun<50> {SpecifierNP} [Property, CommonNoun]
    Property<46> {Word}
    CommonNoun<50> {Noun}
  Props2Noun<105800> {SpecifierNP} [Properties2, CommonNoun]
    Properties2<2116> {} [Property, Property]
    CommonNoun<50> {Noun}
  SpecNoun<300> {SpecifierNP} [NPSpecifier, CommonNoun]
    NPSpecifier<6> {Word}
    CommonNoun<50> {Noun}
  SpecPropNoun<13800> {SpecifierNP} [NPSpecifier, Property, CommonNoun]
    NPSpecifier<6> {Word}
    Property<46> {Word}
    CommonNoun<50> {Noun}
  SpecProps2Noun<634800> {SpecifierNP} [NPSpecifier, Properties2, CommonNoun]
    NPSpecifier<6> {Word}
    Properties2<2116> {} [Property, Property]
    CommonNoun<50> {Noun}
  TheConferenceRoom<1> {SpecifierNP} [THE, CONFERENCE, ROOM]
  TheCopyRoom<1> {SpecifierNP} [THE, COPY, ROOM]
  TheKitchen<1> {SpecifierNP} [THE, KITCHEN]
  TheMainOffice<1> {SpecifierNP} [THE, MAIN, OFFICE]
  TheSoarOffice<1> {SpecifierNP} [THE, SOAR, OFFICE]

```

This hierarchy is recursive, allowing children in the RefExpr type hierarchy to have RefExprs as their constituents. In the figure an occurrence of RefExpr under one of its children is marked with an asterisk, and not expanded further.

## A2.6 Sentences

The Lucia grammar for Rosie has many constructions that form the sentence level. These are composed into four general categories: declarative, imperative, interrogative, and conditional sentences. The last group is not conventional in grammar, but useful in these conditions. The following sequence of tables gives

all the constructions in each category, listed in hierarchical form was done for referring expressions.

### A2.6.1 Declarative sentences

Table A2-16: Declarative Sentences

```
# Declaratives processed Wed Jul 14 17:10:42 MDT 2021
Declarative<1804840131> {VerbWithArguments}
    ConceptIsThatDeclarative<1> {Declarative} [ConceptIsThat, Declarative]
        ConceptIsThat<1511366> {} [RefExpr, FiniteToBe, THAT-complementizer]
        *Declarative<1804840131> {VerbWithArguments}
    DeclarativeAndDeclarative<1> {Declarative} [Declarative, AndDeclarative]
        *Declarative<1804840131> {VerbWithArguments}
        AndDeclarative<1> {} [AND, Declarative]
    EnabledCommandSentence<-1676782454> {Declarative} [EnablerPhrase, Imperative]
        EnablerPhrase<5> {} [Agent, EnablingVerb]
        Imperative<-1194349950> {VerbWithArguments}
    PropertySetIsNotPrepPhrase<2> {Declarative} [PropertySet, Negation, PrepPhrase]
        PropertySet<58> {}
        Negation<2> {} [FiniteToBe, NOT]
        PrepPhrase<15113682> {}
    PropertySetIsPrepPhrase<2> {Declarative} [PropertySet, FiniteToBe, PrepPhrase]
        PropertySet<58> {}
        FiniteToBe<2> {Word, HasVerbFeatures}
        PrepPhrase<15113682> {}
    PropertySetIsProperty<2> {Declarative} [PropertySet, FiniteToBe, Property]
        PropertySet<58> {}
        FiniteToBe<2> {Word, HasVerbFeatures}
        Property<46> {Word}
    Ref IsNotPrepPhrase<1511366> {Declarative} [RefExpr, Negation, PrepPhrase]
        RefExpr<755683> {}
        Negation<2> {} [FiniteToBe, NOT]
        PrepPhrase<15113682> {}
    Ref IsPrepPhrase<1511366> {Declarative} [RefExpr, FiniteToBe, PrepPhrase]
        RefExpr<755683> {}
        FiniteToBe<2> {Word, HasVerbFeatures}
        PrepPhrase<15113682> {}
    Ref IsProperty<3866> {Declarative} [RefExpr, FiniteToBe, Property]
        RefExpr<755683> {}
        FiniteToBe<2> {Word, HasVerbFeatures}
        Property<46> {Word}
    Ref IsRef<-347707758> {Declarative} [RefExpr, FiniteToBe, RefExpr]
        RefExpr<755683> {}
        FiniteToBe<2> {Word, HasVerbFeatures}
        RefExpr<755683> {}
    SituationClause<-521561637> {Declarative}
        TransitiveSituationClause<-521561637> {SituationClause} [RefExpr,
        TransitiveSituationVerb, RefExpr]
    SubjectVerb<34005735> {Declarative} [RefExpr, ActionVerb]
```

```

RefExpr<755683> {}
ActionVerb<45> {}
SubjectVerbObject<755683> {Declarative} [SubjectVerb, RefExpr]
SubjectVerb<34005735> {Declarative} [RefExpr, ActionVerb]
RefExpr<755683> {}
SubjectVerbObjectPP<15113682> {Declarative} [SubjectVerbObject, PrepPhrase]
SubjectVerbObject<755683> {Declarative} [SubjectVerb, RefExpr]
PrepPhrase<15113682> {}
ThereAreNumber<13> {Declarative} [ThereAre, CardinalNumber]
ThereAre<2> {} [THERE, FiniteToBe]
CardinalNumber<13> {Quantifier}
ThereAreRefExpr<1> {Declarative} [ThereAre, RefExpr]
ThereAre<2> {} [THERE, FiniteToBe]
RefExpr<755683> {}
ThisIsAThat<232> {Declarative} [DeicticPronoun, FiniteToBe, PropertySet]
DeicticPronoun<2> {Specifier, BareSpecifier}
FiniteToBe<2> {Word, HasVerbFeatures}
PropertySet<58> {}
ThisIsTheThat<3022732> {Declarative} [DeicticPronoun, FiniteToBe, RefExpr]
DeicticPronoun<2> {Specifier, BareSpecifier}
FiniteToBe<2> {Word, HasVerbFeatures}
RefExpr<755683> {}

```

## A2.6.2 Imperative sentences

Table A2-17: Imperative Sentences

```

# Imperatives processed Wed Jul 14 17:10:42 MDT 2021
Imperative<-1194349950> {VerbWithArguments}
ActAlongDirection<3524264> {Imperative} [SimpleAction, DirectionalPrepPhrase]
SimpleAction<8> {Imperative} [SimpleMotionVerb]
DirectionalPrepPhrase<440533> {} [DirectionalPreposition, RefExpr]
ActAlongDirectionUntil<881067> {Imperative} [ActAlongDirection, UntilClause]
ActAlongDirection<3524264> {Imperative} [SimpleAction, DirectionalPrepPhrase]
UntilClause<1511367> {}
ActInDirection<8> {Imperative} [SimpleAction, Direction]
SimpleAction<8> {Imperative} [SimpleMotionVerb]
Direction<12> {Property}
ActInDirectionForDistance<143> {Imperative} [ActInDirection, NumberOfUnits]
ActInDirection<8> {Imperative} [SimpleAction, Direction]
NumberOfUnits<143> {SpecifierNP} [CardinalNumber, UnitsName]
ActInDirectionUntil<881067> {Imperative} [ActInDirection, UntilClause]
ActInDirection<8> {Imperative} [SimpleAction, Direction]
UntilClause<1511367> {}
ActionForTimePeriod<6435> {Imperative} [ActionVerb, FOR, NumberOfUnits]
ActionVerb<45> {}
NumberOfUnits<143> {SpecifierNP} [CardinalNumber, UnitsName]
AskQuestion<1> {Imperative} [ASK, Question]
Question<1916893510> {}

```

```

DiTransitiveCommand<755683> {Imperative} [TransitiveCommand, RefExpr]
  TransitiveCommand<34005735> {Imperative} [ActionVerb, RefExpr]
    RefExpr<755683> {}
DoUntil<68011515> {Imperative} [ActionVerb, UntilClause]
  ActionVerb<45> {}
  UntilClause<1511367> {}
DriveInDirection<4> {Imperative} [DriveVerb, Direction]
  DriveVerb<4> {SimpleMotionVerb}
  Direction<12> {Property}
DriveVerbToTarget<15113682> {Imperative} [DriveVerb, PrepPhrase]
  DriveVerb<4> {SimpleMotionVerb}
  PrepPhrase<15113682> {}
ImperativeWithLocation<15113682> {Imperative} [Imperative, PrepPhrase]
  *Imperative<-1194349950> {VerbWithArguments}
  PrepPhrase<15113682> {}
ImperativeWithLocationSet<20> {Imperative} [Imperative, PropertySetPrepPhrase]
  *Imperative<-1194349950> {VerbWithArguments}
  PropertySetPrepPhrase<20> {PrepPhrase} [Preposition, PropertySet]
IntransitiveVerbDefinitionSentence<15113682> {Imperative} [UNKNOWN-WORD,
  PrepPhrase]
  PrepPhrase<15113682> {}
LoadWorldFile<1> {Imperative} [LoadVerb, UNKNOWN-WORD]
  LoadVerb<1> {ActionVerb}
MotionOnObject<3022732> {Imperative} [MotionVerb, RefExpr]
  MotionVerb<4> {}
  RefExpr<755683> {}
MoveOnObjectUntil<-1387808300> {Imperative} [MotionOnObject, UntilClause]
  MotionOnObject<3022732> {Imperative} [MotionVerb, RefExpr]
  UntilClause<1511367> {}
RememberAasB<755684> {Imperative}
  RememberConditionasB<755683> {RememberAasB} [REMEMBER, IfConditionAs, RefExpr]
  RememberRefExprasB<1> {RememberAasB} [REMEMBER, RefExpr, AS, RefExpr]
RosieCommand<34005735> {Imperative} [ROSIE, ActionVerb, RefExpr]
  ActionVerb<45> {}
  RefExpr<755683> {}
SimpleAction<8> {Imperative} [SimpleMotionVerb]
  SimpleMotionVerb<8> {ActionVerb}
SubjectHasObject<755683> {Imperative} [SubjectHas, RefExpr]
  SubjectHas<755683> {} [RefExpr, HAS]
  RefExpr<755683> {}
TransitiveCommand<34005735> {Imperative} [ActionVerb, RefExpr]
  ActionVerb<45> {}
  RefExpr<755683> {}
TransitiveCommandParticle<1> {Imperative} [TransitiveCommand, Particle]
  TransitiveCommand<34005735> {Imperative} [ActionVerb, RefExpr]
    Particle<3> {}
TransitiveVerbDefinitionSentence<755683> {Imperative} [UNKNOWN-WORD, RefExpr]
  RefExpr<755683> {}
TurnDirection<3> {Imperative} [TurnVerb, Direction]
  TurnVerb<3> {MotionVerb}
  Direction<12> {Property}

```

```

TurnDirectionByAngle<143> {Imperative} [TurnDirection, NumberOfUnits]
  TurnDirection<3> {Imperative} [TurnVerb, Direction]
  NumberOfUnits<143> {SpecifierNP} [CardinalNumber, UnitsName]
TurnObjectOn<755683> {Imperative} [TURN, RefExpr, ON]
  RefExpr<755683> {}
TurnToDirection<3> {Imperative} [TurnVerb, TO, Direction]
  TurnVerb<3> {MotionVerb}
  Direction<12> {Property}
TurnToTheDirection<3> {Imperative} [TurnVerb, TO, THE, Direction]
  TurnVerb<3> {MotionVerb}
  Direction<12> {Property}

```

### A2.6.3 Questions

Table A2-18: Questions

