SEQSEE: A CONCEPT-CENTERED ARCHITECTURE FOR SEQUENCE PERCEPTION

Abhijit A. Mahabal

Submitted to the faculty of the University Graduate School
in partial fulfillment of the requirements
for the degree of
Doctor of Philosophy
in Computer Science
in the School of Informatics and Computing,
and
the Cognitive Science Program
Indiana University
Bloomington

December 2009

Accepted by the Graduate Faculty, Indiana University, in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

_____
Dr. Douglas R. Hofstadter
(Principal Adviser)

_____
Dr. Michael Gasser

_____
Dr. Robert Goldstone

_____
Dr. David Leake

Bloomington, Indiana
December 16, 2009.

To Ti. Aai and Ti. Dada

ACKNOWLEDGMENTS

For a very long time now, I have been fascinated by how people think. Many people have had a lasting influence on how I think about thinking.

The origin of my interest in how the mind works is partly selfish: when I was sixteen, I desperately and very urgently needed to understand how I solve math problems. In the summer of 1994, I was one of the six students selected to represent India at the International Math Olympiad in Hong Kong. The criteria for selection was a series of tests totaling twenty problems — five problems each from algebra, combinatorics, number theory, and geometry. I solved twelve of the fifteen problems from algebra, combinatorics, and number theory. And none — nada, zero, zilch — from geometry. I had only about a month between my selection and the actual Olympiad in Hong Kong in which to raise my game.

My failure at geometry was not caused by lack of motivation or from being incompetent at math — I was only incompetent at geometry. I spent hundreds of hours trying to get better at geometry and in watching myself solve problems. (This story has a happy ending. Although I repeated in 1995 my unheroic feat of getting selected for the team despite a perfect zero in geometry, I did solve all three geometry problems that I faced at the Olympiad itself.)

In 1996, I read and was inspired by *Fluid Concepts and Creative Analogies*. The computer models described in the book were based on deep and introspective observation of acts of solving problems, of extrapolating sequences, of perceiving analogies, of designing fonts, and so forth. Working with Doug Hofstadter has been very enriching, since, it turned out, the models were also based on observation of acts of transalation, of writing metrical verse, of error-making and slips of tongue, of creating ambigrams, of humor, of mathematical discovery, and much besides. I am grateful to have been around such a vibrant mind.

I fervently hope that one component of Doug's attitude has rubbed off on me — his dogged perseverance in getting to the crux of a matter, in not letting go until he believes that he truly understands an issue.

I also hope that I have learned a thing or two from Doug about clarity in writing. I am thankful for his many detailed comments on my drafts. (The very first comment, on page 1, was "kern this".)

I thank Robert Goldstone, Mike Gasser, and David Leake for supporting me throughout graduate school. Mike Gasser's Grounding Lab meetings and Rob Goldstone's Percepts and Concepts Lab meetings were invaluable, as was Rob's generosity in letting me use his lab to run experiments. David Leake was always supportive and full of encouragement, and I have learned much from him.

I could not have been where I am today without Aai and Dada. Their love, their honesty and integrity, their love for reading, their open-mindedness in not caring if I conformed to the prevalent narrow view of education, their love for the Marathi tongue, and the steady diet of interesting puzzles that they brought me up on have shaped who I am.

Ashish — eight years elder to me — has been a role model since I was in kindergarten. I followed in his footsteps from early on — to atheism; to questioning religious beliefs and wondering about their origin; to numismatics; to astronomy, physics, and math; to birding; and even to Perl.

I could not have finished writing without Shweta's help, patience, and her nagging me to completion. She also endured my many soliloquies about what I was doing in Seqsee and how great Perl is. We started graduate school together, and finished together. We were always at nearly the same stages of our graduate careers, and that was a big help. For both of us, Suhana is an amazing stress buster. It is impossible to simultaneously watch her smile and be stressed out.

Helga deserves a special thank you. Without her vigilance, CRCC would truly stop functioning. For the previous three semesters, I have been away in California, and things would have been much more difficult without her enthusiastic help.

Of my fellow graduate students, two deserve special mention for many intense conversations that I cherish: David Landy and Christopher Honey. Other FARGonauts and Exo-FARGonauts, past and present, helped with many ideas: Harry Foundalis, Francisco Lara-Dammer, Eric Nichols, Damian Sullivan, Matt Rowe, Will York, Michael Roberts, David Moser, Alexandre Linhares, Donald Byrd and Hamid Ekbia. I must single out three ex-FARGonauts for significantly contributing to the present work through their own dissertations: Marsha Meredith — who worked on Seek-Whence for her Ph.D. — and Melanie Mitchell and James Marshall — who wrote, respectively, the Copycat and Metacat programs.

Seqsee is written in Perl, and I have had the good fortune of closely following the development of the next version of the language — Perl 6. This has helped me see the intense amounts of dedication and care that has been put into Perl. I thank Larry Wall, Damian Conway, Audrey Tang, chromatic, Patrick Michaud, and Jonathan Worthington (and countless others) for giving so freely of themselves.

Many authors have shaped my thoughts. A few of these include Douglas Hofstadter, George Pólya, Dan Dennett, Donald Norman, Gilles Fauconnier, John Ellis, Thomas Kuhn, Michael Agar, Ervin Goffman, Larry Wall, Eviatar Zerubavel, William James, Bernard Baars, George Lakoff, and Roger Schank .

I have been influenced by many other people, a few of whom include, in a rough chronological order: Vivek Wagh, Prof. Shukla, Subhash Khot, M. Prakash, S. N. Maheshwari, Amitabha Tripathi, Ashish Raniwala, Rohit Karlupia, Amit Garg, Arjun Prasad Singh, Harshal Pradhan, Siddharth Prakash Jain, people at Landmark Education, Chirag Jain, and Roshan James.

Thank you all very much!

Seqsee: A Concept-centered Architecture for Sequence Perception

Abhijit A. Mahabal

One of the goals of this project is to design and implement a computer program that can extend integer sequences intelligently, and the project has resulted in the creation of the program named "Seqsee" (pronounced "sexy"). Seqsee can extend a wide range of cognitively interesting sequences, including the following sequence (Seqsee is presented the sequence without the groupings indicated by the parentheses):

((1)) ((1) (1 2)) ((1) (1 2) (1 2 3))

If people are shown this sequence (without the parentheses), they quickly form a group consisting of the three initial "1"s, but then realize that each plays a slightly different role in the sequence. Like people, Seqsee is initially distracted by the three consecutive "1"s, but gradually figures out that the second "1" is an ascending group, and that the initial "1" is an ascending group made up of one ascending group.

Architecturally, Seqsee is a descendant of Hofstadter & Mitchell's computer program Copycat, and adds several novel features that allow it to easily modify behavior in response to its recent perceptions, to form specific expectations such as "an ascending group is likely to be located here", to more quickly understand sequences having previously seen similar sequences, to see an entity as something else, and to do all this without the use of brute force.

Seqsee uses several ideas in achieving its goals: William James' notions of the fringe and the stream of thought; analogies between objects; categorization and labeling of objects and of situations, and the detection of categories without using brute-force *tests* for all sorts of categories; the notion of context which influences and is influenced by perception; the notion, similar to affordances, of the "action fringe" of an object; and a category-based long-term memory.

The dissertation describes the program and its principles, which are much more general than integer-sequence extrapolation, and compares its performance with human performance.

Brief Table of Contents

Chapter 1	Introduction	1
Chapter 2	The Seek-Whence Domain	25
Chapter 3	Seqsee's Performance	45
Chapter 4	Codelets, Codelet Trees, and Pressure	85
Chapter 5	Context and Pressure	101
Chapter 6	Categories in Seqsee	133
Chapter 7	Long-term Memory	165
Chapter 8	Conclusions and Next Steps	173
Appendix A	Reading	195
Appendix B	Restrictions on Groups in Seqsee	197
Appendix C	Simon and Kotovsky's Work	201
Appendix D	Human Performance	205

DETAILED TABLE OF CONTENTS

Acknowledgments ix

Brief Table of Contents xv

Detailed Table of Contents xvii

List of Figures xxiii

List of Tables xxvii

Chapter 1 Introduction 1

Section 1.1 The Goals 1

Section 1.2 Brute-force vs. Concept-centered Strategies 3

1.2.1 Superseeker 4

1.2.2 Deep Blue 8

1.2.3 Copycat 11

Section 1.3 The Subgoals 14

1.3.1 Extensibility 15

1.3.2 Generalization 18

1.3.3 Scalability 20

1.3.4 Context-sensitivity 21

1.3.5 Visualization to Gain Insight 22

1.3.6 Subgoals in Tension 22

Section 1.4 The Structure of This Document 23

Chapter 2 The Seek-Whence Domain 25

Section 2.1 What Constitutes a Problem in This Domain? 25

Section 2.2 A Sampler of Sequences 27

2.2.1 Periodic and Quasi-periodic Sequences 27

2.2.2 Ascending Groups 30

2.2.3 Descending Groups and Sameness Groups 31
2.2.4 Combining — or Crossing — Sequences 31
Section 2.3 Pattern–based Sequences 33
Section 2.4 Multiple Ways of Seeing 35
Section 2.5 Squinting 36
Section 2.6 Garden-path Sequences 39
Section 2.7 Blemished Sequences 40
Section 2.8 Seqsee's Deep Connection with Copycat 41
Section 2.9 The Fuzzy Boundary of the Domain 42

Chapter 3 Seqsee's Performance 45
Section 3.2 One Complete Run 49
Section 3.3 Garden-path Sequences 60
3.3.2 A Few More Garden-path Sequences 61
Section 3.4 Quasi-periodic Sequences 66
3.4.2 Comparison with Human Performance 67
3.4.3 The Effect of Template Size 68
3.4.4 The Effect of Revealing More Terms Initially 69
Section 3.5 Squinting, or Seeing As 70
Section 3.6 Remindings 73
Section 3.7 Extending Seqsee: Primes 76
Section 3.8 Further Comparison with Human Performance 77
3.8.1 Different Types of Relationships Between Ascending Groups 79
Section 3.9 Sequences Not Solved by Seqsee 79
3.9.1 Sequences Outside the Domain 80
3.9.2 Sequences Based on Unimplemented Concepts 81
3.9.3 Failure Caused by a Deficient Implementation 82

3.9.4 Sequences Rarely Solved 83
Section 3.10 Parting Thoughts 83

Chapter 4 Codelets, Codelet Trees, and Pressure 85
Section 4.1 The Codelet-level Description 86
Section 4.2 The Coderack 87
Section 4.3 The "Codelet-tree"-level Description 88
Section 4.4 The "Pressure"-level Description 90
4.4.1 "Programming" Seqsee 92
4.4.2 Coderack Demographics 92
4.4.3 Pressure 94
Section 4.5 The Genesis of Pressure 95
4.5.1 Analogy as a Source of Pressure 95
4.5.2 Top-down Expectation as a Source of Pressure 97
4.5.3 Reminding as a Source of Pressure 98
4.5.4 "Reflex" Pressure 98
Section 4.6 Parting Thoughts 98

Chapter 5 Context and Pressure 101
Section 5.1 Serendipity 103
Section 5.2 Copycat's Context-sensitivity 104
5.2.1 Context Influences what is Perceived 104
5.2.2 Context Influences Responses to the Perception 107
5.2.3 Perceptions and Actions Modify Context 108
5.2.4 Other Contexts in Copycat and Metacat 108
Section 5.3 Specificity of Pressures 110
Section 5.4 Perceptual Context 116
Section 5.5 Focusing on an Object 120

5.5.1 One Component of Focusing on an Object 120
5.5.2 The Other Component of Focusing on an Object 121
Section 5.6 Labels and Context....122
Section 5.7 How the Fringe Generates Specific Pressures....126
5.7.1 The First Stop....127
5.7.2 The Second Stop....127
5.7.3 The Third Stop....128
5.7.4 Specificity of Generated Pressures....129
5.7.5 The Fourth Stop....130
Section 5.8 Parting Thoughts....130

Chapter 6 Categories in Seqsee....133
Section 6.1 Categories Missing from Seqsee....135
Section 6.2 A Single Category in Depth....137
6.2.1 Potential Instances....137
6.2.2 Analogies Based on This Category....138
6.2.3 Discovery....143
6.2.4 Gradedness of Category Membership....151
Section 6.3 The Category “Prime Number”....152
Section 6.4 A Family of Generated Categories....153
Section 6.5 A Category with a Faint Odor....157
Section 6.6 Derivative Sequences....159
Section 6.7 The Category “Things Like X”....160

Chapter 7 Long-term Memory....165
Section 7.1 A Different Method for Spreading Activation....165
Section 7.2 A Feature Missing from Seqsee: Forgetting....169
Section 7.3 How New Links are Created....169

Chapter 8 Conclusions and Next Steps 173

Section 8.1 A Few Deficiencies in Seqsee 173

8.1.1 Features in Seqsee Sometimes Interfere With Each Other 174

8.1.2 Granularity of Codelets 174

Section 8.2 A Hard Look at Granularity 175

8.2.1 Benefits of a Finer Granularity 176

8.2.2 Roger Schank on Finer Granularity 177

8.2.3 Challenges of Implementing at a Finer Granularity 179

8.2.4 A Codelet for Multiplication? 181

Section 8.3 Micro-Seqsee 182

8.3.1 Relevant Knowledge 183

8.3.2 Mental Spaces 183

Section 8.4 Mental Spaces in Extrapolating Sequences 188

Section 8.5 Contributions of This Research 193

Appendix A Reading 195

Appendix B Restrictions on Groups in Seqsee 197

Appendix C Simon and Kotovsky's Work 201

Appendix D Human Performance 205

Bibliography 211

LIST OF FIGURES

Figure 1.1 Intermediate and final stages in understanding “iijjkk” 12

Figure 1.2 Activation of three concepts in Copycat 12

Figure 3.1 A few simple sequences that Seqsee solves 45

Figure 3.2 The sequences from Figure 3.1, now shown with ovals 46

Figure 3.3 A percentile graph 46

Figure 3.4 Seqsee’s performance on sequences from Figure 3.1 47

Figure 3.5 Sequence *c* with distractors 49

Figure 3.6 Initial stage: after 0 codelets 50

Figure 3.7 Early stage: some groups seen 51

Figure 3.8 A first question — although hasty! — is asked 52

Figure 3.9 Group of groups formed 54

Figure 3.10 The correct continuation is suggested! 56

Figure 3.11 And the solution is explained. 57

Figure 3.12 Starting out on the wrong foot 59

Figure 3.13 Two structurally similar but cognitively dissimilar sequences 60

Figure 3.14 Comparison of Seqsee (blue) with human (red) performance 61

Figure 3.15 A few more garden-path sequences 62

Figure 3.16 Sequences from Figure 3.15, with ovals 63

Figure 3.17 Seqsee’s performance on sequences from Figure 3.15 63

Figure 3.18 Structure of Sequence *b* 64

Figure 3.19 Part one of Seqsee floundering on Sequence *a* 64

Figure 3.20 Part two of Seqsee floundering on Sequence *a* 64

Figure 3.21 Two quasi-periodic sequences that Seqsee solves 66

Figure 3.22 Sequences from Figure 3.21, with ovals 66

Figure 3.23 Performance on sequences from Figure 3.21 67

Figure 3.24 Three more quasi-periodic sequences 67

Figure 3.25 Sequences from Figure 3.24, now with ovals 67

Figure 3.26 Performance on sequences from Figure 3.24 68
Figure 3.27 A portion of Figure 3.26, magnified 68
Figure 3.28 The effect of template size 69
Figure 3.29 The effect of revealing more terms initially 70
Figure 3.30 Three sequences helped or hindered by squinting 70
Figure 3.31 Screenshot: Seqsee squinting 71
Figure 3.32 Understanding Sequence *a* without squinting 71
Figure 3.33 Sequences from Figure 3.30, with ovals 72
Figure 3.34 Performance on sequences from Figure 3.30 72
Figure 3.35 Effect of having seen the same sequence before 74
Figure 3.36 The effect of having previously seen similar sequences 74
Figure 3.37 Sequences from Figure 3.36, with ovals 75
Figure 3.38 Effect of having seen similar sequences 75
Figure 3.39 Sequences based on the prime numbers 76
Figure 3.40 Sequences from Figure 3.39, with ovals 77
Figure 3.41 Performance on sequences from Figure 3.39 77
Figure 3.42 Sequences to show two types of relations between ascending groups 79
Figure 3.43 Performance on sequences in Figure 3.42 79
Figure 4.1 A single tree of codelets 89
Figure 4.2 Probability of choosing “Attempt Extension of Group” 93
Figure 5.1 A section of Copycat’s Slipnet while solving problem 1 106
Figure 5.2 The same section of Copycat’s Slipnet while solving problem 2 106
Figure 5.3 What move should black play? 112
Figure 6.1 Analogy between “5 6 7” and “5 6 7 8” 138
Figure 6.2 Skeleton of the analogy between “5 6 7” and “5 6 7 8” 139
Figure 6.3 Analogy between “4 5 6” and “7 8 9” 140
Figure 6.4 Skeleton of the analogy between “4 5 6” and “7 8 9” 140
Figure 6.5 Analogy between “(4 5 6) 17” and “(7 8 9) 17” 141
Figure 6.6 Analogy between “3 2 1 2 3” and “4 3 2 1 2 3 4” 143

Figure 6.7 Activation function 147
Figure 6.8 Activation of “Interlaced 3” while solving Sequence 107 148
Figure 6.9 Activation of “Interlaced 2” in Sequence 110, in a particular run 150
Figure 6.10 Analogy between “1 2 3 4 5” and “1 2 3 4 5 6 7” 152
Figure 6.11 Analogy between successive sub-ovals in Sequence 115 154
Figure 6.12 Analogy between the two groups shown above 155
Figure 6.13 Analogy between the starts of the two groups shown above 155
Figure 6.14 Analogy between successive groups in Sequence 118 156
Figure 6.15 Analogy between “(2 3 5)” and “(2 3 5 7)” 156
Figure 6.16 Example of grouping from Phaeaco 161
Figure 7.1 Copycat’s Slipnet 166
Figure 7.2 Mapping between “1 2” and “1 2 3” 171
Figure 7.3 Sequences to demonstrate the effects of long-term memory 171
Figure 7.4 Effects of having seen a similar sequence 172
Figure 7.5 Mapping between successive terms in target sequence 172
Figure 8.1 Mapping between successive terms of groups #4 through #8 188
Figure C.1 My program to solve all the sequences in Simon and Kotovsky (1963) . 203
Figure D.1 Initial screen for each sequence 205
Figure D.2 Screen shown after clicking “Click to view sequence” 205
Figure D.3 Entering a term opens up next box 206

LIST OF TABLES

Table 1.1 Some transforms used by Superseeker 5

Table 4.1 Description of a run at the codelet level 87

Table 4.2 The same part of the run, now shown with creation and wait times 88

Table 4.3 The same part of the run, annotated with tree numbers 90

Table 4.4 Types of codelets on the Coderack at one point during a run 93

Table 5.1 Relative worth of various moves in Figure 5.3 113

Table D.1 Summary of subjects' performance 207

Table D.2 Performance, continued 208

Table D.3 Performance, continued 209

Chapter 1 INTRODUCTION

Section 1.1 THE GOALS

A few years ago, I stumbled across the notion of a *wicked problem*: the idea was that some problems are inherently hard to define precisely, have numerous and contradictory requirements, and have no "stopping rule" (that is, no easy way to determine if a solution is good enough). The descriptions and analyses that I read of a few such problems were fascinatingly apt and seemed particularly compelling, since I was then dealing with a very ill-defined problem at the software company I was working for. I bring up this concept here because it enables me to explain what is being attempted in this project, and to describe the constellation of pressures pulling the project in various directions.

At one level of description, the goal of this project is to design and implement a computer program that can extend integer sequences intelligently, and the project has in fact resulted in the creation of the program named "Seqsee" (pronounced like the word "sexy"). Such a description of the project's goal would be accurate, but also incomplete to the point of being misleading. It leaves out the reasons for choosing this problem and for doing this project in the first place. These reasons — to be explained shortly — greatly constrain the solutions that are deemed acceptable. The reasons also dictate which aspects of the problem are central, and which aspects are likely to get the cold shoulder if time runs out (as it eventually must because of deadlines and such). In the absence of these very special and strict constraints and biases, a project whose goal was "a computer program to extend integer sequences" would have developed quite differently. Indeed, Section 1.2 explores one such program (a part of the *Online Encyclopedia of Integer Sequences*), and the goal of that section is in fact to point out what Seqsee should *not* do.

It would be more to the point to say that the goal of this project is to explore human cognition by creating a computer model of activities that require intelligence, always making sure to avoid using shortcuts that are available with computers but are implausible in people. The extrapolation of integer sequences

merely happens to be the domain in which I have been carrying out the exploration. Chapter 2 demonstrates the richness of (a very restricted subset of) the sequence-extrapolation domain and shows how this is an appropriate choice — indeed, an excellent one — for studying cognition. The restrictions and their justification are also provided there.

Seen this way as a computer model of intelligence, Seqsee is one more program in the long procession to have emerged from the Fluid Analogies Research Group (henceforth, FARG), over the past quarter-century. These programs[1] tackle a wide range of activities, including solving letter-string analogy problems (Marshall, 1999; Mitchell, 1990), solving Bongard problems (Foundalis, 2006) and designing gridfonts (McGraw, 1995; Rehling, 2001). The common thread linking them all is the creation of computer programs that ideally can perceive and understand complex situations in a human-like way, and Section 1.2 spells out some of our beliefs about how such a program should work. Over the years, the models built at FARG have progressed in numerous ways. Another goal of this project is to build on this progress, and to improve the shared underlying architecture along a variety of dimensions. There are a number of ways in which the current Seqsee program could be made faster, better, and stronger, and some of these are outlined in Section 1.3. It is this multitude of realizable improvements — not all of which can simultaneously be achieved — that gives the project the feel of a wicked problem.

The term “wicked problem” is attributed to Rittel and Webber (1972), who used it to characterize socially challenging problems. A modern example of such a problem is “solving terrorism”. Not everyone agrees about what terrorism *is*, let alone what *solving it* would mean. Any “solution” would have to address political issues, cultural issues, economic issues (since poverty is an enabler of desperation and thereby of terrorism), education, and perhaps even climate change (e.g., the Darfur tragedy was caused partly by an acute water shortage, in turn caused by changing weather patterns). All these requirements that the

[1] This list (sorted by the year the work was completed in) includes Jumbo (Hofstadter, 1983), Seek-Whence (Meredith, 1986), Numbo (Defays, 1988), Copycat (Mitchell, 1990, 1993), Tabletop (French, 1995), Letter Spirit (In two parts: McGraw, 1995; Rehling, 2001), Metacat (Marshall, 1999) and Phaeaco (Foundalis, 2006).

solution of a wicked problem must satisfy push in different, incompatible directions, and improvements in one area are likely to cause worsening in others (a cynical example of such a complex interdependency would be the fact that reducing the number of weapons available might help decrease terrorism, but stopping the sale of weapons is not "economically viable"). Wicked problems in developing software systems are far tamer by comparison, of course, but can still be confounding. Several writers have described how software development is a wicked problem (e.g., Conklin and Weil, 1997; DeGrace and Stahl, 1990; Fitzpatrick, 2003).

The main benefit of casting the development of Seqsee as a wicked problem is to enable readers to have a better sense of what to expect. At some spots in this document, readers may feel that the implementation was made "barely good enough". I myself get the distinct feeling, for example, that the implementation of long-term memory (Chapter 7) can be improved and fine-tuned. As Seqsee currently stands, many low-lying fruits remain that would considerably enlarge the set of sequences Seqsee can solve. In other words, I am leaving off at a stage where the relatively low effort-to-gain ratio would favor continued development. However, in that ratio, "gain" merely refers to a Seqsee capable of solving more and more sequences. If the chief goal were to build a program that expertly extrapolates sequences, this would be an unfortunate time to stop; the chief goal, however, is to explore human cognition.

The project has reached a point where simple improvements, despite making Seqsee more *capable*, would not make Seqsee more *intelligent*. Based on what I have learned from implementing Seqsee, I have specific ideas about what would be needed to make it smarter, and I describe them in the last chapter. To convert these ideas into a working program, it would be necessary to tear the current Seqsee down and rebuild it in a new way, and that must wait until after my Ph.D.

Section 1.2 BRUTE-FORCE VS. CONCEPT-CENTERED STRATEGIES

To explain the central tenets of the FARG architecture, we begin by examining its exact opposite: a computer program named "Superseeker" (Sloane,

2007a), whose goal is seemingly identical to that of Seqsee but whose approach is actually profoundly antithetical to it.

1.2.1 SUPERSEEKER

Just to be clear, I must state here that the following discussion is not intended as a criticism of Superseeker. Superseeker's goal, although superficially similar, is actually profoundly different from that of Seqsee, and its techniques may well be appropriate for that goal.

Superseeker is a part of the *Encyclopedia of Integer Sequences* (Sloane, 2007b; Sloane and Plouffe, 1995). That encyclopedia helps scientists or mathematicians identify sequences that arise in their work, and I discovered firsthand what a phenomenal resource the encyclopedia is. I was required to solve a counting problem a few months ago. The problem involves a set of $2n$ people: n teachers and n students. The problem: in how many ways can we choose a subset of these $2n$ people that contains an equal number of teachers and students? Note that we also count the trivial subset consisting of zero teachers and as many students. Let us call this number of ways of choosing $f(n)$.

For small n, I was easily able to figure out the values of $f(n)$: when n is 1, 2, 3, or 4, $f(n)$ is 2, 6, 20 or 70, respectively. No pattern was apparent to me, so I searched for these four terms in the encyclopedia. There I found not just the formula (which happens to be $\frac{(2n)!}{n!n!}$) and the next 20 terms, but also dozens of other problems with the same answer (for example, the number of non-decreasing sequences of length n made up of integers from 0 to n). I also found many references to properties of this sequence. This was exactly what I needed at that time.

With over 100,000 sequences, including our $f(n)$, the encyclopedia is an astounding repository. But consider now the situation if I had been working on a slightly different problem. The new problem: count only those subsets with at least one person in them. Let us call the answer to this modified problem $g(n)$. The first four values of $g(n)$ are 1, 5, 19, and 69 — that is, one less than the corresponding value of $f(n)$. Now we are out of luck: the encyclopedia contains

no entry for $g(n)$. This is where Superseeker enters the picture. To use Superseeker, an email must be sent to *superseeker@research.att.com*, containing a line such as "lookup 1 5 19 69". After a huge amount of computation, Superseeker replies to the email with possible interpretations of the terms.

In order to make sense of the input — $g(n)$, in this case — Superseeker applies 115 distinct transforms to it in order to obtain new sequences that it then looks up in the encyclopedia[2] (Sloane, 2007a). A few of these transforms are shown below. For each transform, I describe how the transformed sequence $h(n)$ is obtained from the original sequence, and what that transformed version of $g(n)$ is. I have included esoteric transformations like *Möbius Inverse* to show that Superseeker casts a wide net. Readers may look up *Möbius Inverse* if they wish, but the concept is not crucial to this argument.

Table 1.1 Some transforms used by Superseeker

Number	Name	Definition	Transformed g(n)
T040	Add 1	$h(n) = g(n) + 1$	2, 6, 20, 70
T041	Subtract 1	$h(n) = g(n) - 1$	0, 4, 18, 68
T018	Take differences	$h(n) = g(n+1) - g(n)$	4, 14, 50
T111	Möbius inverse	$h(n) = \sum_{d\|n} g(d)\mu\left(\frac{n}{d}\right)$	1, 4, 18, 64
T082	Subtract factorial	$h(n) = g(n) - n!$	0, 3, 13, 45
T010	Divide by factorial	$h(n) = \frac{g(n)}{n!}$	1, 2.5, 3.166, 2.45

For the sequence under consideration, transform number 40 ("Add 1") produces a sequence that *is* in the encyclopedia, thus enabling Superseeker to extend the original sequence (by subtracting 1 from each subsequent term of "2, 6, 20, 70", which it can easily do).

Superseeker's strategy is nothing like what a person would do. We do not blindly go through a list of things to try, one by one. In computer science lingo, a strategy like Superseeker's is called *brute-force*. In what follows, I will refine the notion of brute force, and will point out that whether or not a strategy is

[2]It also attempts to fit a polynomial to the data and to apply other heavy-duty tools, such as trying to represent various types of generating functions as hypergeometric series. See Sloane (2007a) for the full list.

considered to be brute-force admits all shades of gray. Both Superseeker and people lie near extremes of the brute-force spectrum, but at opposite ends.

The transform "divide by factorial" underscores the brute force used by Superseeker. Let us ask ourselves if a person would try this transform on this particular sequence (i.e., on "1, 5, 19, 69"). The sequence that would be generated on applying the transform does not even consist of integers, and would by definition be missing from the *Encyclopedia of Integer Sequences*. Thus, in hindsight, the transform had no chance whatsoever of success. People have the foresight to avoid the waste of time of applying this particular transform to this particular sequence, and it is worth asking what the nature of that foresight is and how it might be captured in a computer program. This is a central question confronted by the work presented here, and we will encounter it repeatedly.

To be sure, "divide by factorial" is occasionally useful. For instance, it reduces the sequence "1, 4, 18, 96, 600, 4320, 35280"[3] to the far simpler "1, 2, 3, 4, 5, 6, 7". Still, sequences on which this transform's application is beneficial tend to have certain properties, and if these are not satisfied by the given sequence, it is unlikely to yield promising fruit under this transform. All sequences likely to benefit from this transform grow very rapidly, and, with the possible exception of the first term, all of their terms are even. People notice such shortcuts readily, once they gain familiarity with this type of problem. A person would never try this transform without having a decent reason (however vague) to believe *a priori* that it would succeed.

The nonhuman quality of Superseeker's technique is that the steps it takes are unmotivated. Superseeker has no sense that one transform appears to have a better chance of success than another does, and its decision to apply the "divide by factorial" transform is therefore not purposeful (and it is not really even a decision, since, regardless of what the problem is, Superseeker *always* applies this transform).

[3] This is the number of permutations of the positive integers in which n is the largest element that is not fixed.

Yet, from another point of view, the transforms used by Superseeker *are* purposeful: smart people who knew what they were doing carefully chose these transforms. For the sequences that Superseeker is likely to encounter — that is, sequences arising in science or math — this set of transforms is well-suited. Every single transform in Superseeker's arsenal is an excellent choice, at least for some of these sequences. The nonhuman quality is that Superseeker does not carefully choose the weapon to use; it always tries each and every weapon.

"Weapon" and "arsenal" are doleful words, and so I will switch to the more neutral terms "tool" and "toolset". Is it worthwhile for a program to spend time considering which tool(s) to use? In many cases, the effort required to choose wisely (both machine effort to make that choice, and human effort to write the program to enable such a choice) far outweighs the saving achieved by using only a few tools instead of using all available tools one by one. Implementing a brute-force solution is typically much simpler, and as computers have become quicker, the cost of using more tools than necessary has decreased, making brute force a good choice for many situations.

As the number of tools in the toolset becomes larger, however, brute force becomes less appealing. In any case, when one is attempting to create a program that mimics human thinking, it is crucial to avoid brute force. In order to suggest a way for choosing the right tool, consider what set of transforms is appropriate for three Superseeker-like programs that differ only in the sequences that they are required to solve, as described below:

1. The input sequences are those that the real Superseeker is likely to encounter.

2. The input sequences are a subset of the above: only strictly increasing sequences will be given to the program.

3. The input sequences are a subset of the above: only strictly increasing sequences will be given as input, with the added restriction that no term is more than double the preceding term.

The set of transforms that best fit each of these situations is different. Many transforms that make sense for the unrestricted domain will be pointless in the restricted domain. Conversely, some transforms that are sensible in the restricted domain help only a minuscule fraction of the unrestricted domain, and for that reason, these transforms will not make it to the list of the most useful sequences for the larger domain.

Given a sequence, if the program is able to identify that it is of a certain type (say, it notices that the sequence is strictly increasing), tools with a better chance of success can be chosen. Such categorizing (or labeling) is automatic in people, and it is the principal method I am suggesting to get out of the brute-force thicket.

People — unlike the case of Superseeker, which must choose from only a few dozen possible transforms — have a seemingly limitless variety of choices in deciding how to respond to a situation, or even whether or not to respond at all to a situation. A situation may appear to us to be a win-win situation, or to pose an imminent danger, or to be hilarious, or to be infuriating; each such (possibly subconscious) categorization will change our set of possible responses — narrowing it in some ways by ruling out some responses, but also widening it in other ways by revealing otherwise hidden possibilities.

Responding to a win-win situation with a cooperative gesture, or to a dangerous situation by fleeing to safety, can be said to be purposeful. Superseeker's choices of transforms are purposeful in the sense that, for the sequences it is likely to come across, they are good choices. This purposefulness, however, is shallow — it lacks sensitivity to different types of sequences and does not produce a response tailored to the sequence being considered.

1.2.2 DEEP BLUE

At one extreme of the brute-force spectrum are the purely brute-force systems, and at the other extreme, we find its converse (human-force?). So far, we have been talking about a point near the former end, but now, in the

remainder of this section, we will move away from pure brute force and toward the other extreme.

The system we are about to consider involves the world of chess-playing, and follows a hybrid strategy, partly brute-force. If we were to imagine an analogous program in the domain of Superseeker, it would do roughly this: given an input sequence, it would check which of 8000 different labels are appropriate for it. Labels may include *increasing sequence*, *sequence consisting only of primes*, *sequence consisting only of "0"s and "1"s*, and so forth. Depending on which labels were found to be appropriate, only a subset of the possible transforms would need to be tried. The labeling process is purely brute-force, methodically trying every label, but this allows the program to be more purposeful when applying the few transforms consistent with the labels, and avoids having to throw everything but the kitchen sink at the sequence. Being brute-force in one way allows chess programs to be purposeful in other ways.

The hybrid of brute force and purposefulness that I'm referring to is Deep Blue, the first chess program to beat the reigning world champion (Campbell, Hoane, and Hsu, 2002). When Deep Blue decided on its next move, like most other game-playing programs, it used the classic look-ahead technique. That is, it considered the possible next moves, responses to these moves, responses to responses to these moves, and so forth. There could be many possible moves, many responses to each move, and so on. This forms a tree-like structure, called the *game tree*. Each node of this tree is a possible board position that can be reached from the current position.

How far into the tree Deep Blue looked in order to decide its move is called the *depth* of the search. A purely brute-force approach would search all branches equally deeply (i.e., would search all nodes of the tree up to a certain depth.) But this brute-force method would not be strong enough to beat the reigning human champion, *even* for Deep Blue (which reached the speed of evaluating 300 million board positions per second during its match against Kasparov) as it would be able to look only a few steps ahead. What Deep Blue in fact used, instead, is a technique known as selective extension of the game tree.

For each node, a decision is made whether nodes beyond it should be explored. For making this decision, Deep Blue used a sophisticated method devised by Claude Shannon (1950) — the program estimated the quiescence (or stability) of a given board position to determine how beneficial looking further ahead would be, and it deeply explored only those pathways where it was not clear that one player had a decisive advantage.

I wish to point out that because of their use of selective extension, chess programs are not *purely* brute-force: when they explore a trajectory through potential future moves, it is because there is reason to believe that such exploration is helpful or required. If in some position one player or the other has a substantial advantage, further exploration from that position is not required. The program needs to look deeper only in situations that could turn out either way.

How does a chess program estimate which player has, so to speak, the upper hand? As I will soon elaborate, the function that does such estimation — called the evaluation function — makes use of *categories* (or *labels*).

Deep Blue runs on specialized hardware known simply as "the chess chip". This chip looks for 8,000 distinct patterns of arrangements of pieces on the board, ranging from simple (e.g., the number of enemy rooks on the board) to complex (e.g., enemy minor pieces guarding the 7th and 8th ranks of the file that our rook is on). The evaluation function is in fact an 8000-parameter function, and was almost entirely hand-tuned by chess experts (Campbell, et al., 2002). Many of these patterns are obvious to any expert (human) chess player, and have been present in chess literature for centuries, going back at least to the sixteenth century, at which time the first (unofficial) world chess champion Ruy López de Sigura had already studied several chess openings (de Sigura, 1561)[4]. Deep Blue's evaluation function is therefore an approximation of what chess experts have long known to be some of the most relevant concepts.

[4]As an aside, fans of the author Pierre Menard may enjoy his translation of this chess masterpiece into French (see, for instance, Borges, 1962).

While the ability to estimate who has the advantage allows Deep Blue to avoid being a purely brute-force program, as we have just seen, this ability itself is based almost purely on brute-force. The presence and/or strength of each of the 8000 patterns is always checked, in *all* situations. Of course, this contrasts radically with how human experts evaluate a board.

1.2.3 COPYCAT

Let us take one final sample from the brute-force spectrum, moving further away from pure brute force to briefly consider a pair of programs in which the act of labeling is not brute-force. Although we will be talking about two different programs — Melanie Mitchell's Copycat and Jim Marshall's Metacat — for our purposes, we can treat these as one. I bring Metacat into the discussion because I have access to a running version of Metacat — but not of Copycat — and all screenshots will therefore be from Metacat. A full description of these programs can be found in (Marshall, 1999; Mitchell, 1990).