```

# Questions processed Wed Jul 14 17:10:42 MDT 2021
Question<1916893510> {}
WhQuestion<90681962> {Question}
  WhatClassIsObjectQ<87659228> {WhQuestion} [WHAT, PropertySet, FiniteToBe,
  RefExpr]
  WhatIsObject<1511366> {WhQuestion} [WHAT, FiniteToBe, RefExpr]
  WhatIsPrepPhrase<2> {WhQuestion} [WHAT, FiniteToBe, PrepPhrase]
  WheresWaldo<1511366> {WhQuestion} [WHERE, FiniteToBe, RefExpr]
YesNoQuestion<1826211548> {Question}
  IsObjectClassQ<69522836> {YesNoQuestion} [FiniteToBe, RefExpr, Property]
  IsObjectPropSetQ<87659228> {YesNoQuestion} [FiniteToBe, RefExpr, PropertySet]
  IsObjectRelation<1669029484> {YesNoQuestion} [FiniteToBe, RefExpr, PrepPhrase]

```

### A2.6.4 Conditional sentences

Table A2-19: Conditional Sentences

```

# Conditionals processed Wed Jul 14 17:13:33 MDT 2021
Conditional<611875303> {}
  IfConditionCommand<-525289077> {Conditional} [IF, Declarative, Imperative]
    Declarative<1804840131> {VerbWithArguments}
    Imperative<-1194349950> {VerbWithArguments}
  IfConditionThenCommand<-667675751> {Conditional} [IfConditionThen, Imperative]
    IfConditionThen<1804840131> {} [IF, Declarative, THEN]
    Imperative<-1194349950> {VerbWithArguments}
  IfConditionThenStatement<1804840131> {Conditional} [IfConditionThen, Declarative]
    IfConditionThen<1804840131> {} [IF, Declarative, THEN]
    Declarative<1804840131> {VerbWithArguments}

```

## A2.7 Subordinate Clauses

As the above tables for sentences show, both Declaratives and Imperatives can serve as subordinate clauses in many situations. They can even be used recursively, as the tables show. In addition, there are two other kinds of subordinate clauses: RelativeClauses and UntilClauses. The next two tables show the structures of these clauses.

Table A2-20: Relative Clauses

```
# RelativeClauses processed Wed Jul 14 17:33:02 MDT 2021
RelativeClause<4> {}
  RelativeClausePrepPhrase<2> {RelativeClause} [HeadRelativeClause, PrepPhrase]
    HeadRelativeClause<2> {} [RelativePronoun, FiniteToBe]
    PrepPhrase<15113682> {}
  RelativeClauseProperty<2> {RelativeClause} [HeadRelativeClause, Property]
    HeadRelativeClause<2> {} [RelativePronoun, FiniteToBe]
    Property<46> {Word}
```

Table A2-21: Until Clauses

```
# UntilClauses processed Wed Jul 14 17:33:02 MDT 2021
UntilClause<1511367> {}
  UntilDeclarativeClause<1> {UntilClause} [UNTIL, Declarative]
    Declarative<1804840131> {VerbWithArguments}
  UntilThereIsClause<1511366> {UntilClause} [UNTIL, ThereAre, RefExpr]
    ThereAre<2> {} [THERE, FiniteToBe]
    RefExpr<755683> {}
```

## **Appendix 3 Case Studies**

This appendix gives a number of case studies which, along with their details, are too lengthy to include in the main chapters. Careful study of these examples should provide much insight into how the Lucia comprehension system works.

Each of the case studies below is shown with a box with two columns. On the left is the ID number for the sentence involved, which refers back to the ID numbers given in the corpus listings in Appendix 1. The right side has a list of four important data structures for that sentence: the sentence text, the nested constructions used to build the syntactic analysis, the nested meaning schemas that make up the semantic analysis preceded by “m:”, and the message produced to send to Rosie. Usually these data structures have been abbreviated substantially to show clearly the main structure and any important aspects for this case study. Following the box which gives these main data structures, additional information on key points for this case study are given.

Some notes on notation. In addition to the names of ECG items, many internal symbols used inside Rosie are used to indicate the items that meanings are grounded to. A symbol like “op\_go-to-location1” is what is called a “handle,” which is simply a unique character string used to identify something in memory. Often these handles look like English words or phrases, but internally Lucia and Rosie only care that they are unique. The English is simply to help developers with debugging. A symbol like “O96” or “L94” is a Soar identifier which labels a node in Soar’s working memory or semantic memory. A symbol like “NO-ID3” is a shorthand used here for a handle in Soar that is actually “new-object-id3” with a unique number at the end. These NO-IDs are symbols created by Lucia’s grounding logic for objects derived from the language input that do not yet exist in Rosie’s WM or LTM data.

### A3.1: Simple Commands

*Case Study A0.1: A simple command*

B-054	Go to the kitchen. DriveVerbToTarget[GO, SimplePrepPhrase[TO, TheKitchen[THE, KITCHEN]]] m: MoveToIt[ActionDescriptor[op_go-to-location1], PrepRelation[to1, Kitchen[096]]] command(op_go-to-location1, to1(096))
-------	---

Many of the sentences for Rosie are commands. This one is fairly simple, just saying to go to a particular known location. We will examine it in great detail. Figure A3-1 gives a quick sketch of the constructions and schemas built for this sentence.

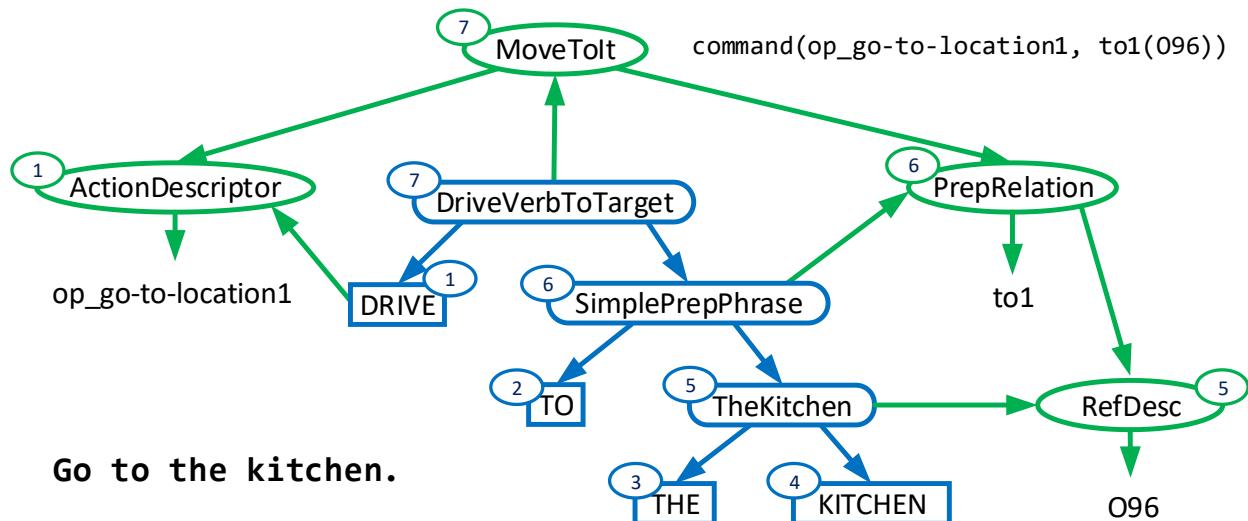


Figure A3-1: Analysis of CS-A3.1

To better understand how ECG works in this case, Table A3-1 shows details of all the constructions used for this sentence. The Supercases show the type hierarchy, and the Constituents the compositional hierarchy based on the types or the orthography for lexical constructions. The generalization process may work up several levels, so that KITCHEN, for instance, is also a CommonNoun and a Noun. The Meaning column shows any schemas invoked plus the meaning constraints used to populate those schemas.

Table A3-1: Constructions for CS-A3.1

Cxn Name	Supercases	Constituents	Meaning
ActionVerb			
SimpleMotionVerb	ActionVerb		ActionDescriptor self.m.class <-- @action
DriveVerb	SimpleMotionVerb		ActionDescriptor self.m.name <-- "go-to-location1"
DRIVE	DriveVerb	"go"	
Preposition			PrepCore
TO	Preposition	"to"	self.m.name <-- "to1"
NPSpecifier			
Determiner	NPSpecifier		
THE	Determiner	"the"	RefDesc self.m.givenness <-- "definite"
Noun			
CommonNoun	Noun		
KITCHEN	CommonNoun	"kitchen"	Kitchen
RefExpr			
SpecifierNP	RefExpr		
TheKitchen	SpecifierNP	THE, KITCHEN	Kitchen self.m.name <--> noun.m.schema-name self.m.category <--> noun.m self.m.givenness <--> spec.m.givenness
PrepPhrase			
SimplePrepPhrase	PrepPhrase	prep: Preposition object: RefExpr	PrepRelation self.m.prep <--> prep.m.name self.m.object <--> object.m
VerbWithArguments			
Imperative	VerbWithArguments		
DriveVerbToTarget	Imperative	verb: DriveVerb target: PrepPhrase	MoveToIt self.m.action <--> verb.m self.m.location <--> target.m

Table A3-2 shows some details of the related meaning schemas. Notice that the generalization process may work up several levels so that Kitchen, for instance, will end up having all these types: Kitchen, MapLocation,

KnownObject, RosieObject, Entity, and RefDesc, along with all the roles that these generalization provide.

Table A3-2: Schemas for CS-A3.1

Schema Name	Supercases	Roles	Constraints
ActionDescriptor		class, name, modifier	
PrepCore		name	
RefDesc		name, category, givenness, modifiers, referent, relation, quantified	
KnownObject	RefDesc	handle	
Entity		rosie-category	
RosieObject	Entity	handle, movable	
MapLocation	KnownObject, RosieObject		rosie-category <-- location
Kitchen	MapLocation		handle <-- loc-kitchen1
PrepRelation		prep, object	
Action		action, direction, location	
MoveToIt	Action	direction, location	

Figure A3-2 shows a diagram of the complete meaning structure.

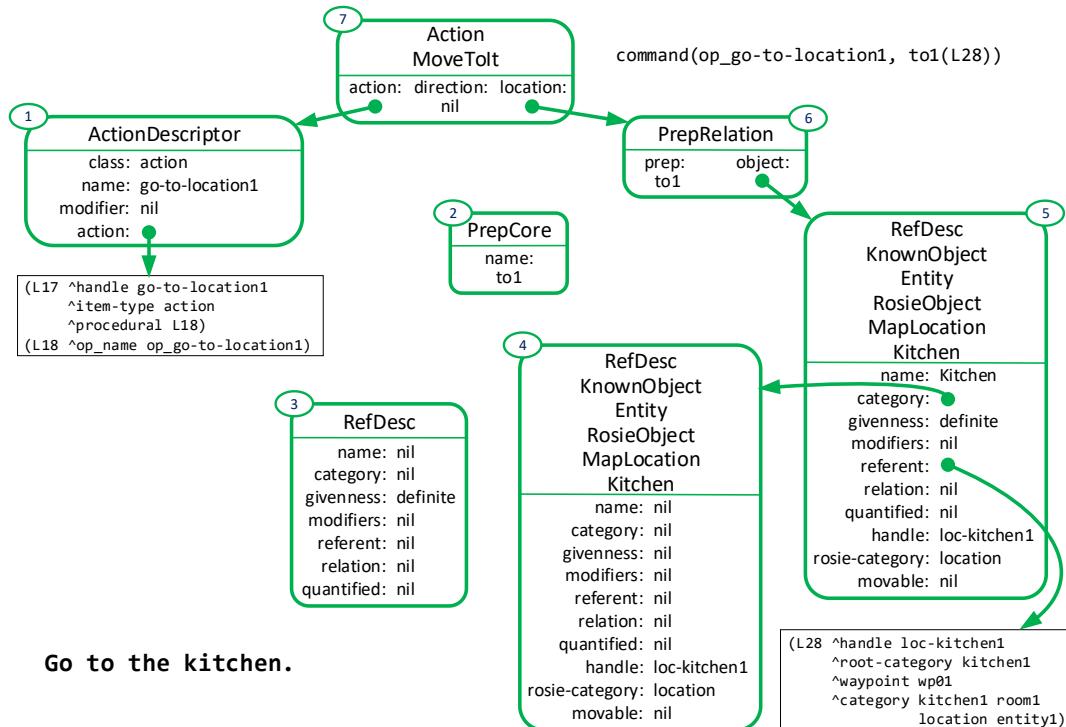


Figure A3-2: Complete meaning structure for CS-A3.1

An interesting detail here is that the ID of the referent for *the kitchen* is O96 as shown in the case study box and in Figure A3-1. However, in Figure A3-2 it is L28. Why is this different? Because when Rosie is doing tasks from the Robot corpus the grounding is dynamic. At the time when the data for Figure A3-2 was captured, Rosie was somewhere else and had to look for the kitchen in its map in LTM, and it got L28 as the reference. In Figure A3-1 it had already gotten to the kitchen so a representation of the kitchen had been put in the World Model, at Lucia grounded to that before looking in the map.

#### Case Study A0.2: A transitive command

B-009	Pick up the green sphere. TransitiveCommand[PickUp[PickVerb[PICK], UP], SpecPropNoun[the green sphere]] m: ActOnIt[ActionDescriptor[op_pick-up1], RefDesc[010]] command(op_pick-up1, 010)
-------	--

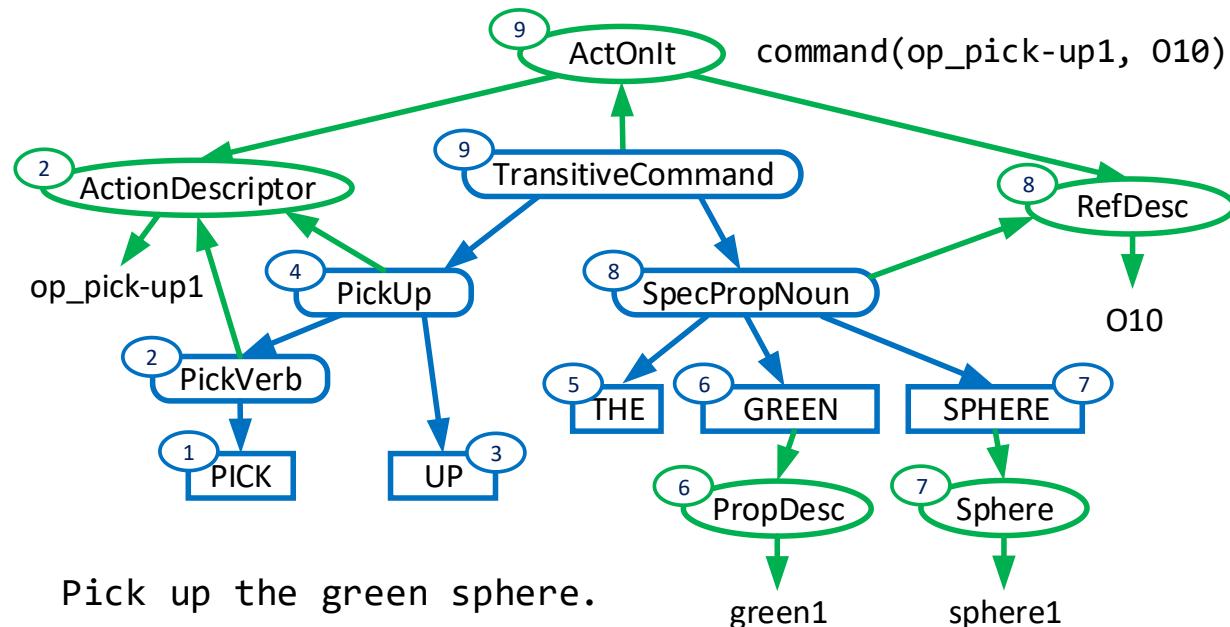


Figure A3-3: Structures built for CS-1

Commands of the type here have an action verb and some object for it to act upon. The action verb here is *pick up*, but its analysis is a bit complicated. In the robotics domain sometimes the word *pick* is used by itself to mean the same thing as *pick up*, so the PickVerb construction recognizes this and evokes

an ActionDescriptor schema, which is grounded to Rosie's symbol for the pick-up action. After *up* has been recognized, a PickUp construction composes PickVerb and UP and attaches to the ActionDescriptor already built and grounded.

The object here is *the green sphere*. The meaning of this phrase is built up piece by piece. First *the* is represented as a determiner, then *green* evokes a PropertyDescriptor (called PropDesc in the figure) defined with a class of color and a name of green1. This grounds to Rosie's perception symbol for the color green. The word *sphere* evokes a schema called Sphere which is a subcase of Block that specifies its shape as sphere1, another symbol that connects to Rosie's visual perception. Finally the SpecPropNoun construction, which is a subcase of RefExpr, evokes a RefDesc schema (short for ReferentDescriptor) and populates its slots with information from its constituents. The resulting RefDesc is then grounded to an object in the World Model that is green and of shape sphere, which in this case happens to be O10.

Finally, the TransitiveCommand construction is built from the ActionVerb supercase of PickUp and the RefExpr supercase of SpecPropNoun. It then evokes an ActOnIt schema, which is composed from the ActionDescriptor and the RefDesc meanings of its constituents. Finally the sentence interpretation logic produces the message shown.

### A3.2: Complex Referring Expressions

*Case Study A0.3: A local repair*

B-020	<pre>Pick up the green block on the stove. TransitiveCommand[PickUp[PickVerb[PICK], UP],   RefExprPrepPhrase[SpecPropNoun[the green block]     SimplePrepPhrase[ON, SpecNoun[the stove]]]] m: ActOnIt[ActionDescriptor[op_pick-up1], RefDesc[08]] command(op_pick-up1, 08)</pre>
-------	--

This sentence is similar to B-009 *Pick up the green sphere.*, but has some additional complications. The current world has several green blocks of different sizes and shapes, and so the grounding of *the green block* is ambiguous, even

though just *Pick up the green block* appears to be a complete sentence and builds a TransitiveCommand. The speaker added the phrase *on the stove* to resolve this ambiguity, but this brings up a second issue: how should this prepositional phrase be attached? The solution to this problem requires an operation we call a *local repair*. Figure A3-4: A local repair shows an example.

In part a) of the figure we see that node 9 has built a TransitiveCommand with its meaning for *Pick up the green block*, but that the object is ambiguous with four options and more words have built up the prepositional phrase in node 14. To attach the PP two constructions are possible, an ImperativeWithLocation to modify the complete command and the other with a RefExprPrePhrase to modify *the green block*.

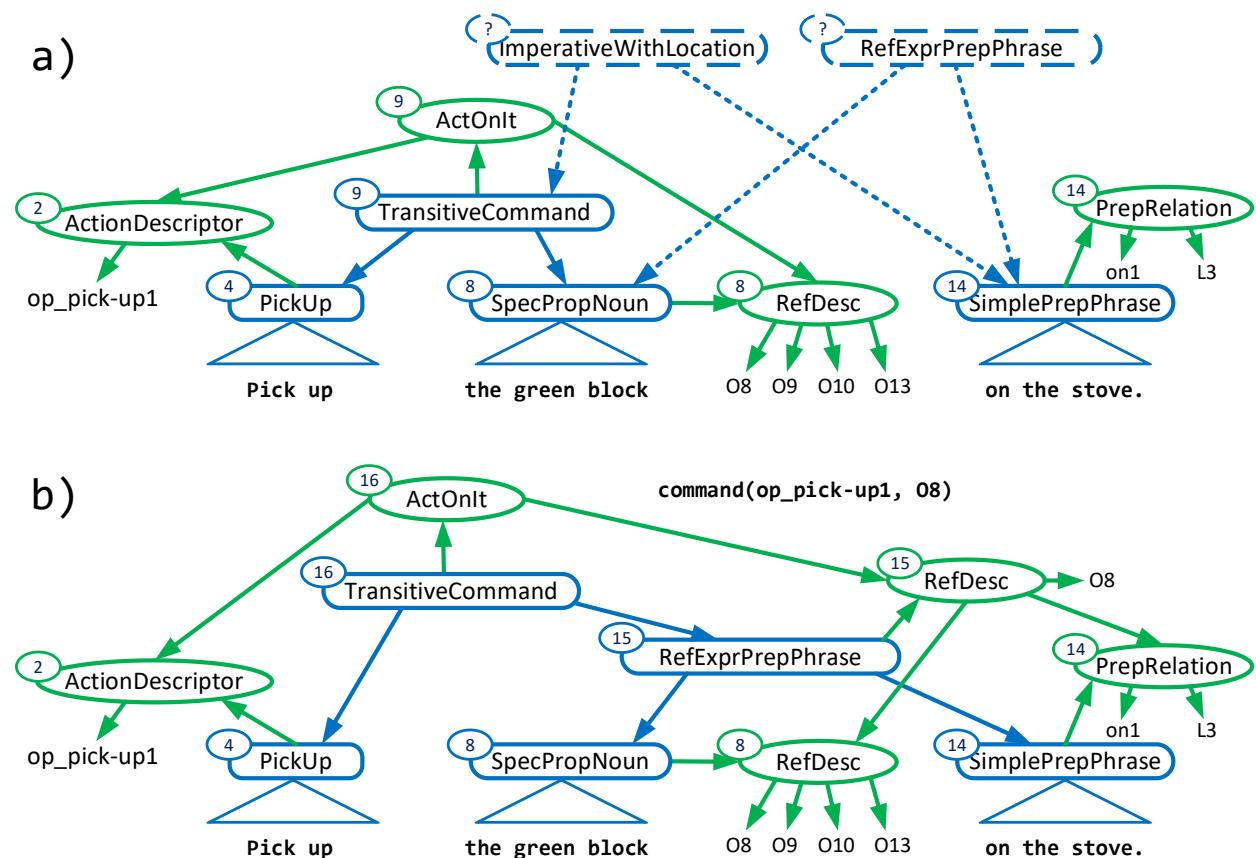


Figure A3-4: A local repair

The ECG formalism does not have a direct mechanism for resolving this ambiguity, so additional Soar code is used to decide that the PP should be attached to the first RefExpr. However, that first RefExpr in node 8 has already been consumed as a constituent of node 9. Following the idea of Lewis (1993), we perform an operation called *snip* to completely discard both the construction and the schema for node 9, which allows node 15 to be built and grounded in part b) of the figure. With the additional information from *on the stove*, this grounding produces a unique referent of O8. With this in place, a new TransitiveCommand is built in node 16 and the sentence can now produce the command message shown.

Although the ECG formalism does not have a specific mechanism for making the attachment decision, a more judicious design of the grammar with the existing formalism could have resolved this problem. With the sentence as is, connecting the PP to the RefExpr is the right solution, but had the verb been *Put* instead of *Pick up* the other attachment would have been better. Had these two verbs been defined with different types, then TransitiveCommand could have been written to only work with verbs of the same type as *Pick up*. Then a different construction could have been used for the other type of verb. At the time when the PP had been processed, the existing structure would determine the attachment. However, the snip to attach the PP to the RefExpr would still have been necessary.