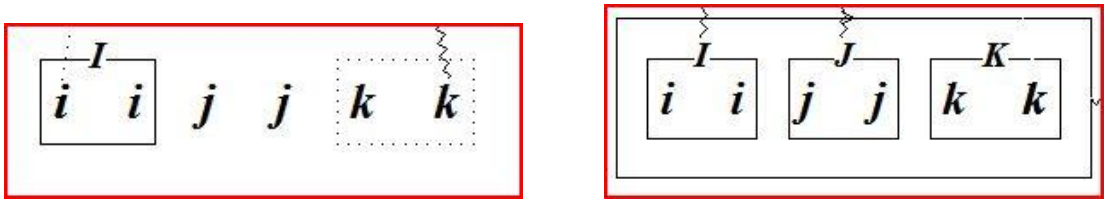
Copycat's task is to solve letter-string analogy problems such as:

Copycat Problem 1.	If **abc** changes to **abd,** what does **iijjkk** change to?
Copycat Problem 2.	If **abc** changes to **cba,** what does **iijjkk** change to?
Copycat Problem 3.	If **abc** changes to **abd,** what does **kji** change to?

Let us focus on how the structure of "**iijjkk**" is understood by Copycat while solving problem 1 from the list above. For "**iijjkk**" to be understood as analogous to "**abc**", the substring **ii** must be seen as a single group[5], or more specifically, as a *sameness group* since both its elements are identical. Furthermore, "**iijjkk**" should be seen as a successor group. Copycat successfully labels all three pairs of letters ("**ii**", "**jj**", and "**kk**") as sameness groups, but this process is not instantaneous. Figure 1.1 is a screenshot that shows an intermediate stage (Subfigure a) and the final stage (Subfigure b) in Copycat's understanding of this string. Notice how in the intermediate stage the two "**i**"s have been seen as a single group. The dotted rectangle around the two "**k**"s

[5] This is not absolutely true. Copycat sometimes produces an answer to Problem 1 that does not require **ii** to be seen as a group — "**iijjkl**". This answer ignores the structures of "**abc**" and "**iijjkk**", merely changing the last letter to its successor. The very shallow analogy there is "both '**abc**' and '**iijjkk**' are strings".

indicates that Copycat is considering grouping them into a chunk but has not done so yet.



(a) (b)

Figure 1.1 Intermediate and final stages in understanding "iijjkk"

At stage (a) above, there is reason to believe that sameness groups are present in and relevant to understanding the string, as one such group has already been seen (namely, the "**ii**") and a second one looks promising (the "**kk**"). This set of "hints" puts pressure on Copycat to explicitly search for sameness groups, and this leads to finally seeing all three sameness groups.

What is not shown in the figure is that Copycat was simultaneously trying to make sense of the "**abc**" in the input, leading it to believe that successorship was relevant in the problem, and therefore it also explicitly searched for successors, in the end seeing the entire "**iijjkk**" as a large successor group (as is represented by the next-to-largest rectangle in Subfigure b).

Copycat keeps track of how relevant various concepts appear in a component called the Slipnet. The screenshot in Figure 1.2 shows how relevant Copycat considers three concepts (*predecessor*, *successor*, and *sameness*) at stage (a). The bigger a circle is, the more relevant that concept is considered to be at that moment, and the likelier Copycat is to take action based on that relevance (e.g., to search for sameness groups).

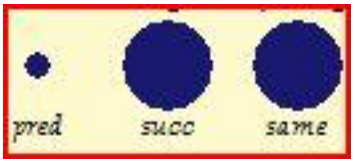


Figure 1.2 Activation of three concepts in Copycat

Copycat's use of concepts is much more explicit than that of Deep Blue. As with Deep Blue, concepts allow Copycat to follow promising paths. Unlike the case for Deep Blue, the number of paths Copycat might follow is not well-defined or even bounded, and the use of concepts is therefore indispensable. Concepts allow Copycat to produce insightful answers, as occasionally happens when it is given the following problem:

Copycat Problem 4. If **abc** changes to **abd,** what does **mrrjjj** change to?

Copycat almost always sees "**abc**" as a successor group, but it sometimes sees "**mrrjjj**" as a successor group as well, with the *length* increasing (as opposed to the *letter category* in the case of "**abc**"), and on such occasions it produces the answer "**mrrjjjj**".

It is our belief that concepts and their activation must be deeply understood if we are to make headway in cognitive science. I quote a paragraph from Fluid Concepts and Creative Analogies (Hofstadter and the Members of the Fluid Analogies Research Group, 1995, p. 466):

> We believe that if AI and cognitive science are to clarify the workings of the human mind, and particularly the human mind as a creative engine, they must pay far more explicit attention to the level of *concepts* and *analogies*, and move away from the magical hope that such phenomena, with their extraordinary richness and complexity, will simply emerge somehow all by themselves, as a result of training networks of artificial neurons. Of course, neural hardware underpins all conceptual phenomena, but then again, so does elementary-particle physics. The real question is: What kinds of intermediate–level structures and mechanisms, located somewhere between quarks and the cortex, do the work that counts?

I will have the opportunity to argue (in Chapter 5) that Seqsee is a tiny bit further away from the brute-force end of the spectrum than Copycat or Metacat are — its actions are more specific, and in that sense, more purposeful. I will show how concepts help Seqsee keep focused, and indeed help it decide what to focus on.

Section 1.3 THE SUBGOALS

Seqsee is an attempt to build on lessons learned and progress made in the development of earlier FARG models. Like Seqsee, each of the earlier programs represents about half a dozen person-years of work, and often more. All of these programs have built on their predecessors, sometimes directly, by sharing the domain and the code (for example, Metacat (Marshall, 1999) extends Copycat (Mitchell, 1990)[6]), but usually less directly, by just borrowing ideas and architectural insights (but not the code or the problem domain).

A careful analysis of how FARG models have progressed over the years reveals a few definite trends. This section describes some of these trends, and thus catalogs some of the aspects in which successive FARG projects have made progress. I also discuss the areas in which I hoped to take Seqsee further than earlier projects. I believe I have made substantial progress in some areas, little or none in others, and have slid back in an area or two. Additionally, I describe one particular improvement that has not yet been realized, but that in some form or other has been wishful thinking routinely expressed on our research group's mailing list: a generic FARG library that would make it easier to create future implementations targeting other domains.

A few remarks are in order regarding the grouping of potential improvements under the similar-sounding headings of *extensibility*, *scalability,* and *generalization*. These are interdependent, and the following split is necessarily somewhat artificial.

Extensibility refers to the ease with which Seqsee's ability to extrapolate sequences can be augmented. It is a convenient fiction to think of Seqsee as consisting of two layers — although the separation between the two is not clean — and to call these the "hardware" layer and the "software" layer, even though both are of course implemented in software. Naming the layers in this way allows me to use the phrase "programming Seqsee" in a meaningful way — the process of programming Seqsee does not include making changes to the

[6]In order to be accurate, I must point out that Marshall translated the Copycat code from Lisp to Scheme before extending it. However, these languages are close enough that saying that he extended "the same code" is only a small fib.

underlying architecture. We can think of the lower layer as a virtual machine, and of the upper layer as the software running on this machine. By the word "extensibility", I mean the idea of augmenting Seqsee's ability by modifying only its software layer.

How comfortably Seqsee can deal with inputs containing hundreds of terms or with situations involving hundreds of categories is the concern called *scalability*. In such situations, Seqsee may be unable to cope for two types of reasons. First, Seqsee may be using up too many resources (for instance, memory and time), and a quicker computer and a greater amount of RAM would solve the problem. Though I have spent a significant amount of time in making sure that Seqsee is not very slow, that is not my main concern here. I worry more about the second source of non-scalability: Seqsee's inability to deal with thousands of categories or thousands of groups may be caused by deeper shortcomings of its architecture, resulting from an incorrect understanding or a faulty implementation of the underlying cognitive processes. If Seqsee had a repertoire consisting of thousands of categories, for instance, it might happen that it would pay attention to hundreds of these instead of being able to zoom in on just a few that seemed relevant to a given problem, and for that reason it would be unable to extrapolate effectively. The same spreading-attention-too-thin effect can potentially occur when thousands of groups are present. Just throwing a hundred–times–faster computer at this deficiency would not make it go away, and therefore, unless it is fixed, this issue would limit Seqsee to having only a small set of categories. The subsection *scalability* considers this issue.

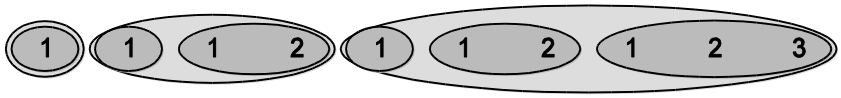
Finally, the subsection on *generalization* looks at how well suited Seqsee's techniques are for other domains. As in the section on extensibility, I'm primarily referring to modifications at the software level.

1.3.1 EXTENSIBILITY

1.3.1.1 EXTENSIBLE ABILITY TO SEE COMPLEX STRUCTURES

The ability of FARG programs to see complex structures has increased over time. Copycat was unable to see how the string "**aabbcc**" could be changed into the string "**aabbccdd**". In fact, Mitchell gave Copycat only the ability to

describe changes between strings of equal lengths where only one letter had changed. Metacat was able to perceive far more general changes, including between the two strings just mentioned. It could also see changes involving swapping letter-categories, as in the change from "**aabaa**" to "**bbabb**". Seqsee takes this further, and it can successfully extrapolate Sequence 1, for example.

Sequence 1. 

A better way of identifying potentially analogous structures can be credited for this increased ability, as I will describe in Chapter 5. This ability is extensible — for example, adding to Seqsee a rudimentary notion of the prime numbers (to identify primes and to be able to say if two numbers are successive primes) allows it to see, with no extra work, a large number of sequences based on primes. Chapter 6 contains a careful discussion of such an addition to Seqsee.

It should be pointed out that sequence-extrapolation challenges are similar to Copycat challenges. Sequence 1, for instance, is like the following set of Copycat problems:

Copycat Problem 5.	If **aab** changes to **aababc,** what does **aababc** change to?
Copycat Problem 6.	What does **a** change to?
Copycat Problem 7.	What does **aababcabcd** change to?

Section 2.8 explores the deep connections — including historical — between the Seqsee and Copycat domains.

Seqsee's ability to see complex structures is extensible in the sense that it is possible to add categories — without having to modify the underlying architecture — that allow Seqsee to see more types of structures than it could before. We will see examples of some such extensions in Chapter 6.

1.3.1.2 Ability to Discover Long-Distance Relations

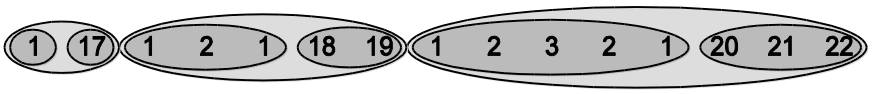
Section 5.7.4 shows how Seqsee's mechanism of discovering relations allows it to see relations between objects far from each other (Copycat and

Metacat are unable to do this, and for that reason they cannot solve problems such as Copycat Problem 8).

Copycat Problem 8. If **abpqabcpqr** changes to **abpqabcpqrabcd**, what does **ewefwx** change to?

This ability is needed if Seqsee is to understand, for example, Sequence 2.

Sequence 2.



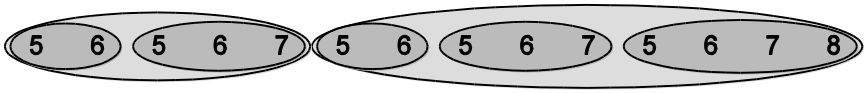
This ability is also extensible in that addition of categories allows Seqsee to draw analogies between objects that it otherwise could not, and hence it can create a greater variety of relations.

1.3.1.3 Remembering Between Runs

Seqsee can remember aspects of solutions and relations that it has seen. This helps it in subsequent runs to see similar sequences more quickly (Chapter 7). Copycat (1990) did not have such a mechanism, but Metacat (1999) had the beginnings of one: although it could not make use of what it remembered to help it solve a problem, it could nonetheless point out at the end of a run that an earlier problem was similar to the one it just solved. Phaeaco (2006) has the most functional permanent memory of all FARG models so far.

There are psychologically interesting issues here. The fact of having been frequently exposed to Sequence 1 should enable Seqsee to understand Sequence 3 more quickly, as it is "exactly the same sequence", in a certain sense of "exactly".

Sequence 3.



1.3.1.4 Ease of Adding New Goals

Apart from *extrapolating* sequences, we may wish Seqsee to *describe* sequences and to *make variations* on a given sequence. Ideally, it should be easy

to add novel goals of this sort. Metacat, for instance, can run in two modes: it can either find an answer to a letter-string analogy problem, or, in case it is told the answer, it can attempt to justify it. Seqsee has unfortunately made no progress in this area.

1.3.2 Generalization

1.3.2.1 Separation of Architecture and Domain

As described in Section 1.1, Seqsee is intended to model key mechanisms of human cognition, not just some curious, highly specialized ability to perceive integer sequences. If the implementation is very tightly tied with and optimized for extending sequences, changing Seqsee to work in another domain would involve an impossible amount of work. It would really amount to writing a new program from scratch. Consequently, I have attempted to design Seqsee so that most of its components do not care about the domain. The Coderack (Chapter 4), the stream of thought (Chapter 5), several visualization tools (Chapters 3 and 6), and (for the most part) long-term memory (Chapter 7) would work essentially unchanged in any domain. Other components, however, such as the Workspace or individual codelets, are tied to the domain, and would need to be redone if one were trying to retarget Seqsee.

A goal of this separation is to create a reusable library that would allow rapid creation of FARG implementations. The current work does not go far enough in this direction — no reusable library has been produced — but at some point in the future, I plan to tear down and rebuild Seqsee to incorporate the lessons I have learned in this project. When I do so, I will attempt to make a yet cleaner separation between the domain-specific aspects and the domain-independent aspects of the program.

1.3.2.2 Ability to Represent Arbitrary Categories

People routinely create categories that constitute big challenges to model faithfully (i.e., without cognitively implausible shortcuts). Consider the category *Einstein*, which, at first blush, seems to contain one individual (or perhaps also includes other people named “Einstein”). But the word “Einstein” is routinely used to creatively describe other individuals, including Charles Hartshorne (who

has been called “the Einstein of religious thought”), Dr. Magnus Hirschfeld (“the Einstein of sex”), and Eric Drexler (“the Einstein of nanotechnology”). We also manufacture categories on the fly (e.g., “things to pack for the Mexico trip”). Lakoff (1987) discusses how even an everyday category such as “mother” is full of nuances and in extreme cases splits into several related categories such as *biological mother*, *stepmother*, *surrogate mother*, *adoptive mother*, *foster mother*, and *donor mother*. Is Seqsee’s way of using categories rich enough to give rise to such diversity and complexity?

Before answering this question, I need to draw a distinction among three types of categories. First, there are categories that the current incarnation of Seqsee already has when the program starts a run (the “built-in” categories). Second, there are categories that the current incarnation can generate on the fly if needed (“auto-generated” categories). Third, there are categories that a programmer can add to Seqsee with a small amount of effort.

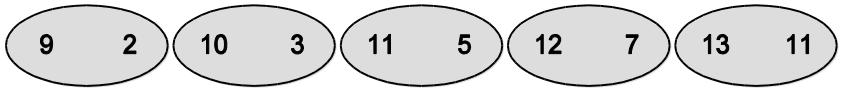
The question “Is Seqsee’s way rich enough?” splits into a couple of questions: “Are the current categories (whether built-in or auto-generated) rich enough?”, and “Can deep categories be added easily?”

It might be argued that this third set of categories is sufficiently vaguely described to give me enough wiggle room to claim fantastic things as being easily realizable with just a little more effort than I have bothered to put in. I will therefore try to be a bit more precise about what adding a category to Seqsee requires of the programmer, and more fundamentally, I will try to discuss the shades of gray in the notion of Seqsee (or a person) “possessing a category”.

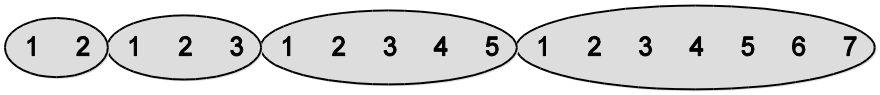
All this we will discuss in Chapter 6. For now, I will provide two examples of built-in categories in Seqsee: one shallow, the other deeper. First, we look at the category “prime number”, which was recently added to Seqsee. Primes are not a part of Seqsee’s domain (which is described in Chapter 2), but they were added nonetheless, in order to test certain aspects of the implementation, and they are turned off by default. Seqsee has a switch that can be flipped to imbue it with a rudimentary “working knowledge” of primes. Even after the switch has

been flipped on, however, Seqsee's understanding of primes is extremely shallow — as opposed to completely nonexistent before. Seqsee has no conception of division, of quotients and remainders, or of factors, let alone of counting a number's number of factors. It does know enough about primes, however, to allow it not just to see the canonical "2 3 5 7 11" but also to successfully extend sequences such as:

Sequence 4.



Sequence 5.



Now that I have described the shallowness of a particular category in Seqsee — and provided an example of how having even a shallow category can help in extending moderately complex sequences, thereby giving an illusion of understanding — I must point out that Seqsee's understanding of such categories as *ascending group* is far from completely hollow. Seqsee has the ability to recognize instances of this concept, to see relations between instances, to create specific subcategories on the fly (e.g., *ascending group starting at 2*), even to "smell" the category at a distance (by noting the presence of successor relations and guessing that successor groups may well exist and be relevant), and to create more complex categories based on this category (an example of which can be seen in the outer ovals of Sequence 1).

One of my main goals was to give Seqsee the ability to add categories easily. Copycat and Metacat had a fixed list of categories, whereas Seqsee creates categories on the fly and is also easily extensible. Phaeaco also has an extensible category system.

1.3.3 SCALABILITY

People have the ability to deal with hundreds of thousands of categories, if not millions. Consequently, being able to deal with a large number of categories was an important goal of Seqsee.

1.3.3.1 Ability to Cope with Large Inputs

Seqsee seems to have little trouble in dealing with sequences even if hundreds of initial terms are given. If a greater number of initial terms are revealed, in fact, Seqsee is a bit quicker in reaching the solution (Sections 3.2 and 3.3 contain examples of this phenomenon).

There are cases, however, when being given a large number of terms confuses Seqsee: such situations include cases where parts of the sequence seem to fit together nicely, but do so misleadingly. We will see examples of such situations also in Chapter 3.

1.3.3.2 Ability to Cope with Large Numbers of Categories

As the number of *active* categories grows (because Seqsee comes to believe that certain categories are relevant to understanding the current sequence, or because optional features in Seqsee are turned on that activate more categories), Seqsee's attention can get spread quite thinly, and its performance can thus deteriorate. Chapter 8 contains specific ideas that hold promise for improving the situation.

I must hasten to add that although in the presence of a large number of categories, Seqsee's performance occasionally deteriorates, the problem is usually caused by categories whose presence Seqsee cannot "smell" easily — that is, Seqsee does not have good intuitions about what situations these categories are relevant in, and so it needlessly pays attention to these. Most categories in the core of Seqsee do not suffer from this problem.

1.3.4 Context-sensitivity

All of Chapter 5 is dedicated to the exploration of context-sensitivity in Seqsee and its predecessors. To avoid repeating myself, here I will present just two concrete examples of how, over the decades, FARG programs have become increasingly sensitive to context.

First, Copycat and Metacat can detect the importance of successor groups in a problem and can adapt their actions accordingly. They do not, however, become sensitive to the presence of "successor groups that start with c", even if

many of these are discovered in the problem. Seqsee is more sensitive to this finer sort of distinction.

Second, many of Metacat's advances over Copycat directly increased the degree of context-sensitivity. For example, the introduction of the architectural component Thematic Spaces allowed Metacat to realize in a particular problem that successor groups are relevant in some of the strings in a given problem but not in all of them. When Copycat thinks of successor groups as important, it seeks such groups in *all* strings. The existence of Thematic Spaces allows Metacat to be more focused — it can search for successor groups in those strings where there is a greater reason to suspect their presence. The additional component of Metacat called the Temporal Trace, likewise, allows Metacat to be more sensitive than Copycat is to its own processing, as it can detect when it gets stuck in a rut, whereas Copycat or Seqsee are never aware of going over the same territory time and time again.

Seqsee is sensitive to a wider range of contexts, I think, than either Copycat or Metacat is, as I will attempt to show in Chapter 5.

1.3.5 Visualization to Gain Insight

If Seqsee is to succeed as a tool for exploring cognition by simulating it, it is not enough to see just the final result generated by Seqsee on a given problem. Much more important is to see how it works in general and what it does on specific runs. Without a good visualization tool, however, understanding such things is hard. An important goal has been to build tools for this purpose. These have had the added benefit of being useful in the development of the program, and they also make it easier to explain the ideas behind Seqsee.

1.3.6 Subgoals in Tension

Designing Seqsee to be easy to extend and designing it to be easy to generalize to other domains are distinct goals that pull in different directions. Improving either one may cause a worsening of the other. The Workspace (the area where perceived groups and relations are kept) is an example. In Seqsee, the Workspace contains groups and relations. For each group, Seqsee knows its

location in the input sequence. This is clearly optimized for sequences. In moving to another domain (say, to the domain of playing Go), the workspace structures would have to be different. Seqsee has several subroutines that make it easy to write codelets solely for the domain of understanding sequences, and these will not generalize to the domain of playing Go.

Generality and visualization are also in tension. Displaying a sequence and chunks within it is easy, as is displaying a Go board. However, creating a generalized display that can be used for both the domain of sequences and for the domain of Go is trickier (indeed, I have no ideas whatsoever on how one might attempt this).

In the same vein, extensibility and scalability are at odds. As more categories and goals are added, the likelihood of their interfering with each other goes up, and Seqsee may become less scalable.

Section 1.4 The Structure of This Document

Chapter 2 describes the Seek-Whence domain and, using dozens of sequences, points out its cognitive richness. Chapter 3 shows how well Seqsee performs on these sequences, and compares its performance with human performance. Chapter 4 begins a description of the internals of Seqsee by giving a bird's-eye view of the architecture, and Chapters 5, 6, and 7 continue the description of Seqsee's architecture showing, respectively, how the key notions of context, categories, and long-term memory have been implemented. Finally, Chapter 8 points out some of the shortcomings of the current implementation with specific remedies suggested for a few of them.

Chapter 2 THE SEEK-WHENCE DOMAIN

The Seqsee project is an attempt to model fundamental and general aspects of human cognition. The aspects being modeled are not limited to any single domain. However, the actual domain that Seqsee works in is of necessity constrained. In this chapter, I describe this domain in detail, and I explain the limitations on the domain and show how these constraints promote certain types of explorations more deeply.

Before diving into the details, I would like to clarify my use of the names *Seek-Whence* and *Seqsee*. The Seek-Whence *domain* was introduced by Douglas Hofstadter around 1977, and he describes it along with the ideas leading up to it in *Metamagical Themas* (Hofstadter, 1985) and in *Fluid Concepts and Creative Analogies* (Hofstadter and the Members of the Fluid Analogies Research Group, 1995). The Seek-Whence *program*, on the other hand, is a program — yet unrealized — that would be able to solve problems in the Seek-Whence domain as well as a human can, and in the same way. It is thus a wishful abstraction not achievable anytime soon. Seqsee can be thought of as a stepping-stone on the long path to the Seek-Whence program.

Section 2.1 WHAT CONSTITUTES A PROBLEM IN THIS DOMAIN?

The notion of a problem in the Seek-Whence domain can be explained with an example. A human being first thinks of an integer sequence. For the sake of concreteness, assume that the infinite sequence thought of is

Sequence 6. **1 1 2 1 2 3 1 2 3 4 1 2 3 4 5**

This sequence is made up of rising blocks of increasing lengths: 1 — 1 2 — 1 2 3. The person then chooses how many initial terms to reveal to another person or to the program. Depending on this choice, the solver will receive as input “1 1 2 1 2 3” or “1 1 2” or any of the other possibilities, including those ending in an incomplete block. Clearly, each of these inputs presents a slightly different cognitive challenge to the solver. For example, the input “1 1 2 1”,

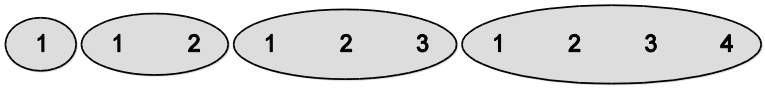
ending in an incomplete final block, is likely to be more confusing than the two inputs shown earlier.

Let us now look at the situation from Seqsee's perspective. For concreteness's sake, assume that it is given "1 1 2 1" as input. Seqsee sees only these initial four terms, and, of course, infinitely many sequences have exactly these initial terms. Many such sequences could be random, but even if not random, infinitely many start out this way. What to do?

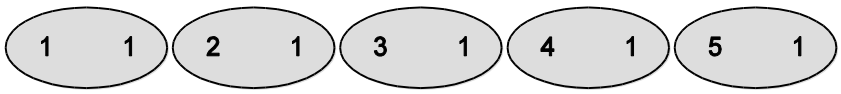
The problem is not as bad as it may seem. Even though there really are infinitely many possibilities, not all of them are equally plausible. An analogy will make this precise. Imagine that on a cold December morning your car refuses to start. There are several *possible* explanations for this, including the explanations that somebody stole the engine, or that this is a prank and your frustration is being taped for the benefit of YouTube. These explanations are highly implausible, though technically possible in a nitpicking sort of way. These, and infinitely many others, do not enter your mind at all — at least not at that early a stage of frustration. Instead, the thoughts that you do entertain are usually simpler and informed by similar episodes from earlier in your life.

In the case of Seqsee, analogously, only a few of the possible continuations are plausible. Still, multiple simple extrapolations remain in contention. I show seven possible extrapolations of "1 1 2 1" below, including the one that the human inventor intended. These are shown in a format that brings out their internal structure. However, the reader must keep in mind that what Seqsee is given as input is nothing but raw numbers, and it must draw its own ovals if needed.

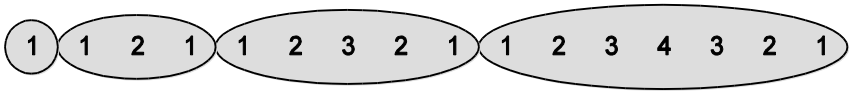
Sequence 7.



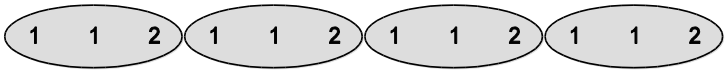
Sequence 8.



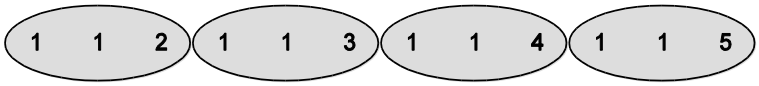
Sequence 9.



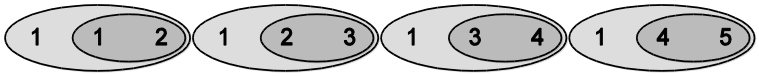
Sequence 10.



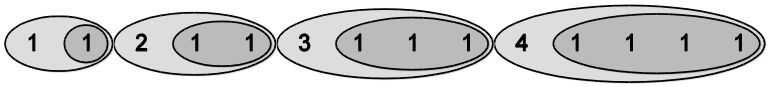
Sequence 11.



Sequence 12.



Sequence 13.



Assuming that Seqsee is able to imagine these possible extrapolations, it needs to try to single out the one that is the correct sequence — namely, the sequence thought of by the human. To pick out the correct answer or to check the answer that it currently believes in, Seqsee may ask the human a question like “Are the next two terms 3 and 1?”. In this case, it will get “no” as the answer since the next two terms are “2 3”. When Seqsee has successfully extrapolated the sequence by a few terms and finally trusts the theory of the sequence that it has come up with, it finishes up by describing the sequence in words.

A “problem-solving” episode with Seqsee is thus really a back-and-forth dialogue between the program and the sequence-inventor as opposed to being a simple “sequence in, extrapolation out” affair.

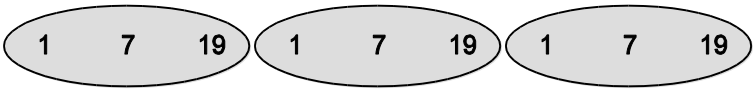
Section 2.2 A Sampler of Sequences

I present below a variety of sequences from the Seek-Whence domain, chosen to showcase the domain’s breadth.

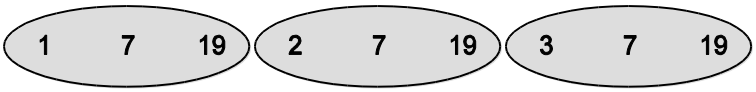
2.2.1 Periodic and Quasi-periodic Sequences

Let us begin with simple periodic sequences. Here are two examples with periods of 1 and 3, respectively:

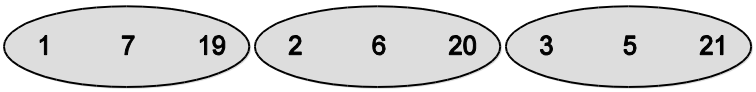
Sequence 14. 

Sequence 15. 

Throughout, I will point out metaphorical knobs that we can twist to increase or decrease the cognitive difficulty of the sequence. A knob available for periodic sequences slightly alters the periodic nature — instead of repeating the same thing, a slight variation is made. Although no longer periodic in a strict mathematical sense, the resulting sequence retains a periodic *feel*, and I shall call such sequences *quasi-periodic*. Sequence 16 is just a spicier, quasi-periodic version of the periodic Sequence 15:

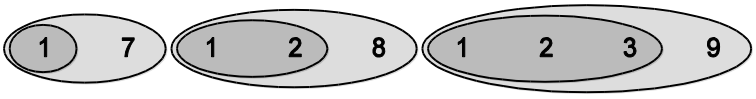
Sequence 16. 

Of course, we can twist this knob further, and make variations in more than one place, even heterogeneous variations:

Sequence 17. 

The variation may also play with lengths, and in that case, the quasi-periodic nature of the resultant sequence is harder to discern in the raw form, but it stands out when shown with ovals:

Sequence 18. 

Sequence 19. 

Although understanding this sequence is simple for people, the mental skills needed for its understanding are not so simple. To understand even this

simple sequence, when it is shown to us without ovals, we need to be able to recognize "(1 2 3)" as a single chunk, to recognize that it is similar to the chunk "(1 2)", to recognize that the solitary "1" at the very beginning is really a degenerate chunk — "(1)" — and also to recognize "((1) 7)" and "((1 2) 8)" as analogous structures.

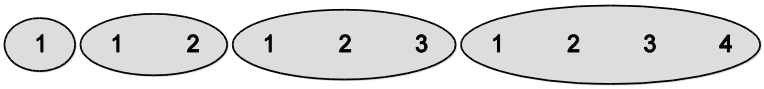
Before we move on to other sequences, I would like to point out that Sequence 17 affords an interesting example of a phenomenon that appears paradoxical at first blush — that restrictions can actually enrich a domain. The next few terms are "(4 4 22) (5 3 23) (6 2 24)". As the middle element in each block of the sequence keeps getting smaller, it will eventually reach "1". How should the sequence continue? Several possibilities come to mind. If negative numbers are admissible[7], the simplest solution is to just continue with "… (7 1 25) (8 0 26) (9 -1 27) …," and this is what Seqsee currently does. However, the original Seek-Whence domain does *not* include negative numbers and this "solution" is therefore sweeping the problem under the rug. Restricted to using only the natural numbers, the possible ways of dealing with the middle element get creative, as shown below. Note that all of these solutions are extravagant and uncalled for if negative numbers are perfectly acceptable within the domain.

- The "sticky" solution: "… (7 1 25) (8 0 26) (9 0 27) (10 0 28) …"
- The "bouncing" solution: "… (7 1 25) (8 0 26) (9 1 27) (10 2 28) …"
- The "pretend non-existence" solution: "… (7 1 25) (8 0 26) (9 27) (10 28) …"
- The "balking" solution: "… (7 1 25) (8 0 25)." (i.e., the sequence stops here).

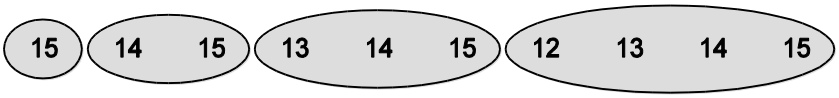
[7] Negative numbers are not a part of the original Seek-Whence domain. The domain was designed in order to have nothing to do with math notions of any sophistication, and to have all numbers have meanings as counting numbers. I initially left the negative numbers in in Seqsee because it was easier to code that way. I always intended to take them out later, but never got around to it.

2.2.2 ASCENDING GROUPS

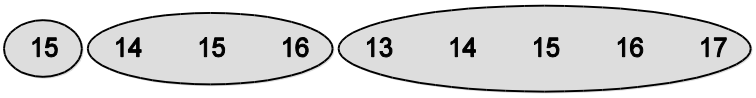
An instance of the category *ascending* is the sequence fragment “3 4 5 6”, and we have already seen sequences Involving this category. One of the simplest sequences based on this category is:

Sequence 20. 

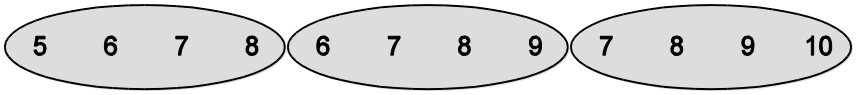
Many knobs come to mind that can increase the difficulty of understanding such a sequence. In Sequence 20, the left end of each group stays at “1”, and only the right end changes. This is the natural end, in a certain sense, because the following sequence seems slightly harder:

Sequence 21. 

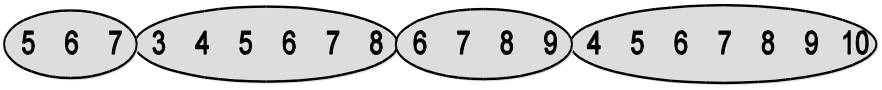
Of course, both ends can move:

Sequence 22. 

Or both can move without changing the length:

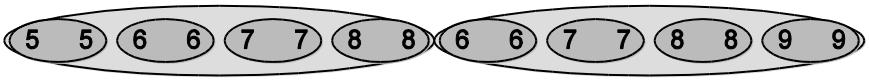
Sequence 23. 

Or length can play an important role, with the start of each group equal to the length of the previous group:

Sequence 24. 

And I have barely scratched the surface, since I have not mentioned changes to the numbers themselves without changing the overall structure of the sequence:

Sequence 25.



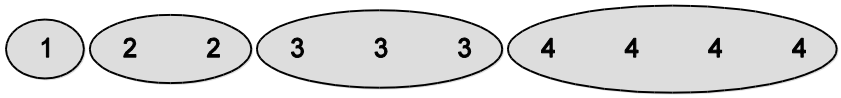
Throughout this document, I have displayed sequences by indicating groups with ovals. The obvious reason for doing so is to prevent slowing the reader down. Even though I have half a dozen years of working in this domain under my belt, and despite my great familiarity with all of these sequences, some sequences I have displayed here still take me a few seconds to understand if they are shown to me without the ovals.

With the ovals present, however, it is a different story. Even if seeing the sequence for the first time, the reader will trivially understand it, with very little or no effort. There is no need for me to spell out what is happening in, say, Sequence 23. This ability to easily and effortlessly grasp an enormous variety of simple patterns is the standard, vanilla human cognitive ability that we wish to capture in Seqsee or Seek-Whence.

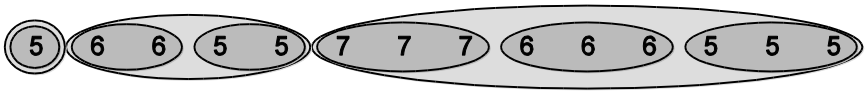
2.2.3 Descending Groups and Sameness Groups

Let us return to the goal of exhibiting more sequences. Analogous to *ascending* are the categories *descending* and *sameness*, and these participate in the following sequences:

Sequence 26.



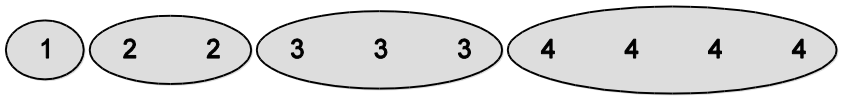
Sequence 27.



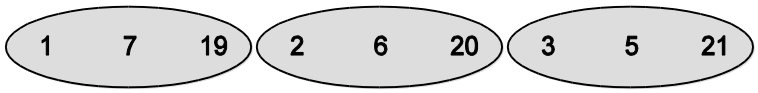
2.2.4 Combining — or Crossing — Sequences

The complexity of sequences can be stepped up by "combining" two or more sequences. I do not have a formal definition of "combination", but by looking at the following sequences, the reader should be able to see what I am doing. Combining Sequence 26 with Sequence 17 (repeated below) produces, for example, Sequence 28, which is more complex than either:

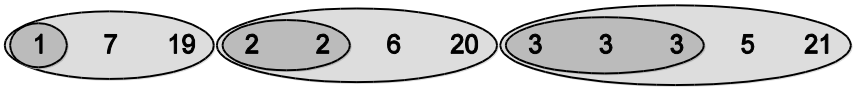
Sequence 26



Sequence 17

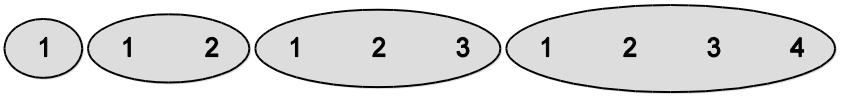


Sequence 28.