#### *Case Study A0.4: Double prepositional phrases*

B-032	<pre> Move the green rectangle to the left of the large green rectangle to the pantry.  ImperativeWithLocation[     TransitiveCommand[MoveVerb[Move],         RefExprPrepPhrase[SpecPropNoun[the green rectangle],             SimplePrepPhrase[T0-THE-LEFT-OF,                 SpecProps2Noun[the large green rectangle]]]],         SimplePrepPhrase[T0, SpecNoun[the pantry.]]     ]     m: Action[         ActOnIt[ActionDescriptor[move1],             RefDesc[RefDesc[08], PrepRelation[left-of1, RefDesc[09]]]],             PrepRelation[to1, RefDesc[L2]]]     command(move1, 08, to1(L2)) </pre>
-------	--

This sentence illustrates a complicated case which also needs a local repair. First note the *to the left of* is one of the multi-word lexical items that is recognized by a single lexical construction and then treated as if it were a single preposition. When *to the left of the large green rectangle* has been processed, this PP is attached to *Move the green rectangle* in the opposite way as in the previous example, since *move* is a verb that requires a target location. Then when another PP, *to the pantry*, comes along, the ImperativeWithLocation that had been built is discarded, the first PP is integrated with the first RefExpr, and the second PP is used to build a new ImperativeWithLocation.

*Case Study A0.5: A subject relative clause and the word that*

B-028	Pick the green block that is small. TransitiveCommand[PickUp[PickVerb[PICK], UP], RefExprRelClause[SpecPropNoun[the green block] RelativeClauseProperty[ HeadRelativeClause[THAT-relative, IS], SMALL]]] m: ActOnIt[ActionDescriptor[op_pick-up1], RefDesc[013]] command(op_pick-up1, 013)
-------	--

This sentence shows an example of what linguists call a subject relative clause: *that is small*. This clause looks like a complete declarative sentence where the subject of *is* is *that*. The clause relates *that* to the property *small*, thus selected out of the green blocks in the world only one of small size. This case study shows several points about how Lucia does comprehension. Natural language also has object relative clauses, but we have none of these in our corpora. We will consider an example when we look at parsing breakdown below. Here is a list of several sentences in our corpora that all use some form of subject relative clause.

- (1) a B-028 Pick the green block *that is small*.
- b B-030 Pick the green block *that is on the stove*.
- c B-031 Pick a green block *that is larger than the green box*.
- d G-016 A location *that is not below a red block* is accessible.
- e G-053 If a clear location is adjacent to a red block  
 then you can move the red block onto the clear location  
 and move a clear available blue block onto a location  
*that was below the red block*.

f G-011 If the number of the locations between a location and a covered location is the number of the blocks *that are* on the covered location then you can move it onto the former location.

If we compare (1a) to (1b), they both have the same clause structure, except that in (1b) the clause relates *that* to the fact that it is *on the stove*, giving an alternative way to resolve the ambiguous grounding. Notice that B-030 is the same as B-020, our CS-A3.3, except that *that is* has been added, eliminating the ambiguity which required a local repair. Generally speaking, relative clauses serve this purpose of eliminating semantic ambiguity.

The next point is about the way the grammar recognizes these relative clauses. A construction called HeadRelativeClause has the constituents THAT-relative and FiniteToBe. FiniteToBe is a general construction that is a supercase of the lexical constructions for *is*, *are*, and *was* in our grammar. We see examples of all three of these in (1d-f). The HeadRelativeClause captures greedily the fact that *that* followed by some finite form of *to be* is a sure signal that a subject relative clause is beginning, and that this clause will need to attach to the previous referring expression.

Another point is about the word *that*. This function word is used in several different ways, as shown in these examples.

- (2) a B-028 Pick the green block *that is* small.  
b B-075 Pick up *that*.  
c B-145 The goal *is that* the box is in the office.

To handle these cases, Lucia's grammar has three different lexical constructions for *that*. THAT-relative is a subcase of RelativePronoun and from there of Pronoun, THAT-deictic is a subcase of DeicticPronoun and thus Pronoun, and THAT-complementizer is a subcase of Complementizer. In (2a) RelativePronoun and FiniteToBe are combined to form a HeadRelativeClause, which then combines with a Property to form a RelativeClauseProperty, a subcase of RelativeClause, which in turn combines with a RefExpr to form a RefExprRelClause. In (2b), since Pronoun is a subcase of RefExpr, *that* simply

becomes a RefExpr by itself. In (2c) a ConceptIsThat construction spans *The goal is that* by combining a RefExpr with a FiniteToBe and a THAT-complementizer, and this in turn is combined with a Declarative to form a complete sentence with ConceptIsThatDeclarative. This way of grouping things together greedily works very well for doing incremental comprehension with a construction grammar like ECG that assigns meaning structures to every level of syntactic structure and that allows for three-way branching of composite constructions. However, these structures are very different from those found in generative grammar analyses.

Another important point for this case study is that the algorithm must decide which sense of *that* to select. In all cases when lexical access is done for *that* all three options are temporarily posted to the Soar state. Then the following operator makes a selection based on the syntactic context. In (2a) the current stack has a TransitiveCommand with RefExpr[*the green block*] as its last constituent. An attachment operator recognizes that a RelativePronoun can be attached to that RefExpr, snips the TransitiveCommand, and selects the THAT-relative option, allowing a HeadRelativeClause to be recognized after *is*. In (2c) when the sequence at the top of the stack is RefExpr[*The goal*] <- FiniteToBe[*is*] <- *that*, a ConceptIsThat is recognized, and this forces the selection of THAT-complementizer. In (2b) there is no direct or indirect RefExpr preceding *that*, nor is preceded by a RefExpr and a FiniteToBe, so the algorithm proceeds to try grounding *that* as a RefExpr, and selecting THAT-deictic in the process. In all these cases once a lexical sense is selected the others are remembered as alternatives for possible use later on with a local repair.

One more point about *that* is about how it is grounded. THAT-deictic in (2b) is simply grounded to whatever object in the World Model is currently being pointed to by the instructor. The meaning of THAT-relative in (2a) is set to a RefDesc which has all the referents that were found for the grounding of the RefExpr that *that* is attached to. This provides the set that the relative clause can select from for grounding the RefExprRelClause. THAT-complementizer in (2c) serves only a syntactic function and needs no grounding.

### A3.3: Simple Declaratives

*Case Study A0.6: A declarative statement*

B-067	The red triangle is behind the stove. RefIsPrepPhrase[SpecPropNoun[The red triangle], IS, SimplePrepPhrase[behind the stove]] m: PrepPhraseAssertion[RefDesc[07], PrepRelation[behind1, RefDesc[L3]]] object-description(07, behind1(L3))
-------	--

This is a simple statement stating a fact about the world that could be useful to Rosie in learning a task. It illustrates an interesting point about this approach to defining grammar. Notice that the word *is*, represented by the IS construction, has no meaning associated with it. Linguists call *is* a *copula*, a verb that connects two things together. It can be used in many different ways depending on the two things being connected. Some approaches might consider that *is* itself has many different senses for these different situations. Our construction grammar approach uses some specific larger construction to provide a *meaning in context* (Goldberg, 2019). Since *is* can be used in many different ways, in this case study we briefly describe a number of examples from our baseline corpus.

*Case Study A0.7: Lexical ambiguity: uses of the word is*

B-001	The sphere is green. RefIsProperty[SpecNoun[The sphere], IS, Property[green]] m: PropertyApplication[RefDesc[010], PropertyDescriptor[green1]] object-description(010, predicate(green1))
B-007	The red triangle is on the stove. RefIsPrepPhrase[SpecPropNoun[The red triangle], IS, SimplePrepPhrase[ON, SpecNoun[the stove]]] m: PrepPhraseAssertion[RefDesc[07], PrepRelation[on1, RefDesc[L3]]] object-description(07, on1(L3))
B-044	Octagon is a shape. PropertyDefinitionSentence[UNKNOWN-WORD[Octagon], IS, Property1Set[shape]] m: PropertyDefinition[PropertySetDescriptor[predicate[shape]], octagon] adjective-definition(word(octagon), property(shape))

The lexical construction IS is a subcase of FiniteToBe, which as we have seen is used as a constituent of many larger constructions that really determine

what *is* and its related forms mean in larger contexts such as relative clauses or other sentences with subordinate clauses.

### A3.4: Complex Sentences

#### *Case Study A0.8: Commands with conditions*

B-046	If the green box is large then go forward. IfConditionThenCommand[ IfConditionThen[IF, RefIsProperty[the green box is large], THEN], ActInDirection[SimpleAction[go], FORWARD-direction] ] m: IfThenCommand[ PropertyApplication[the green box is large], DoItInDirection[op_go-to-location1, forward] ] conditional(if-subclause(013, predicate(large1)), then-subclause(action(op_go-to-location1)))
-------	--

An Imperative can be preceded by a condition subordinate clause introduced with *If*. This example shows the auxiliary construction IfConditionThen which surrounds the Declarative for the condition with IF and THEN. The main construction combines this with the Imperative for the then clause.

#### *Case Study A0.9: Commands with terminations*

B-119	Drive until you sense a wall. DoUntil[DRIVE, UntilDeclarativeClause[UNTIL, TransitiveSituationClause[YOU, SENSE, SpecNoun[a wall]]] ] m: TerminatedAction[op_go-to-location1, UntilDeclarative[RefDesc[R5], sense1, RefDesc[NO-ID1]]] command(op_go-to-location1, until-clause(agent(R5), action(sense1), 055))
-------	--

An Imperative can also be followed by a conditional subordinate clause introduced with *until*. Here are examples. Many of the Rosie sentences have this form of an action to perform and a condition that will terminate the action. In this case the termination condition uses *you* to refer to Rosie itself, which has the id R5 in the current World Model.

*Case Study A0.10: Enabled commands*