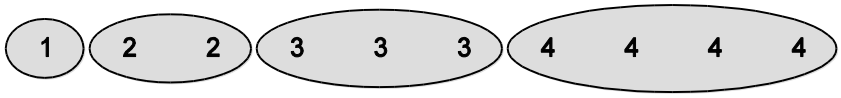


In the same combining spirit, we can also combine Sequence 7 with Sequence 26 (both repeated below), and obtain Sequence 29 or Sequence 30.

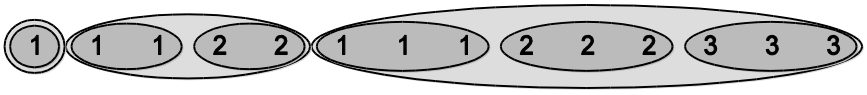
Sequence 7



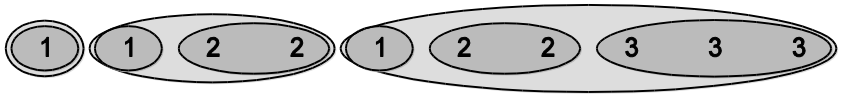
Sequence 26



Sequence 29.

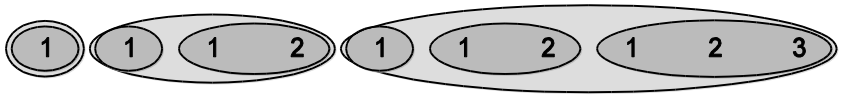


Sequence 30.



We can take our newfound zeal for combination further by combining Sequence 7 with itself to yield:

Sequence 31.



Sequence 31 is the most complex sequence we have created so far. Apart from being interesting as a test case because of its complexity and hierarchical structure, it will also serve as a good example of a couple of new ideas to be introduced later — "squinting" in Section 2.5 and garden-path sequences in Section 2.6.

Section 2.3 PATTERN-BASED SEQUENCES

Canonical examples that many people come up with when thinking about integer sequences include the following:

Sequence 32. 1 3 5 7 9 11

Sequence 33. 2 3 5 7 11 13 17 19

Sequence 34. 1 1 2 3 5 8 13 21 34

Sequence 32 consists of the odd numbers; Sequence 33 consists of the prime numbers; and the sequence after that is made up of the Fibonacci numbers (with each term being the sum of the previous two terms). As members of the mathematical category *integer sequences*, these are quite central. And yet, all of these sequences lie outside the Seek-Whence domain.

Why banish these sequences? To answer this question, let us begin with Sequence 33. Recognizing what this sequence is requires, among other things, the ability to recognize "5" and "7" as primes. If we open the gates to requiring such knowledge as "7 is a prime", logically we must also allow in such factoids as "21 is a triangular number" and even "60 is the size of a simple group"[8]. This last factoid leads to a bizarre sequence (Sequence 35) that consists of the primes all the way through "59", and then, perplexingly, is followed by a "60", and then goes on. This sequence would have been frustratingly hard to fathom, even for some mathematicians, if I had not just let the cat out of the bag:

[8]A *simple group* is a mathematical object that is defined as "a group that has no proper normal subgroup". In other (but vague) words, a simple group has no "factor" apart from "1" and itself, and is therefore like a prime number. In fact, any group whose number of elements is a prime number is necessarily a simple group. The Icosahedral group (the group of symmetries of the regular icosahedron) has 60 elements, and it is the smallest simple group with non-prime size. Obviously, I do not expect Seqsee ever to understand this.

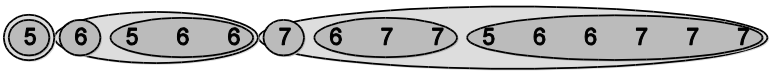
Sequence 35. **2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 60**

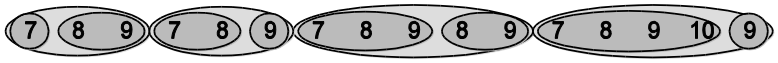
Is such specialized knowledge central to cognition? No, certainly none of the individual pieces of knowledge is needed for intelligent thinking. A vast majority of humanity is happily oblivious of these mathematical facts and yet gets by just fine. An average person can easily handle lots of sequences without any specialized technical knowledge at all.

However, although the specific categories "prime numbers" and "Fibonacci numbers" can be dispensed with, the ability to create and use categories in general is decidedly indispensable. As will be shown, the Seek-Whence domain — though stripped of mathematical knowledge of the sort just mentioned — is rich in categories, both pre-existent and created on the fly.

If even sequences like the prime numbers and odd numbers are outside Seqsee's domain, just what is left? This chapter gives dozens of examples, but the basic criterion is easily summarized: sequences created using the concepts *numerical successor*, *numerical predecessor*, *equality*, and *length* are within the domain. This description intentionally leaves out addition and subtraction, let alone multiplication, division, and factors. This is so because even keeping addition in the picture brings in sequences like the Fibonacci numbers (where each term is the sum of the preceding two), the odd numbers (where each term is the sum of the preceding term and the number 2), the squares (where the n^{th} term is the sum of the first n odd numbers) and so on.

Despite the extremely stripped-down, austere nature of the Seek-Whence domain, there are plenty of complicated challenges in it, such as Sequence 36 below, and also challenges that people initially find perplexing, such as Sequence 37.

Sequence 36. 

Sequence 37. 

As was stated in the previous chapter, the primary goal of this project is to explore cognition, and Seek-Whence merely happens to be the domain chosen in which to conduct such an exploration. It is therefore not the primary goal of this project to create an expert sequence-extrapolator, though obviously we would like to make Seqsee good at its task. Keeping addition "mathematizes" the domain, and since it is not the goal of the project to develop a model of expert knowledge in a technical domain, addition (and other mathematical concepts) was kept out deliberately. (See further discussion in (Hofstadter and the Members of the Fluid Analogies Research Group, 1995)). It makes sense to limit the complexity of the domain so long as we do not oversimplify. A dazzling array of cognitively rich sequences remains. Various sections in this chapter present samplers of sequences that amply demonstrate that the simplification achieved by barring addition and other mathematical concepts does not throw out the baby with the bathwater.

Section 2.4 MULTIPLE WAYS OF SEEING

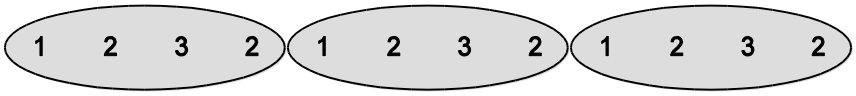
There is nothing in the world that has a unique way of being perceived: someone's trash is another's art, someone's terrorist is another's liberation fighter, and someone's honest criticism is another's heinous blasphemy. More mundanely, George W. Bush can be seen as "the previous president" or as "the 43rd president" or perhaps even as "George III" (having been preceded as presidents by George Washington and George Bush Sr.).

Two ways of understanding something might be mathematically identical and yet vastly differ psychologically, as Richard Feynman points out (1965, p. 53). Having described inverse-square laws of force in three different — but mathematically equivalent — ways, he goes on to say, "Psychologically [these theories] are different because they are completely unequivalent when you are trying to guess new laws." While Seqsee needs to guess no new laws at the cutting edge of particle physics, it must still be capable of seeing a sequence in

multiple ways, just as people do. The following sequence can be seen as oscillating between a low point of "1" and the peak of "3":

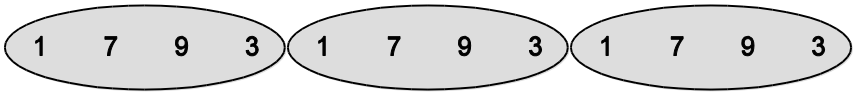
Sequence 38. 1 2 3 2 1 2 3 2 1 2 3 2 1

It could also be seen in a radically different way that nonetheless predicts exactly the same continuation:

Sequence 39. 

Which of these diverse ways the sequence is perceived in influences how similar that sequence appears to another. For example, Sequence 38, more than Sequence 39, appears similar to Sequence 40. Conversely, Sequence 39, but not Sequence 38, appears similar to Sequence 41.

Sequence 40. 5 6 7 8 7 6 5 6 7 8 7 6 5

Sequence 41. 

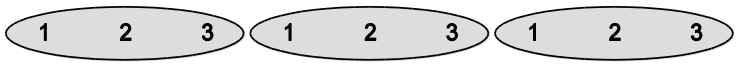
Section 2.5 SQUINTING

Consider the following sequence:

Sequence 42. 1 2 3 1 2 2 3 1 2 2 2 3

This can be understood in a mechanical and lifeless fashion as an interlacing of the three sequences "1 1 1 …", "(2) (2 2) (2 2 2) …", and "3 3 3 …".

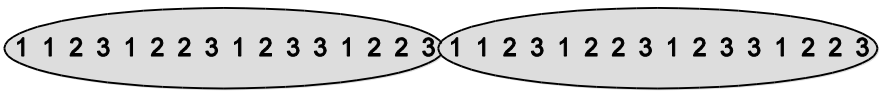
However, the sequence has a cleaner description that is readily apparent to people: the sequence is merely an embellished version of the sequence shown next. If we "squint" as we look at Sequence 42, we can see it as Sequence 43.

Sequence 43. 

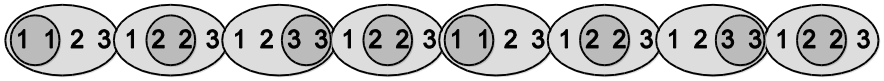
Though the embellishments are not the same in successive ovals, their variation is regular and predictable, thus allowing us to extend the sequence.

People very readily see something *as* something else. This phenomenon is so commonplace that it is hard to know what examples to start with. People routinely understand others despite speech errors. For instance, the sounds "a three-plage letter" is heard as "a three-page letter" with no noticeable effort — in fact, most people will not even hear the extra "l" at all. In mathematics, integration is seen as being just an extreme case of summation, linear transformations are seen as matrix multiplications, and so forth. The sociologist Ervin Goffman (1986) explores the question of how we understand what is going on around us, and a core component of his theory involves what he terms "keying", which is reinterpreting some activity as being something else. The book is filled with examples of such reinterpretations. For example, when we watch the play "Julius Caesar", what we are literally seeing is some actors on stage, in the United States. However, we can (and unconsciously do) pretend that we are witnessing a scene in Europe with a very different set of people. We may personally know some of the artists in the play, but from the time the curtain rises to the time it falls, we are transported into a different world. Hofstadter (1995) provides plenty of other examples of "seeing as". As can be seen from Sequence 42, the Seek-Whence domain is rich in examples requiring this ability to squint.

Other sequences that benefit from squinting include

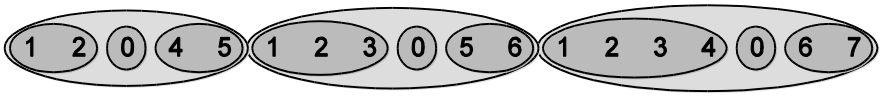
Sequence 44. 

which can be seen in a lifeless fashion as 16 terms repeating endlessly, or it can be seen as a jazzed-up version of Sequence 43, with some term in each group being doubled and the position of this doubled item bouncing around in its size-3 cage, as is shown in Sequence 45 below.

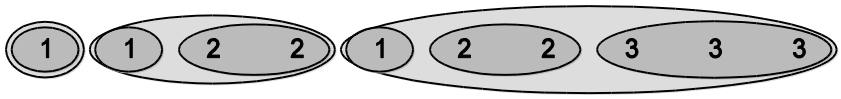
Sequence 45. 

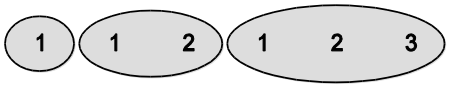
Similarly, Sequence 46 can be seen as made up of three interlaced sequences, as shown in Sequence 47, but there is a much simpler way of seeing it: each group is just an ascending group, but the third term from the end of each group is hidden behind a 0.

Sequence 46. 1 2 0 4 5 1 2 3 0 5 6 1 2 3 4 0 6 7

Sequence 47. 

In this same spirit, the following three sequences are related:

Sequence 48. 

Sequence 49. 

Sequence 50. 1 2 3 4 5

The "direct" way of seeing Sequence 48 is as shown by the ovals, as a hierarchical structure. However, it can also be seen as an embellished version of Sequence 49, with each simple number being replaced by the more ornate ascending group. By the same token, Sequence 49 is a resplendent version of the unassuming sequence "1 2 3 4". In that sense, Sequence 48 is just an embellishment of "1 2 3 4".

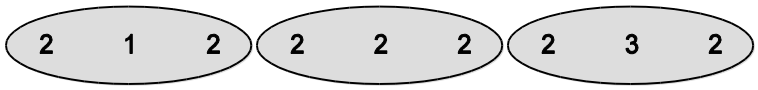
Section 2.6 GARDEN-PATH SEQUENCES

A "garden-path sentence" is a sentence like "The old man the boat", whose initial words suggest (in fact, almost shove down the throat) an incorrect interpretation. Subsequent words do not fit this initial interpretation and force the reader to reassess the meaning of the earlier part of the sentence. Garden-path sequences, analogously, contain terms that cry out to be seen in some (misleading) way. Two such sequences are shown next.

Sequence 51. 1 1 1 2 1 1 2 1 2 3

Sequence 52. 2 1 2 2 2 2 2 3 2

Sequence 51 is likely hard for the reader to guess, despite the fact that it has already been seen and discussed in this chapter. Sequence 52 is more clearly seen as

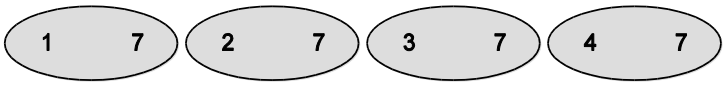
Sequence 53. 

In both cases, a block of identical terms seemed to cry out to be grouped: the block of three "1"s in the first and the block of five "2"s in the second. The highly tempting grouping in each case leads one astray, making the solution harder to find, but people nonetheless tend very reliably to form these groups quickly. Even if, in searching for another way of seeing the sequence, they then tentatively break these groups apart, they find it hard to not form the same groups again.

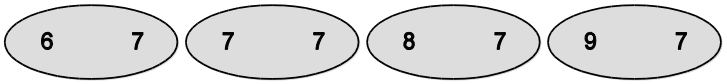
In experiments that I conducted with human subjects in Robert Goldstone's laboratory (these are described in Appendix D), the following two sequences (without the ovals) were among many that were shown to subjects, and the time required for the subjects to understand each sequence was

observed. Sequence 55 took over three times as long as Sequence 54 did[9], for obvious reasons. Human vision makes it virtually impossible to *not* notice identical things when they are neighbors.

Sequence 54.

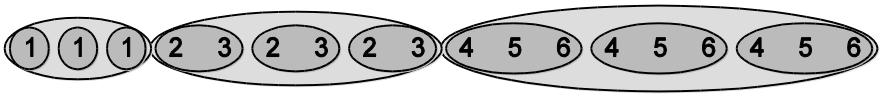


Sequence 55.



A final example of garden-path sequences involves "1 2 3" and "2 3 4 5 6" as misleading groups, as becomes obvious when you mentally erase the ovals.

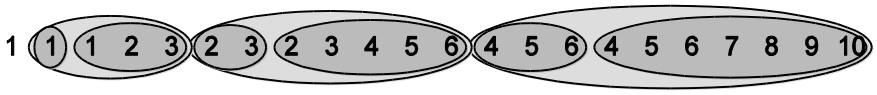
Sequence 56.



Section 2.7 BLEMISHED SEQUENCES

Sequence 56 admits of another interpretation, in which "1 2 3" and "2 3 4 5 6" do not need to be split apart, although this way of seeing it results in an initial blemish:

Sequence 57.



Sequences with initial blemishes are also part of this domain. Seqsee should be able to look at a sequence like

Sequence 58. 7 1 2 3 4 5 6 7 8 9

and see it as "ascending, except with the initial misfit of '7'".

[9] 21 out of 25 subjects solved Sequence 54, taking 5.13 seconds on average when successful. 13 out of 23 subjects solved Sequence 55, taking 16.87 seconds on average when successful.

Section 2.8 SEQSEE'S DEEP CONNECTION WITH COPYCAT

Seqsee would not have been possible had the very rich Copycat and Metacat projects not already explored many of the ideas used here and demonstrated that these can actually be made to work. Although many ideas implemented in Copycat and Metacat were already in place in the early 1980's, a working program increases the amount of trust one places in those ideas and makes ambitious projects conceivable.

In this section, I point out the deep connections between these programs and Seqsee. Work on Seqsee began after Copycat and Metacat had been completed, and it builds on these programs' architecture. Curiously, though, these programs themselves were born out of an earlier FARG attempt at a sequence-extrapolating program.

In the first half on the 1980's, Hofstadter designed the Seek-Whence domain and the original ideas for the architecture (Hofstadter, Meredith, and Clossman, 1982). Marsha Meredith (1986) worked on a program called Seek-Whence. The task turned out to be quite difficult, and one problem that Hofstadter repeatedly ran into was the ubiquity of analogies in integer sequences. Melanie Mitchell (1990, p. 232) describes the role of analogy-making in solving these sequences:

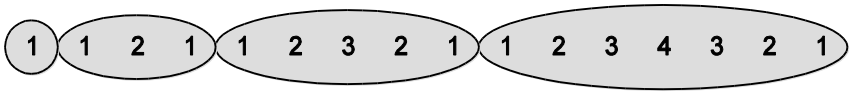
> To find a coherent interpretation for the sequence "1 2 1 1 3 1 1 4 1…", one must map hypothesized segments against each other, perceiving corresponding *roles* within segments (for example, a reasonable parsing is "121-131-141…", with the role played by the "2" in "1 2 1" corresponding to the role played by the "3" in "1 3 1", and so on). What originally gave rise to the Copycat project was Hofstadter's desire to further isolate this essential role of analogy-making in Seek-Whence.

The essential role of analogy-making in Seek-Whence was indeed isolated, and became the Copycat microdomain. Melanie Mitchell (1990) produced the program Copycat, and Jim Marshall (1999) extended her work to produce Metacat. Seqsee comes full circle, and applies some of their ideas to the Seek-Whence domain.

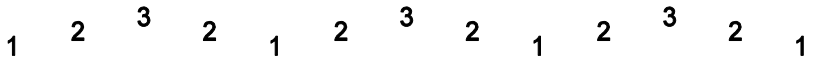
Section 2.9 THE FUZZY BOUNDARY OF THE DOMAIN

I must confess to having two seemingly contradictory views about what to include in the domain. On the one hand, mathematical ideas such as *addition*, *multiplication*, and *primality* have been shied away from. On the other, fuzzier ideas such as *mountain* (Sequence 59), *oscillation* (Sequence 60), *hiding* (Sequence 61) and *doubling* (Sequence 62) have been welcomed. This seeming inconsistency needs an explanation.

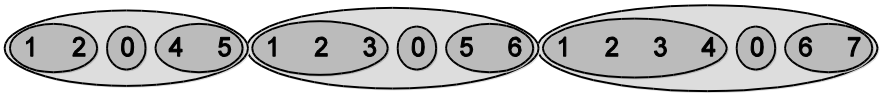
Sequence 59.



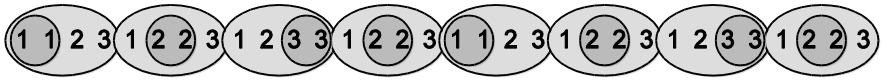
Sequence 60.



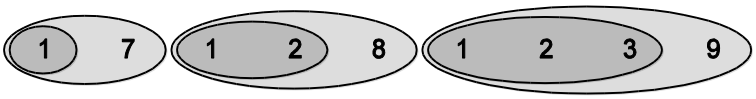
Sequence 61.



Sequence 62.

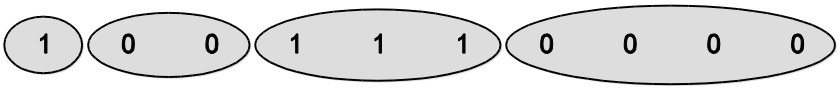


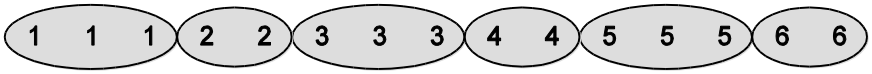
There are strong reasons to prefer, say, *hiding* for inclusion within the Seek-Whence domain over, say, *prime number*. For one, the concept of *hiding* surely arose many millennia before anybody conceived of the notion *number* — let alone the notion *prime number* — and in that sense it is a more important and basic concept. Mathematical terms such as *multiplication* are fairly recent, by contrast, and are certainly not among the most typical and frequent of human concepts. A program such as Superseeker — which can recognize, without even blinking, a sequence such as (to pick something at random) the Möbius transform of the Catalan numbers — appears impressive and superhuman, but it stumbles on Sequence 63, where even a third grader would not. The ability to extrapolate the humble Sequence 63 — with its "anybody can solve that!" feel — is, it seems to me, far more central to human cognition than being able to recognize the Möbius transform of the Catalan numbers.

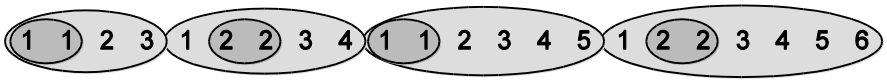
Sequence 63. 

A second (and related) reason is that although the prime numbers are important in the world of integer sequences, they are inconsequential in most others. Hiding is a far more general phenomenon and thus is a central human concept.

Given the discussion above, it is clear that the boundary of the Seek-Whence domain is quite fuzzy. We could throw into it, for instance, the notion of *alternation* between two things, exemplified in sequences such as:

Sequence 64. 

Sequence 65. 

Sequence 66. 

Indeed, recognizing the presence of the abstract notion of alternation in its countless guises is many orders of magnitude harder than recognizing primes, as the latter can be achieved in some programming languages in a single line of code.

Having defined the problem domain, we look next at how well Seqsee performs on some of these sequences, and we also compare its performance with human performance.

Chapter 3 SEQSEE'S PERFORMANCE

The primary reason for promoting the chapter on performance from its customary location near the end of the dissertation is motivational. It is an opportunity for the reader to decide if the remainder of this document is worth their time. There is no point in dragging the reader through a detailed description of how Seqsee works if, at the end, the performance is felt to be mediocre. I hope, however, that this chapter proves to be a sufficient hook.

In this chapter, I describe sequences that Seqsee can solve and sequences beyond its reach. I also compare Seqsee's performance on some sequences with human performance. Seqsee has a few optional features that can be turned on or off, resulting in different modes in which Seqsee can be run, and I compare Seqsee's performance in various modes. Lastly, I describe the effects that previously seen sequences have on Seqsee when it is a run with the optional feature "long-term memory" turned on.

Let us begin on a positive note by looking at a few sequences that Seqsee successfully solves. The relative difficulty that Seqsee (or a human) has in understanding different sequences is front and center to the current discussion, and annotating sequences with ovals greatly reduces such differences. Figure 3.1 below therefore shows the raw form in which these are presented to Seqsee. The annotated versions of these sequences are shown on the next page.

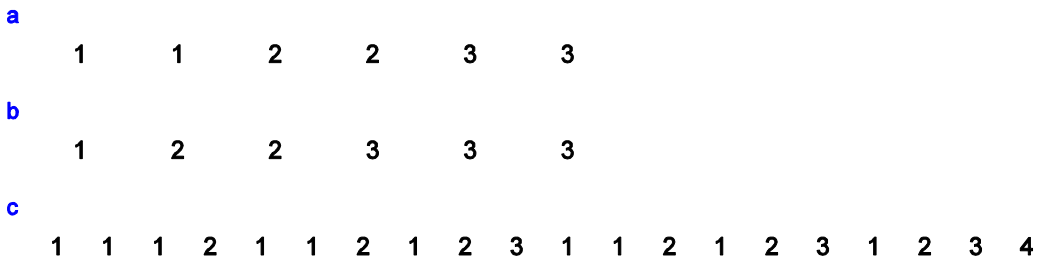


Figure 3.1 A few simple sequences that Seqsee solves

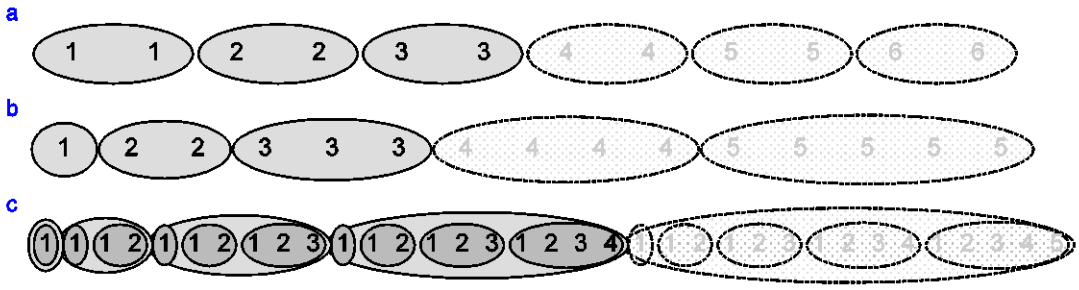


Figure 3.2 The sequences from Figure 3.1, now shown with ovals

The claim that Seqsee "solves" a particular sequence is blurry, and the next section therefore describes one complete session of Seqsee and specifically points out what I mean by "solving". Having made that clear, I will then be able to describe Seqsee's performance in terms of the percentage of runs where Seqsee successfully tackles a particular sequence, and how long it took to succeed. "How long" is measured in "number of codelets run", a notion that will be described in detail in the next chapter. For the time being, the term "time-steps" can safely be substituted for "codelets".

I use percentile charts to display the time taken by Seqsee on a particular sequence. A single percentile chart is shown in Figure 3.3, and described shortly thereafter.

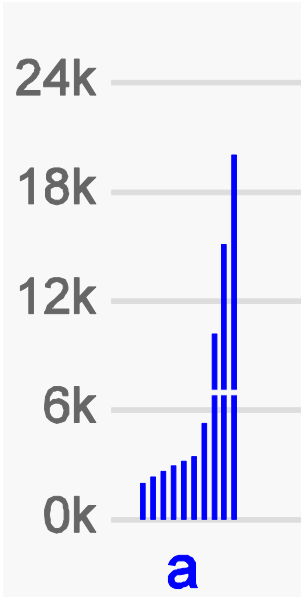


Figure 3.3 A percentile graph

The percentile chart consists of 10 vertical lines and a horizontal cut across them. Each vertical line represents a specific percentile of the time Seqsee took on some problem. The rightmost vertical line represents the 100th percentile — this is the maximum amount of time that Seqsee took when it successfully solved a problem. In this particular chart, that number is around 20,000 codelets. The line next to it represents the 90th percentile — in 90% of cases, Seqsee took this much time or less, which in this chart is about 15,000 codelet. Thus, in the slowest 10% of its runs, Seqsee took between 15,000 and 20,000 codelets. As we work leftward, the third line from the right represents the 80th percentile, and so forth, until the leftmost line, which represents the 10th percentile. It can thus be seen that about half the time, Seqsee solves this problem in 3000 codelets or less, but often takes more. On the average, it takes about 7000 codelets, and this average is represented by the white horizontal cut.

In comparing Seqsee's performance on two sequences, looking only at the rightmost vertical line (that is, the maximum time Seqsee took) or only at the horizontal cut (that is, the average time) is informative, but the shape of the graph tells a richer story.

The figure below shows Seqsee's performance on the three sequences from Figure 3.1.

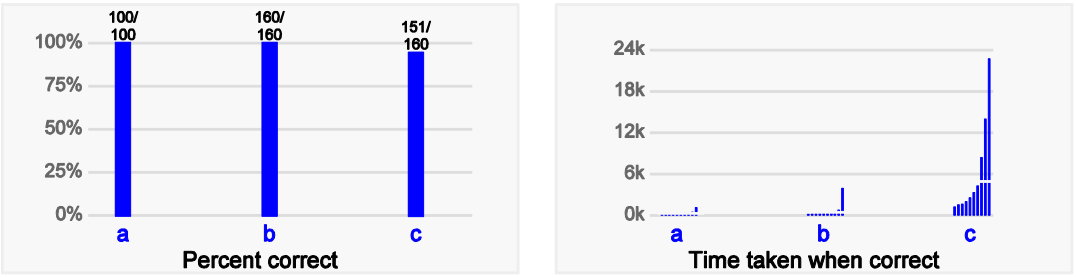


Figure 3.4 Seqsee's performance on sequences from Figure 3.1

In this figure, and in all such figures in this chapter, the labels below the bars correspond to sequences with the same label in the preceding figure that displays sequences.

Labels above the bar in the left chart are of the form “a/b” (for example, “151/160” can be seen above the third bar), and they indicate the number of times that Seqsee was tested on this sequence (160 times) and the number of times that it succeeded (151 times). Seqsee almost always solves these three sequences, dipping to a success rate of 94% only for the third sequence. Failure on a sequence can have either of two causes: either Seqsee crashed (which it occasionally does), or the program had not succeeded within 25000 codelets, at which point the run was terminated.

The right half of Figure 3.4 displays percentile charts showing the number of codelets that Seqsee required on each of the three sequences. It is apparent that Seqsee does not even break a sweat on the first two sequences — their percentile graphs are nearly flat, except for 10% of the worst runs in each case, which took noticeably longer but still finished in under 1000 and 4000 codelets respectively. The third sequence makes Seqsee work somewhat harder, requiring about 4000 time steps on the average. About 10% of the time, however, Seqsee takes between 15,000 and 20,000 time steps (this can be seen by looking at the difference in the heights of the two rightmost vertical lines).

In each of these three sequences, the slowest runs were significantly slower than most runs. The slowest run took several times longer than the average run (represented by the horizontal cut). This pattern of occasionally taking much longer than average will be repeated in many sequences, as the charts in the rest of this chapter will show. Often, the reason for Seqsee taking long in a particular run is an initial misstep — an unfortunate interpretation of some part of the sequence — from which it does not quickly recover. The previous chapter gave examples of garden-path sequences where some part of the sequence cries out for a particular interpretation that happens to be misleading. The vast majority of sequences can be thought of as being garden-path sequences at least to some extent: that is, locally, in small pockets, short pieces of the sequence seem to fit together in a way that is not consistent with the global interpretation. In the third sequence of the figure above, there are many such small local pockets, some of which are indicated by distracting blue ovals below:

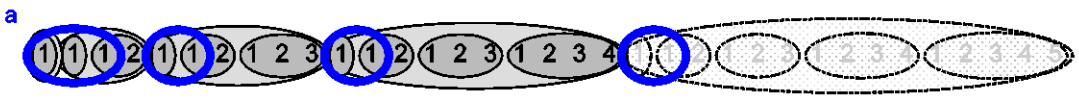


Figure 3.5 Sequence *c* with distractors

Only the more prominent distractors have been marked: less prominent groups — such as a few "2 1"s — have been left out to avoid clutter.

In general, the greater the number of distracting things, the slower it is for Seqsee and for people to understand that sequence. As Seqsee's ability to perceive patterns has grown over the course of this project, so has the range of things by which it is distracted. This is the primary reason for hiding some of Seqsee's abilities behind optional features: these features, when turned on, enable Seqsee to see sequences it could not otherwise see, but hinder its performance on some other sequences. Ideally, such hiding should not be required. Indeed, many of Seqsee's features have their own checks and balances, and are always kept turned on without any issue. Chapter 6, which deals with categories, shows what those checks and balances look like in some cases. Section 3.5, on the other hand, describes one particular overzealous feature — "seeing as", which I also call *squinting* — and discusses some sequences on which Seqsee performs better when that feature is turned off.

Section 3.2 One Complete Run

Let us "watch" Seqsee run. Over the next six pages, I present screenshots of successive stages of Seqsee solving Sequence 67 below.

Sequence 67. 6 1 2 7 1 2 3 8 1 2 3 4

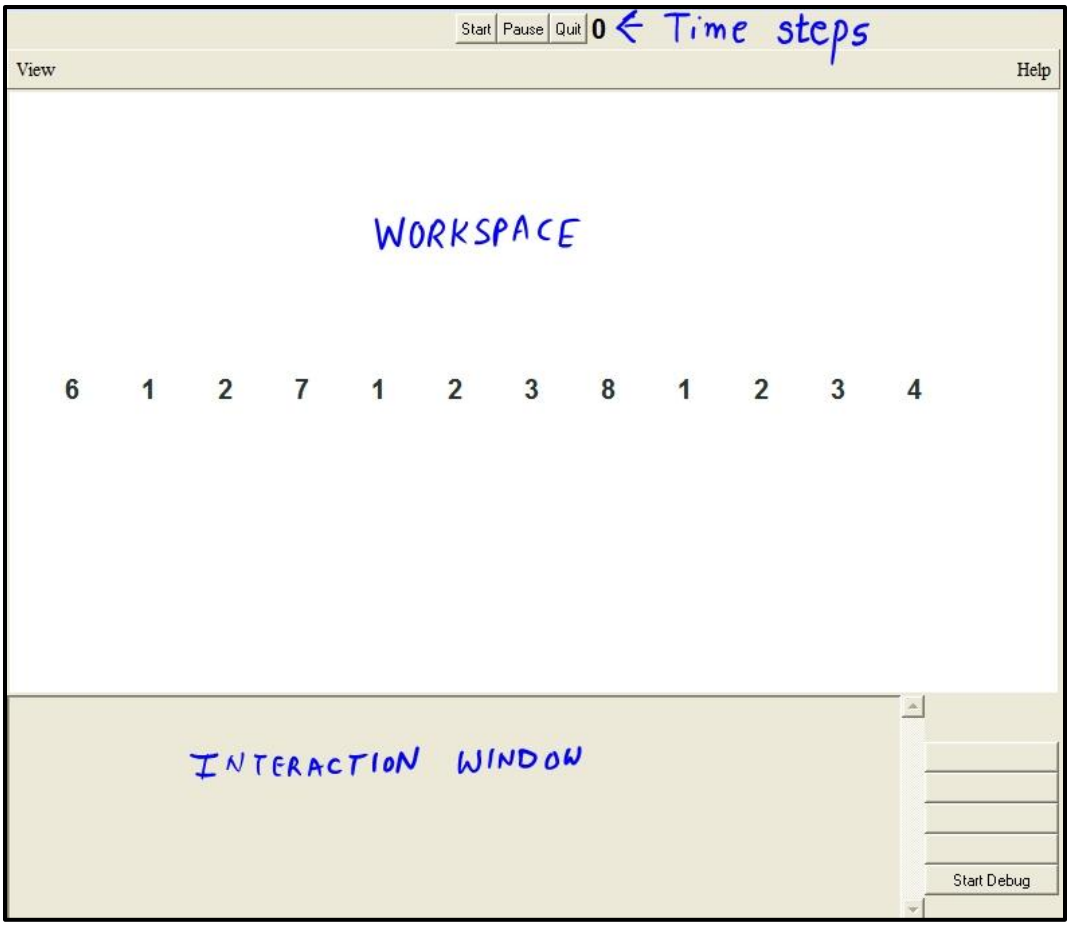


Figure 3.6 Initial stage: after 0 codelets

This screenshot shows the initial state of Seqsee before any processing has happened. The numeral "0" can be seen, near the center at the very top of the image, representing the number of time steps that have elapsed in the run. Each time step corresponds to the execution of a single codelet, as will be described in the next chapter.

The bottom of the image is occupied by the interaction window. This is where Seqsee asks the human who is interacting with it such questions as "Are the next three terms 3, 4, and 5?" Naturally, this window is empty at this stage.

Finally, the central part of the figure represents the Workspace. One might consider this to be a blackboard on which Seqsee makes notes (and this imagery is explained further in Appendix A). At this stage, the Workspace contains just the initial terms that Seqsee was given. This will soon change,

however, and the Workspace will fill up as pieces of the sequence and relationships among them begin to be seen. This will be the main area to watch in subsequent screenshots.

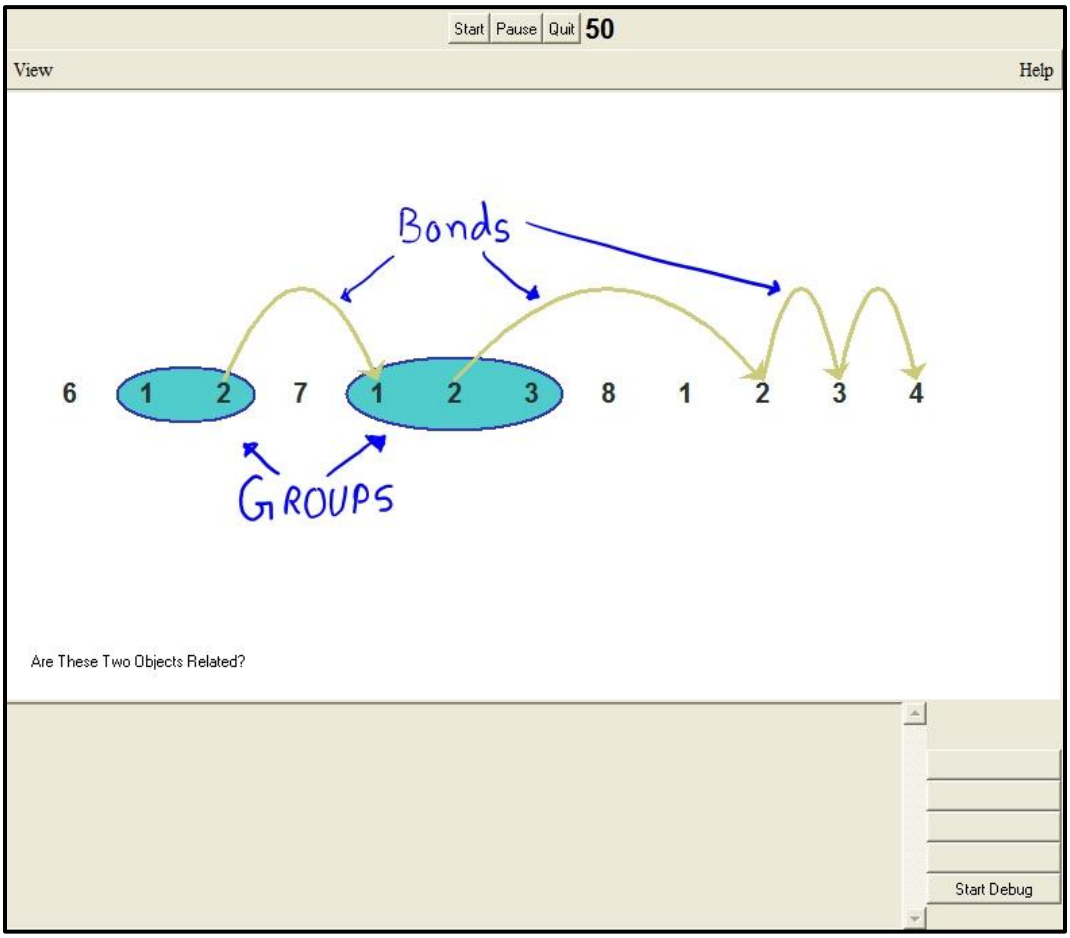


Figure 3.7 Early stage: some groups seen

The figure above is a snapshot fifty steps into the run. The reader must keep in mind that this is only how Seqsee looks after fifty steps *in this particular run,* and because of the thousands of very small, local decisions that Seqsee makes probabilistically, on different runs, it follows different trajectories even if the input is the same.

Seqsee has discovered some structure at this stage — small bits of the sequence have started to make sense. Four yellow arrows are visible. I will follow Copycat terminology and call these arrows “bonds”. Bonds represent any kind of similarity or relatedness that Seqsee has discovered between elements

or between groups of elements. Often, a more appropriate term for these bonds is *analogies* and this term will also be occasionally used. On a black-and-white printout, the arrowheads may be hard to discern, and I should mention that all arrows point to the right.

A couple of blue ovals can be seen, and these denote *groups:* chunks that Seqsee can treat as single units. These chunks are not black boxes (i.e., opaque). Items inside such a box are visible to Seqsee, and it is able to form bonds or other groups among these. However, once a chunk is formed — and until it is destroyed, as might happen sometimes — Seqsee has a probabilistic tendency to ignore its contents. Thus, the chunk is a translucent box — not quite opaque, and not quite transparent. The discussion of Figure 3.9 (on page 54) provides an illustration.

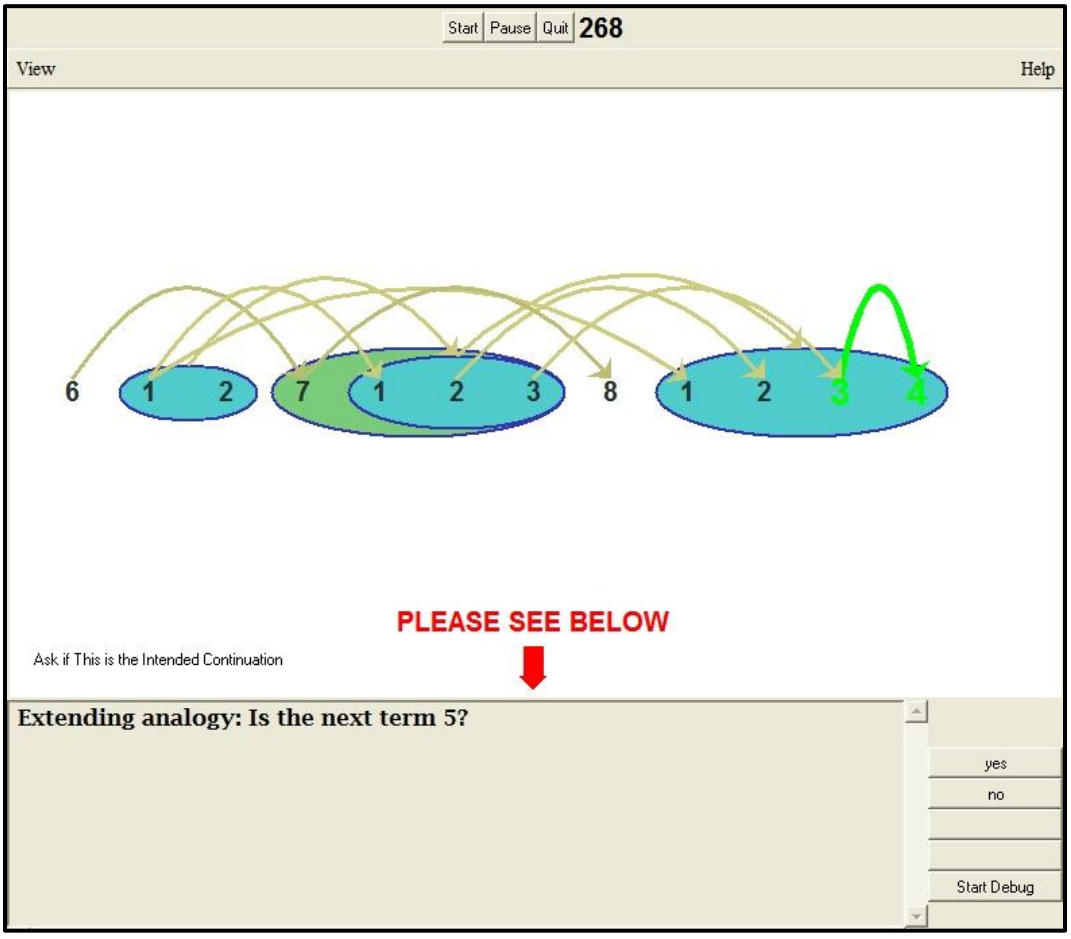


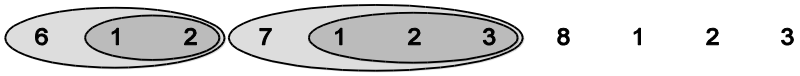
Figure 3.8 A first question — although hasty! — is asked

The interaction window now contains a question ("Is the next term 5?"). The preamble to the question — "Extending analogy" — briefly describes why Seqsee believes that the next term may be a "5".

The question asked by Seqsee may sound myopic — it certainly is hasty — but there is some justification to it. Had Seqsee been working on the almost identical Sequence 68 (below), then the corresponding question "Is the next term 4?" — would have been exactly the right question to ask.

(As a sidenote, notice how easily we humans toss off such phrases as "the corresponding question" (meaning, of course, "the *analogous* question"), presuming, taking totally for granted, that *other* humans will effortlessly and trivially see, understand, and agree with what we're saying. That is, we are *assuming* an analogy-making capacity that is both *universal* and *objective*!)

Since Seqsee does not and should not assume that the initial terms presented to it were neatly cut off at a group boundary, it is willing to ask for subsequent terms based only on the last few known terms, and doing this makes perfect sense, up to a point. It is simply a very *local* point of view, not taking into account any overarching global structure. And indeed, at this point, no global structure has been seen, so there is none to take into account as of yet. For that reason, one can be less harsh in one's judgment of Seqsee's question and simply say that it was a bit overeager and perhaps jumped the gun, although in the light of Sequence 68, perhaps it didn't.

Sequence 68. 

Of course, if Seqsee were less hasty and could hold its horses until more of the sequence made sense, it would make fewer queries that got "no" for an answer. How to make Seqsee less hasty and yet allow it to extend sequences such as Sequence 68 is a deep problem — indeed, a problem that is central to achieving that elusive quality called intelligence — that I have not solved satisfactorily.

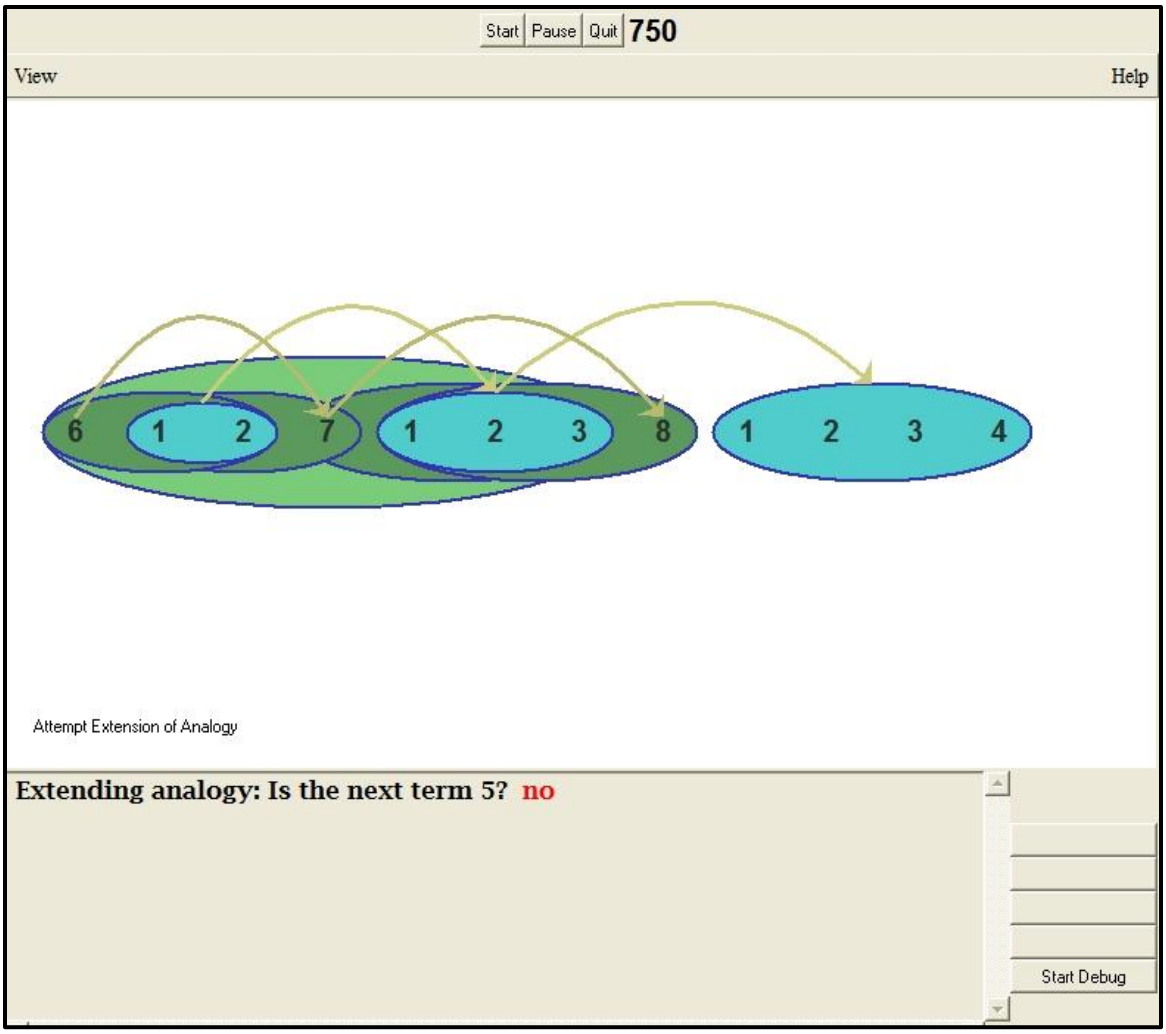


Figure 3.9 Group of groups formed

At the stage shown in Figure 3.9, some higher-level groups have been created. The groups "(1 2)" and "(1 2 3)" seen earlier are now seen as parts of larger and higher-level groups — "(6 (1 2))" and "(7 (1 2 3))", and these groups themselves, put together, form a still higher-level group. "(1 2)" is in fact also a part of another group: "((1 2) 7)", which reveals that Seqsee is unsure about where the group "(1 2)" belongs, and luckily Seqsee can entertain both rival possibilities simultaneously until further evidence is available. It does not believe in each of these two competing groups equally strongly, however.

Now that this large group whose two members are "(6 (1 2))" and "(7 (1 2 3))" has been seen, there is a strong reason, thanks to analogy, to believe

that the next few elements to the right of the "(7 (1 2 3))" will be "(8 (1 2 3 4))", and indeed Seqsee should — and does — actively look for these right there. The five elements "8", "1", "2", "3", and "4" are indeed present at that location, but they have not yet been seen as forming a single hierarchical group "(8 (1 2 3 4))", but the analogy between "(6 (1 2))" and "(7 (1 2 3))" suggests the formation of such an analogous group. Groups exert pressure to look for their analogues on either side. For example, the "(1 2 3)" group in the center exerts some pressure to look for a "4" immediately to its right. However, because it is enclosed in a bigger group, this pressure is much weaker. This is an illustration of the earlier-mentioned fact that groups are not black boxes but are translucent — their parts can exert some (even if little) influence.

Apart from arrows between *elements* — for example, between the "6" and the "7" — there are arrows between *groups* — for example, between the "(1 2 3)" group in the center and the "(1 2 3 4)" group at the end. Using the term *analogy* to describe this arrow connecting two blatantly analogous structures is more in keeping with the conventional use of the term, but it's important to point out that even the much simpler relation between "6" and "7" is really cut from the same cloth, and is also an analogy — simply a humbler one.

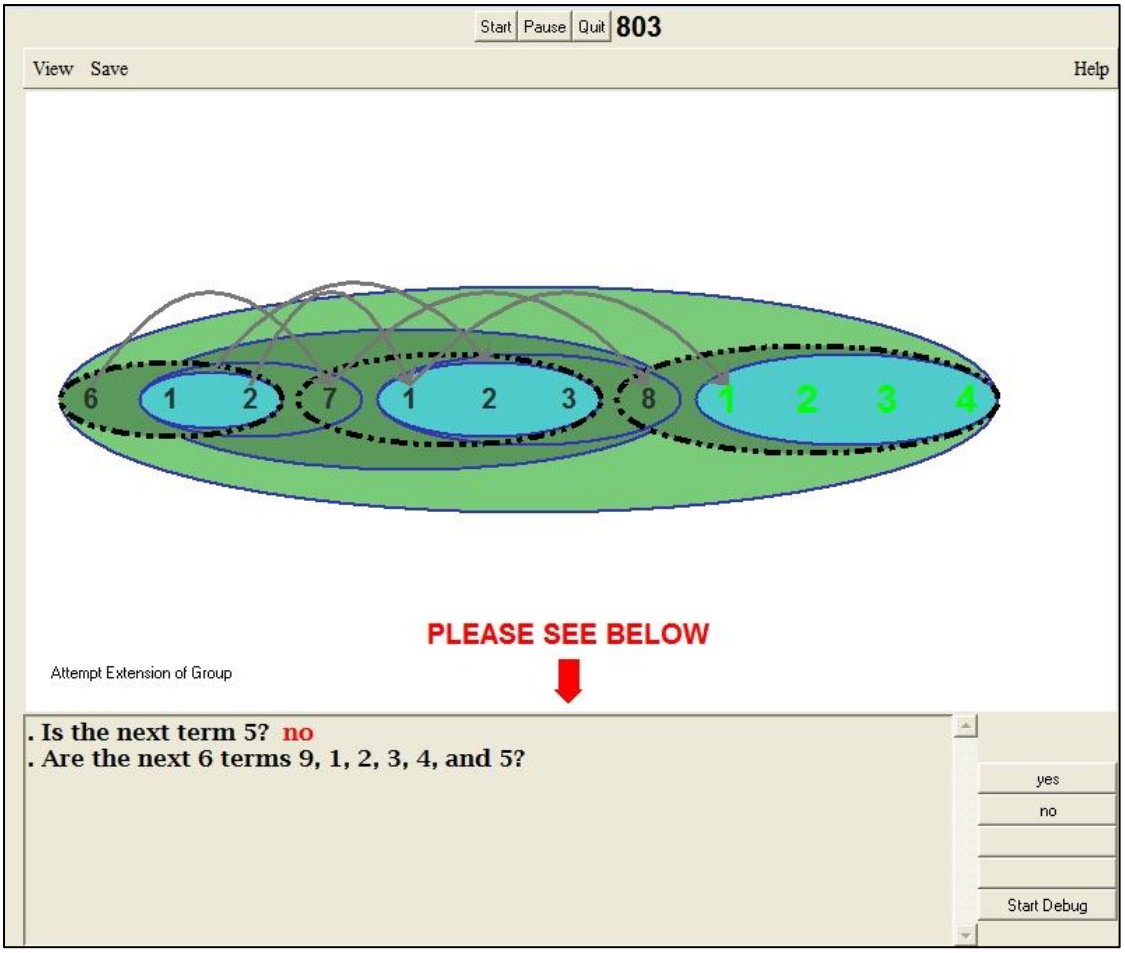


Figure 3.10 The correct continuation is suggested!

At the stage displayed in Figure 3.10, Seqsee has posed another question: "Are the next 6 terms 9, 1, 2, 3, 4, and 5?" When a question is posed, Seqsee uses dotted ovals such as those seen above to indicate evidence in support of the question. Note that Seqsee's question is not restricted to the next single term but instead it predicts an entire large group — made up of 9, 1, 2, 3, 4, and 5 — as a potential continuation to the pattern.

Though Seqsee considers these three groups as sufficient evidence to ask the question, other groups that are inconsistent with this view of the sequence — "((1 2) 7)", for example — still linger in the background. Seqsee will eventually destroy these, but only after it is more convinced that it has put its finger on what the essence of the sequence is. This cleanup will already have happened by the next and final screenshot.

In the picture above, a far larger group encompassing all the known elements at all levels of structure has been seen, and it should be mentioned that this group is hierarchical: it has three members (shown by dotted ovals in the same figure) each of which contains smaller groups as members.

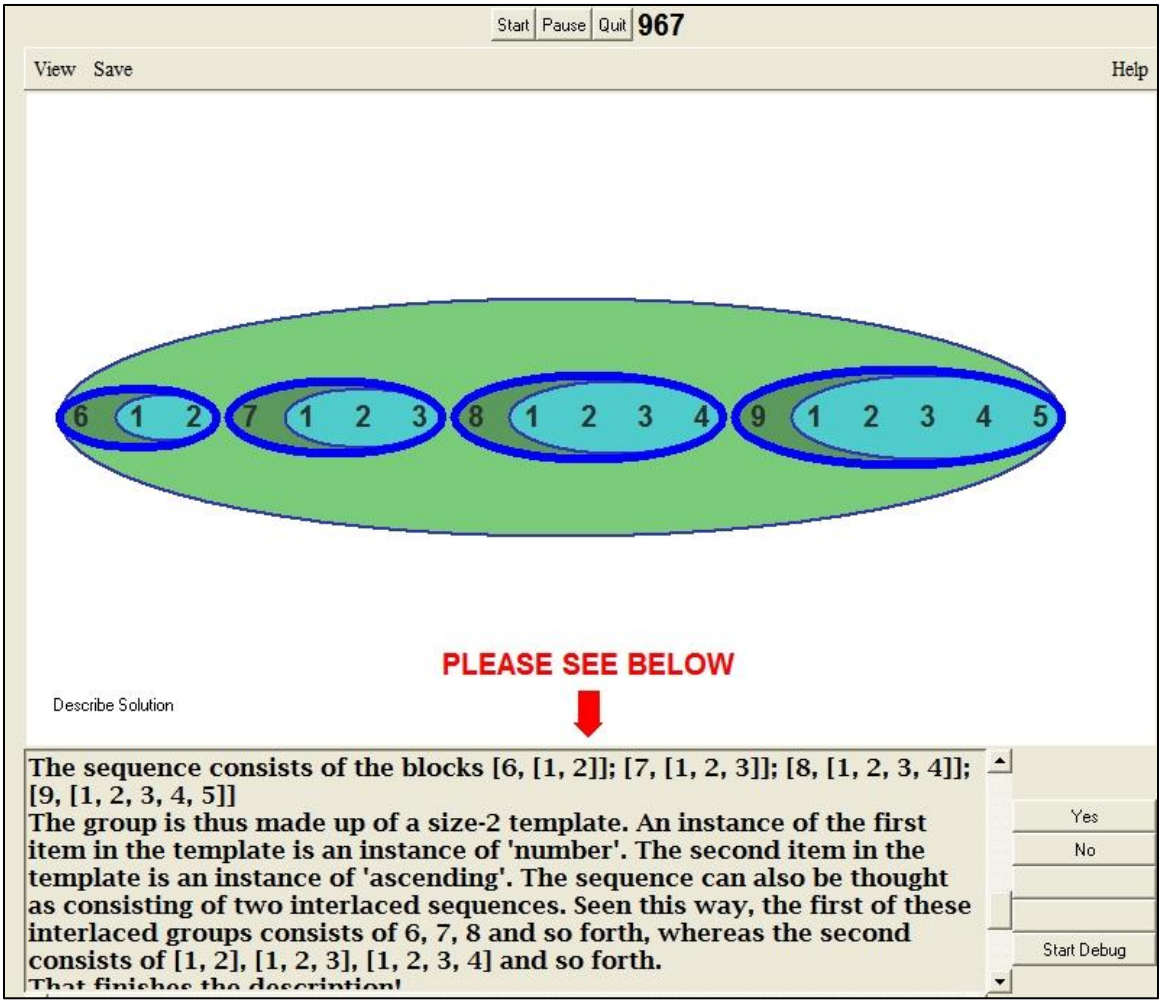


Figure 3.11 And the solution is explained.

We have now reached the final screenshot in this series. A few more terms are visible now, thanks to the human's having answered "yes" to the question in the previous screenshot. The group "(9 (1 2 3 4 5))" was added at that point.

Since Seqsee's guess about the next term was correct, it is now more certain about the nature of the sequence, and it can consequently delete

inconsistent groups such as "((1 2) 7)", doing which results in a tidier Workspace.

Seqsee then describes the sequence:

> "The sequence consists of the blocks (6 (1 2)); (7 (1 2 3)); (8 (1 2 3 4)); The group is thus made up of a size-2 template. An instance of the first item in the template is an instance of "number". The second item in the template is an instance of "ascending". The sequence can also be thought of as consisting of two interlaced sequences. Seen this way, the first of these interlaced groups consists of 6, 7, 8 and so forth, whereas the second consists of (1, 2), (1, 2, 3), (1, 2, 3, 4) and so forth. That finishes the description."

In the remainder of this chapter, I use the following definition of "success": Seqsee is deemed to have succeeded in solving a problem if it correctly extends the sequence for a few terms and also comes up with a description that accounts for all terms that it knows about. This definition is somewhat lacking: according to this definition, Seqsee always fails on sequences with an initial blemish (for example, the sequence "7, 1, 2, 3, 4, …"). Seqsee describes that sequence as

> The sequence contains an initial blemish: 7 does not fit. The sequence consists of the blocks 1; 2; 3; 4 , where each term is the successor of the previous term.

It is dangerous to consider "7 does not fit" as fully accounting for that term, even though it happens to be spot-on in this particular case. The danger is that, by analogy, other descriptions that are dubious would become acceptable. For example, a description of the sequence "(1), (1, 2), (1, 2, 3) …" that says "The sequence contains an initial blemish: 1, 1 and 2 do not fit." but that is otherwise accurate would unfortunately become acceptable.

This, then, was a short tour of Seqsee solving one problem. In a different run, things would have proceeded somewhat differently. There is one particular variation that occurs frequently and that showcases an important feature of Seqsee, and so a screenshot from this other type of run is presented next.

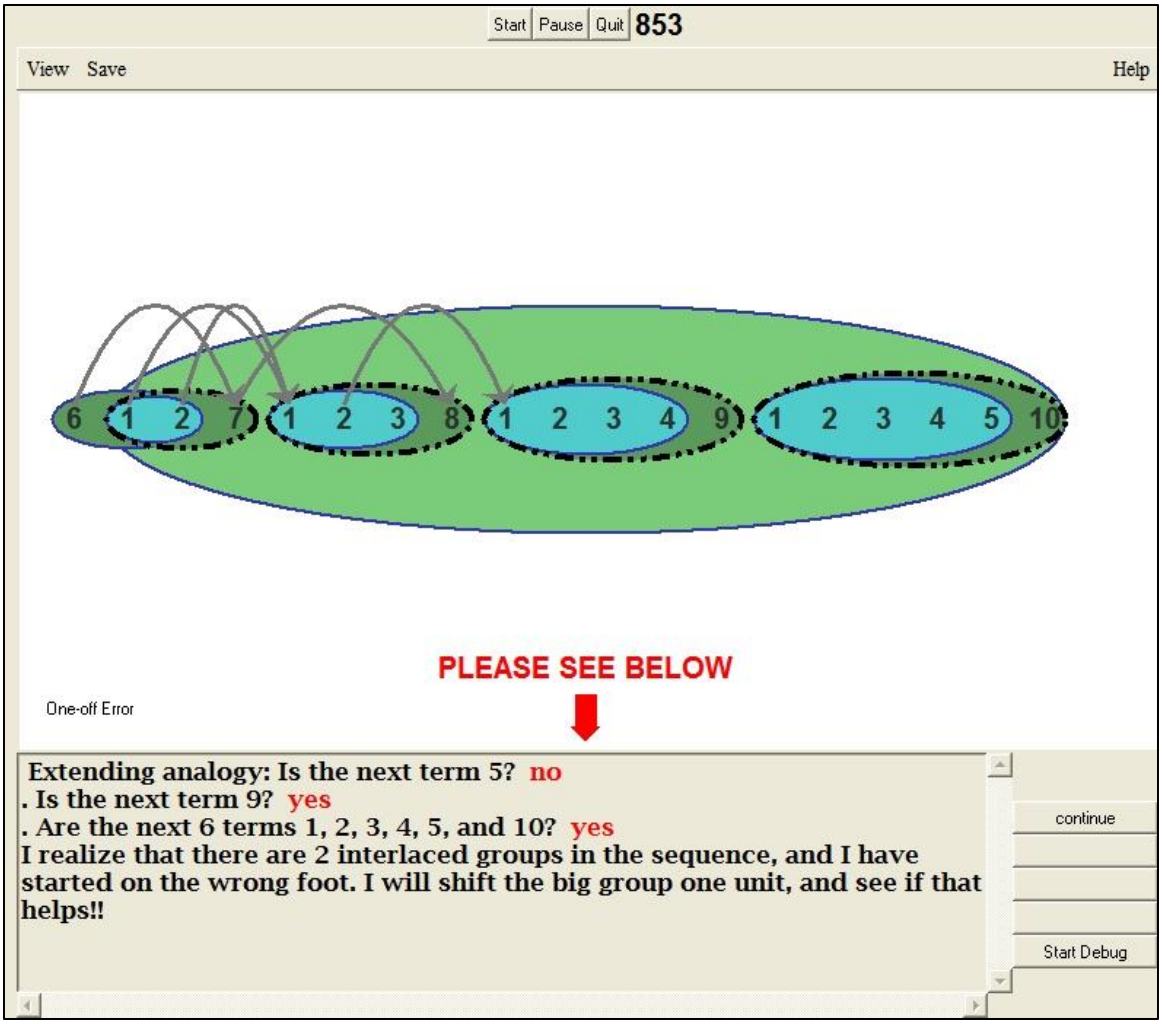
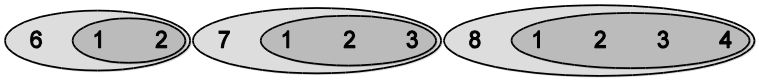
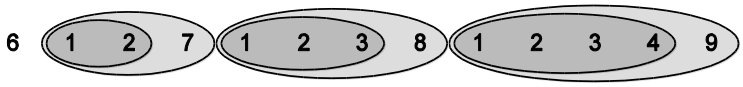


Figure 3.12 Starting out on the wrong foot

Here, instead of understanding the sequence as shown below in Sequence 69, Seqsee has started off on the wrong foot, as it were, seeing it as Sequence 70 instead.

Sequence 69. 

Sequence 70. 

Seqsee realizes that something is amiss when it tries to extend leftward the large group whose three members are the ovals shown in Sequence 70. In

this it fails, as it expects to see a "1" and "6" to the left of "((1 2) 7)" but does not find them. Since its current theory about the nature of the sequence is that it is made up of blocks based on a template of size 2, one likely cause of its not being able to extend the pattern all the way back to the beginning of the sequence is that it may have started midway through some block (such as midway through "(6 (1 2))").

Section 3.3 GARDEN-PATH SEQUENCES

Chapter 2 described sequences whose initial terms suggest some interpretation that turns out to be incorrect, thereby making the sequence somewhat harder to understand. The following two sequences are essentially identical in their formation rule, but are very different in how quickly they are understood by people and by Seqsee.

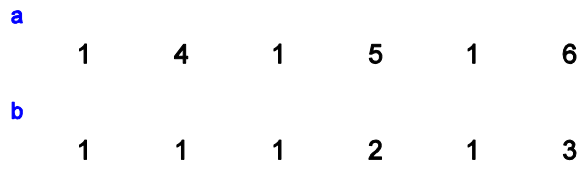


Figure 3.13 Two structurally similar but cognitively dissimilar sequences

The next figure compares Seqsee's performance with human performance on these two problems. "Human performance" refers to data that I collected in Robert Goldstone's laboratory in 2008. The experiment is described in depth in Appendix D. The task faced by the human subjects was subtly different: they were shown the terms of the sequence and were required to provide the next few terms. The difference from what Seqsee does is of course that Seqsee can ask questions such as "Are the next three terms 1, 2, and 3?", possibly receive "no" as an answer, and then give it another go. Because of this important difference, these comparisons are somewhat criticizable. However, when the difference in human performance on two such sequences is stark, and when Seqsee displays the same qualitative difference, it suggests that Seqsee might be doing something right, in the sense that it is doing something very humanlike.

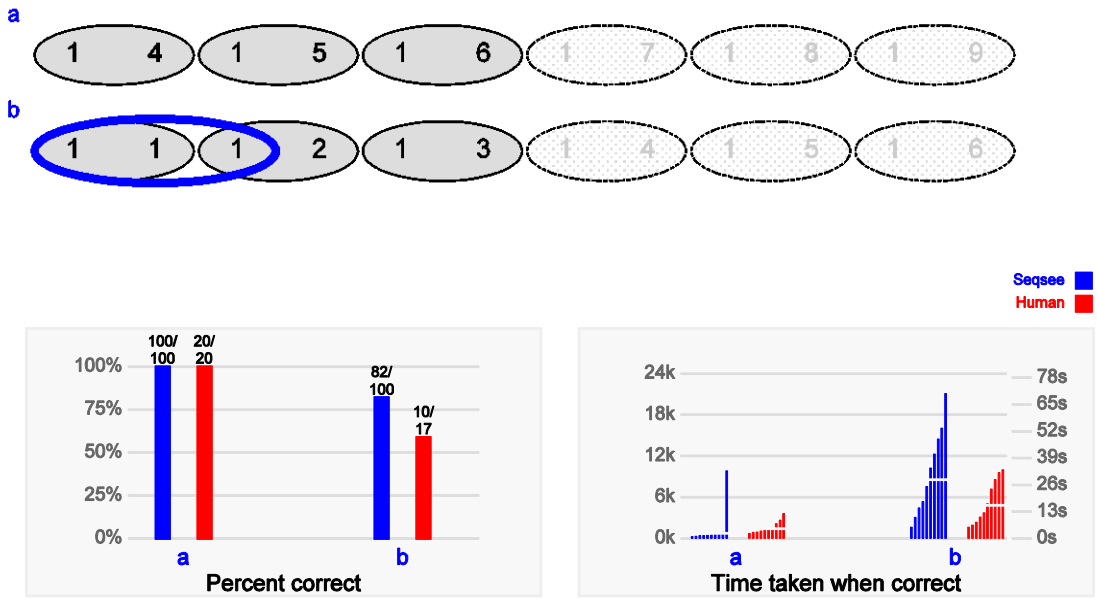


Figure 3.14 Comparison of Seqsee (blue) with human (red) performance

The second sequence contains a distracting blue oval that serves the same purpose as the blue oval in Figure 3.5 earlier — it points out a misleading chunk of the sequence. Since the first sequence contains no distractors, it is seen much more quickly.

One extra feature in the second chart that must be noted is the presence of labels such as '13s' at the extreme right. The data regarding humans presented in this chart were of course measured in seconds, not in codelets. The relative scaling of codelets versus seconds uses the formula "1 second = 300 codelets", which is roughly the speed at which Seqsee runs.

The difference in how easily people extrapolate the two sequences is qualitatively similar to the difference in how easily Seqsee solves these. Both get the second sequence wrong quite often, though neither one ever makes a mistake on the first. The time taken is also much greater — in fact, many times greater — for the second sequence.

3.3.2 A Few More Garden-path Sequences

Let us look at a few more garden-path sequences. However, I have no corresponding human data for these.

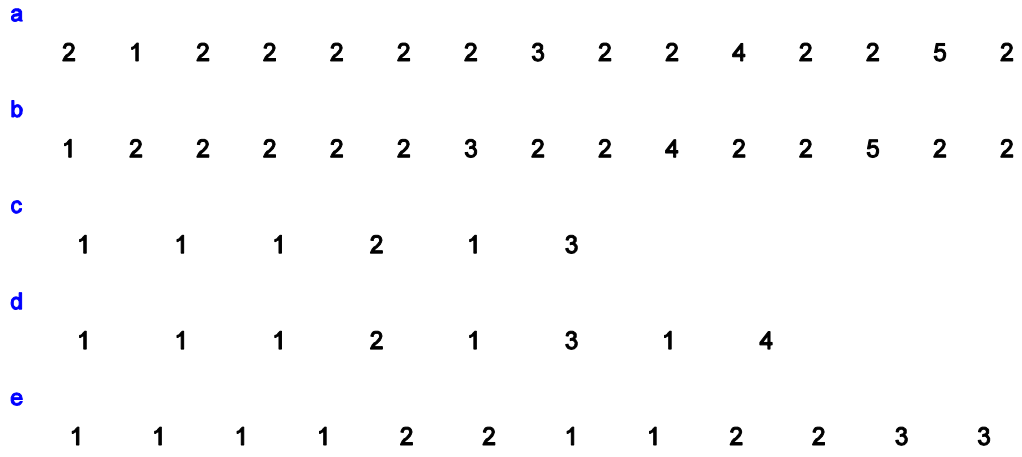


Figure 3.15 A few more garden-path sequences

Sequences *a* and *b* above are nearly identical. The only difference between the two is that Sequence *a* contains an extra "2" at the beginning. There is reason to include both: Seqsee trivially gets sequence *b*, but struggles mightily with sequence *a*. This is unfortunate — Sequence *a* was one of the "target" sequences for this project. Hofstadter (Hofstadter and the Members of the Fluid Analogies Research Group, 1995, p. 51) went so far as to call it "the theme song of the Seek-Whence domain". A small consolation is that Seqsee *nearly* succeeds in solving this sequence almost every time, and I will describe shortly why it is only rarely that it fully succeeds.

Sequences *c* and *d* are identical — they differ only in that a different number of initial terms has been revealed in the two cases. The purpose of including both here is to show that misleading interpretations suggested by some terms can be overcome more easily if more non-misleading terms are present. This is hardly surprising, and both the sequences have been included here to show that having more terms does indeed make the sequence easier for Seqsee. If there are no misleading terms, revealing more terms does not cause a corresponding drop in the time required to solve (Figure 3.29, on page 70, provides one such example).

Figures 3.16 and 3.17 below show the sequences with some of their distracting pieces clearly marked, and Seqsee's performance on these

sequences. I should remind the reader again to not be distracted by the rightmost vertical line in a percentile chart. The rightmost line is the tallest and forces itself into our attention, but one only has to look at the charts corresponding to sequences *a* and *b* in Figure 3.17 below to realize the importance of the shape — if one ignores the slowest 10% of the runs, it becomes clear that sequence *b* is solved much more quickly than sequence *a* is, although the tallest lines would have us believe otherwise.