G-004	You can move a clear block onto a clear object. EnabledCommandSentence[EnablerPhrase[You can], ImperativeWithLocation[move a clear block onto a clear object.]] m: EnabledCommand[Enabler[RefDesc[R5], ability[can]], Action[ActOnIt[op_move1, RefDesc[057]], PrepRelation[on1, RefDesc[083]]] command(agent(R5), action(op_move1), action-modifier(can), 057, on1(083))
-------	--

The Games corpus has many sentences like this starting with *You can* in its scripts to define actions, possibly predicated on the current situation in the discourse. These are another example of how the incremental, construction grammar approach of Lucia structures sentences differently than what traditional theories of grammar will do. A traditional approach would consider *can* as an auxiliary attached to the verb *move*. Lucia instead combines *can* with the subject *You* to form what we call an EnablerPhrase which is used like a condition to precede the rest of the sentence, which is a complete Imperative all by itself. A bit of the traditional approach still survives here in the definition of the message structure where *can* is considered a modifier on the action.

*Case Study A0.11: Learning to comprehend a new sentence structure*

G-004	You can move a clear block onto a clear object. EnabledCommandSentence[ EnablerPhrase[You can], ImperativeWithLocation[move a clear block onto a clear object.] ] command(agent(R5), action(move1), action-modifier(can), NO-ID1, on1(NO-ID2))
-------	--

A traditional analysis of the syntax of this sentence would probably say that *can* is an auxiliary verb that modifies *move* and that *you* is the subject of the sentence. The developer of Games apparently was thinking this way when he designed the structure of the action message, since *can* appears as a modifier for the action. However, in the Rosie domain, this does not make a lot of sense since the whole part *move ... object* is a Rosie command all by itself that Lucia already knows how to comprehend. Therefore, leveraging this prior knowledge, we decided to analyze the *You can* at the beginning as a phrase that modifies

this whole command to make it an action that Rosie can remember abstractly for future use. This is an interesting example of how incremental acquisition may produce a very different grammar from one produced by a linguist analyzing the language as a whole.

To comprehend this sentence by this analysis, two new lexical constructions were added: CAN-verb for *can* as a subcase of EnablingVerb, and ONTO for *onto* as a synonym for *on* and a subcase of Preposition. Two new composite constructions were needed: EnablerPhrase with constituents of Agent and EnablingVerb and an Enabler schema, and EnabledCommandSentence with EnablerPhrase and Imperative constituents and an EnabledCommand schema. EnablingVerb is a new general construction with an AuxiliaryFeatures schema, and CAN-verb sets the type role of the AuxiliaryFeatures to “can”. Agent is another new general construction and the lexicals for *you*, *we*, *Rosie*, and the proper names for people were made subcases of Agent. All these new ECG items generate a total of 24 new ECG rules.

These new ECG items gave Lucia the ability to analyze the sentence and are used later for many other *you can* sentences in the Games corpus. In addition, several new hand-built rules were needed: 2 rules to snip away an EnabledCommandSentence sentence when a later PrepPhrase needs to attach to the Imperative that was created after *block*, 11 new grounding rules to deal with issues of hypothetical grounding that had not be addressed before, and 2 new rules in sentence interpretation to format the message for this type of sentence.

This sentence illustrates aspects of all four of the steps in learning to comprehend a new sentence. Most of these new ECG items and hand-built rules are used over again for many other sentences, so they provide a good deal of generality. In terms of the three-phase LAE theory, this sentence shows how a single experience can be reasoned about to produce a lot of new knowledge. However, the generalization done here was not done automatically but depended

on the insights of the human Lucia developer, so this example does not show an easy way to an algorithm for generalization.

#### *Case Study A0.12: Declaratives with conditions*

G-005	<pre>If a location is not below an object then it is clear. IfConditionThenStatement[   IfConditionThen[IF,     PropertySetIsNotPrepPhrase[a location is not below an object],     THEN],   RefIsProperty[it is clear] ] m: IfThenStatement[   PrepPhraseAssertion[a location is not below an object],   PropertyApplication[it is clear] ] conditional(   if-subclause(action(is1), modifier(negation),     below1(NO-ID2)),   then-subclause(action(is1), NO-ID1, predicate(clear)))</pre>
-------	--

In the previous example of G-004, the phrase *You can* refers to some hypothetical future situation while solving the puzzle. At this point Rosie is not sure what *clear* means in this context, so it asks the instructor: *Please describe the meaning of 'clear' in this context*. The instructor responds with the definition given in G-005. Note that *it* is an anaphoric reference. This is discussed further under grounding in Chapter 5.

#### *Case Study A0.13: Complex declaratives*

G-006	<pre>The goal is that a red block is on a green block and the red block is below an orange block. ConceptIsThatDeclarative[ConceptIsThat[The goal is that],   DeclarativeAndDeclarative[     RefIsPrepPhrase[a red block is on a green block],     AndDeclarative[AND,       RefIsPrepPhrase[the red block is below an orange block]]] ] m: ConceptIsThatAssertion[RefDesc[The goal],   CompoundAssertion[     PrepPhraseAssertion[a red block is on a green block],     PrepPhraseAssertion[the red block isbelow an orange block] ] object-description(concept(goal),   subclause(     subclause(action(is1), NO-ID1, on1(NO-ID1)),     subclause(action(is1), NO-ID1, below1(NO-ID3)))) )</pre>
-------	--

G-006 illustrates a complex sentence with two levels of subordinate clauses and two clauses combined with a conjunction. The Games corpus has a number of sentences of this type of complexity.

*Case Study A0.14: Declaratives with conditions and functions*

G-024	<pre>If the volume of a block is more than the volume of an object then the block is larger than the object.  IfConditionThenStatement[   IfConditionThen[IF,     RefIsPrepPhrase[       FunctionWithArgument[THE-VOLUME-OF, SpecNoun[a block]],       IS,       SimplePrepPhrase[MORE-THAN,         FunctionWithArgument[the volume of an object]]]     THEN],     RefIsPrepPhrase[SpecNoun[the block], IS,       SimplePrepPhrase[LARGER-THAN, SpecNoun[the object]]] ]   m: IfThenStatement[     PrepPhraseAssertion[       FunctionApplication[volume, RefDesc[an block]],       PrepRelation[more-than1,         FunctionApplication[volume, RefDesc[an object]]]],     PrepPhraseAssertion[RefDesc[the block],       PrepRelation[larger-than1, RefDesc[the object]]] ]   conditional(     if-subclause(action(is1),       volume-of(NO-ID1), more-than(volume-of(NO-ID2))),     then-subclause(action(is1), larger-than(NO-ID1, NO-ID2)))</pre>
-------	--

G-024 is used to define *larger than* using hypothetical objects. It also illustrates a grammatical form used in many Games sentences called a *function*, in this case *the volume of*. Notice how the message looks simpler than the comprehension structures because the new object id's are used to represent all the properties of an object.

*Case Study A0.15: Yes/No questions*

B-098	<pre>Is this red? IsObjectClassQ[IS, RefExpr[this], Property[red] ] m: ObjectIsClassQuestion[RefDesc[06],   PropertyDescriptor[color, red1] ] object-question(06, color(red1))</pre>
-------	--

B-077	Is this a sphere? IsObjectPropSetQ[IS, RefExpr[this], Property1Set[a sphere] ] m: ObjectIsClassQuestion[RefDesc[06], PropertySetDescriptor[shape, sphere1] ] object-question(06, shape(sphere1))
B-078	Is the green sphere on the table? IsObjectRelation[IS, SpecPropNoun[the green sphere], SimplePrepPhrase[ON, SpecNoun[the table]]] m: ObjectIsRelationQuestion[RefDesc[010], PrepRelation[on1, L6] ] object-question(010, on1(L6))

Chapter 3 showed the top-level grammar network for questions, and here we give some detailed examples. Notice that there is no “movement” involved, and even that the word *is* is represented in the syntax but its function and meaning are absorbed into the sentence-level meaning schema with no further trace of the word itself. This illustrates how composite constructions are used to represent meaning that is not represented at the lexical level in this theory of comprehension.

#### *Case Study A0.16: Wh- questions*

B-096	What is this? WhatIsObject[WHAT, IS, RefExpr[this]] m: WhatIsObjectQuestion[RefDesc[06]] what-is-question(06)
B-062	What is on the red triangle? WhatIsPrepPhrase[WHAT, IS, SimplePrepPhrase[ON, SpecPropNoun[the red triangle]]] m: WhatIsRelation[PrepRelation[on1, RefDesc[07]]] what-is-question(predicate(on1, 07))
B-090	What color is the large sphere? WhatClassIsObjectQ[WHAT, Property1Set[color], IS, SpecPropNoun[the large sphere]] m: WhatClassIsObjectQuestion[PropertySetDescriptor[color], RefDesc[010]] predicate-question(06, predicate(color))
B-048	Where is the red triangle? WheresWaldo[WHERE, IS, SpecPropNoun[the red triangle]] m: WhereIsObject[RefDesc[07]] where-is-question(07)

Here again we see how complexity in the string of lexical items is absorbed by high-level constructions and schemas.

### *Case Study A0.17: Dynamic grounding for Robot tasks*

Most of the sentences used in the Robot corpus for teaching Rosie a variety of tasks involving navigation and manipulation of objects in a real environment appear refreshingly simple when compared to many of the Games sentences. However, the grounding process for these sentences is complex because the agent is moving around in the environment and thus objects move in and out of its visual perception and may or may not be remembered from prior experience. Here we look at a few examples that illustrate the sorts of issues involved.

R-001	Find the fork. TransitiveCommand[FIND, SpecNoun[the fork]] m: ActOnIt[op_find1, RefDesc[NO-ID1]] command(op_find1, NO-ID1)
-------	---

Suppose Rosie is in the kitchen and the instructor wants to tell it to move the fork onto the table. However, the fork is in the kitchen but not currently visible to the agent. When the instructor says R-001, *Find the fork.*, Lucia has nothing to ground *the fork* to. Lucia handles this by creating a new object id for the fork, and passing that new object along in the message produced.

R-002	Move the fork onto the table. ImperativeWithLocation[ TransitiveCommand[Move the fork], SimplePrepPhrase[onto the table] ] m: Action[ActOnIt[move1, 0530], PrepRelation[on1, 0118] ] command(move1, 0530, on1(0118))
-------	---