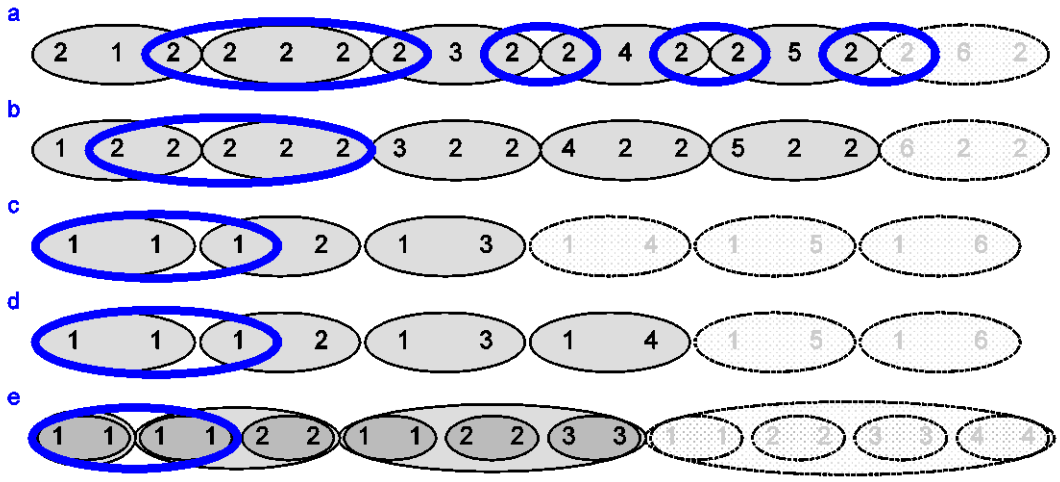


Figure 3.16 Sequences from Figure 3.15, with ovals

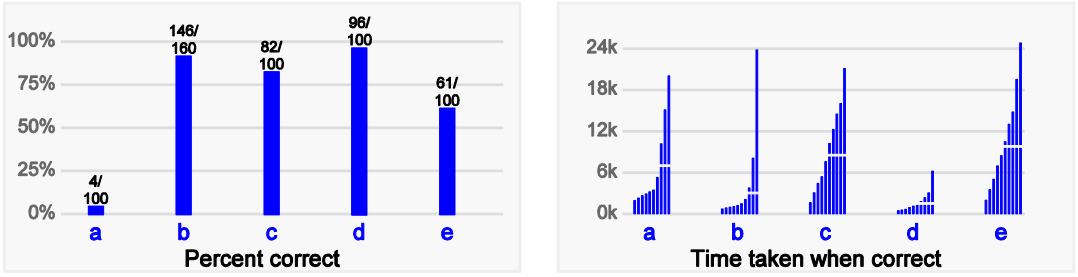


Figure 3.17 Seqsee's performance on sequences from Figure 3.15

The sequences shown above also contain the distracting blue ovals. These are the "garden paths" in the sequence, as it were. Large chunks consisting of only one type of element ("2"s in the first two sequences, and "1"s in the next three) are quickly seen as a single group, and need to be broken up before the

sequence can be understood. The difference in performance on the nearly-identical sequences *a* and *b* is immense — Seqsee almost always fails on the first, and almost never fails on the second — but the number of distractions in the two cases is also vast. Sequence *b* may be seen thus:

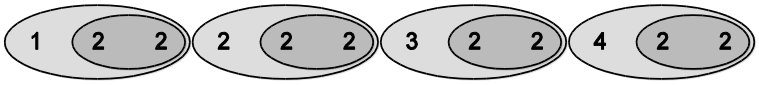


Figure 3.18 Structure of Sequence *b*

Looked at this way, the many pairs of "2" are consistent with the global understanding of the sequence. In sequence *a*, however, these pairs of "2"s *are* a distraction. The two figures below show Seqsee floundering between two equally bad alternatives when solving sequence *a*. In understanding the sequence, Seqsee often reaches the stage shown below, in which it has split the sequence into 3-element chunks, but starting at the second element ("1").

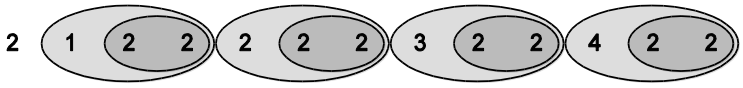


Figure 3.19 Part one of Seqsee floundering on Sequence *a*

This level of understanding is enough to extrapolate the sequence: the analogous structures "(1 (2 2))", "(2 (2 2))" through "(4 (2 2))" are enough to allow it to guess that the next few groups are "(5 (2 2))", "(6 (2 2))", and so forth. However, the initial "2" has not yet been accounted for. Extending leftward from the current configuration is not possible, as that would require the presence of "(0 (2 2))" to the left of "(1 (2 2))". Seqsee tries to fix this by moving the boundaries of the groups by one unit. However, all the pairs of "2" form a single unit, and unless these units are broken into their constituent elements, shifting boundaries leads to a situation such as Figure 3.20 below.

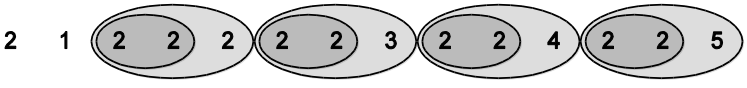
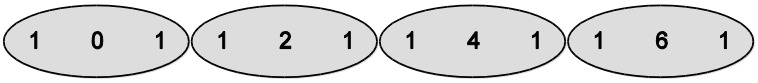


Figure 3.20 Part two of Seqsee floundering on Sequence *a*

Here, nothing has been improved. Indeed, if anything, the situation has worsened: *two* elements are now unaccounted for. Trying the same boundary-moving trick again leads right back to Figure 3.19. Seqsee oscillates between these two positions, and in the meanwhile it keeps asking questions such as "are the next 3 terms 2, 2 and 6?". Since both of the explanations it has found are adequate to extrapolate the sequence, it keeps getting "yes" as an answer.

The interest and importance of this problem for exploring the mechanics of cognition lies firstly in the seductive, almost irresistible length-5 run of "2"s that Seqsee must sooner or later find a way to correctly split apart. In this task, Seqsee succeeds. But there is a second key interest of this sequence's challenge, and that is the challenge of splitting the many length-2 runs of "2"s. This is a much harder problem, because of the presence of *many* such pairs, each of which lends credence to, and thereby strengthens, all of the others. Seqsee fails in this aspect of intelligence. It cannot break all the "(2 2)" groups up simultaneously. It has no understanding of them as being essentially *one single group*, and it cannot thus get the key insight.

Rather than be disappointed at this shortcoming of Seqsee, there may be reason to feel elated. Seqsee's difficulty understanding this sequence is eerily similar to the trouble that Hofstadter had — described in his own words in (Hofstadter and the Members of the Fluid Analogies Research Group, 1995, p. 39) — on a similar sequence:



Hofstadter describes why it took him a while to understand it:

> What I had perceived at first was this structure:
>
> 1 – 0 – (1 1 2) – (1 1 4) – (1 1 6) – (1 1 8) - …
>
> and then, by shifting the packet's boundaries, this structure:
>
> 1 – (0 1 1) – (2 1 1) – (4 1 1) – (6 1 1) - …

But for some reason, and don't ask me why, it had been much harder for me to come up with this view of the new sequence:

(1 0 1) – (1 2 1) – (1 4 1) – (1 6 1) – (1 8 1) - …

Actually, *do* ask me why, because I think I can tell you! The answer is partly *inertia* — after all, for years, I had perceived [a nearly identical sequence] in terms of a "[1 1 2n]" pattern — and partly a *default esthetic preference*, which caused me to strongly resist cleaving a pair of 1's down the middle. Specifically, the two identical numbers "wanted", in my mind, to be kept together, so it took considerable external pressure to knock me out of this default way of seeing things.

Section 3.4 QUASI-PERIODIC SEQUENCES

This section reports Seqsee's performance on quasi-periodic sequences. The following two are examples of what Seqsee can solve.

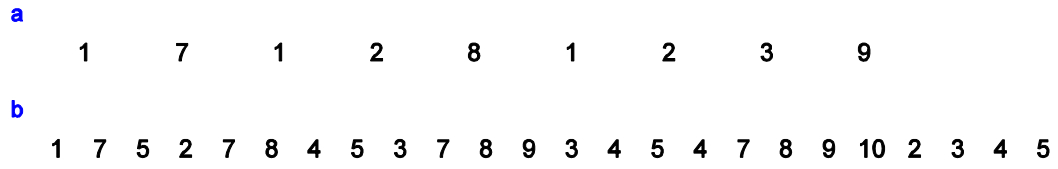


Figure 3.21 Two quasi-periodic sequences that Seqsee solves

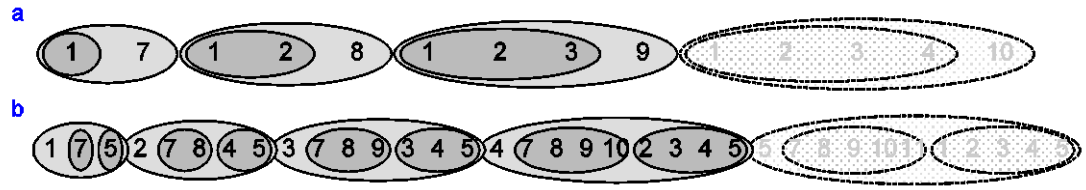


Figure 3.22 Sequences from Figure 3.21, with ovals

In both cases, the template underlying the sequence is of a fixed size — 2 and 3 top-level constituents, respectively — but at the level of individual numbers, the instantiations of these templates are not of a fixed length in either sequence. Seqsee has no trouble with either sequence[10]:

[10] However, if Seqsee were not allowed to use negative numbers (as was the case in the original Seek-Whence domain), Sequence *b* would not have the simple pat answer shown by the grayed-out terms — it would need to worry about what would happen once it hit a "0" and needed to take its predecessor.

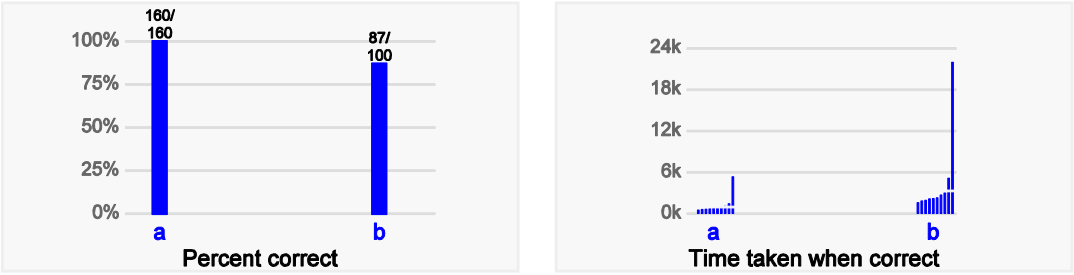


Figure 3.23 Performance on sequences from Figure 3.21

3.4.2 Comparison with Human Performance

Let us look at a few more quasi-periodic sequences. In the experiment with human subjects, the following three sequences were used, and we can thus compare human performance with that of Seqsee:

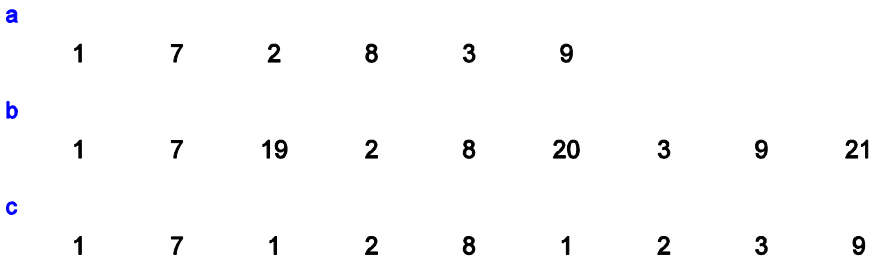


Figure 3.24 Three more quasi-periodic sequences

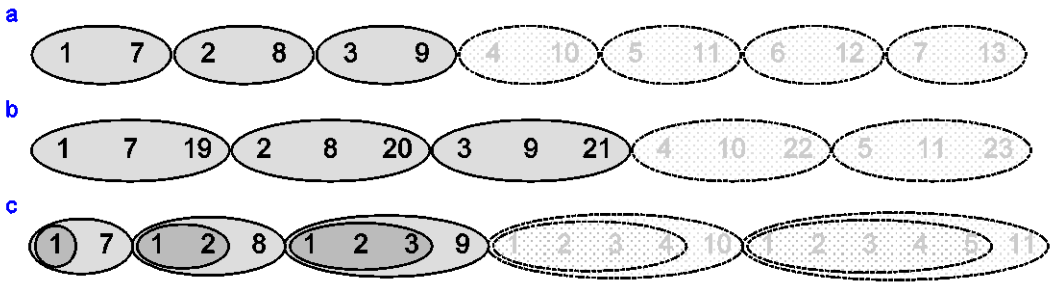


Figure 3.25 Sequences from Figure 3.24, now with ovals

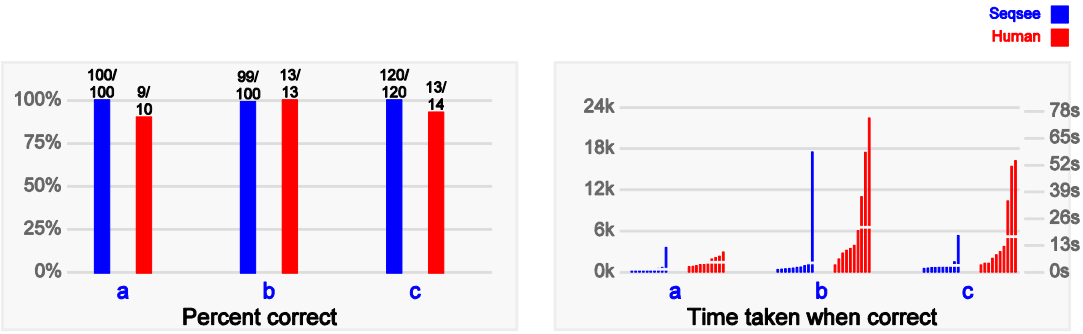


Figure 3.26 Performance on sequences from Figure 3.24

As the reader can see, both Seqsee and people get everything right essentially all the time, and the second and third sequences are roughly equally difficult for both, and significantly more difficult than the first sequence for both. Since the relative difficulty of the three sequences is hard to see in the case of Seqsee, that portion of the chart above has been magnified below.

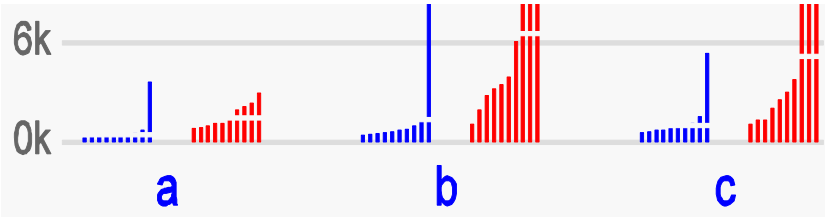
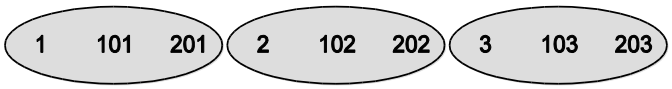


Figure 3.27 A portion of Figure 3.26, magnified

3.4.3 The Effect of Template Size

Seqsee can, in principle, see quasi-periodic sequences of any period, but the larger the period, the harder it becomes for Seqsee. In Figure 3.28, the numbers below the bars refer to the period. The sequence with a period of 3 that I used was Sequence 71. Sequences with other periods were analogous. For example, for period 2, the first four terms of the sequence were “1 101 2 102”. Incidentally, these sequences contained no distractors for Seqsee — it does not see the number “101” as consisting of two “1”s and a “0”. To Seqsee, the second element of the sequence bears absolutely no resemblance to the first element, or to the third.

Sequence 71. 

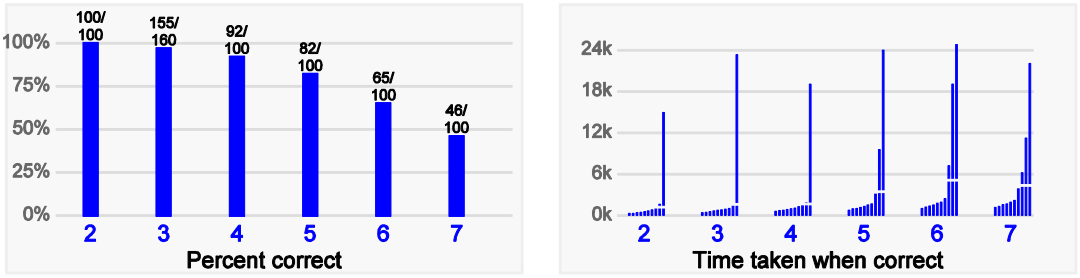


Figure 3.28 The effect of template size

As the period becomes higher, Seqsee fails more frequently (because of lingering bugs in the program) and takes longer (because a slight resistance to seeing sequences with higher periods has been built into Seqsee to defend against the many spurious long-distance similarities that are present in almost all sequences).

3.4.4 The Effect of Revealing More Terms Initially

The next figure shows how well Seqsee fares on a single sequence — on Sequence 71 — when different numbers of initial terms are revealed. The number of initial terms that were given as input to Seqsee is shown below the bar.

There is no appreciable difference between the various bars in the average amount of time taken. However, as more terms are revealed, Seqsee is less likely to get stuck in an inappropriate initial hunch about the nature of the sequence, and consequently the worst runs when *many* terms are shown to Seqsee are not as bad as the worst runs when *fewer* terms are shown.

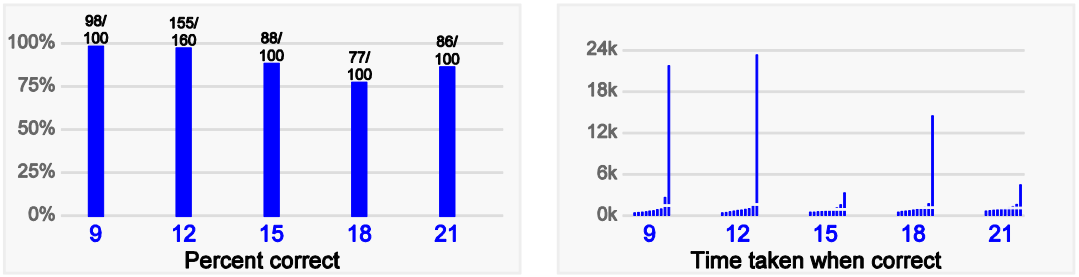


Figure 3.29 The effect of revealing more terms initially

Section 3.5 SQUINTING, OR SEEING AS

We come now to the first optional feature in Seqsee. When this feature is turned on — the default is to keep it off — Seqsee acquires a limited ability to see something *as* something else. Such "squinting", as I have called it, enables Seqsee to solve sequences that it could not, in practice, solve earlier, although, in principle, all these sequences could occasionally have been solved even without this feature. Much more important, though, is the fact that the presence of this feature brings Seqsee's way of looking at certain sequences closer to how a person would look at them.

We will look at Seqsee's performance on the following three sequences with the "squinting" feature turned off and with it turned on. As I mentioned at the start of this chapter, this feature helps Seqsee on some sequences, but hinders it on others.

a
1 1 2 3 1 2 2 3 4 1 2 3 3 4 5

b
1 2 2 2 2 2 3 2 2 4 2 2 5 2 2

c
1 2 2 3 4 4 5 6 7 7 8 9 10 11

Figure 3.30 Three sequences helped or hindered by squinting

To make "seeing something *as* something else" more explicit, here is a screenshot of Seqsee solving the first sequence above in this special mode:

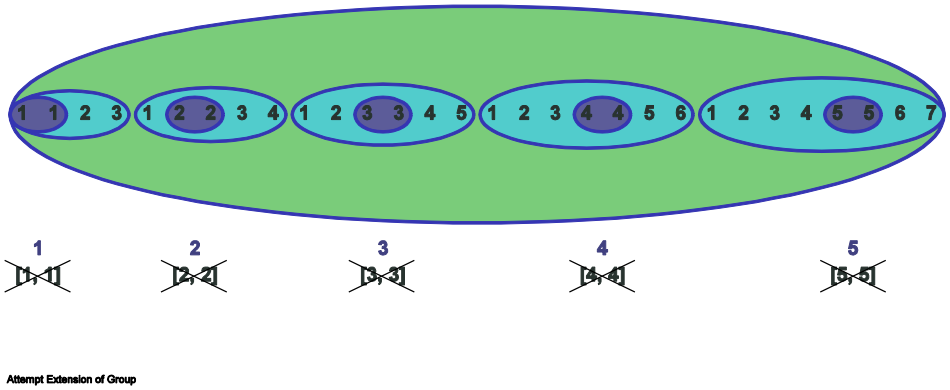


Figure 3.31 Screenshot: Seqsee squinting

At the bottom left of the screenshot, a crossed-out "(1, 1)" can be seen along with a "1" above it. This is directly below the purple oval "(1 1)" and indicates that Seqsee is seeing "(1 1)" as a "1". In fact, Seqsee understands the sequence as a spiced-up version of "(1 2 3) (1 2 3 4) (1 2 3 4 5)…".

This is not the only way to understand the sequence: it can also be understood as "((1) (1 2 3)) ((1 2) (2 3 4))…", and this involves no squinting. However, that way of understanding it requires splitting up the very salient sameness groups "(1 1)", "(2 2)", and so on, as shown in Figure 3.32. If Seqsee's ability to squint is turned off, it sometimes understands the sequence in this fashion, but only rarely.

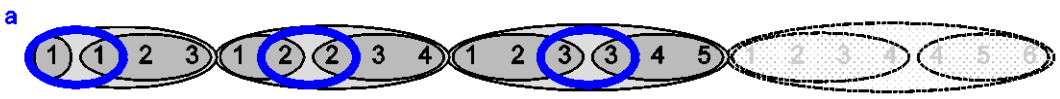


Figure 3.32 Understanding Sequence *a* without squinting

Sequences *b* and *c* in Figure 3.30 above will partially answer the question of why it is that this feature is turned off by default. When the feature is turned on, these two sequences do not show the improvement that the first sequence does. In fact, Seqsee's performance on Sequence *c* significantly deteriorates.

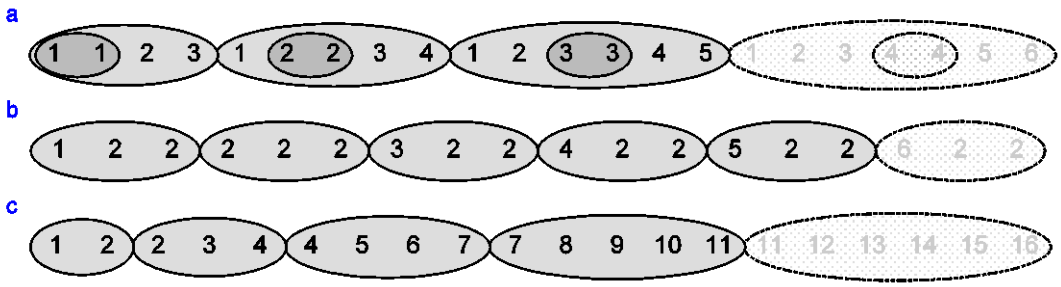


Figure 3.33 Sequences from Figure 3.30, with ovals

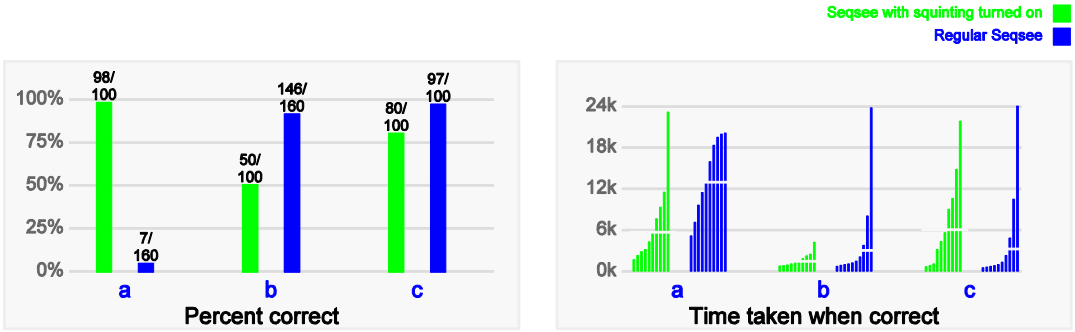


Figure 3.34 Performance on sequences from Figure 3.30

Why does allowing it to squint cause Seqsee to take longer on Sequence *c*? This is because Seqsee's ability to squint is limited: it can squint only once in a single group[11]. This limitation makes it impossible to see sequence fragments such as "2 3 3 4 5 6 6 7" as an ascending group, since that would require that the "3 3" be seen as a "3" *and* that the "6 6" be seen as "6". A mathematically adept person might see Sequence *c* above as being "1 2 3…" with the nth term doubled whenever $n-1$ is a triangular number, but such a sophisticated representation is, naturally, miles beyond Seqsee's reach. In sequence *c*, with squinting turned off, Seqsee never arrives at certain misleading interpretations

[11] The reason for this restriction is described in Appendix B along with other restrictions on groups in Seqsee. Briefly, allowing multiple squints is a significantly harder problem.

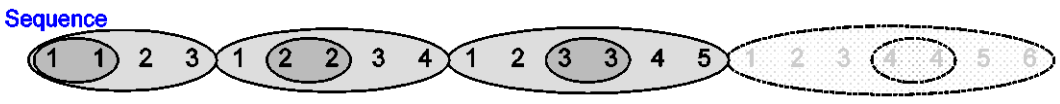
that the presence of squinting makes reachable, including, for example, seeing the first five terms as "1 2 3 4". Seeing those five terms in this way makes it impossible for Seqsee to find a consistent interpretation for the entire sequence.

In sequence *b*, similarly, when squinting is turned off, Seqsee never sees the initial seven terms as "1 2 3". Since seeing those terms in that fashion leads nowhere for that sequence, allowing it to squint makes Seqsee slower. This is a shortcoming in Seqsee: it is overzealous in its use of squinting.

A natural thing to wonder is whether the problem would go away if we could turn down the "zealousness knob" for squinting. In a sense, I already do that — squinting is turned off in the "factory setting". In the default setting, although I do not see the problems described above, I obviously also lose all the benefits of squinting. If squinting were merely dialed down, we'd see fewer problems but we'd also see fewer benefits. The main problem to be fixed is that the current way of sniffing "the need to squint" is a noisy signal that leads to many false alarms.

Section 3.6 REMINDINGS

Ideally, having seen a sequence before should make understanding it again quicker. This does in fact happen when another optional feature — long-term memory — is turned on. Consider the *marching doubler* that we have seen before (it is repeated at the top of Figure 3.35 below). With squinting turned on, Seqsee perceives this sequence in about 6000 codelets, on the average. What if Seqsee had seen this sequence once before, or twice, or five times? The following chart shows how the performance on the marching doubler improves from having seen it earlier. The labels below the bars indicate how many times Seqsee had seen this sequence earlier.



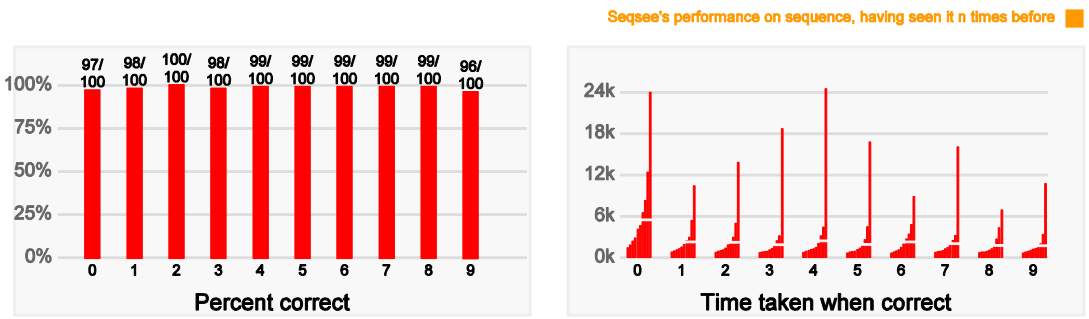


Figure 3.35 Effect of having seen the same sequence before

In comparing the ten percentile charts shown above, it is important to not focus merely on the rightmost bar (which represents the time taken for the worst run) in each chart. Attention must also be paid to the average time taken (indicated by the horizontal cut). Starting from a fresh slate, Seqsee takes on an average around 6000 codelets to see the sequence. When it has seen the sequence even once, this number drops to around 2500. Having seen the same sequence more than twice before only marginally improves the performance beyond this.

Moving on, understanding a sequence should be easier if *similar* sequences have been seen earlier. The target sequence in the figure below is the *marching tripler*.

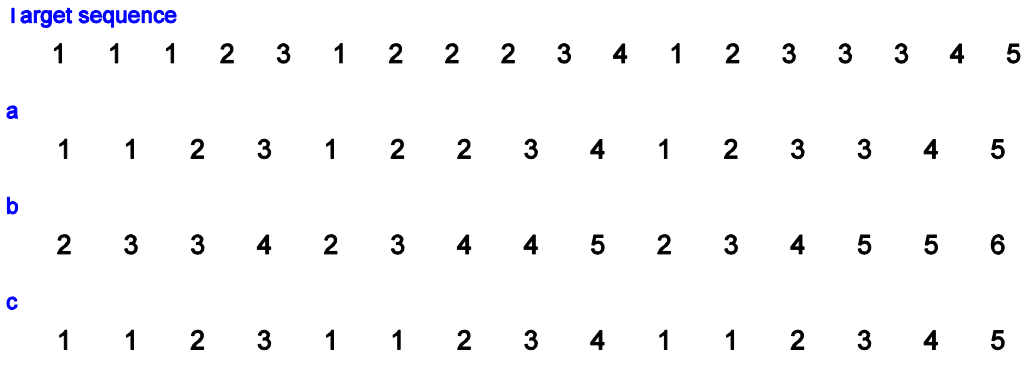


Figure 3.36 The effect of having previously seen similar sequences

Without the benefit of prior memories, but with squinting turned on, Seqsee requires around 4000 codelets to solve it, on the average. But when

Seqsee had already seen the marching doubler (sequence *a*) ten times, this number dropped dramatically to half that value. The first group, both in the target sequence and in Sequence a, is a "1 2 3", but the transfer effect does not depend on such similarities. Having seen the superficially different but fundamentally similar Sequence *b* has an equally strong effect. Even Sequence *c* has a beneficial effect — lesser, but noticeable — although Sequence *c* is different from the target sequence in important ways.

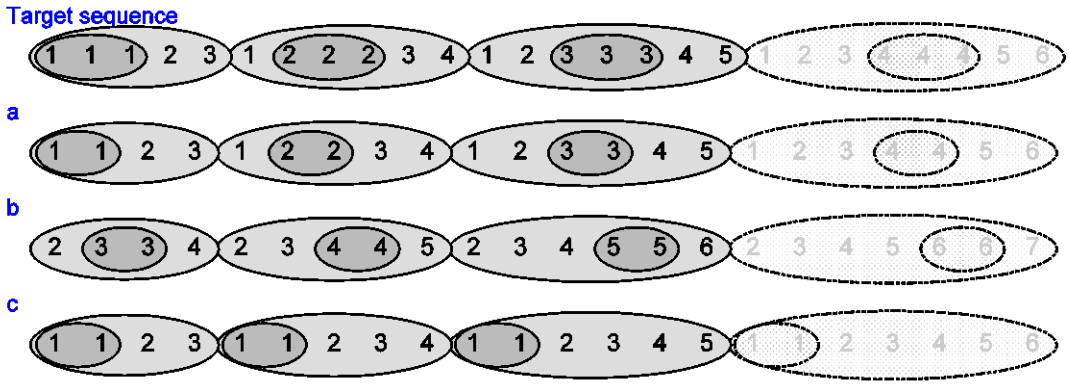


Figure 3.37 Sequences from Figure 3.36, with ovals

Figure 3.38 Effect of having seen similar sequences

Long-term memory has another benefit: it allows Seqsee to take a stab at certain sequences that it otherwise could not solve. With no memories, Seqsee does not have any suggestions for extrapolating a sequence given only the first

term — say, '9'. Given just this one term, and without prior memories, Seqsee is unable to do anything. However, if Seqsee has already seen the sequence "(1), (2, 2), (3, 3, 3), (4, 4, 4, 4), …", it has seen numbers followed by the same number (for example, it has seen "2" followed by a "2" within a single group), and can suggest "9" as a continuation of the sequence. If it receives a "yes" to the question whether the next term is "9", it will create a sameness group "(9 9)", and will likely ask if the next term — the third — is also a "9". Moreover, since it has already seen sameness groups followed by longer sameness groups, it will possibly ask if the terms following the sameness group "(9 9)" are three terms "(10 10 10)". Chapter 7 describes how reminding works in Seqsee.

Section 3.7 EXTENDING SEQSEE: PRIMES

The third and final optional feature that I will mention is the addition of the category "prime number". With this addition, the following sequences become solvable by Seqsee.

Figure 3.39 Sequences based on the prime numbers

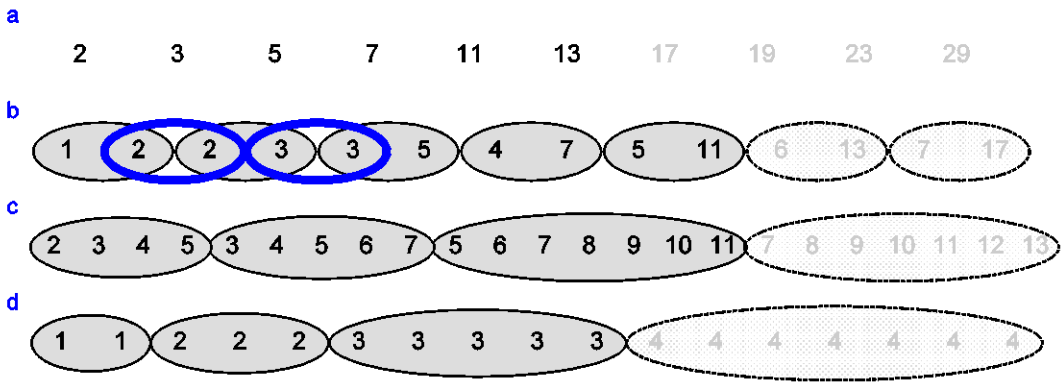


Figure 3.40 Sequences from Figure 3.39, with ovals

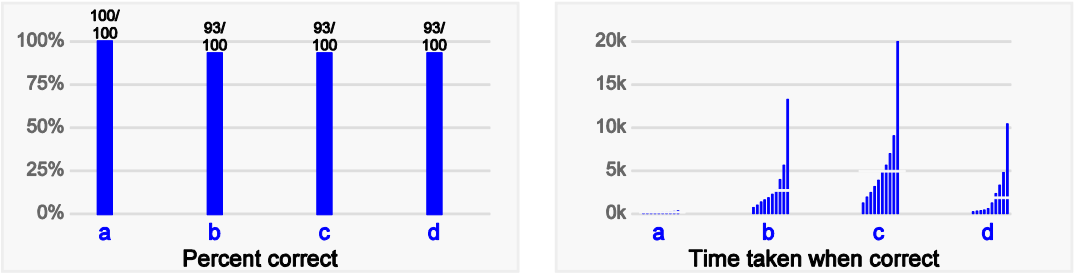


Figure 3.41 Performance on sequences from Figure 3.39

Seqsee's extremely limited understanding of the concept "prime number" is discussed in Section 6.3.

Section 3.8 Further Comparison with Human Performance

Robert Goldstone suggested comparing Seqsee's performance with human performance on the following sequences.

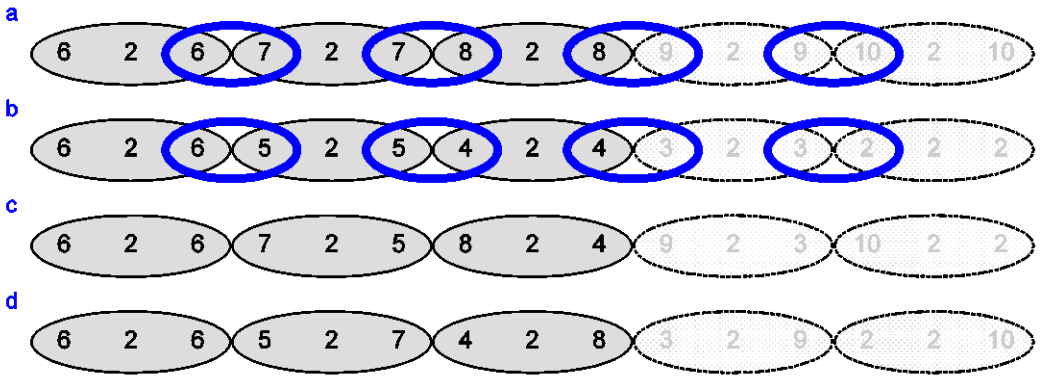
a
6 2 6 7 2 7 8 2 8

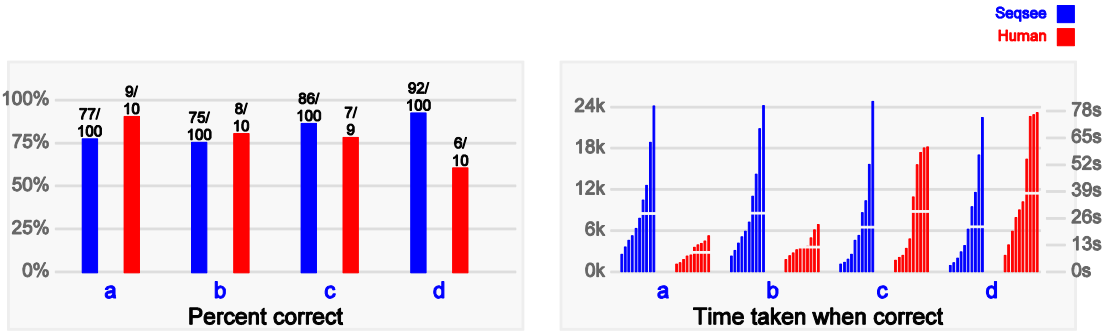
b
6 2 6 5 2 5 4 2 4

c
6 2 6 7 2 5 8 2 4

d
6 2 6 5 2 7 4 2 8

In the first two, both “6”s change in the same way — either both to “7”, or both to “5”. In the last two, on the other hand, they go in different directions. This should make, so the theory goes, the first two easier. And, indeed, Goldstone’s guess was correct: human subjects found the first two significantly simpler. For Seqsee, however, exactly the reverse happens. My guess at the reason for this odd disparity is indicated by the thick blue ovals, which Seqsee finds attractive (I know this to be a fact) but perhaps people don’t (this is only a guess).





3.8.1 Different Types of Relationships Between Ascending Groups

a

1 7 19 2 8 19 20 3 9 19 20 21

b

1 7 19 2 8 20 21 3 9 22 23 24

Figure 3.42 Sequences to show two types of relations between ascending groups

Understanding how “2 3 4” can be changed to “2 3 4 5” is perhaps slightly easier than understanding how “2 3 4” can be changed to “5 6 7 8”. More importantly, noticing the similarity between “2 3 4” and “2 3 4 5” is quicker than in the other pair. Consequently, it is a fair expectation that Sequence *a* is understood more easily. This turns out to be the case both for people and for Seqsee.

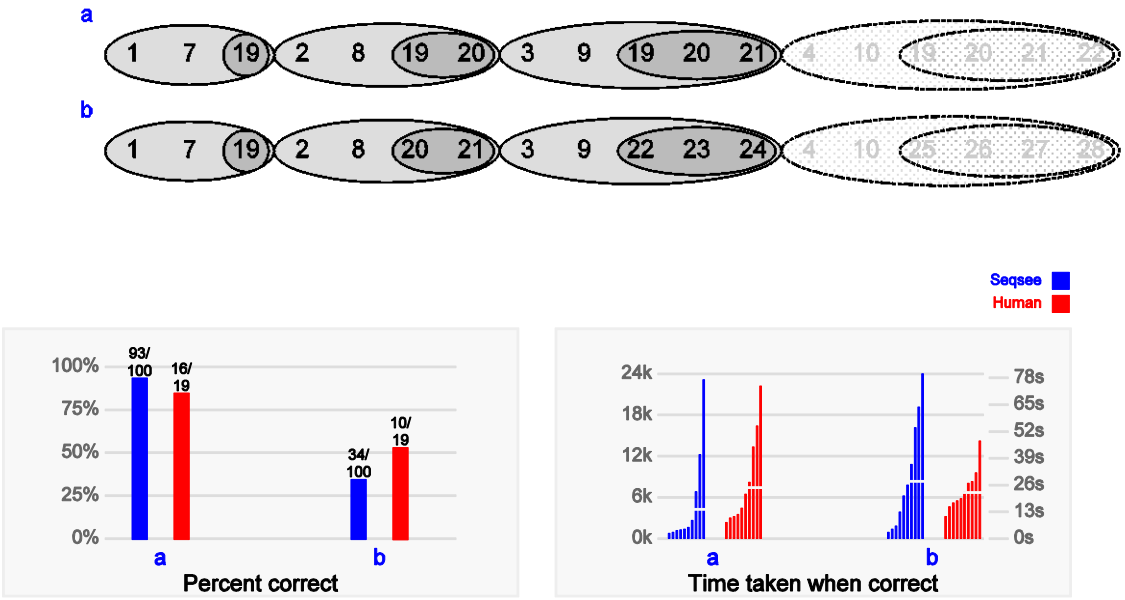
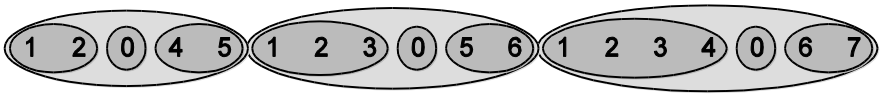


Figure 3.43 Performance on sequences in Figure 3.42

Section 3.9 Sequences Not Solved by Seqsee

Sequences that Seqsee does not solve fall into a few categories. Some of these sequences lie outside the domain. Within the domain, Seqsee is unable to solve some sequences that depend on concepts that it knows nothing about

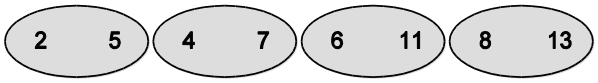
(such as “hiding” or “masking”, as seen in Sequence 91, which I have repeated from the previous chapter).

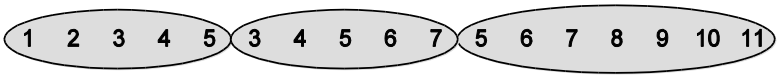
Sequence 72. 

Also within the domain are sequences that depend on ideas that Seqsee knows about, but whose implementation is lacking in some way (such as “alternation”). Finally, there are sequences that Seqsee solves only very rarely — that is, if it has been so lucky as not to have been seduced by any of the very tempting garden paths.

3.9.1 Sequences Outside the Domain

As described in the previous chapter, Seqsee does not possess such categories as the Fibonacci numbers, or even the odd numbers. This often strikes people as strange, and therefore I decided one morning to bite the bullet and add the categories *even numbers* and *odd numbers* to Seqsee. It took me all of 20 minutes to do so, and Seqsee can now be made to solve, for instance, the following sequences:

Sequence 73. 

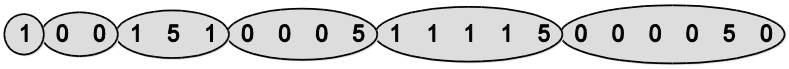
Sequence 74. 

The esoteric Sequence 74 is a succession of ascending groups that begin with an odd number and end with a prime number. I am not particularly proud that Seqsee is able to solve that sequence: without the ovals, people would find the sequence quite hard. The only point I wanted to make with the addition of odd numbers to Seqsee was that adding a shallow version of this category would be trivial and would allow Seqsee to solve moderately impressive-looking problems. I have hidden this addition behind an optional feature — “parity” — since this modified version of Seqsee *automatically* labels all odd numbers as “odd” numbers — a cognitively absurd shortcut. It does not at all address the question of how a person would realize that odd numbers are relevant to the

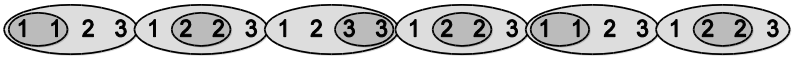
given problem. It thus shoves the hard problem of how to label appropriately under the rug and solves only the easy problem of manipulating labeled entities.

3.9.2 Sequences Based on Unimplemented Concepts

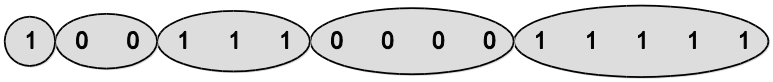
Sequence 75 below consists of lengthening blocks of "0" and "1", with every fifth term replaced by a "5". Seqsee cannot solve this.

Sequence 75. 

It also cannot solve Sequence 76, which is the original "bouncing doubler".

Sequence 76. 

In order for Seqsee to successfully solve Sequence 75, some ideas from Section 6.6 ("Derivative sequences") might come in handy. By using a mechanism similar to the one it uses for squinting, Seqsee can be relatively easily enabled to "fix" the blemishes in Sequence 75 to produce Sequence 77.

Sequence 77. 

If Seqsee could simultaneously work on the original and the derived sequence and try to describe how they differ, it might make headway on this sequence. Getting Seqsee to honestly understand Sequence 75 will not be easy, but I am unable to estimate just how hard it will turn out to be.

As for Sequence 76, Seqsee can already understand it, given enough terms — but only in a very uninspired fashion, which consists of the initial sixteen terms repeating.

Seqsee can already describe how one block there relates to the next in terms of the movement of what is doubled. It can see how "(1 1) 2 3" changes to "1 (2 2) 3", and how this latter group changes in the same way to "1 2 (3 3)". However, it is not possible to change "1 2 (3 3)" in the same way, and Seqsee will

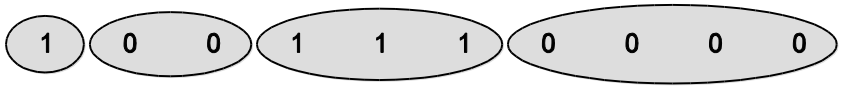
therefore hit a snag. There are multiple possible responses to such a snag, as we saw in Section 2.2.1: the next term might be "1 2 3 (4 4)" (by extending), or "1 2 3" (where the non-existent "4" is doubled), and so forth. Seqsee will need to be made sensitive to different types of snags.

As a bonus, efforts at extending Seqsee to solve Sequence 76 will almost certainly have another benefit. Although this is not obvious, there is deep similarity between being able to solve the bouncing doubler (Sequence 76) and being able to work only with positive numbers. Solving any of these satisfactorily will involve making Seqsee aware of the notion of "hitting a wall". In the first case, Seqsee will try to double the fourth element of a length-3 group as discussed above, and in the second case, it may need to extend a group such as "(4 3 2 1)" rightward without using negative numbers. Extending Seqsee to solve Sequence 76 will, therefore, make it easy to banish negative numbers (which are not present in the original Seek-Whence domain).

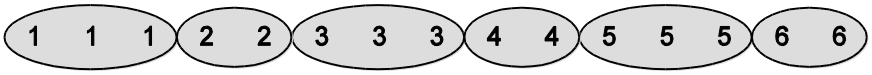
3.9.3 Failure Caused by a Deficient Implementation

While Seqsee can solve several sequences that are based on alternation between two items (for example, Sequences 78 and 79 below), it cannot perceive cycling among more than two items. Furthermore, it cannot perceive alternation in position, such as in Sequence 80.

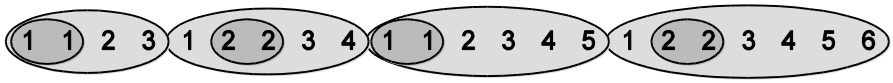
Sequence 78.



Sequence 79.

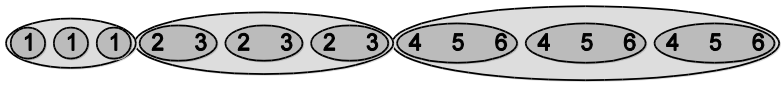


Sequence 80.



3.9.4 SEQUENCES RARELY SOLVED

I will present just a single example:

Sequence 81. 

Here, the “1 2 3”-group and the “2 3 4 5 6”-group are quickly formed and subsequently are almost never correctly split. Once in a while, however, Seqsee stumbles across the solution.

Section 3.10 PARTING THOUGHTS

This chapter has been focused on evaluating Seqsee in terms of how well it does at solving different types of sequences. Of course, there are other aspects that must also be considered in evaluating a system like Seqsee.

The coming parts of this dissertation will shed light on some of these aspects. Extensibility was an important goal for Seqsee, and Chapter 6 shows how easy it is to extend Seqsee by adding some types of new categories. Another goal was to construct tools that enable visualization of what Seqsee is doing, and screenshots are peppered throughout this dissertation to that effect.

In the next chapter, we will begin to take a deeper look at Seqsee's architecture.

Chapter 4 CODELETS, CODELET TREES, AND PRESSURE

In this chapter, I will begin a description of Seqsee's architecture. Two aspects of Seqsee's working are worth noting at the outset: its blackboard architecture and its nondeterminism. Seqsee, like its predecessors in the Fluid Analogies Research Group, has a blackboard architecture. This architecture is a way for hundreds of entities to collaborate. I would draw an analogy to the way that a police department working on a murder might coordinate its work. Officers must work somewhat independently of each other and yet must also coordinate closely. One officer may be busy verifying the alibi of one suspect, another may be checking out a different suspect in another part of the city. Through a blackboard is how the officers may communicate: each will write down significant findings on the board and read what others have written. What they read will probably influence the leads that the various officers follow next. Thus, the team as a whole can stay on track and in synchrony, even without any two officers ever meeting face to face.

In the case of Seqsee, determining what the sequence is is the mystery to solve. Significant findings include some piece of the sequence that has started to make sense or perhaps some discovery about how two pieces of the sequence are related. The component called the Workspace corresponds to the blackboard, and this is where observations about the sequence are noted.

The second aspect of the architecture worth pointing out is its nondeterminism. Even on the same input, Seqsee's understanding may follow extremely different trajectories and this can sometimes lead to very distinct interpretations of the sequence.

Everything that happens in Seqsee is carried out by codelets. A codelet is a tiny piece of code with a small, local impact on Seqsee's internal world. A single codelet, for example, may create a bond between two elements in the sequence if it notices that they are analogous. Another codelet may attempt to extend the ascending group "2 3 4", by looking for a "5" to its right or a "1" to its left.

The effect of a single codelet is so small that solving all but the simplest sequences requires at least several thousand codelets' joint effort. Solving even moderately complex sequences can require the services of tens of thousands of codelets. Moreover, such tasks as explaining the solution once it is found are carried out not by one but by about a dozen different types of codelets.

Because of the tininess of individual codelets, any description of a single run of Seqsee that lists all the codelets that ran would be very long and incomprehensible. A description at that level would be almost as unenlightening as describing climate at the level of atoms, and we must therefore find higher levels of description. In this chapter, I will roughly explain, at various levels of description, one single run of Seqsee solving the following sequence:

Sequence 82. **6 1 2 7 1 2 3 8 1 2 3 4**

Section 4.1 THE CODELET-LEVEL DESCRIPTION

Seqsee contains several types (or families) of codelets. To implement each family, I have had to write a few lines of code. Creating an individual codelet, on the other hand, is just "rubbing a copy off of the master copy", and is thus a very simple operation.

An example of a codelet family is "Attempt Extension of Group". When a codelet of this family is created, it is told what group to operate on, and optionally, whether to try to extend leftward or rightward. If no direction is specified, it chooses one probabilistically when it is run. Because of the extra information imparted at creation time, different copies of "Attempt Extension of Group" are not identical in the effect they would have when run, but of course the effects are similar for each.

The following table lists the codelets run in a fraction of one complete run on Sequence 82 — the notion of a blow-by-blow account taken to an extreme. I have shown only 12 out of over a thousand codelets. The leftmost column shows the time step at which the codelet was run, and the second column shows the type of the codelet. I have omitted details such as what each

individual codelet did, but in the final column, I have included a brief description of what each type of codelet does.

Table 4.1 Description of a run at the codelet level

Timestep	Type of Codelet	Actions typically taken by this type of codelet
127	Read from Workspace	Probabilistically chooses a single object (which includes elements, groups, and even analogies) to focus on. Appendix A describes the image behind the choice of the verb "to read".
128	Focus on a Group	Given a group, this codelet chooses an appropriate set of actions to take next, and creates codelets for these actions. If it is an ascending group, for instance, and another ascending group has recently been seen, then a codelet of type "Are These Two Objects Related?" might be created. Section 5.5 discusses in depth the notion of focusing.
129	Are These Two Objects Related?	This codelet is created if two objects appear related, and when run, it tries to discover in what way the two structures are analogous.
130	Attempt Extension of Group	This codelet, given the group, attempts to extend it rightward or leftward. As an example, if the group happens to be "(2 2) (3 3 3)", extending rightward involves checking if four "4"s are present to the right of the three "3"s, and including these in the group if they are there.
131	Create Group	Given a few adjacent objects related to each other in a particular way (for example, the objects could be "2", "3", and "4", where each element is the numerical successor of the previous element), this codelet creates a group consisting of these objects.
132	Attempt Extension of Analogy	Similar to "Attempt Extension of Group", if an analogy is seen between "(2 2)" and "(3 3 3)", extending rightward involves checking if four "4"s are present to the right of the three "3"s, and if they are, a codelet to create the group "(2 2) (3 3 3)(4 4 4 4)" may be created.
133	Read from Workspace	See above.
134	Focus on a Single Element	This is analogous to focusing on a single group.
135	Read from Workspace	See above.
136	Focus on a Single Element	See above.
137	Attempt Extension of Analogy	See above.
138	Are These Two Objects Related?	See above.

Section 4.2 THE CODERACK

When a codelet is created, it is not immediately run. Instead, it is put into a staging area called the Coderack. At any given time, dozens of codelets might be waiting to run. Associated with each such inert codelet is a number between 0 and 100, called its *urgency*. At every time step, one codelet is probabilistically chosen from the Coderack and run. This choice is biased by the urgency, with

higher-urgency codelets being likelier to be chosen. By design, the Coderack has limited capacity, and if adding new codelets causes this capacity to be exceeded, a few of the earlier codelets are expunged (lower-urgency codelets are more likely to be expunged). A codelet may wait for an arbitrarily long time in the Coderack before it is either run or expunged. Table 4.2 shows the same part of the run that Table 4.1 did, but along with each codelet, it also shows at which time step the codelet was created, and how long it had to wait before being run. It is apparent that waiting times can be many time steps long.

Table 4.2 The same part of the run, now shown with creation and wait times

Execution time	Creation time	Duration of wait	Type of Codelet
127	110	17	Read from Workspace
128	126	2	Focus on a Group
129	127	2	Are These Two Objects Related?
130	127	3	Attempt Extension of Group
131	110	21	Create Group
132	82	50	Attempt Extension of Analogy
133	122	11	Read from Workspace
134	132	2	Focus on a Single Element
135	127	8	Read from Workspace
136	134	2	Focus on a Single Element
137	120	17	Attempt Extension of Analogy
138	133	5	Are These Two Objects Related?

Section 4.3 THE "CODELET-TREE"-LEVEL DESCRIPTION

Some types of codelets, when run, create more codelets. As we saw in the rightmost column of Table 4.1, "Focus on a Group" and "Attempt Extension of Analogy" are examples of codelet types that manufacture other codelets. This is the principal way in which new codelets are created — that is, as a result of some other codelet running.

Two exceptions to this rule are the codelet types "Read from Workspace" and "Check Progress". Seqsee periodically creates codelets of these types. These codelets are the neutrons that get the chain reaction going, as it were, since their running results in the generation of more neutrons that create even more neutrons.

Starting with a "Read from Workspace" or a "Check Progress" codelet, we can create its family tree: the codelet will possibly have descendants, which in

turn will possibly have descendants, and so forth. Part of one such tree is shown below. Codelet that were created but expunged before being run have been crossed out (in this particular figure, “Look for Similar Groups” is the only such codelet).

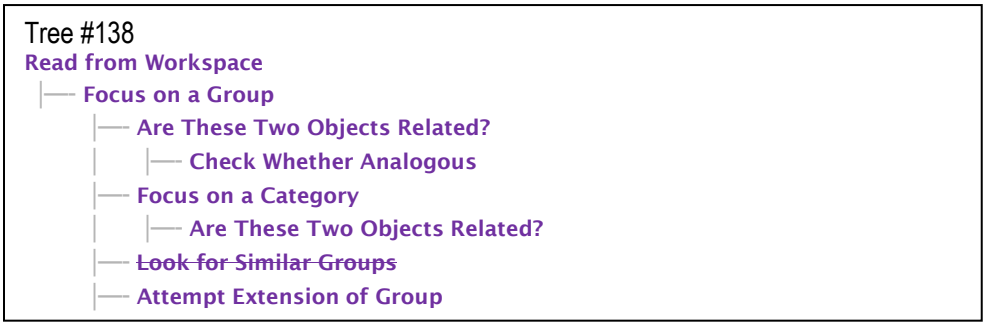


Figure 4.1 A single tree of codelets

The codelet-level description of a run that was seen in Section 4.1 is unsatisfactory, because it gives a disjointed account that does not convey the causal links between codelets. A second level of describing the run consists of listing the codelet-trees that make up a run. This slightly higher-level description brings out the causal structure by making explicit what leads to what, but it, too, gives only a disjointed account and does not make it clear that, as we shall soon see, a few codelet-trees can be active simultaneously and compete for resources.

Recall from Section 4.2 that a codelet may have to wait in the Coderack for many time steps before being chosen to run. This implies that not all codelets in a codelet-tree will run contiguously. Unlike the case of codelets, where each finishes before the next begins, codelet-trees are temporally intertwined. Usually, a few trees of codelets are simultaneously active. Table 4.3 below again shows the same codelet-level view as Table 4.1 does, but now each codelet is annotated with the tree to which it belongs. Note how codelets from tree #24 are intermingled with, for example, codelets from tree #25.

Table 4.3 The same part of the run, annotated with tree numbers

Timestep	Tree#	Type of Codelet
127	#22	Read from Workspace
128	#22	Focus on a Group
129	#22	Are These Two Objects Related?
130	#22	Attempt Extension of Group
131	#13	Create Group
132	#16	Attempt Extension of Analogy
133	#24	Read from Workspace
134	#24	Focus on a Single Element
135	#25	Read from Workspace
136	#25	Focus on a Single Element
137	#17	Attempt Extension of Analogy
138	#24	Are These Two Objects Related?

Since many trees of codelets are active simultaneously, there is an inherent parallelism of exploration. Many avenues are explored at the same time. Each codelet has a certain urgency, and therefore we can calculate the likelihood that a codelet from a particular set would be chosen next. It turns out, unsurprisingly, that the various codelet trees that are currently active are not equally urgent. Some pathways appear more promising to Seqsee, and these are followed more aggressively. Less promising ideas can still be explored, but more slowly and using fewer resources. This is basically the idea of the parallel terraced scan described by Hofstadter (1983; also see 1995).

Section 4.4 THE "PRESSURE"-LEVEL DESCRIPTION

The third level of description is more abstract than the other two presented thus far. In order to motivate this view of the run, imagine that Seqsee is working on some sequence and that some number of codelets have run so far. Consider for a moment this innocuous question: What will Seqsee do next?

The question may seem fuzzy and imprecise: after all, Seqsee's next step is determined probabilistically. Any of a vast number of different things can happen next, and it is impossible to predict which of these possibilities will in fact come to pass. However, the question may be answered precisely if we borrow a standard tool from the computer scientist's theoretical toolbox. The trick is that the answer would not be of the form "this will happen next", but

instead a probability distribution: "with 3% probability this will happen, and with 7.2% that will happen and…" Such a distribution can be calculated accurately and constitutes the most accurate description of Seqsee's immediate next step.

This analysis can be taken further. Not only can we calculate where Seqsee will be a single step later (if only as an unsatisfactory superposition of dozens of possibilities), but we can also calculate where Seqsee will be ten or ten thousand steps later (but again only as a superposition of billions of possibilities). While a full analysis for a thousand steps is prohibitively expensive to undertake, it can be approximated cheaply by running Seqsee multiple times.

It can be observed from such multiple runs that, at a more abstract level of description, Seqsee is in fact far more deterministic than at its lowest level. It tends very regularly, almost predictably, to come up with similar groupings of terms, to fall into similar traps, and so forth. Seqsee is not completely deterministic at this abstract level, of course, but when described at a high level of abstraction, in different runs with the same input, it appears to follow fewer than a dozen different macroscopic pathways rather than the billions of potential paths suggested by having dozens of choices at each of thousands of steps. I can draw an analogy to describing a person's activities in a town: go to school, go to the grocery store, go to the post office, etc. These are of course high-level abstractions of incredibly detailed physical actions, which are always different at a microscopic level. But at a chunked level, many different pathways are perceived as being identical.

Let us return now to providing yet another answer to the post-hoc question "What did Seqsee do in that run?" While the two descriptions thus far told us what Seqsee *actually did* at each step, this third level of description tells us what Seqsee *might have done* at each step, and with what probability. For example, at time step 1000, let's say that Seqsee actually ran a codelet of the family "*Attempt Extension of Group*" — in this particular run. This is what *happened,* but it is only one of many things that *might possibly have happened*.

Other things might have happened — another codelet might have been chosen to run, or even if the same codelet had been chosen, it might have taken a slightly different action, since some types of codelets can probabilistically choose, for instance, where to act. If Seqsee had been run millions of times from exactly that configuration, a wide range of things might have happened next.

4.4.1 "Programming" Seqsee

Seqsee can be modified at two distinct levels: at the "hardware" level (by modifying how the most basic components of Seqsee, such as the Coderack or the Workspace, operate), or at the "software" level (by adding new types of codelets, or by manipulating the urgency values assigned to fresh codelets). For the moment, I wish to pretend that the "hardware" level is inviolate, and by the phrase "programming Seqsee" I will be referring only to modification at the "software" level.

Programming Seqsee is different from programming a traditional computer system. In traditional programming, the programmer specifies exactly what the computer should do next at each step. By contrast, the programmer of Seqsee can only *bias the relative probabilities* of Seqsee taking a particular step next, by making sure that a codelet of low or high urgency gets placed on the Coderack as needed. It may seem a tautology that programming Seqsee is really an exercise in manipulating what Seqsee does, but Seqsee's nondeterminism implies that the programmer is really manipulating the probability distribution. Thus, possibly the most useful view of Seqsee's doings exposes the probability distribution and how it shifts during the run. Such a view may be dubbed *the intention-level view*, since the distribution of Seqsee's likely next move can be considered its intention.

4.4.2 Coderack Demographics

Table 4.4 below shows the demographics of the Coderack at one point during the run. To the right of several codelet families' names are two numbers, one representing how many codelets of that family are currently in the Coderack, and the other the probability that a codelet of this family will be chosen next. From the table, we see that, more likely than not, the next codelet

chosen will be of the family “Attempt Extension of Analogy”. Each of the five codelets of this family is associated with a particular analogy whose extension it will attempt if the codelet is ever run, but the figure does not capture such subtleties and thus tells only a part of the story.

Table 4.4 Types of codelets on the Coderack at one point during a run

	Number of codelets	Percentage of urgency
Are These two Objects Related?	2	17
Read from Workspace	1	7
Attempt Extension of Analogy	5	60
Look for Similar Groups	1	5
Create Group	1	11

From the figure above, it can be seen that, at this stage, the probability of choosing a codelet of the family “Attempt Extension of Group” is zero, as there are no codelets of this family currently on the Coderack. However, this number will change throughout the run. Figure 4.2 shows how the probability of choosing an “Attempt Extension of Group” codelet changes over the course of a run.

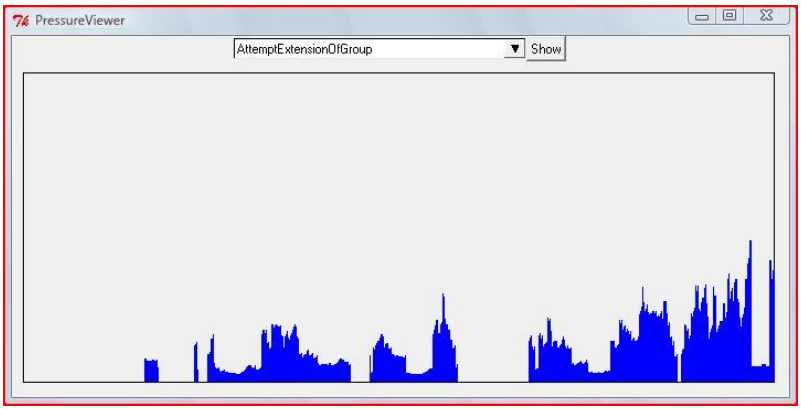


Figure 4.2 Probability of choosing “Attempt Extension of Group”

Looking at Figure 4.2, one may make the following observations. Early in the run, Seqsee felt no pressure to attempt extension of groups. Indeed, there were no groups to extend. One might be tempted to say that group-extending activity picked up later, especially toward the end. However, phrasing it this way is not quite accurate — the graph represents not what *was* done at each step but

how likely this particular action was. A more accurate statement is "The willingness to extend groups picked up later". But I'm being pedantic here — usually, what is likely and what actually happens will be qualitatively similar.

4.4.3 PRESSURE

This section's title, "The pressure-level description", can be explained through an analogy to pressure in physics. Pressure in a gas is due to the activity of myriads of molecules pushing in many different directions. Each molecule's push is minuscule, but jointly they can pack enough punch to move a train (i.e., in the boiler of a steam engine). The situation is analogous here. Each codelet has negligible impact, but working together they get the job of sequence-understanding done. Various codelets push, moreover, in different "directions", and the pressure-level description gives a sense for the amount of push in various directions.

One component of the "direction" that a codelet is pushing in is the type of action the codelet promises to carry out, but there are other components as well. When a codelet of the family "Attempt Extension of Group" is created, it is given the job of extending a *particular* group in the Workspace, and therefore we know what part of the sequence the codelet would act in if it is chosen to run. However, other codelets probabilistically choose the object to operate on, and since we know how they choose, we can again figure out the probability distribution of where such codelets will work. In this way, we can determine another component of pressure: what part of the sequence Seqsee is most likely to operate in next.

Pressure is a very central notion in Seqsee, and I want to ensure that this notion is clear to the reader, and therefore I will spell out the meanings of a couple of phrases I am likely to use. "There is pressure to do X" means that there are codelets on the Coderack that will achieve X if they are run, and "generating pressure to do X" denotes the creation of codelets to do X.

The next three chapters deal with three distinct topics: context, categories, and long-term memory. The theme unifying these chapters is that each describes mechanisms that generate pressure in Seqsee — pressure to do

such things as create groups, label a group as an "ascending group", describe the solution, impose a uniform way of describing different groups, and so forth. Chapter 5 shows how context (as captured using William James' notion of the fringe) can generate specific targeted pressures. Chapter 6 discusses labels, which are an important ingredient in Seqsee's ability to discover analogies between objects. Chapter 7 describes how previous runs on similar problems shove Seqsee toward similar solutions to the current problem. The remainder of this chapter takes a brief look at some of these ideas.

Section 4.5 THE GENESIS OF PRESSURE

As has been just explained, pressure is generated by the addition of codelets to the Coderack. This section discusses various sources of pressure. It describes four distinct mechanisms — analogy, expectation, reminding, and reflex — that push Seqsee in a particular direction, by adding particular codelets to the Coderack. These mechanisms are interdependent, but for ease of description, I will discuss them as if they were independent.

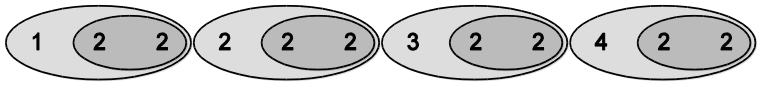
4.5.1 ANALOGY AS A SOURCE OF PRESSURE

In order for Seqsee to understand the sequence "1 2 3 7 7 7 2 3 4 7 7 7", it must realize that the "(1 2 3)" and the "(2 3 4)" groups are *analogous*, and it must act on this realization. If it so happens that Seqsee focuses[12] in quick succession on the groups "(1 2 3)" and "(2 3 4)", it will suspect that the two groups are potentially analogous — by a process based on William James' notion of the fringe of a concept, described in Chapter 5 — and it may create codelets that lead to the creation of a bond between the two groups. Thus, perceived analogies generate pressure that helps the sequence-understanding process.

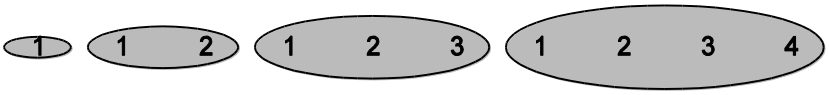
It is not just analogy to some particular *object* that generates pressure, however. In one possible way of solving the sequence below (which we have already discussed in the last two chapters), the long procession of "2"s must be broken, and this eventually takes place as a consequence of a perceived similarity between the many "[2 2]"s that have been seen. In other words, "[2 2]" is a refrain in the sequence, and its constant repetition makes it seem highly

[12] What I mean by "focusing" will be described in Section 5.5.

relevant to Seqsee. Each new "[2 2]" that it notices makes the abstraction "[2 2]" — which is added to the Slipnet if it was not already present in it — stronger, and this in turn makes the other "[2 2]"s in the sequence stronger. When, finally, a couple of "[2 2]"s get carved out of "2 2 2 2 2", they will appear strong not just because of an analogy to some *particular* "[2 2]", but also because of an analogy to the *abstract notion* of "[2 2]".

Sequence 83. 

It is important to note that the notion of what objects are seen as analogous to each other evolves over the course of a run. For example, how similar two groups appear depends on what aspects are currently considered important. To illustrate, consider these two sequences (both of which contain three contiguous elements "1 2 3" and three other contiguous elements "2 3 4"):

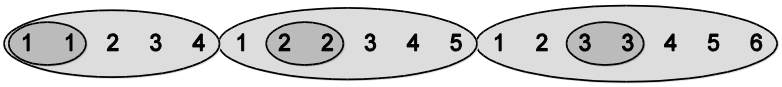
Sequence 84. 

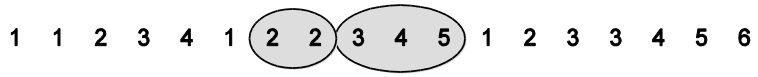
Sequence 85. 

When the program starts, it has seen no structure yet, and if by chance the groups "1 2 3" and "2 3 4" are seen at that stage, their degree of similarity to each other would be the same in one sequence as in the other. However, by the end of the run, in one case *ascending groups starting at "1"* would be important, whereas *ascending groups of length 3* are seen to be more important in the other. Consequently, at that late stage, "1 2 3" and "2 3 4" would not appear to be equally analogous in the two examples, and seeing these two groups in quick succession while solving Sequence 85 will generate a stronger pressure than would be generated by seeing the same groups while solving Sequence 84 (by dint of codelets of higher urgency being produced, for example).

4.5.2 TOP-DOWN EXPECTATION AS A SOURCE OF PRESSURE

Consider Sequence 86 below, which can be understood as a spiced-up version of "(1 2 3 4) (1 2 3 4 5) (1 2 3 4 5 6)", and imagine a situation where Seqsee has just started working on this problem, and has understood only the small subsection shown in Sequence 87.

Sequence 86. 

Sequence 87. 

Seqsee attempts to extend groups that it has seen. It is likely that it will attempt to extend "(3 4 5)" leftward, and it would, naturally, expect to see a "2" there. At this stage, there are two different objects immediately to the left of "(3 4 5)": the pair "(2 2)" and the "2" itself (which is a part of the pair, but no matter). Since what this extension expects is literally present, a very likely possibility is that Seqsee will attempt to construct the plain-vanilla group "(2 3 4 5)". However, for this attempt to succeed, it would need to destroy the "(2 2)" group, but if this group seems strong to Seqsee, "(2 3 4 5)" may well not get created at this time. A second, less likely possibility is that Seqsee will attempt to see the "(2 2)" group as a "2" through squinting. How likely it is to succeed in *this* endeavor is influenced by how many other such squintings it has seen. This example is somewhat blurred as I have had to use phrases such as "likely" and "with a higher probability" more frequently than I would have liked, but the main thrust of my argument has been amply demonstrated: namely, expectations exert pressures affecting what Seqsee looks for and builds. The pressure to squint (i.e., to see something as something else) came from a top-down expectation about what object should be present at a particular location.

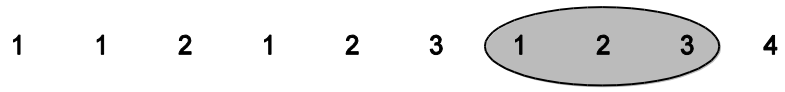
An especially significant type of pressure is generated when the expected terms lie beyond what has been seen so far. When more of the sequence above has been understood, Seqsee expects to see "1 2 3 (4 4) 5 6" next, and pressure is created (in the form of codelets) to ask the person if these are indeed the next few terms.

4.5.3 Reminding as a Source of Pressure

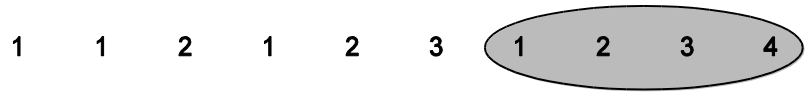
As we saw in Section 3.5, it is possible to run Seqsee in a special mode so that it remembers some aspects of sequences that it has solved earlier. Suppose, for example, that in some run it had seen "1 1 2 2 3 3", and had also noticed that the group "2 2" followed the group "1 1". If it is now presented with the new sequence "5 5 5", although there is a strong pressure just to see the sequence as an endless procession of "5"s, there is also pressure generated by the reminding — by "analogy to past experiences" — to think about "6 6 6" as a possible follow-up.