Rosie acts on R-001 by scanning around the kitchen until it sees the fork on the counter and enters it into the World Model. When Lucia processes R-002, it grounds the fork to that visible object, O530 here, and the table to O118.

R-009	The only goal is that the fork is on the table. ConceptIsThatDeclarative[ConceptIsThat[The only goal is that], RefIsPrepPhrase[the fork is on the table] ] m: ConceptIsThatAssertion[RefDesc[The only goal], PrepPhraseAssertion[the fork is on the table] ] object-description(concept(goal), modifier1(only1), subclause(action(is1), 0530, on1(0118)) )
-------	--

When teaching this script, Rosie does not yet know what the verb *move* means, so it asks the instructor what the goal is and the instructor answers with R-009. The word *only* is used here because in these tasks there are sometimes multiple goals to be considered for the same action.

R-026	Turn right twenty-five degrees. TurnDirectionByAngle[TurnDirection[TURN, RIGHT], NumberOfUnits[TWENTY-FIVE, DEGREES] ] m: TurnByAngle[] command(action(op_turn1), relative-direction1(right1), number(25), unit(degrees) )
-------	---

R-026 illustrates another area of the grammar to deal with complex verbs and measurements using specialized constructions that provide semantic precision. TurnDirectionByAngle is a semantically precise subcase of Imperative, NumberOfUnits is a precise subcase of SpecifierNP, and TurnByAngle is a precise subcase of Action. TWENTY-FIVE is a subcase of CardinalNumber that gives its value as 25. Rosie's operational subsystem knows how to interpret all this and physically turn the agent by 25 degrees to the right.

R-008	Go to the starting location. DriveVerbToTarget[DRIVE, SimplePrepPhrase[to the starting location] ] m: MoveToIt[op_go-to-location1, PrepRelation[to the starting location] ] command(action(op_go-to-location1), predicate(NO-ID1, modifier1(starting1)) )
-------	---

R-008 is tricky since *the starting location* is a phrase whose exact meaning is very context dependent. Rosie has to remember where it was when it started learning or performing a given task. This information will not be in the World Model, so Lucia creates a new object and marks it as having the modifier *starting1*. Then Rosie does some reasoning to find what actual location this should refer to.

### A3.5: Sentences That Are Problematic for Humans

The English language has many sentences that are considered grammatical by traditional standards of analysis, but are hard or impossible for humans to

understand, at least using just automatic processing without a lot of deliberate analysis. Lewis (1993) considers a number of these cases. Here we consider two sentences from his lists, a garden-path sentence and a center-embedded sentence that causes what Lewis calls “parsing breakdown.” We also consider one of the several very large sentences in the Games corpus that a normal human would have grave difficulty with. All three of these case studies will illustrate how the Lucia theory of comprehension based on construction grammar and incremental processing using finite cognitive resources gives at least one plausible explanation for the difficulty humans have with these sentences. Note that the sentences from Lewis have been analyzed using the Lucia theory and syntactic parsing, but do not produce grounded action messages.

*Case Study A0.18: A garden-path sentence*

Lewis GP-14	The horse raced past the barn fell. <i>See Case Study 6.1 for this sentence.</i>
----------------	---

*Case Study A0.19: Sentences with center embedding*

Lewis PB1	The man that the woman that the dog bit likes eats fish. <i>See Case Study 6.2 for a similar sentence.</i>
Lewis PB2	The man the woman the dog bit likes eats fish. <i>See Case Study 6.2 for a similar sentence.</i>

*Case Study A0.20: A monster sentence*

G-011	<pre>If the number of the locations between a location and a covered location is the number of the blocks that are on the covered location then you can move it onto the former location.  IfConditionThenStatement[   IfConditionThen[IF,     RefIsRef[the number of ... the covered location], THEN],     EnabledCommandSentence[EnablerPhrase[you can],       ImperativeWithLocation[move it onto the former location]]   m: IfThenStatement     EqualComparison[       FunctionApplication[the number of the locations         between a location and a covered location],       FunctionApplication[the number of the blocks         that are on the covered location]],       EnabledCommand[Enabler[you can],         Action[move it onto the former location]]     conditional(       if-subclause(action(is1),         number-of(NO-ID1, between(NO-ID2, NO-ID3)),         number-of(NO-ID4, on1(NO-ID3))),       then-subclause(agent(R5), action(move1), action-modifier(can), NO-ID4,         on1(NO-ID2)))</pre>
-------	---

We call this a monster sentence because it is so long and complicated that a normal human can't really understand it without a lot of deliberate effort. Nevertheless, it is in the Games corpus, Lucia has been made to comprehend it, and Rosie understands how to act on it.

One difficulty with this sentence is resolving the anaphoric reference for *it*. The intended coreferent is *the blocks*, which doesn't agree by number, and is also far away and buried within other phrases so that normal human processing will not find it. Therefore, to achieve correct operation of the script in Rosie using Lucia, a special Soar rule for handling this particular coreference was added to Lucia. Obviously this violates our attempt to keep things as general as possible.

In Chapter 5 we discuss the Lucia theory of structured working memory and its limits. As mentioned there, this sentence violates the rule that the maximum depth of the comprehension state is about 7, while this sentence requires a depth of 10. Lucia's implementation does not actually enforce this limit, so Lucia can process the sentence. This violation seems one reasonable explanation for why a human cannot process this sentence easily.

Despite these problems, Figure A3-5 shows an abbreviated version of the construction hierarchy that Lucia produces for this sentence. Several local repairs are needed to build this structure, the sentence uses the function *the number of* in two places.

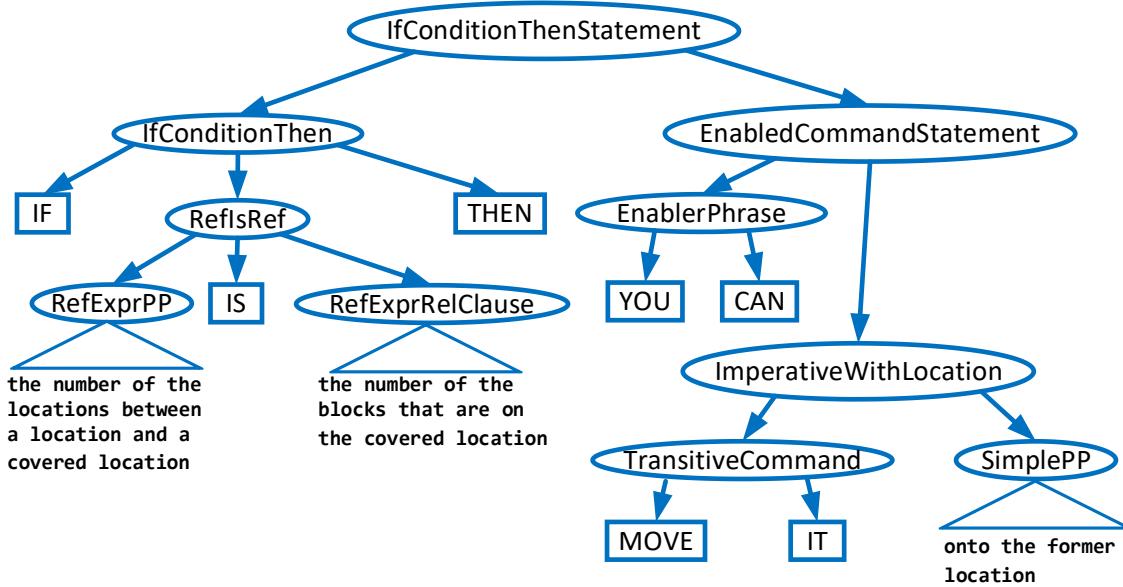


Figure A3-5: Syntactic structure of a monster sentence

Figure A3-6 shows a very abbreviated version of the semantic hierarchy. In this figure the schema nodes are represented as green ovals with simplified names of their schemas. Each schema also has one or more blue rectangles containing the part of the original text that produced that schema. An important observation is that there are several coreferences between pairs of referring expressions that refer to the same hypothetical object. For example, obj-7 is an object referred to by both phrase 5 *a location* and phrase 19 *the former location*. Similarly obj-8 represents an object referred to by both phrase 7 *a covered location* and phrase 12 *the covered location*. A strange case is obj-10, referred to by both phrase 10 *the blocks* and phrase 17 *it*. This is not grammatical English since the number property of these two phrases does not match, and it is very hard to see how *it* refers to *the blocks*. Both humans and Lucia have a hard time with this coreference. A very special custom Soar rule had to be added to Lucia to make this connection.

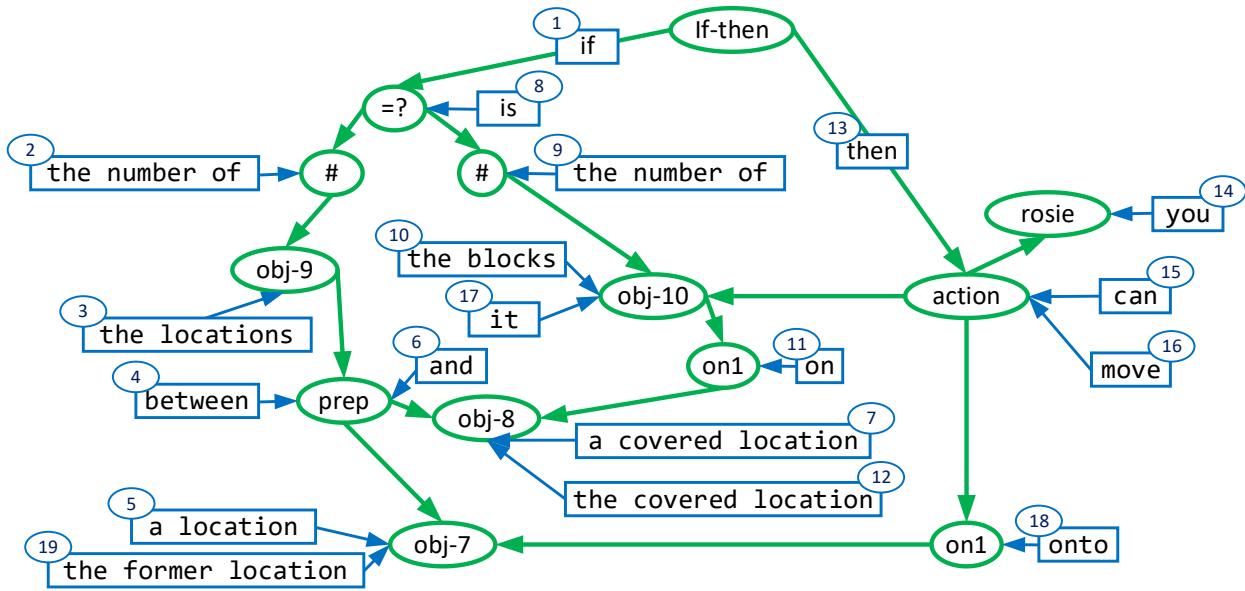


Figure A3-6: Semantic structure of a monster sentence

In spite of the challenges of this sentence, it is useful to illustrate some of the limits of what humans can process easily.

## Bibliography

- AAAI. (2017). Computational Construction Grammar and Natural Language Understanding. *Papers from the 2017 AAAI Spring Symposium*.
- Adger, D. (2003). *Core Syntax: A Minimalist Approach*. Oxford University Press.
- Anderson, J. R. (1977). Computer Simulation of a Language Acquisition System: A Second Report. In D. LaBerge & S. J. Samuels (Eds.), *Basic Processing in Reading: Perception and Comprehension* (1st ed., pp. 305–360). Lawrence Erlbaum Associates, Inc.
- Anderson, J. R. (2007). *How Can the Human Mind Occur in the Physical Universe?* Oxford University Press.
- Anderson, J. R., & Bower, G. H. (1973). *Human Associative Memory*. Lawrence Erlbaum Associates, Inc.
- Artzi, Y., Lee, K., & Zettlemoyer, L. (2015). Broad-coverage CCG Semantic Parsing with AMR. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1699–1710. <http://yoavartzi.com/amr>
- Baddeley, A. (2012). Working memory: Theories, models, and controversies. *Annual Review of Psychology*, 63, 1–29. <https://doi.org/10.1146/annurev-psych-120710-100422>
- Ball, J., Freiman, M., Rodgers, S., & Myers, C. (2010). Toward a Functional Model of Human Language Processing. *32nd Annual Meeting of the Cognitive Science Society*, 1583–1588.
- Ball, J., Myers, C., Heiberg, A., Cooke, N., Matessa, M., & Freiman, M. (2009). The Synthetic Teammate Project. *18th Annual Conference on Behavior Representation in Modeling and Simulation 2009, BRiMS 2009, April*, 73–80.
- Ball, J. T. (2004). *THE DOUBLE R THEORY OF LANGUAGE COMPREHENSION*. <http://stinet.dtic.mil>
- Ball, J. T. (2012). Explorations in ACT-R Based Language Analysis - Memory Chunk Activation, Retrieval and Verification without Inhibition. *Proceedings*

*of the 11th International Conference on Cognitive Modeling, ICCM 2012*, 131–136.

Ball, J. T. (2013). Modeling the Binding of Implicit Arguments in Complement Clauses in ACT-R/Double-R. *Proceedings of the 12th International Conference on Cognitive Modeling*, 83–88.

Barsalou, L. W. (1999). Perceptual symbol systems. *Behavioral and Brain Sciences*, 22, 577–660. [www.service.emory.edu/~barsalou/](http://www.service.emory.edu/~barsalou/)

Barsalou, L. W. (2008). Grounded cognition. *Annual Review of Psychology*, 59, 617–645. <https://doi.org/10.1146/annurev.psych.59.103006.093639>

Bergen, B. K. (2012). *Louder Than Words: The New Science of How the Mind Makes Meaning*. Basic Books.

Bergen, B. K., & Chang, N. C. (2013). Embodied Construction Grammar. In T. Hoffmann & G. Trousdale (Eds.), *The Oxford Handbook of Construction Grammar* (pp. 168–190). Oxford University Press.

Bhattasali, S., Hale, J., Pallier, C., Brennan, J., Luh, W.-M., Nathan, R., Shohini Bhattasali, A., Nathan Spreng, R., & Brennan, J. R. (2018). Differentiating Phrase Structure Parsing and Memory Retrieval in the Brain. *Proceedings of the Society for Computation in Linguistics*, 1, 9. <http://scholarworks.umass.edu/scil> <http://scholarworks.umass.edu/scilhttp://scholarworks.umass.edu/scil/vol1/iss1/9>

Bloom, P. (2000). *How Children Learn the Meanings of Words*. The MIT Press.

Bock, K., Dell, G. S., Chang, F., & Onishi, K. H. (2007). Persistent structural priming from language comprehension to language production. *Cognition*, 104(3), 437–458. <https://doi.org/10.1016/j.cognition.2006.07.003>