4.5.4 "Reflex" Pressure

Seqsee takes certain actions reflexively. When looking at a group, for example, it always is under strong pressure to extend it. This makes sense given the way structure within a sequence is discovered. It might happen, for example, that at some point during the run the input has been partially understood:

Sequence 88. 

Extending the group rightward results in a larger structure:

Sequence 89. 

Seqsee has an "innate" desire to extend groups, to extend relations, and so forth, and these innate pressures are evolving and situation-dependent. We will take an in-depth look at such pressures in Section 5.5.2.

Section 4.6 Parting Thoughts

In this chapter, we have seen that a single run is made up of thousands of small codelets, and we have seen different levels at which to describe what was going on in the run. The most important thing to take away from this chapter is the idea of pressure (i.e., what kinds of things Seqsee is likely to do next), and its manifestation in Seqsee (what codelets are waiting to run). This view is the most important for programming Seqsee, and it is also the most important

cognitively if one is willing to look at pressure as “what Seqsee (subconsciously or unconsciously) wants to do next”.

Each of the next three chapters highlights a different component of the creation of pressure, or if you will, of the generation of motivation. We shall see, respectively, how context, categories, and prior experiences guide Seqsee towards a solution.

Chapter 5 CONTEXT AND PRESSURE

This chapter explores the role of context in Seqsee. The idea that context influences our thoughts and actions is an obvious one, and as a quick Google search using the phrase "context influences" reveals, scientific literature is full of examples of the effects of contexts. Effects of many types of context on diverse phenomena have been studied: the visual context's effect on binocular rivalry (Sobel and Blake, 2002), the nutritional context's effect on food choice (for example, sheep and goats select foods with a higher ratio of protein to energy after they consume a high-tannin food item (Villalba, Provenza, and Banner, 2002)), the social context's effect on life decisions (for example, neighborhood socioeconomic status exerts considerable influence on educational aspirations of youth (Sewell and Armer, 1966)).

What I meant by "context" in the previous paragraph was anything (in the environment or in the head) that influences what we perceive, what we assume, what conclusions we jump to, and how we act. This is very general, but we shall henceforth be concerned only with the context created by what is going on in the head — current beliefs and recent thoughts that alter subsequent beliefs and thoughts.

William James observed in *Principles of Psychology* (James, 1890) that desires, goals, and interests color our perception:

> Let four men make a tour in Europe. One will bring home only picturesque impressions — costumes and colors, parks and views and works of architecture, pictures and statues. To another all this will be non-existent; and distances and prices, populations and drainage-arrangements, door — and window — fastenings, and other useful statistics will take their place. A third will give a rich account of the theatres, restaurants, and public balls, and naught beside; whilst the fourth will perhaps have been so wrapped in his own subjective broodings as to tell little more than a few names of places through which he passed. Each has selected, out of the same mass of presented objects, those which suited his private interest, and has made his experience thereby.

One last example I will mention in support of the centrality of context to cognition is Bernard Baars' (1988) cognitive theory of consciousness, which is centered around the notion of context. Chapter 4 of his book is titled "Unconscious Contexts Shape Conscious Experience" and it looks at empirical evidence for contexts and identifies various types of contexts, including those created by firmly held beliefs that make us blind to alternatives (as Albert Einstein (1949, p. 21) claimed happened to nineteenth-century physicists, since they "saw in classical mechanics a firm and final foundation for all physics, yes, indeed for all natural science").

Given this range of examples, it should be no surprise that context plays a major role even in the Seek-Whence domain. What might be slightly surprising is the depth of its involvement.

An important question regarding context is how to represent it. It is plain as day that without representation in *some* form — as activations of neurons, as predicates in first-order logic, as changes made to the environment by means of writing, or in some other fashion — context can have no effect.

We should therefore keep in mind the question of how to represent any of the contexts that we discuss. Representations of contexts, however, need not be extremely complex; they can be parsimonious and still effective. Subsection 5.2.4, for example, looks at the Copycat notion of *temperature*, which plays an important role despite consisting in just a single number.

The remainder of this chapter is structured as follows. It begins by making a small detour into the world of serendipitous discoveries — chance observations that certain keen minds were able to latch on to, resulting in surprising revelations. Section 5.2 affords a close-up view of some contexts tracked by Copycat. Section 5.3 undertakes a discussion of diffuse and specific pressures. Two sections explore two types of contexts: those generated by recent perceptions (Section 5.4) and those generated by the act of labeling (Section 5.6). Section 5.4 also introduces the key notion of *fringes*, first described by William James. Sandwiched between these, Section 5.5 gives a detailed description of what it means for Seqsee to focus on something. Finally,

Section 5.7 ties all these pieces together by showing how fringes can be used to represent perceptual and labeling contexts and thereby can enable the generation of very specific pressures. It will take a good deal of exposition to unpack the phrase in the previous sentence — "fringes can be used to represent perceptual and labeling contexts and thereby can enable the generation of very specific pressures" — but it forms the core of what Seqsee has realized, over and beyond what Copycat and Metacat had already achieved.

Section 5.1 SERENDIPITY

In his book *Happy Accidents: Serendipity in Modern Medical Breakthroughs*, Morton Meyers (2007) presents a litany of examples of accidental discoveries. He says:

> Lithium, Viagra, Lipitor, antidepressants, chemotherapy drugs, penicillin and other antibiotics, Coumadin — all were discovered not because someone set out to find a specific drug that did a specific thing but because someone found something he or she wasn't even looking for. Similarly, the use of surgical gloves, the Pap smear, and catheterization of the heart's arteries leading to bypass surgery were all stumbled upon.
>
> This is the essence of serendipity. […] Discovery requires serendipity. But serendipity is not a chance event alone. It is a process in which a chance event is seized upon by a creative person who chooses to pay attention to the event, unravel its mystery, and find a proper application for it. (Meyers, 2007, p. xii)

This phenomenon is of course not limited to medicine, and plenty of examples from other domains — microwave ovens, X-rays, vulcanized rubber — are presented by many authors (Andel, 1994; Austin, 1978; Cannon, 1940, 1945; Roberts, 1989). These authors insist that "chance is but one element, perhaps the catalyst for creativity in scientific research" (Meyers, 2007, p. 7) and this is also the gist of Louis Pasteur's famous remark: "In the field of observation, chance favors only the prepared mind".

While many of the discoverers whose work is described by Meyers had strokes of good luck, it was hard work that paved the way to their serendipitous discovery. Take the case of Paul Ehrlich, who created *salvarsan*, the first cure for syphilis. This was the 606th chemical compound he had created in his many years of long search for a cure; clearly, with such dogged perseverance Ehrlich

increased his chances of getting "lucky". In this view, hard work is like buying many lottery tickets: it improves the odds of winning the Jackpot. This is similar to the humorous advice proffered to his readers by Indiana University's former president and chancellor Herman B Wells — "Be lucky!" — in his autobiography bearing the same title (H. B. Wells, 1980).

As we shall see later in this chapter, most discoveries Seqsee makes en route to understanding a sequence are based on lucky coincidences, and I will describe in what way Seqsee can be said to possess a "prepared mind". We will also look at how "hard work" by Seqsee virtually guarantees a steady stream of lucky coincidences.

Section 5.2 COPYCAT'S CONTEXT-SENSITIVITY

Before we look at how Seqsee is sensitive to context, I will describe a few ways in which Seqsee's predecessor, Copycat, is context-sensitive. I am aware of at least three types of contexts that Copycat tracks, and taking a close look at these will afford me an opportunity to point out subsequently in precisely what ways Seqsee goes beyond Copycat. My goals here are threefold. I wish to show first how context can influence what is perceived by Copycat; second, how context influences how Copycat responds to what is perceived; and finally, how Copycat's perceptions alter its context, thereby influencing its future perceptions and responses. I have chosen simple examples in order to keep the discussion short, but hopefully they are clear enough to indicate how the same ideas would apply to complex situations.

To keep this discussion concrete, consider these two Copycat challenges:

Copycat Problem 9. If **abc** goes to **abd**, what does **rs** go to?

Copycat Problem 10. If **gwk** goes to **gwl**, what does **rs** go to?

5.2.1 CONTEXT INFLUENCES WHAT IS PERCEIVED

In each case, "**rt**" is a sensible answer, but the path to this solution is different. The difference in the way Copycat reaches that solution hinges on how it perceives the input string "**rs**". This string is common to both problems, but how it is typically seen in each case is different. The string "**rs**" can be seen

as a rightward successor group — since "**s**" is to the right of and is the successor of "**r**" — or as a leftward predecessor group, or as composed of two unrelated letters. The difference between the first two of these interpretations might appear to be mere nitpicking and hairsplitting, but in Copycat's domain, it can be significant.

To illustrate, consider what different answers make sense for the following Copycat challenge and how these answers differ:

Copycat Problem 11. If **abc** goes to **abd**, what does **kji** go to?

The string "**abc**" is a rightward successor (or leftward predecessor) group, whereas "**kji**" is not — it is a rightward predecessor (or leftward successor) group. Thus, if "**kji**" is to be interpreted in the same way as "**abc**" is (after all, we are trying to make an analogy between the two), either rightwardness or successorship must give way, and "**kji**" would be seen either as a rightward group (but one that, unlike "**abc**", happens to be a predecessor group), or as a successor group (but one that, unlike "**abc**", happens to be leftward). In fact, people do indeed see it in these two ways. In a survey involving 10 subjects conducted by Melanie Mitchell (1990, p. 95), two of the answers provided by subjects for this Copycat challenge were

- "**kjh**" (since the original analogy contains the successor group "**abc**" and its last letter is replaced by its successor, replace the last letter of the rightward predecessor group "**kji**" by its predecessor), and
- "**lji**" (by seeing "**kji**" as a leftward successor group, replacing the last letter — the "**k**" — by its successor).

Thus, the two ways of looking at "**kji**" that at first blush are indistinguishable can lead to quite distinct answers. It is a general phenomenon that a seemingly unitary concept can be teased apart into two (or more) concepts. George Lakoff discusses this at length in *Women, Fire, And Dangerous Things*, and points out that what are now two different concepts — *genetic mother* and *gestational mother* — were but a single concept not long ago.

In order to get back to how context influences what relationship is discovered between "**r**" and "**s**", we need to revisit the Copycat component called the Slipnet, which was introduced in the first chapter. As the reader may recall, among its other activities, the Slipnet stores the activations of various concepts. This belief about which concepts are relevant to the given problem and which concepts are irrelevant forms the first type of Copycat context that we will discuss. Figures 5.1 and 5.2 show one part of Copycat's Slipnet during two different runs — on the first and the second Copycat challenges, respectively — and one easily sees how the sizes of some of the corresponding circles differ in the two cases.

The figures show several circles, but not all of these are of relevance to the current discussion — only the circles labeled "left" and "right" (second and third from the left in the figures) are currently of interest. These represent, respectively, how relevant Copycat believes leftward and rightward relations are to the problem that it is working on.

Figure 5.1 A section of Copycat's Slipnet while solving problem 1

Figure 5.2 The same section of Copycat's Slipnet while solving problem 2

Between the two equally valid but slightly different ways of seeing the string "**rs**" — rightward successor group or leftward predecessor group — how does Copycat choose? In the absence of any leftward or rightward bias — as happens in the second problem, where the concepts *left* and *right* are almost equally active (see Figure 5.2 above) — Copycat just picks one or the other randomly. If, instead, there is a rightward bias (as happens in the first run as shown in Figure 5.1), Copycat is more likely to create a rightward relation.

Copycat is more likely to see "**rs**" as a successor relation, therefore, in the case of the first problem than in the second.

I will complete the story of how Copycat solves these two problems only in the next subsection, but I have already completed my first goal. I have shown how context influences — biases — perception. Although this was a very trivial example, the idea is more generally applicable, both to more complex situations that arise in Copycat and in human cognition.

5.2.2 Context Influences Responses to the Perception

Though Copycat is more likely to see a rightward relation in one problem than in the other, it is *possible* for it to see it this way in either problem. Even if it made this same observation in both cases, we shall see in this section that because of the differing contexts of the observation, Copycat responds differently.

I must now draw the reader's attention to three more circles in Figures 5.1 and 5.2 . These are the three rightmost circles with cryptic labels, and they represent Copycat's estimation of the importance of the concepts *successor group*, *predecessor group* and *sameness group* in the given problem. As is obvious from the figures, Copycat considers the notion of *successor group* to be far more relevant in the first problem (understandably so, as it contains "**abc**", and also "**ab**" inside "**abd**") than in the second problem, where successor relations are scant.

The reaction of Copycat to seeing a successor relation in the Workspace depends on the activation of the "successor group" concept. If it is highly activated, as happens in the first problem, Copycat is likely to proceed with "upgrading" that relation into a group. If, on the other hand, the activation is low, Copycat is likely to ignore the newly perceived relationship, considering it to be spurious or insignificant.

The observation that "**s**" is the successor of "**r**" is therefore likely to be ignored in one context and not in the other. I have thus shown one example (albeit a rather simple one) of how context biases Copycat's reaction to an observation, and this completes my second goal.

In the second problem, not only is there no strong bias towards either of the concepts *left* or *right*, none of the three concepts being discussed in this section is very active. There is a fifty-fifty chance that Copycat will consider creating a rightward or a leftward group made up of the two letters "**rs**", but only a small chance that the groups will actually be constructed. And this is as it should be: the initial string "**gwk**" to which an analogy is being made consists of disjointed letters. Copycat comes up with the answer "**rt**" by following the rule "replace rightmost letter by successor". In the first problem, though, "**rs**" will probably be seen as a group, and rightly so, by an analogy to the group "**abc**", and although the same answer is reached, a different road led to it. This difference would have been much more pronounced if the problems had instead been:

Copycat Problem 12. If **abc** goes to **abd**, what does **sr** go to?

Copycat Problem 13. If **gwk** goes to **gwl**, what does **sr** go to?

Here, the first of these two problems is similar to Copycat Problem 11, and either of "**sq**" or "**tr**" are reasonable answers. In the second problem, however, all three input strings will likely be seen as disjointed, and the only reasonable answers are "**ss**" (by replacing the rightmost letter by its successor) or "**sl**" (by replacing the rightmost letter by an "**l**").

5.2.3 Perceptions and Actions Modify Context

The perception that "**s**" is the successor of "**r**" increases the activation of the concept *successor-relation* and also of the concept *right*. Future similar observations are more likely to get a favorable treatment. The more successor groups Copycat sees, the stronger is its tendency to look for and therefore to discover even more such groups. This concludes my third goal.

5.2.4 Other Contexts in Copycat and Metacat

Activation levels of concepts are just one form of context in Copycat. Distances between concepts and temperature are two others.

5.2.4.1 Distance Between Concepts

The distance between two concepts controls how quickly activation spreads from one to the other. If the distance between two concepts is small,

then high activation of one concept quickly translates into both concepts being highly activated; if it is large, activation tends not to spread and only one concept will be strongly activated. This type of context both influences and is influenced by perception. Copycat Problem 14 below will serve well to illustrate this idea.

Copycat Problem 14. If **abc** goes to **abd**, what does **xyz** go to?

The concepts of *predecessor* and *successor* are in general not very close to each other. However, they are related by being each other's antonym. If several *other* pairs of antonyms are seen — if, in that problem, the "a" is seen as rhe *first* letter of the alphabet, and if the "z" is seen as the *last* letter of the alphabet (its antonym), and if one of them is seen as the *leftmost* letter of its string, and the other is seen as its antonym, the *rightmost* letter of the string — then the activation of the concept *antonym* goes up, and the distance between any two concepts in the Slipnet that are related by antonymy goes down. That is, perceptual activity reduces the distance between *predecessor* and *successor*.

In this problem, if the activation of "successor group" goes up because of the presence of "**abc**", the activation of the now nearby concept "predecessor group" will go up, making it more likely for Copycat to see "**xyz**" as a leftward predecessor group — an example of reduced distance influencing perception — finally leading to the answer "**wyz**", which was reported by the subjects in Melanie Mitchell's experiments as the most satisfactory answer.

5.2.4.2 TEMPERATURE

In both Copycat and Metacat, temperature is a number between 0 and 100 that represents how satisfied the program is with its current understanding of the problem. A temperature of zero represents perfect satisfaction. A low temperature indicates that Copycat may be close to a solution, and that the groupings already created are probably appropriate. At high temperatures, on the other hand, Copycat does not have qualms about destroying weak groups.

Being but a single number, temperature is a very succinct representation. It is nonetheless extremely crucial to Copycat. Mitchell (1990) demonstrates how

disabling this aspect of Copycat causes serious degradation in Copycat's abilities. Temperature turns out to be essential in keeping Copycat flexible (by allowing it to destroy groups in certain situations and thus not getting stuck) and yet not trigger-happy (in destroying groups when it nears a solution).

Section 5.3 SPECIFICITY OF PRESSURES

The following is a quote from the H. G. Wells' novel *The Research Magnificent* (1915):

> The need for action became so urgent in him, that he got right out of his bed and sat on the edge of it. Something had to be done at once. He did not know what it was but he felt that there could be no more sleep, no more rest, no dressing nor eating nor going forth before he came to decisions. Christian before his pilgrimage began was not more certain of this need of flight from the life of routine and vanities.
>
> What was to be done?

"Him" refers to William Porphyry Benham, the novel's protagonist, who clearly feels an internal pressure to do something. This pressure is not weak — indeed, it is so strong that "he felt that there could be no more sleep, no more rest, no dressing nor eating nor going forth before he came to decisions". The pressure, however, is diffuse — it does not translate immediately into any particular action. After more thought, what he currently describes as "something had to be done at once, don't know what" will be translated into action.

In contrast to such diffuse pressures, specific pressures will feel refreshingly clear. An anecdote about the mathematician Srinivas Ramanujan has stuck in my mind. Once, Ramanujan was cooking when an elated friend stopped by, wishing to share the solution of a problem that he had seen in a magazine and had solved. When the friend read out the problem, Ramanujan said — still stirring his curry — "please take down the solution", and dictated a continued fraction that gave infinitely many solutions to the problem. On being asked how he had so quickly solved the problem, Ramanujan said, "It was clear that a continued fraction was needed. I just had to figure out which one".

Both diffuse and specific pressures are necessary for cognition, as we will see in the rest of this section.

Our first task is to get the fuzzy notion of "specificity of pressure" into sharper focus. Roughly speaking, a specific pressure is one that will result in one of only a small set of outcomes, whereas a diffuse pressure results in one out of a large number of possible outcomes.

Of course, I do not wish to assign to a given pressure a number indicating its specificity. Such an endeavor to quantify specificity is doomed to failure, as it is pointless to count how many different actions a pressure engenders — as pointless as trying to count how many words the English language has. The problem is that it is unclear when two actions count as being the same. A silly example: in how many ways can you wave to a friend? Infinitely many, differing in how fast you move your hand, at what angle you hold your wrist and so forth. But seen another way, all those different actions are equivalent for most purposes. So long as a precise number indicating the specificity is not sought, however, the description above can give a reasonable qualitative feel for how specific a given pressure is.

An example from chess — a finite game with only a handful of moves at each stage — will help crystallize these ideas concerning specificity of pressures. Consider the chess position shown in Figure 5.3, where white has just pushed the pawn in front of the queen two squares forward. What should black do?

32 different moves are legal at this stage, including silly ones that move the king or the rook. Nobody (other than chess programs) systematically goes through all these possibilities. An intelligent person playing chess for the very first time may think through several moves, but already after playing just a few games will consider only a few moves that appear to have some chance of success. This narrowing-down of choices continues as one's expertise builds. Some statistics are shown below that support this "narrowing-down" hypothesis.

Figure 5.3 What move should black play?

The computer program Toga II — which plays at the level of human grandmasters — was used to analyze the position just shown. For each possible move, Toga can provide its worth in centipawns[13]. The best move (according to Toga) is to take the white pawn with the black pawn, and this move is represented in chess notation as "**exd4**". All other moves are substantially worse (according to Toga) — by at least 50 centipawns. The first two columns in the table below show, respectively, various moves and how much worse they are than the best move (according to Toga). Using a database containing 1 million chess games, I analyzed what moves are actually played in this position. The next three columns show how frequently a particular move is made by players at three different levels of chess sophistication — those with an Elo rating[14] below 1400 (beginners or not-very-good veterans), those rated between 1500 and 1800 (intermediate), and those rated above 2400 (International Masters).

[13] The evaluation of a position in chess is expressed by chess programs in *centipawns*. A score of +100 refers to situation advantageous to white, equivalent to being ahead by one pawn. Likewise, a score of -100 is equally advantageous for black. This evaluation can be used to find the relative worth of various moves.

[14] The Elo rating system is the most widely accepted method for calculating the relative skill levels of chess players, and a player is awarded titles such as "International Master" and "Grandmaster" only on attaining certain specific Elo-rating thresholds.

Table 5.1 Relative worth of various moves in Figure 5.3

Move	Loss in Centipawns (as compared to best move)	% times played		
		Novices or inept veterans	Intermediate	International Masters
exd4	0	**78.0**	**94.2**	**99.5**
Nxd4	51	1.4	0.3	
Nf6	56	2.1	0.2	
d6	65	10.2	3.4	0.2
Bd6	70	1.5	0.3	
Bb4+	94	1.5		
d5	96	1.9	0.4	
f6	126	2.4	0.7	

To an experienced chess player, this board position suggests specific actions, but to a novice, this particular configuration of pieces evokes forces that pull in different directions and a handful of moves compete with each other. In that sense, for a novice, this configuration generates a diffuse pressure. As expertise grows, the pressures generated by the same situation on the board become more specific (that is, it is converted to a narrower range of actions). It is as if the expert can zoom in onto what needs to be done much better than the non-expert can.

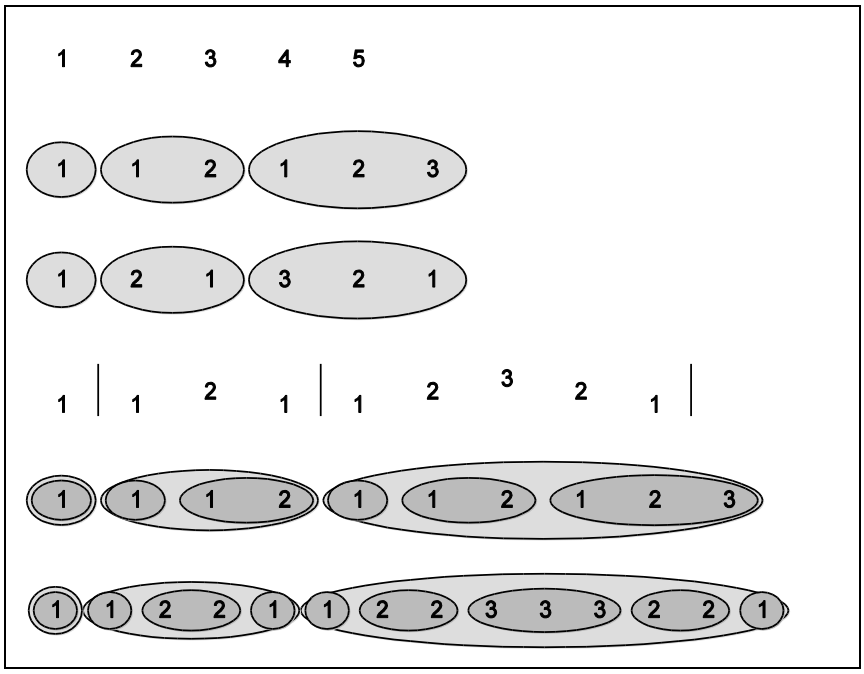
In her analyses of potential improvements to Copycat, Melanie Mitchell (1990, p. 192) expresses the need for specific pressures when she writes:

> One problem seems to be that Copycat's top-down codelets are too global: they are not targeted specifically enough. A Top-down successor-bond codelet will attempt to build a successor bond *anywhere*, but it seems that what is needed here is codelets that try to build specific types of bonds in *specific* places.

A codelet that attempts to build a successor group anywhere in the string represents a spatially diffuse pressure. The pressure represented by this codelet is converted into one action out of a set of many similar potential actions. One way of defining the diffuseness of a pressure is in terms of the range of actions that the pressure can get converted into. In other words, if Copycat were run one million times from exactly the same configuration with exactly the same

codelet poised to run, because of the many stochastic choices made at every stage, the actions taken by that codelet would not be the same in each run.

If a string is long and contains very few successor groups, a codelet that probabilistically chooses the location to look for a successor group has a low likelihood of succeeding. In Seqsee, there are situations where a codelet, if its pressure is diffuse, would have virtually no chance of doing anything useful. As an example, consider the many possible interpretations of a "1" in a sequence. The table below contains six sequences, each starting with a "1". The meaning of this initial "1" is radically different in each case, ranging from the simple "the numeral 1" and "a degenerate case of an ascending group" to the esoteric "a degenerate case of a top-heavy mountain" (see the last sequence to know what I mean by that phrase). It is very easy to create sequences where the lowly "1" needs to be interpreted in even zanier ways.



In Copycat, a solitary "**c**" may be interpreted in two ways — as a letter, or as a group of length 1. The group of length 1 can itself be a successor group, a predecessor group, or a sameness group. Even just these few interpretations already cause difficulties, summarized by Mitchell (1990, p. 192):

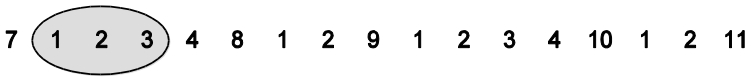
> [T]he program is currently too willing to build single-letter groups without sufficient pressure. It seems more plausible that constructing such an unusual group should be done in response to a strong *location-specific* pressure […], rather than (as is currently the case) in response to more general pressure from other groups in the string.

Seqsee's challenge in interpreting single elements is more difficult than in the case of Copycat. Seqsee must *absolutely never* try on for size the interpretation "a degenerate top-heavy mountain" without a very strong reason. In the last sequence above, such an interpretation would apply only to one specific instance of a "1" in it and not to any of the four other "1"s in it, none of which cry out at all for such an unusual interpretation. Notice that the pressure generated will have to be not just specific spatially (it should say "look at *this* location"), but also specific in other ways ("look for this *particular* kind of object at that location"). Seqsee can indeed generate very specific pressures, as we will see in the last section of this chapter.

I do not wish to imply from the discussion above that specific pressures are the only desirable types of pressures. At the beginning of a run, the sequence has not been understood at all, and Seqsee needs to be fluid in interpreting what it sees. A very specific pressure at this stage would be tantamount to a strong *a priori* belief about what the rule behind the sequence is. As the understanding of the sequence deepens, more specific expectations about the sequence should result in more specific pressures. The initial pressure in Seqsee can be translated into English roughly as "look for local islands of meaning within the sequence" without specifying what sort of islands or where. As the run progresses, more specific types of structures will be actively sought in more specific locations.

Although diffuse pressures predominate at the beginning of a run, after even tiny sections of the sequence have begun to make sense, *local* specific pressures can play a role. Take, for example, a sequence such as the one shown below, where Seqsee's limited current understanding is indicated by the solitary oval. No *global* specific pressures make sense at this early stage of

understanding, but a *local* pressure to seek "4" to the right of the oval is still appropriate.

Sequence 90. 

I rest my case that both specific and diffuse pressures are indispensable.

Section 5.4 PERCEPTUAL CONTEXT

Of the vast array of types of contexts, let's zoom in on one. This is the context made up of recent perceptions and thoughts. Consider the following sentence from *The Joke and its Relation to the Unconscious* by Sigmund Freud (2003, p. 88), and think about how each word strikes us as we read it:

> Experience consists of experiencing what we wish we had not experienced.

When we read the first occurrence of the word "experience" — that is, before the rest of the sentence has been read — there is nothing unusual to notice. Many perfectly ordinary sentences start out with this word. What happens, though, is that every word slightly alters the context that subsequent words are read in, so that when we read "experiencing" we cannot help but feel that we have just encountered this word, and perhaps we may ask ourselves if its meaning differs from that of "experience", and whether the author of the sentence is being playful or merely sloppy, and so forth. The recently-read word "experience" alters our experience of the just-read word "experiencing" (and I am sure that my readers asked themselves if, when I used the word again, I was being sloppy or amateurish or cutesy or laying it on too thick).

In the Freudian sentence, by the time we reach "experienced", the deck is stacked against our understanding this as an ordinary, vanilla, run-of-the-mill instance of that word. Its feel is very different from the feel of the same word in sentences such as "we experienced some unexpected heavy traffic" or "John McCain is experienced". Indeed, all theories that attempt to explain how people can deal with polysemous words tip their hats to context. In a dictionary, polysemous words such as *bank* — which can be a river bank or a financial

institution — have different subentries, and context must be used to disambiguate them, as was just done by using the phrases “river bank” and “financial institution”. I am ascribing a little more influence to context here by saying that even if a word is used in the *same* sense in two sentences — that is, both uses fall under the purview of the same subentry in the dictionary — the context can cause the two occurrences to feel different.

The three words — “experience”, “experiencing”, and “experienced” — are almost identical, and this explains how reading one slightly alters the perception of others, but much less than blatant repetition is necessary for this effect. Consider this sentence attributed to Pascal, quoted in Esar (1978, p. 808):

> If Cleopatra’s nose had been shorter, the face of the whole world would have been changed.

Here, the word “face” reminds us — I hope the reader will allow me to use the verb *remind* to refer to events less than a second ago — of the word “nose” that was just read, and some effort is channeled to understanding how the two are related in this sentence. Esar (1978, p. 808) gives several more sentences of this sort where a word harks back to a just-used word. Just using colors:

> We may ridicule romance as something made up of red roses and white lies; advise motorists to keep their eyes on two things: red lights and green drivers; mock girls who prefer men with blue eyes and greenbacks.

What is going on here is that reading a word (or a phrase, or a sentence, or an idea) makes us sensitive — if only momentarily — to *related* words (or phrases, sentences, ideas). What we do when we notice (or feel) similarity depends on what types of things we notice relationships between. If similarity is perceived between two problems, for instance, we may attempt to adapt the solution of one to solve the other. If similarity is perceived between a problem we have been struggling with (take, for example, the case of Alfred Nobel struggling to find something to mix with nitroglycerine to decrease unintended explosions) and a chance observation (as happened when Nobel dropped some nitroglycerin by mistake on sawdust and the feared explosion did not happen),

we may realize that the observation might somehow help us solve the problem and follow the lead. If, on the other hand, similarity is perceived between a failed attempt at solving some problem and another recent failed attempt, we may attempt something radically different.

Since every word that we read temporarily alters the context, and since it is impossible to know in advance what related ideas might ring similar, how to represent perceptual context in a computer program is a big challenge. A representation of perceptual context must be able to explain how similarity can be perceived — between two problems, between a problem and possible solutions, between two obstacles hit while solving a problem.

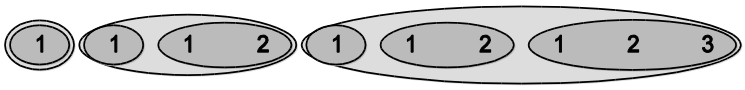
This is precisely where William James' notion of the *fringe* enters the picture. In his *Principles of Psychology*, James (1890) described the fringe as a *penumbra* of vague experiences, sometimes using the terms *psychic overtones* or *suffusion*. Consider the concept of *Vietnam*. When that concept is evoked, it does not come alone. It brings along with it, to various degrees, such concepts as war, Agent Orange, and Iraq; or perhaps the beautiful beaches and the brilliant green rice paddies and women in conical hats; or perhaps just idle facts like "its capital is Hanoi" and "it's near China". To each person the package that is Vietnam is differently filled, but for most, it is not empty. If I next mention another place, it would appear more or less similar, depending on how, in the reader's mind, its package overlaps with the reader's version of the Vietnam package. Florida with its beaches, China with its rice paddies, Kosovo with its war, or even New York with its "I enjoyed my vacation there" may appear similar.

Concepts in Seqsee also have such fringes ("packages") of related concepts. If Seqsee has noticed that, by squinting, the group "(1 1) 2 3" can be seen as the ascending group "1 2 3", then the fringe of "(1 1) 2 3" will contain, among other things, the concept *ascending group*, the concept *blemished version of an ascending group*, and the concept "1 2 3"-*group*. In this context, the group "4 5 6 (7 7) 8", being both ascending and blemished-ascending, will appear more similar to this group than the merely ascending "5 6 7" does.

Fringes — in people or in Seqsee — are not black-and-white. The fringe of the concept *Vietnam* does not contain in equal degrees all the concepts just mentioned — some are prominently present, while others can only be discerned faintly. In Seqsee, the fringe of a concept is a weighted set (i.e., to each element of the fringe is attached a number between 0 and 1 indicating the degree to which this element is in the fringe). Moreover, what the fringe of a concept contains changes based on the context.

In Seqsee, the fringe of a group in the Workspace (say, a "(1)") includes nodes in the Slipnet that are near the Slipnet node "1" as well as the categories that this particular group has been seen as belonging to. A consequence of including currently perceived categories as part of the fringe is that two different "1"s will not necessarily have identical fringes. For instance, in the following now-familiar sequence, the fringe of the initial "1" will be quite different from the fringes of the other "1"s.

Sequence 91.



This description of the fringe is a slight simplification. Seqsee contains a subroutine to calculate the fringe, and it works differently for groups and for bonds. For groups, the subroutine works precisely as was just described. For bonds, the fringe includes the two objects that the bond connects, as well as the type of the bond (e.g., *successor*).

When the fringes of two objects overlap, it suggests that the two are similar. Moreover, the greater the overlap, the greater is the potential similarity. The fringes of the two groups "1 2 3" and "5 6 7" may both contain *ascending of length three*, thereby making them appear similar. Similarly, the fringes of the two bonds between the three contiguous groups "5 6", "5 6 7", and "5 6 7 8" — that is, the fringe of the bond between the first two of these groups and the fringe of the bond between the last two — will overlap quite a bit, since the middle group is common to both fringes, as is the type of the bond.[15] By contrast, the fringes of the bonds between the three contiguous groups "5 6",

[15] The notion of the type of a bond is explored in the next chapter.

"5 6 7", and "5 6 7" will overlap somewhat less, since the types of bonds are different.

Section 5.5 FOCUSING ON AN OBJECT

This section describes what Seqsee is doing when I describe it as *focusing* on an object. I used the word "object" in a very general way, including groups, analogies, and even categories under that umbrella term.

Let us first look at how codelets of types such as "Focus on Group" and "Focus on Analogy" are created. They are most commonly created by a "Read from Workspace" codelet. When this codelet is run, it chooses an object from the Workspace randomly, but in a manner biased by how *important* Seqsee considers that object to be. It then creates a codelet to focus on that object; this section will describe what form such focusing takes. A second source of focusing is other codelets, such as "Create Group" — codelets of this family create a group and add to the Coderack a codelet to focus on the newly created group.

Recall from Chapter 4 that Seqsee periodically creates "Read from Workspace" codelets, and since each such codelet leads to focusing on some object, focusing constitutes a large part of all work done by Seqsee.

5.5.1 ONE COMPONENT OF FOCUSING ON AN OBJECT

When focusing on an object, Seqsee calculates the fringe of this object. Seqsee also stores the fringes of a few[16] objects recently focused on. The fringe of the object currently under focus may or may not overlap with stored fringes, but if it does, this suggests possible actions to take — that is, it suggests specific codelets to add to the Coderack.

If the fringe-overlap is between two groups, a codelet is created to check if they are analogous. If the fringe-overlap is between a bond connecting some object A to some object B and another bond connecting B to some object C, a codelet is created to investigate if a group can be created using A, B, and C.

[16] Currently, 10. It is simplistic to remember a fixed number of prior fringes — events don't fade from memory uniformly, and there should be a way in Seqsee to remember the fringe of an older salient event even after more recent (but unremarkable) fringes have faded away.

In the current version of Seqsee, these are the only types of fringe-overlap that have any effect. In subsequent versions, more general types of overlap will be handled — for example, between a problem (such as "I want to find an ascending group in this small part of the sequence") and a solution (for instance, stumbling upon such a group).

Every object focused on influences how Seqsee responds to subsequent objects. If a "7" is seen, this observation changes what Seqsee does if it sees another "7" or another object that is similar in the sense that their fringes overlap. This opens up rich opportunities for serendipity to play a role — as Seqsee's attention jumps from object to object, it is virtually guaranteed that hundreds of episodes will occur where two similar objects are focused upon, one soon after the other.

Such episodes get the ball rolling, since these are likely to result in the discovery of analogies, which leads to the construction of groups that can then generate specific pressures that lead to the understanding of the entire sequence.

An important point to note is that the strength of stored fringes decays over time, so that more recently observed objects are likelier to appear similar.

5.5.2 The Other Component of Focusing on an Object

Similar to the notion of the fringe and superficially related to the idea of an *affordance* is the concept of *action-fringe*, which contains potential actions. The following quote from Donald Norman's *The Design of Everyday Things* (2002, p. 9) describes affordances:

> The term affordance refers to the perceived and actual properties of the thing, primarily those fundamental properties that determine just how the thing could possibly be used […].
>
> Affordances provide strong clues for the operations of things. Plates are for pushing. Knobs are for turning. Slots are for inserting things into. Balls are for throwing or bouncing.