Bothell, D. (2021). *ACT-R 7.21 Reference Manual*. <http://act-r.psy.cmu.edu/actr7.x/reference-manual.pdf>

Brady, T. F., Störmer, V. S., & Alvarez, G. A. (2016). Working memory is not fixed-capacity: More active storage capacity for real-world objects than for simple stimuli. *Proceedings of the National Academy of Sciences of the United States of America*, 113(27), 7459–7464. <https://doi.org/10.1073/pnas.1520027113>

Brennan, J. R., & Hale, J. T. (2019). Hierarchical structure guides rapid linguistic predictions during naturalistic listening. *PLoS ONE*, 14(1). <https://doi.org/10.1371/journal.pone.0207741>

- Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Openai, D. A. (2020). *Language Models are Few-Shot Learners*.
- Bryant, J. E. (2008). *Best-fit Constructional Analysis*. PhD dissertation. University of California at Berkeley.
- Carroll, D. W. (2008). *Psychology of Language* (Fifth Edit). Wadsworth.
- Chai, J. Y., Fang, R., Liu, C., & She, L. (2016). Collaborative Language Grounding Toward Situated Human-Robot Dialogue. *AI Magazine*, 32–45.
- Chang, N. C. (2008). *Constructing grammar: A computational model of the emergence of early constructions*. PhD dissertation. University of California at Berkeley.
- Chen, D. L. (2012). Fast Online Lexicon Learning for Grounded Language Acquisition. *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics, ACL 2012*, 430–439. <https://www.aclweb.org/anthology/P12-1045/>
- Chen, D. L., & Mooney, R. J. (2011). Learning to interpret natural language navigation instructions from observations. *Proceedings of the National Conference on Artificial Intelligence*, 1, 859–865.
- Chomsky, N. (1957). *Syntactic structures*. Mouton.
- Chomsky, N. (1965). *Aspects of the Theory of Syntax*. The MIT Press.
- Chomsky, N. (1980). *Rules and Representations*. Columbia University Press.
- Chomsky, N. (1986). *Knowledge of Language: Its Nature, Origin, and Use*. Praeger Publishers.
- Chomsky, N. (2017). Language architecture and its import for evolution. *Neuroscience and Biobehavioral Reviews*, 81, 295–300. <https://doi.org/10.1016/j.neubiorev.2017.01.053>
- Christiansen, M. H., & Chater, N. (2016). The Now-or-Never bottleneck: A fundamental constraint on language. *Behavioral and Brain Sciences*, 1–72. <https://doi.org/10.1017/S0140525X1500031X>

- Cowan, N. (2017). The many faces of working memory and short-term storage. *Psychonomic Bulletin and Review*, 24(4), 1158–1170. <https://doi.org/10.3758/s13423-016-1191-6>
- Dąbrowska, E. (2015). What exactly is Universal Grammar, and has anyone seen it? *Frontiers in Psychology*, 6(June). <https://doi.org/10.3389/fpsyg.2015.00852>
- Damonte, M., & Monti, E. (2021). One Semantic Parser to Parse Them All: Sequence to Sequence Multi-Task Learning on Semantic Parsing Datasets. *Proceedings of the 10th Conference on Lexical and Computational Semantics*, 173–184.
- Demir, M., McNeese, N. J., Cooke, N. J., Ball, J. T., Myers, C., & Friedman, M. (2015). Synthetic teammate communication and coordination with humans. *Proceedings of the Human Factors and Ergonomics Society*, 2015-January, 951–955. <https://doi.org/10.1177/1541931215591275>
- Devlin, J., Chang, M.-W., Lee, K., Google, K. T., & Language, A. I. (2019). *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. <https://github.com/tensorflow/tensor2tensor>
- Dodge, E. K. (2010). *Constructional and Conceptual Composition*. PhD dissertation. University of California at Berkeley.
- Dove, G. (2014). Thinking in words: Language as an embodied medium of thought. *Topics in Cognitive Science*, 6(3), 371–389. <https://doi.org/10.1111/tops.12102>
- Eppe, M., Trott, S., & Feldman, J. (2016). Exploiting Deep Semantics and Compositionality of Natural Language for Human-Robot-Interaction. *IEEE International Conference on Intelligent Robots and Systems*, 2016-Novem, 731–738. <https://doi.org/10.1109/IROS.2016.7759133>
- Eppe, M., Trott, S., Raghuram, V., Feldman, J., & Janin, A. (2016). Application-Independent and Integration-Friendly Natural Language Understanding. *GCAI 2016. 2nd Global Conference on Artificial Intelligence Application-Independent*, 41, 340–352.
- Feldman, J. A. (2006). *From Molecule to Metaphor: A Neural Theory of Language*. The MIT Press.

- Feldman, J., Dodge, E., & Bryant, J. (2009). Embodied Construction Grammar. In *The Oxford Handbook of Linguistic Analysis*. <https://doi.org/10.1093/oxfordhb/9780199544004.013.0006>
- Fellbaum, C. (Ed.). (1998). *WordNet: An Electronic Lexical Database*. The MIT Press.
- Ferrand, L., & New, B. (2004). Semantic and Associative Priming in the Mental Lexicon. In P. Bonin (Ed.), *Mental lexicon: "Some words to talk about words"* (pp. 25–43). Nova Science Publishers.
- Fillmore, C. J. (1976). Frame Semantics and the Nature of Language. *Annals of the New York Academy of Sciences*, 280(1), 20–32. <https://doi.org/10.1111/j.1749-6632.1976.tb25467.x>
- Fillmore, C. J. (1988). The Mechanisms of “Construction Grammar”. *Proceedings of the Fourteenth Annual Meeting of the Berkeley Linguistics Society*, 35–55. <https://doi.org/10.3765/bls.v14i0.1794>
- Fillmore, C. J., & Baker, C. F. (2009). A Frames Approach To Semantic Analysis. In B. Heine & H. Narrog (Eds.), *The Oxford Handbook of Linguistic Analysis* (pp. 313–340). Oxford University Press. <https://doi.org/10.1093/oxfordhb/9780199544004.013.0013>
- Fillmore, C. J., Kay, P., & O’Connor, M. C. (1988). Regularity and Idiomaticity in Grammatical Constructions: The Case of Let Alone. *Language*, 64(3), 501–538.
- Fincher-Kiefer, R. (2019). *How the Body Shapes Knowledge: Empirical Support for Embodied Cognition*. American Psychological Association.
- Fitts, P. M., & Posner, M. I. (1967). *Hunan performance*. Brooks/Cole Pub. Co.
- Gallese, V., & Lakoff, G. (2005). The brain’s concepts: the role of the sensory-motor system in conceptual knowledge. *Cognitive Neuropsychology*, 22(3/4), 455–479. <https://doi.org/10.1080/02643290442000310>
- Garcia-Casademont, E., & Steels, L. (2016). Insight Grammar Learning. *Journal of Cognitive Science*, 17, 27–62.
- Glenberg, A. M. (2015). Few Believe the World Is Flat: How Embodiment Is Changing the Scientific Understanding of Cognition. *Canadian Journal of Experimental Psychology*, 69(2), 165–171. <https://doi.org/10.1037/cep0000056>

Gluck, K. A., & Laird, J. E. (Eds.). (2018a). *Interactive Task Learning: Humans, Robots, and Agents Acquiring New Tasks through Natural Instructions*. The MIT Press.

Gluck, K. A., & Laird, J. E. (Eds.). (2018b). *Interactive Task Learning: Humans, Robots, and Agents Acquiring New Tasks through Natural Interactions*. The MIT Press.

Goldberg, A. E. (1995). *Constructions: A Construction Grammar Approach to Argument Structure*. The University of Chicago Press.

Goldberg, A. E. (2003). Constructions: A new theoretical approach to language. *Trends in Cognitive Sciences*, 7(5), 219–224. [https://doi.org/10.1016/S1364-6613\(03\)00080-9](https://doi.org/10.1016/S1364-6613(03)00080-9)

Goldberg, A. E. (2006). *Constructions at work: The nature of generalization in language*. Oxford University Press.

Goldberg, A. E. (2013). Constructionist Approaches. In T. Hoffmann & G. Trousdale (Eds.), *The Oxford Handbook of Construction Grammar* (pp. 15–31). Oxford University Press. <https://doi.org/10.1093/oxfordhb/9780195396683.013.0002>

Goldberg, A. E. (2019). *Explain Me This: Creativity, Competition, and the Partial Productivity of Constructions*. Princeton University Press.

Goldwasser, D., & Roth, D. (2011). Learning from Natural Instructions. *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence*, 1794–1800.

Gopalan, N., Rosen, E., Konidaris, G., & Tellex, S. (2020, July). Simultaneously Learning Transferable Symbols and Language Groundings from Perceptual Data for Instruction Following. *Robotics: Science and Systems*.

Hauser, M. D., Chomsky, N., & Fitch, W. T. (2002). The Faculty of Language : What Is It , Who Has It , and How Did It Evolve? *Science*, 298(5598), 1569–1579. <https://www.jstor.org/stable/3832837>

Hoffman, T., & Trousdale, G. (Eds.). (2013a). *The Oxford Handbook of Construction Grammar*. Oxford University Press.

Hoffman, T., & Trousdale, G. (Eds.). (2013b). *The Oxford Handbook of Construction Grammar*. Oxford University Press. <https://doi.org/10.1093/oxfordhb/9780195396683.001.0001>

- Isbilen, E. S., & Christiansen, M. H. (2020). Chunk-Based Memory Constraints on the Cultural Evolution of Language. *Topics in Cognitive Science*, 12(2), 713–726. <https://doi.org/10.1111/tops.12376>
- Jackendoff, R. (2003). Précis of Foundations of Language: Brain, Meaning, Grammar, Evolution. *Behavioral and Brain Sciences*, 26, 651–707. <https://doi.org/10.1017/s0140525x03000153>
- Johnson, M. (1987). *The Body in the Mind: The Bodily Basis of Meaning, Imagination, and Reason*. The University of Chicago Press.
- Johnson, M. (2018). The Embodiment of Language. In *The Oxford Handbook of 4E Cognition* (pp. 622–640). Oxford University Press. <https://doi.org/10.1093/oxfordhb/9780198735410.013.33>
- Johnson, M., & Lakoff, G. (2002). Why cognitive linguistics requires embodied realism. *Cognitive Linguistics*, 13(3), 245–263.
- Jones, S. M. (2020). A grammatically robust cognitive model of English and Korean sentence processing. *Proceedings of the 18th International Conference on Cognitive Modelling (ICCM 2020)*. <https://iccm-conference.neocities.org/2020/ICCM2020Proceedings.pdf>
- Jurafsky, D. (1992). An On-Line Computational Model of Human Sentence Interpretation. *AAAI-92 Proceedings*, 302–308.
- Kay, P., & Fillmore, C. J. (1999). Grammatical Constructions and Linguistic Generalizations : The What's X doing Y? Construction. *Language*, 75(1), 1–33.
- Kemmerer, D. (2015). *Cognitive Neuroscience of Language*. Psychology Press. <https://doi.org/10.4324/9781315764061>
- Kirk, J., Mininger, A., & Laird, J. (2016). Learning task goals interactively with visual demonstrations. *Biologically Inspired Cognitive Architectures*, 18, 1–8. <https://doi.org/10.1016/j.bica.2016.08.001>
- Kirk, J. R. (2019). *Learning Hierarchical Compositional Task Definitions through Online Situated Interactive Language Instruction*. PhD dissertation. University of Michigan.
- Kirk, J. R., & Laird, J. E. (2016). Learning General and Efficient Representations of Novel Games Through Interactive Instruction. *Advances in Cognitive Systems*, 4.

- Kirk, J. R., & Laird, J. E. (2019). Learning hierarchical symbolic representations to support interactive task learning and knowledge transfer. *IJCAI International Joint Conference on Artificial Intelligence, 2019-Augus*, 6095–6102. <https://doi.org/10.24963/ijcai.2019/844>
- Kotseruba, I., & Tsotsos, J. K. (2020). 40 Years of Cognitive Architectures: Core Cognitive Abilities and Practical Applications. In *Artificial Intelligence Review* (Vol. 53). Springer Netherlands. <https://doi.org/10.1007/s10462-018-9646-y>
- Krashen, S. (1985). *The Input Hypothesis: Issues and Implications*. Longman.
- Krashen, S. (2003). *Explorations in Language Acquisition and Use: The Taipei Lectures* (Heineman, Ed.).
- Kuhl, P. K. (2000). A new view of language acquisition. *Proceedings of the National Academy of Sciences*, 97(22), 11850–11857. [www.pnas.org](http://www.pnas.org)
- Laird, J. E. (2012). *The Soar Cognitive Architecture*. The MIT Press.
- Laird, J. E. (2021). *An Analysis and Comparison of ACT-R and Soar*. <http://laird.engin.umich.edu/wp-content/uploads/sites/331/2021/08/formal-3.pdf>.
- Laird, J. E., Anderson, J., Forbus, K. D., Lebiere, C., Salvucci, D., Scheutz, M., Thomaz, A., Trafton, G., Wray, R. E., Mohan, S., & Kirk, J. R. (2017). Interactive Task Learning. *IEEE Intelligent Systems*, 32(4), 6–21 (invited). <https://doi.org/10.1109/MIS.2017.3121552>
- Laird, J. E., Lebiere, C., & Rosenbloom, P. S. (2017). A Standard Model of the Mind: Toward a Common Computational Framework across Artificial Intelligence, Cognitive Science, Neuroscience, and Robotics. *AI Magazine*, 38(4), 13. <https://doi.org/10.1609/aimag.v38i4.2744>
- Laird, J. E., Mohan, S., Kirk, J., & Mininger, A. (2018). Characteristics of the Learning Problem in Situated Interactive Task Learning. In K. A. Gluck & J. E. Laird (Eds.), *Interactive Task Learning: Humans, Robots, and Agents Acquiring New Tasks through Natural Interactions* (pp. 273–291). The MIT Press.
- Laird, J. E., & Rosenbloom, P. S. (1996). The Evolution of the Soar Cognitive Architecture. In D. Steier & T. M. Mitchell (Eds.), *Mind Matters: A Tribute to Allen Newell*. Lawrence Erlbaum Associates, Inc.

- Lakoff, G. (1987). *Women, Fire, and Dangerous Things: What Categories Reveal about the Human Mind*. The University of Chicago Press.
- Lakoff, G. (1990). The Invariance Hypothesis: is abstract reason based on image-schemas? *Cognitive Linguistics*, 1(1), 39–74.
- Lakoff, G. (2012). Explaining Embodied Cognition Results. *Topics in Cognitive Science*, 4(4), 773–785. <https://doi.org/10.1111/j.1756-8765.2012.01222.x>
- Lakoff, G., & Johnson, M. (1980). *Metaphors We Live By*. University of Chicago Press.
- Lehman, J. F., Laird, J. E., & Rosenbloom, P. S. (1996). *A Gentle Introduction to Soar, an Architecture for Human Cognition*. <https://apps.dtic.mil/sti/pdfs/ADA314690.pdf>
- Lewis, R. L. (1993). *An Architecturally-based Theory of Human Sentence Comprehension*. PhD dissertation. Carnegie Mellon University.
- Lewis, R. L., & Vasishth, S. (2005). An Activation-Based Model of Sentence Processing as Skilled Memory Retrieval. *Cognitive Science*, 29, 375–419.
- Liang, P. (2016). Learning Executable Semantic Parsers for Natural Language Understanding. *Communications of the ACM*, 59(9), 68–76. <https://doi.org/10.1145/2866568>
- Lindes, P. (2018). The Common Model of Cognition and humanlike language comprehension. *Procedia Computer Science*, 145, 765–772. <https://doi.org/10.1016/j.procs.2018.11.032>
- Lindes, P. (2019). Predictions of a Model of Language Comprehension Compared to Brain Data. *Proceedings of the 17th International Conference on Cognitive Modeling (ICCM 2019)*.
- Lindes, P. (2020). Constructing Meaning in Small Increments. In *Poster presented at CogSci 2020*.
- Lindes, P., & Laird, J. E. (2016). Toward Integrating Cognitive Linguistics and Cognitive Language Processing. *Proceedings of the 14th International Conference on Cognitive Modeling (ICCM 2016)*. <http://acs.ist.psu.edu/iccm2016/proceedings/lindes2016iccm.pdf>

Lindes, P., & Laird, J. E. (2017a). Ambiguity Resolution in a Cognitive Model of Language Comprehension. *Proceedings of the 15th International Conference on Cognitive Modeling (ICCM 2017)*.

Lindes, P., & Laird, J. E. (2017b). Cognitive Modeling Approaches to Language Comprehension Using Construction Grammar. *AAAI Spring Symposium on Computational Construction Grammar and Natural Language Understanding - Technical Report*, SS-17-02. <https://www.aaai.org/ocs/index.php/SSS/SSS17/paper/view/15285>

Lindes, P., Mininger, A., Kirk, J. R., & Laird, J. E. (2017). Grounding Language for Interactive Task Learning. *Proceedings of the First Workshop on Language Grounding for Robotics*, 1–9. <https://doi.org/10.18653/v1/w17-2801>

Liu, C., Yang, S., Saba-Sadiya, S., Shukla, N., He, Y., Zhu, S.-C., & Chai, J. Y. (2016). Jointly Learning Grounded Task Structures from Language Instruction and Visual Demonstration. *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, 1482–1492.

Mandler, J. M., & Pagán Cánovas, C. (2014). On defining image schemas. *Language and Cognition*, 6(4), 510–532. <https://doi.org/10.1017/langcog.2014.14>

McNeese, N. J., Demir, M., Cooke, N. J., & Myers, C. (2018). Teaming With a Synthetic Teammate: Insights into Human-Autonomy Teaming. *HUMAN FACTORS*, 60(2), 262–273. <https://doi.org/10.1177/0018720817743223>

McNerney, S. (2011). *A Brief Guide to Embodied Cognition: Why You Are Not Your Brain*. Scientific American Blog Network. <http://blogs.scientificamerican.com/guest-blog/2011/11/04/a-brief-guide-to-embodied-cognition-why-you-are-not-your-brain/>

McShane, M., & Nirenburg, S. (2019). *Linguistics for the Age of AI - Draft for review only*.

McShane, M., & Nirenburg, S. (2021). *Linguistics for the Age of AI*. The MIT Press.

Merlo, P. A., & Stevenson, S. (2000). Lexical Syntax and Parsing Architecture. In M. W. Crocker, M. Pickering, & C. Jr. Clifton (Eds.), *Architecture and Mechanisms for Language Processing* (pp. 161–188). Cambridge University Press.

- Michaelis, L. A. (2006). Construction Grammar. In K. Brown (Ed.), *Encyclopedia of Language & Linguistics* (Second Edi, Issue 3, pp. 73–84). Elsevier Science. <https://doi.org/10.1016/b0-08-044854-2/02031-9>
- Mininger, A. (2021). *Expanding Task Diversity in Explanation-Based Interactive Task Learning*. PhD dissertation. University of Michigan. [https://aaronmininger.com/media/amwebsite/docs/Mininger\\_thesis\\_final.pdf](https://aaronmininger.com/media/amwebsite/docs/Mininger_thesis_final.pdf)
- Mininger, A., & Laird, J. E. (2018). Interactively Learning a Blend of Goal-Based and Procedural Tasks. *Thirty-Second AAAI Conference on Artificial Intelligence*. <https://www.aaai.org/ocs/index.php/AAAI/AAAI18/paper/viewPaper/16903>
- Mok, E. H. (2009). *Contextual Bootstrapping for Grammar Learning*. PhD dissertation. University of California at Berkeley. <http://www.eecs.berkeley.edu/Pubs/TechRpts/2009/EECS-2009-12.html>
- Newell, A. (1990). *Unified Theories of Cognition*. Harvard University Press.
- Newell, A., & Simon, H. A. (1961). GPS, A Program that Simulates Human Thought. In H. Billing (Ed.), *Lernende automaten* (pp. 109–124). Oldenbourg.
- Newell, A., & Simon, H. A. (1976). Newell & Simon 1976 - CS as Empirical Inquiry. *Communications of the ACM*, 19(3), 113–126.
- Nguyen, T., Gopalan, N., Patel, R., Corsaro, M., Pavlick, E., & Tellez, S. (2020, July). Robot Object Retrieval with Contextual Natural Language Queries. *Robotics: Science and Systems*.
- Oberauer, K. (2019). Working Memory and Attention – A Conceptual Analysis and Review. *Journal of Cognition*, 2(1), 1–23. <https://doi.org/10.5334/joc.58>
- Parr, T. (2012). *The Definitive ANTLR 4 Reference*. The Pragmatic Bookshelf.
- Pylkkänen, L. (2019). The neural basis of combinatory syntax and semantics. *Science*, 366(6461), 62–66. <https://doi.org/10.1126/science.aax0050>
- Ramaraj, P. (2021). Robots that Help Humans Build Better Mental Models of Robots. *HRI '21 Companion, March 8–11, 2021, Boulder, CO, USA*, 595–597. <https://doi.org/10.1145/3434074.3446365>

- Ramaraj, P., & Laird, J. E. (2018). Establishing Common Ground for Learning Robots. *RSS 2018: Workshop on Models and Representations for Natural Human-Robot Communication*.
- Reitter, D., Keller, F., & Moore, J. D. (2011). A Computational Cognitive Model of Syntactic Priming. *Cognitive Science*, 35(4), 587–637. <https://doi.org/10.1111/j.1551-6709.2010.01165.x>
- Ritter, F. E., Tehranchi, F., & Oury, J. D. (2019). ACT-R: A cognitive architecture for modeling cognition. *Wiley Interdisciplinary Reviews: Cognitive Science*, 10(3), 1–19. <https://doi.org/10.1002/wcs.1488>
- Ruppenhofer, J., Ellsworth, M., Schwarzer-Petruck, M., Johnson, C. R., & Scheffczyk, J. (2006). *FrameNet II: Extended theory and practice*. <https://nbn-resolving.org/urn:nbn:de:bsz:mh39-54153>
- Schank, R. C. (1972). Conceptual Dependency: A Theory of Natural Language Understanding. *Cognitive Psychology*, 3(4), 552–631. [https://doi.org/10.1016/0010-0285\(72\)90022-9](https://doi.org/10.1016/0010-0285(72)90022-9)
- Schank, R. C., & Abelson, R. (1977). *Scripts Plans Goals and Understanding: An Inquiry into Human Knowledge Structures*. Lawrence Erlbaum Associates, Inc.
- Stearns, B. (2021). *A Comprehensive Computational Model of PRIMs Theory for Task-Independent Procedural Learning*.
- Steels, L. (2013). Fluid Construction Grammar. In T. Hoffmann & G. Trousdale (Eds.), *The Oxford Handbook of Construction Grammar* (pp. 153–167). Oxford University Press. <https://doi.org/10.1093/oxfordhb/9780195396683.013.0009>
- Steels, L., & Hild, M. (Eds.). (2012). *Language Grounding in Robots*. Springer.
- Tanenhaus, M. K., Spivey-Knowlton, M. J., Eberhard, K., & Sedivy, J. C. (1995). Integration of Visual and Linguistic Information in Spoken Language Comprehension. *Science*, 268(5217), 1632.
- Tellex, S., Kollar, T., Dickerson, S., Walter, M. R., Banarjee, A. G., Teller, S., & Roy, N. (2011). Understanding Natural Language Commands for Robotic Navigation and Mobile Manipulation. *Proceedings of the National Conference on Artificial Intelligence (AAAI 2011)*.
- Tomasello, M. (2003). *Constructing a Language: A Usage-Based Theory of Language Acquisition*. Harvard University Press.

Trueswell, J. C., Medina, T. N., Hafri, A., & Gleitman, L. R. (2013). Propose but verify: Fast mapping meets cross-situational word learning. *Cognitive Psychology*, 66(1), 126–156.  
<https://doi.org/10.1016/j.cogpsych.2012.10.001>

van Eecke, P., & Steels, L. (2016). The role of pro- and anti-unification in insight grammar learning. *Proceedings of Benelearn 2016*.

van Elk, M., & Bekkering, H. (2018). The Embodiment of Concepts. In *The Oxford Handbook of 4E Cognition* (pp. 640–660). Oxford University Press.  
<https://doi.org/10.1093/oxfordhb/9780198735410.013.34>

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention Is All You Need. *31st Conference on Neural Information Processing Systems, Nips*, 5998–6008.  
<http://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf>

Winograd, T. (1972). Understanding Natural Language. *Cognitive Psychology*, 3, 1–191. <https://doi.org/10.1016/B978-0-86576-089-9.50009-0>

Young, R. M., & Lewis, R. L. (1999). The Soar Cognitive Architecture and Human Working Memory. In A. Miyaki & P. Shah (Eds.), *Models of working memory: Mechanisms of active maintenance and executive control* (pp. 224–256). Cambridge University Press.  
<https://doi.org/https://doi.org/10.1017/CBO9781139174909.010>