All manner of objects — both physical entities and mental constructs — have affordances (here used in the limited sense of "possible actions suggested by the object"). A humorous example is suggested by Mark Twain's advice to

writers, “If you find an adjective, kill it”, and another by Jeremy Silman’s advice to chess players (1998, p. 4): “The existence of an open center should be likened to a neon sign saying, ‘Castle immediately! Castle immediately!’” Elsewhere, Silman writes (1999, p. 1):

> A player can’t do [just] anything he wishes to do. For example, if you love to attack, you can’t go after the enemy king in any and all situations. Instead, you have to learn to read the board and obey its dictates. If the board wants you to attack the king, then attack it. If the board wants you to play in a quiet positional vein, then you must follow that advice to the letter.

To make this long story short, objects read by Seqsee suggest possible actions. Groups suggest that they should be extended, large groups suggest that the search for the solution may be nearing an end, and large groups that cannot be extended all the way leftwards to the beginning of the sequence may suggest that the sequence is blemished at the beginning (for example, it may be a sequence such as “17 1 2 3 4 …”).

As we saw in the previous section, part of the fringe of an object includes nodes that are near it in the long-term memory. The action-fringe of an object, similarly, is influenced by the nearby contents of long-term memory. For example, the action-fringe of a “1 2” group can come to include looking for a “1 2 3” to the right, if sequences where such relations were abundant have been seen earlier. Similarly, if a “1 2 3 2 1” group is seen and labeled as a *mountain*, Seqsee will be quicker in labeling as mountains any “1 2 3 2 1” groups that it encounters in subsequent runs, and therefore the label “mountain” will become a part of the fringe of those groups more quickly.

The second component of focusing is the creation of codelets suggested by the action-fringe.

Section 5.6 Labels and Context

We routinely apply labels to objects, people, and events. For instance, we may label a book as being a “big-buzz debut novel”, a “mediocre detective novel”, or even a “too-ambitious second novel”. How the mind produces labels is relevant — perhaps central — to this dissertation, and it is discussed in Chapter

6. In this section, however, I am concerned only with the effects of labels on subsequent thoughts. In short, I am interested in exploring how labels form a context.

In his book *Predictably Irrational*, in the chapter "The cost of zero cost", Dan Ariely (2008) gives a nice example of the effect of labels. He demonstrates how the label "free" can make us behave irrationally. On Halloween night, to half the kids who came trick-or-treating, he gave three small chocolates (each weighing 0.16 ounces), with the option of upgrading one small chocolate to a one-ounce Snickers bar, or exchanging two small chocolates for a two-ounce Snickers bar. The logical thing to do, in order to maximize the amount of chocolate gained, is to take the second offer, and this is what almost all the kids did. The second set of kids were again given three small chocolates, with the option of getting the one-ounce Snickers bar for free, or exchanging one small chocolate for the larger Snickers bar. Again, the logical move is to take the second offer, but a large number of kids were thrown off by the word "free" and chose the first option instead.

As another example, the following is a plea before the United States Senate Special Committee on Aging to disallow the labeling of something in a particular way because of the way it influences perception (Baratz, 2001).

> Somehow calling something a "traditional Chinese medicine" implies it has been in use for centuries and, because of that name alone, is somehow valid, safe, and effective. The truth lies elsewhere. Let us not mince words. These substances are being promoted as drugs, in the common everyday usage of that term, clever language, creative terminology and nosological acrobatics, notwithstanding.

Each of these examples demonstrates that choice of words has an effect on subsequent thoughts. This is hardly surprising — *of course* labeling influences perception. If I believe that a certain dog is aggressive, I will tend to avoid it.

To pave the way to describing how Seqsee uses labels, I present a few more examples of labeling below. There is a caveat to keep in mind, though. Labels have a sociological component that makes them hard to discuss. When

somebody labels something in a particular way (for instance, Sarah Palin labeling Barack Obama as somebody who "pals with terrorists"), one cannot always take that labeling at face value. All sorts of social complexities enter the picture, and it may or may not be a description that the person who is doing the labeling believes to be accurate. This is unfortunate, since I want to ponder the effects of labels on cognition, and grains of salt attached to such labels would require that the reader's ruminations about the potential effects be filled with hedges. I would urge the reader, therefore, to momentarily brush such complexities aside and simply to try to step into the shoes of someone who actually believes the labeling statement, and to think about what thoughts and actions the label engenders.

To the examples, then. We label not just objects but also entire situations. The following have been collected from Google by searching for phrases such as "this is a situation that *" and seeing what occupied the place of the *:

- This is a situation that requires rectification.
- This is a situation that we cannot afford to repeat.
- This is a manageable situation.
- This is a win-win situation.

Such labels influence how we act, since our response to a situation requiring rectification is different from our response to a win-win situation. The descriptions above may feel generic and vague, and the pressures that they generate correspondingly diffuse, so consider more specific characterizations gathered from the same source:

- This is clearly a case of a large corporation trying to squash competition.
- This is clearly a case of the scientific establishment elite protecting its own.
- This is clearly a case of "Denial of Due Process".
- This is clearly a case of putting the fox in charge of the chicken coop.

- This is clearly a battle for the future of the Republican Party.
- This is a situation that calls for genital mutilation to be performed on all 3 officers involved.

If how we describe things has such a significant effect, any successful cognitive model would need to account for this. If people react differently to "a win-win situation" and to "a situation requiring rectification", so too must Seqsee (for some rough analogue of these labels in its domain). Seqsee has no notion, of course, of "a win-win situation", but there are situations that it may encounter during a run where it seems to be making good progress and believes that it is close to a solution. "A situation requiring rectification" is likewise missing from Seqsee, but there are situations when it is spinning its wheels, continually getting stuck in the same spots, and recognizes the impasse that it is in.

Labels in Seqsee come in two flavors. The first flavor applies to objects such as groups and analogies between groups, and is thus local. Examples of such labels include "ascending group" and "mountain". The second flavor describes global aspects of the entire sequence or of the process of understanding the sequence — for example, that the sequence is an interlacing of two sequences, that the sequence involves squinting, that the sequence is initially blemished, that Seqsee believes that it has understood the sequence, or that Seqsee believes it is making no progress. I will use the terms "object label" and "situation label" for these two types of labels.

Object labels "warp" similarity: two objects that carry the same label appear more similar to Seqsee for that reason. The effect of a perceived similarity depends on how relevant Seqsee currently considers the shared label to be.[17] As the perceived importance of some label goes up, objects that have been given this label appear to Seqsee to be more similar to each other. Section 5.7 shows how similarity generates specific pressures.

As I have pointed out, Seqsee applies such labels only in the presence of pressure to do so. One source of such pressure comes from the existence of

[17] Chapter 6 describes how the perceived relevance of categories changes over time.

other groups in the sequence that have the same label. A second source of pressure to apply some types of labels comes from a mounting sense that objects deserving such labels are present. The presence of mountains (that is, sequence fragments such as "3 4 5 6 5 4 3") is signaled by the presence of upslopes (for example, "3 4 5 6") overlapping on the right with a downslope (such as "6 5 4 3"). A solitary example of such overlap may well be spurious, but when many instances are seen, the concept *mountain* is likely to be a key in the understanding of the sequence. These ideas are more fully discussed in Chapter 6.

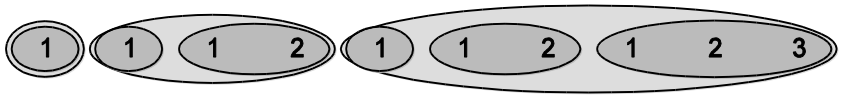
Situation labels, on the other hand, are implemented as codelet types. One type of situation that Seqsee encounters in every successful run is the presence of a very large group: that is, a large chunk of the sequence, generally covering more than half the known terms, makes sense to Seqsee. The appropriate response to noticing such a chunk is to try and convert it into an entire solution, which means predicting the next few terms by extending this group and completing the problem-solving session by describing the solution. A single codelet family, by adding to the Coderack the right sorts of codelets, generates the appropriate set of pressures for this set of actions.

Section 5.7 How the Fringe Generates Specific Pressures

It is now time to see how fringes and action-fringes together provide a rich perceptual context and enable the generation of specific pressures. We will use a suitably complex sequence — Sequence 92 — to describe these effects. Below each number in the sequence is a letter between *a* and *j*, which will make it easy to pinpoint specific parts of the sequence. Sequence 93 shows the same sequence in the notation we have been using throughout.

Sequence 92.

Sequence 93.



I will use a hypothetical run of Seqsee on this sequence in order to describe the effects of the fringe and the action-fringe. I will "pause" the "run" at a few strategic places and describe the state Seqsee is in (i.e., groups it has seen, the codelet that is just about to run, and so forth) and the imminent likely actions (i.e., the pressures).

5.7.1 THE FIRST STOP

Our first stop is shown below in Sequence 94, where two groups have been seen.

Sequence 94.

When Seqsee was frozen in its tracks in this hypothetical run, it was just about to run a "Focus on Group" codelet that, at creation time[18], was told to focus on the "$(1_h\ 2_i)$" group.

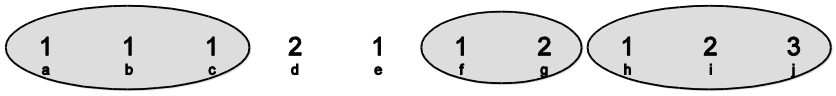
As we saw in Section 5.5, focusing has two components — one based on fringe overlap, and the other based on the action-fringe. At this early stage in the run, the fringe of the group under consideration does not significantly overlap any recent fringes, so only the second component of focusing has any effect. The action-fringe of this group — or affordance, or set of suggested actions, or whatever else you want to call this — includes the creation of a codelet to attempt to extend this group. Thus, the result of running the "Focus on Group" codelet is the creation of an "Attempt Extension of Group" codelet. Put differently, this codelet generates pressure to extend this group. If this newly created codelet is run, it might try to extend the group rightward and thereby create the "$(1_h\ 2_i\ 3_j)$" group, or it might try extending leftward and fail to achieve anything.

5.7.2 THE SECOND STOP

At our second stop, Seqsee has seen one more ascending group, as shown in Sequence 95.

[18] Recall that codelets of this family are usually created by Read-from-Workspace codelets. A Read-from-Workspace codelet probabilistically chooses some object to focus on, and creates a codelet to make that happen.

Sequence 95.

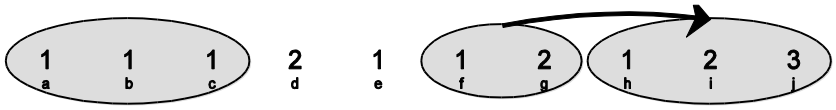


It so happens that Seqsee had recently focused on the ascending group "(1_h 2_i 3_j)". Since the fringe of this group contains the category *ascending*, Seqsee's ears are perked up for any other ascending groups. Said differently, the presence of the label "ascending group" on a recently seen group modifies the context in a way that changes Seqsee's response to subsequent sightings of other ascending groups. The fringe of any other ascending group will overlap that of "(1_h 2_i 3_j)", and so the two will appear analogous to Seqsee. If the concept *ascending group* is highly active, the similarity will appear even stronger. If the focus happened to fall on "(1_f 2_g)" next, then Seqsee would be reminded of the recent similar group "(1_h 2_i 3_j)", and would add a codelet to explore how the two structures are analogous.

5.7.3 THE THIRD STOP

At our third stop, Seqsee has seen how "(1_h 2_i 3_j)" is analogous to "(1_f 2_g)". This analogy is based on both being ascending groups. Extending this analogy rightward would be like solving "If '1 2' goes to '1 2 3', what does '1 2 3' go to?"

Sequence 96.



Extending the analogy leftward, similarly, amounts to attempting to answer the query "If '1 2 3' goes to '1 2', what does '1 2' go to?" It so happens that, in this hypothetical run, the codelet selected to run is a "focus on analogy" codelet with the mandate to focus on this analogy. The action-fringe of an analogy between groups includes trying to extend the analogy leftwards or rightwards. Extending the analogy rightwards would result in Seqsee seeking the ascending group "(1 2 3 4)" to the right of "(1_h 2_i 3_j)". Since this goes beyond the known elements, it would result in pressure to ask the question "Are the next four terms 1, 2, 3 and 4?" Extending leftwards would likewise cause Seqsee to seek the ascending group "1" to the left of "(1_f 2_g)", and it would succeed in this

endeavor, thereby perceiving "1_e" as a (degenerate) example of an ascending group.

5.7.4 Specificity of Generated Pressures

I would like to draw the reader's attention to how specific the pressures generated in these three cases have been. In the first case, Seqsee was looking for a "3" in a particular location, and in the third, it was looking for a "1 as an ascending group" in a specific location. The second case is slightly different: Seqsee was looking for the relationship between two particular groups, but these groups had a high likelihood of being related by virtue of belonging to a common category.

Contrast this with how Copycat works. Copycat has the codelet family called *group scout*, which works as follows. The codelet takes a specific category as argument. A random object is chosen (probabilistically), and the input is scanned for a potential group of that category that starts or ends at the chosen object (Mitchell, 1990, p. 275). Copycat runs this codelet only when there appears to be a good chance that objects of that category exist *somewhere*. However, there is no specific reason to believe that an instance of the category would exist at a *randomly* chosen location. Consequently, most attempts to look for these specific types of groups are bound to fail. This approach is problematic because of the wasted effort, and because it is not scalable. Copycat has exactly three categories for groups (namely, *successor*, *predecessor*, and *sameness*) and input strings tend to be short, usually being between three and eight characters in length, so this method of searching is perhaps adequate there. Seqsee, by contrast, with its larger repertoire of categories and its far longer sequences, cannot afford such unmotivated guesswork, unless it is based on hunches that have a reasonable chance of panning out.

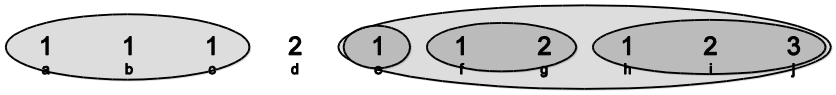
A very similar difference exists between Seqsee and Copycat in how relations are formed between objects. Here, too, Seqsee is more focused than Copycat is. Relations are called "bonds" in Copycat, and one of the codelet families responsible for bond formation is the *bond scout* (Mitchell, 1990, p. 273). This codelet works by choosing an object probabilistically, choosing an adjacent object, and seeing if they are related. This limits relations to being

between *adjacent* objects. This approach cannot be generalized to also find long-distance relations. To illustrate, consider the letter string "abcdefgh…z", and the naïve generalization of the algorithm that would choose two objects at random and would then check their relatedness. The probability of success of this codelet for this string can be calculated[19], and it turns out to be about 7%. If the naïveté of my generalization is decreased by biasing the random choice to choose nearby objects with greater likelihood, the codelet works better for this string, but less well for strings in which there are long-distance relations. Seqsee, on the other hand, checks the relatedness of two objects only if they stand a chance of being related, as suggested by their overlapping fringes.

5.7.5 THE FOURTH STOP

Let's look at one more point in the run, which occurs in our hypothetical run soon after the point we just looked at. It will be very useful to look at this, as it will illustrate shortcomings in (the current version of) Seqsee.

Sequence 97.



It would have been nice if at this juncture Seqsee felt the pressure to look for "((1) (1 2))" to the left of the large group. However, unless the long-term memory is turned on, and unless similar sequences have been seen before, Seqsee is incapable of imagining this group at that location. In other words, Seqsee does not innately know that groups such as "((1) (1 2) (1 2 3))" are often preceded by "((1) (1 2))". It would be easy enough to enhance Seqsee with this particular ability, but that misses the point — it would simply be a small patch on a very major, general problem.

Section 5.8 PARTING THOUGHTS

In this chapter, we have seen the positive role that context can play in directing Seqsee towards a solution, and we have also seen a compact yet flexible method of representing context. This method is easy to extend, and Seqsee barely scratches the surface of the uses that it can be put to — for

[19]For this specific sequence, this is equal to the probability that two randomly chosen objects are adjacent.

example, I have not even implemented “goal contexts”, which play a starring role in other systems such as Baars’ Cognitive Theory of Consciousness.

In the next version of Seqsee, on which I expect to begin to work soon after finishing my Ph.D., I expect to entrust contexts with greater responsibilities. One particularly fertile area of investigation would be to see how fringes can be used for self-watching. At the very least, the fringes of two different events in which Seqsee gets stuck “in the same place” could have a large overlap — even for a liberal interpretation of “in the same place”. In its current incarnation, Seqsee does not watch itself at all, and in that sense it is a step backward from Metacat, which can occasionally spot itself repeating the same futile maneuvers and can take corrective action. Implementing self-watching using fringes will be a robust addition to Seqsee.

Chapter 6 CATEGORIES IN SEQSEE

In this chapter, I describe the categories in Seqsee — labels that Seqsee can attach to objects. I fully agree with George Lakoff (1987, p. 5) when he says that "categorization is not a matter to be taken lightly. There is nothing more basic than categorization to our thought, perception, action, and speech", and an important goal of this project has been to endow Seqsee with a rich and flexible set of categories.

I already mentioned in the previous chapter that labels play an important role in how Seqsee realizes that two objects are similar. What I have not yet spelled out is the role of categories (or labels — I use the two terms interchangeably here) in describing in precisely what ways two objects are related.

Consider how one might describe the difference between "1 2 3 2 1" and "1 2 3 4 3 2 1". A succinct description would surely depend on noticing the symmetrical mountain-like structure shared by the two. We can describe the change from the former to the latter as "the peak becomes taller". There are a few points worth observing there. First, notice how the difference between the two sequences is expressed in terms of their similarities (both are *mountains*). This is true about analogies in general, and this is why it is impossible to come up with anything but contrived answers to riddles like "What is the difference between a cat and a comma?", the answer to which is that one has claws at the end of its paws, and the other is a pause at the end of a clause. This answer justifies the claim that differences are understood in terms of similarities, as two disparate things (cats and commas) were differentiated only after the artificial imposition of a type of similarity (both can be described using the word *end* and homonyms of the words *clause* and *pause*).

The second thing to note is that we were able to talk of *peaks* only because we were able to cast the sequences as *mountains*. Imposing a category onto some object allows its description in terms that were simply unavailable before. A stark example is the occasional description of the star Aldebaran as

"the eye" of the constellation Taurus. Such a fanciful use of the term *eye* is possible only because the constellation can be considered an instance of the category "bull", however tenuously. Refusing to see an animal there makes it impossible to see Aldebaran as an eye.

Notwithstanding the fact that "taurus" literally means *bull* in Latin, it is of course very far from a prototypical bull. In describing a "real" bull, we may describe many of its parts besides its eyes, but most of those ways of describing do not carry over to such a fringe instance of the *bull* category as the constellation Taurus. On the flip side, Taurus is an instance of other categories besides "bull" (such as *constellation* and *sun sign*), and these allow for other ways to describe Taurus that have nothing to do with bulls, such as "having Aldebaran as its brightest star".

To cut a long story short, categories in Seqsee, once awakened, provide extra words to describe aspects of their instances. To illustrate, consider the category *sameness group*. Instances of this category include "5 5 5" and "(1 9) (1 9) (1 9) (1 9)". The terms that this category supplies for descriptions of its instances include *length* and *each item*. One can thus say that each item in the sameness group "(1 9) (1 9) (1 9) (1 9)" is "(1 9)". This also gives us a simple way to describe relations. The relation of this group to the group "(2 8) (2 8) (2 8)" can be described in terms of how the *length* changes (it decreases) and how *each item* changes.

In the goal of implementing a rich and supple system of categories, I have been moderately successful, but there are important aspects of categories that the current version of Seqsee lacks, and it is these that we will look at next (Section 6.1); then we move on to discuss what categories Seqsee does possess (Sections 6.2through 6.5), and, armed with that information, we return to a discussion of how the missing categories might be added to Seqsee (Sections 6.6 and 6.7). There are of course many other categories that Seqsee is incapable of handling without significant overhaul, and we will have occasion to discuss some of these along the way.

Section 6.1 CATEGORIES MISSING FROM SEQSEE

I begin the discussion of categories in Seqsee in an unusual way, starting with deficiencies in what Seqsee currently does. Consider the following sequence, which the current version of Seqsee does not understand:

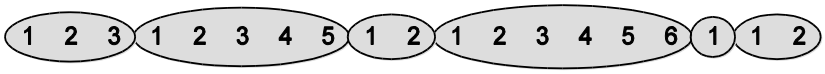
Sequence 98. **1 7843 4 43280 3 1174**

Seqsee has no notion of the admittedly blurry concept "small number". Thus, Seqsee is unable to extend Sequence 98, consisting of alternating small and large numbers. True, the sequence is not unambiguously defined (for example, is "59" — neither clearly large nor clearly small — acceptable as the next term?). True, since the next term can be *any* small number, the sequence does not have a *unique* correct extrapolation. But such ambiguities do not stop *people* from effortlessly suggesting plausible extensions, and we would like Seqsee to be able to easily extend the sequence as well.

Seqsee does no better on Sequence 132 below, although it does realize — thanks to the relationship between "1", "2", and "3", which Seqsee quickly spots — that this is a quasi-periodic sequence with a period of two.

Sequence 99. **1 7843 2 43280 3 1174**

Seqsee's inability to solve these two sequences is just the tip of the iceberg. Let's explore this further, starting with another sequence that Seqsee is currently incapable of extrapolating. Sequence 100 consists of a sequence of ascending groups, all starting at "1".

Sequence 100. 

What comes next? Recall that the human sequence-inventor is the sole arbiter of what constitutes a valid continuation, and in this case, I as the designer of the sequence wish to accept any ascending group starting at "1" as the continuation. Both "(1 2 3) (1 2 3 4 5)" and "(1) (1 2 3) (1 2 3 4)" are

acceptable as the next eight terms, but "(2 3 4)" or "1 1 3" are not acceptable as the next three. People may catch on to the fact that there is no deeper pattern relating the ovals, and having thus recognized the non-uniqueness of the continuation, they may randomly choose some fitting group as the next few elements. Or it may happen instead that they do not suspect the absence of pattern and believe they have just not spotted it yet, but can still confidently say "I don't know what comes next, but it's definitely some ascending group starting at '1'". The current version of Seqsee cannot create incompletely specified groups, and so it can do neither. It can easily build an ascending group starting at "1" and ending at "3", but it cannot — yet — "build some ascending group, don't care which!"

The problem runs deeper. A person might phrase what comes next in the above sequence slightly differently:

> I don't know what comes next, but it is definitely something like "1 2" or "1 2 3 4 5".

This strange category — "something like '1 2' or '1 2 3 4 5'" — surely contains "1 2 3 4 5 6" as a central member, but it contains only barely, if at all, the group "1 2 3 4 … 100000". This category is thus graded — instances range from those that are certainly inside the category to those that are marginal. Graded categories are ubiquitous, and various authors (such as Ludwig Wittgenstein (1953/2001), George Lakoff (1987), Douglas Hofstadter (for example, 1995, pp. 263-267), John Ellis (1993), to name but a few) have discussed such categories and their importance at length. Seqsee's categories are also graded (Section 6.2.4), but they are nearly black-and-white in comparison.

Can Seqsee be modified to handle such "things-like-these" categories? Perhaps it can be. We shall return to this question in Sections 6.6 and 6.7 after seeing what categories are already present in Seqsee.

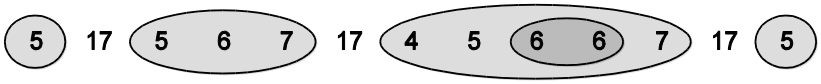
Section 6.2 A SINGLE CATEGORY IN DEPTH

I have programmed certain categories into Seqsee, including the categories *ascending group* and *sameness group*. I refer to these as the "built-in categories". In addition, I have given Seqsee the ability to *create* some other categories as needed, and these I call the "generated categories". In this section, we will look in depth at the built-in category *ascending group*. Subsequent sections will treat a few other categories at length.

6.2.1 POTENTIAL INSTANCES

Consider the four outermost ovals in the nonsensical Sequence 101. All of these are instances of the category *ascending group*, though to differing degrees. The two solitary "5"s are degenerate ascending groups, whereas seeing "4 5 (6 6) 7" as an ascending group requires some amount of squinting — that is, one has to pretend that the "6 6" is a simple "6".

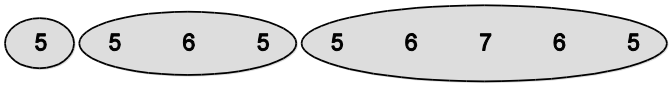
Sequence 101.



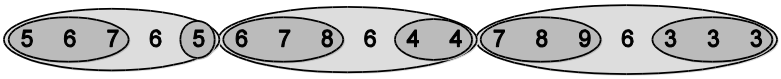
By the time that Seqsee finishes understanding the sequence presented to it, it annotates several groups as belonging to certain categories. For example, "5 6 7" will quickly get annotated as an ascending group. The process of annotation is not automatic: there is, for example, no good reason to automatically annotate a single "5" as an ascending group. Seqsee makes that annotation only if there are sufficiently compelling reasons. In the case of "5 6 7", though, the annotation is likely to be made as soon as the group is created, since Seqsee almost certainly will have created that group precisely because of noticing that successive items in "5 6 7" are successors of previous items.

When Seqsee does annotate a group as belonging to a certain category, it also remembers a category-specific description of that group. For "5 6 7" seen as an ascending group, for example, such a description states that it starts at "5", ends at "7" and has length 3. To make this clearer, consider the description of the group "5 6 7 6 5" in Sequence 102 below when seen as an instance of the

category *mountain*: the *foot* is “5” and the *peak* is “7”. It is by comparing such descriptions of two groups belonging to the same category that Seqsee discovers how one group may be transformed into the other.

Sequence 102. 

To stress that the description is specific to a category, I present Sequence 103 below, which also contains the group “5 6 7 6 5”. In this sequence, seeing this group as a mountain turns out to be of no use. Here, the group is instead a particular instantiation of a certain template of length 3. Speaking of the “peak” of this group makes only a little sense — no more sense than talking about the peak of the group “6 7 8 6 4 4” — and in this context it merely means “the largest element of the group”.

Sequence 103. 

The word “potential” in this subsection’s title needs to be stressed. In a given run, Seqsee might or might not see “5 6” as ascending. Similarly, in Sequence 101, either, both, or neither of the two ovals containing just one “5” might be seen as ascending groups, independently of each other.

6.2.2 Analogies Based on This Category

Seqsee sees similarities between two things based on their shared categories. “5 6 7” and “5 6 7 8”, for instance, are both ascending groups, and the description of how the first of these can be changed into the latter can be seen schematically in Figure 6.1.

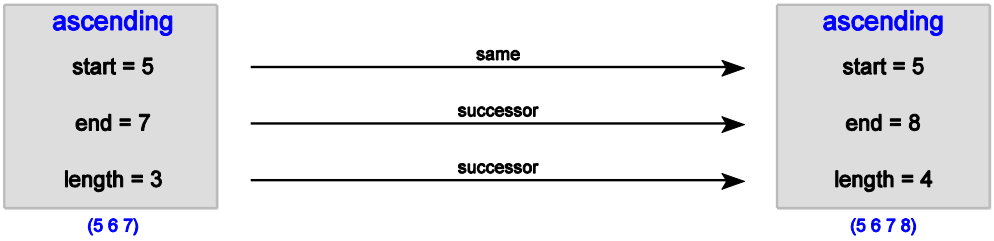


Figure 6.1 Analogy between “5 6 7” and “5 6 7 8”

There are many things going on in that figure, so let's unpack some of these. The shaded rectangle in the leftmost part of the figure contains one possible description of the group shown below that rectangle — the group "(5 6 7)". The description is based on the category "ascending group", and says that the group starts at "5", ends at "7", and has length 3. The right part of the figure also contains a shaded rectangle describing the group shown below it. This description is also based on the category "ascending group" — in Seqsee, all analogies are between objects that are seen as belonging to some common category[20].

The central portion of the figure, which consists of labeled arrows, is called the *mapping*, and is a recipe for converting (the description of) the first object into (the description of) the second: leave "start" untouched, and replace "end" and "length" by their successors. The essence of this mapping has been extracted below in Figure 6.2.

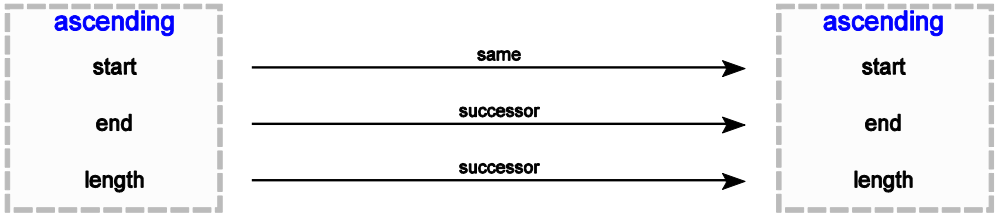


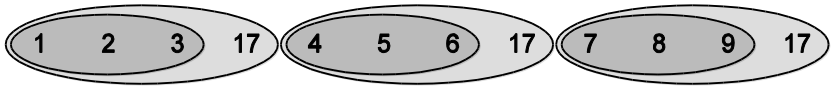
Figure 6.2 Skeleton of the analogy between "5 6 7" and "5 6 7 8"

Here, the actual groups have been erased and only a skeleton is left. It should be clear that this skeleton, if applied to any other ascending group, would change it in the same way: applying it to "2 3" would produce "2 3 4", and applying it to "7" seen as an ascending group of length one would produce "7 8".

[20] There is a lot of subtlety here: objects can be seen by Seqsee as belonging to multiple categories simultaneously (such as to the categories *mountain* and *group of size 3*). Also, in the process of the creation of a bond between two entities, Seqsee may discover a particular way to categorize some object (for example, we saw in Section 5.7 how Seqsee came to see the category *ascending group of ascending groups* as being applicabale to the initial "1" in the sequence "(1) ((1) (1 2)) ((1) (1 2) (1 2 3))". Nevertheless, when Seqsee sees an analogy between two objects, both have been seen as belonging to some common some category.

Analogical connections between groups can be more intricate than this. We will now look at how an outer oval in Sequence 104 can be transformed into the next oval.

Sequence 104.



Note that the inner ovals in that sequence are ascending groups, and the *start* of each inner oval is directly related to the *end* of the previous inner oval, as shown in Figure 6.3.

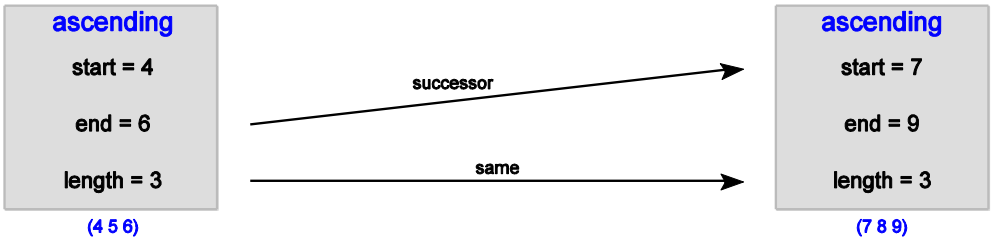


Figure 6.3 Analogy between "4 5 6" and "7 8 9"

This analogical mapping can also be stripped of specific groups to produce a skeleton, as shown below in Figure 6.4.

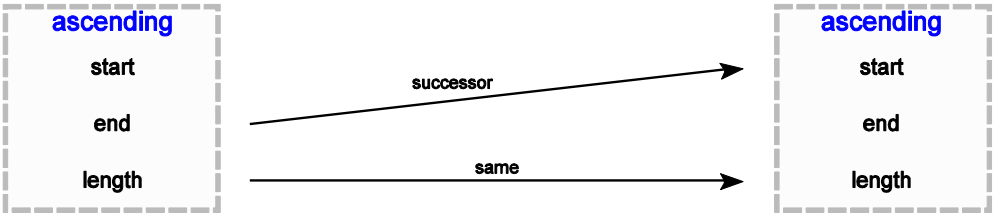


Figure 6.4 Skeleton of the analogy between "4 5 6" and "7 8 9"

A mapping may rely on simpler mappings (as it does in the two mappings shown above, each of which depends upon the simpler mappings "successor" and "same"). By the same token, the mapping may be a piece in a larger mapping. We can draw the relationship between the outer ovals in Sequence 104

as shown in Figure 6.5. Building up new mappings out of simpler mappings in this fashion allows Seqsee to perceive and describe a wide range of analogies.

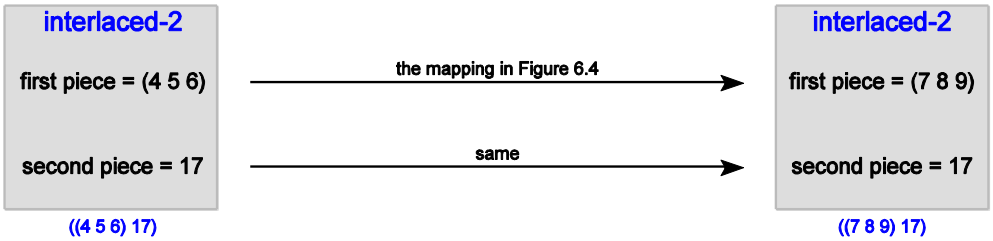
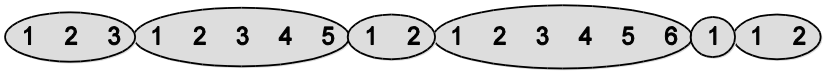


Figure 6.5 Analogy between "(4 5 6) 17" and "(7 8 9) 17"

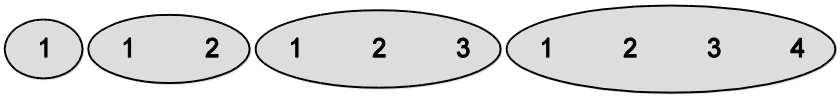
The mapping in Figure 6.4 only predicts the *start* and the *length* of the next group, but that is enough to build that group. Seqsee knows how to take a description and use it to build the corresponding group, if possible. Seqsee also knows how to check if a description is enough to let it build the corresponding structure. For the category *ascending group*, a description that includes both *start* and *length* is enough, whereas the description "it starts at 1" is inadequate for Seqsee. This is partly responsible for Seqsee's inability to extrapolate sequences such as Sequence 100 (repeated below).

Sequence 100



One natural question that may strike the reader is whether in each mapping Seqsee compulsively encodes too much information even when much less would do. Take Sequence 105 below, for instance.

Sequence 105.



In describing the relationship between the ovals in this sequence, people would find it adequate to say that the largest element, the *end*, is getting bigger, whereas Seqsee (it might appear) is stuck with being complete and needlessly pointing out that the *start* stays put at "1". However, this worry is unnecessary. It is indeed true that when Seqsee sees these ovals as plain vanilla ascending

groups, it includes the *start* in its description. However, Seqsee can and often does generate the category "ascending group starting at '1'". When Seqsee sees the ovals as instances of this more specialized category, each oval and the relations between ovals will be described in terms of the *end* alone.

For any category, the set of possible descriptions can be considered to form a tiny specialized language. Each category has a different language, a different vocabulary. The vocabulary for *ascending groups* includes *start*, *end,* and *length*; for *mountains*, it contains *foot* and *peak*; for the category to which the outer ovals of Sequence 104 belong[21], it includes *first piece* and *second piece*.

To make this clearer, and to motivate some other ideas later in the chapter, let us look at how Seqsee would need to be modified in order for it to understand sequences containing *valleys*, which we can define as follows. Let us stipulate that instances of valleys include "3 2 1 2 3" and "4 3 2 1 2 3 4", but not "4 3 2 3 4". That is, we artificially add the restriction that the bottom must be a "1" for us to count it as a valley. In order to enable Seqsee to use this category, we need to impart the following abilities to it:

- The ability to create a valley from its description. The description "the depth is five" is adequate, and Seqsee should be able to convert it to the group "5 4 3 2 1 2 3 4 5".
- The ability to say whether a group is an instance of the category *valley*. If a certain group actually is a valley, Seqsee should be able to create a valley-specific description for it. Consider, for instance, the group "3 3 3 2 2 1 2 2 3 3 3". If Seqsee recognizes this as a valley, the description would include the fact that the depth is three, as well as a description of the squinting involved in seeing this as a spiced-up version of "3 2 1 2 3".

In essence, we will need to add two subroutines[22] (one corresponding to each of these two abilities), and in this case this works out to about a dozen lines of code. Once this code is in place, Seqsee can see how two valleys are

[21]This generated category bears the name "Interlaced 2", which denotes an interlacing of two unrelated sequences. Put differently, each group in the sequence is based or a template of size 2.

[22] I am using the word "subroutine" literally here, not metaphorically, and I do mean adding a few lines of code.

related. If “3 2 1 2 3” and “4 3 2 1 2 3 4” were seen to be valleys, for instance, Seqsee would be able to describe their relationship simply by the analogy shown in Figure 6.6 below, and it would even be able to extend that analogy by doing the same thing to “4 3 2 1 2 3 4”, thereby obtaining “5 4 3 2 1 2 3 4 5”.

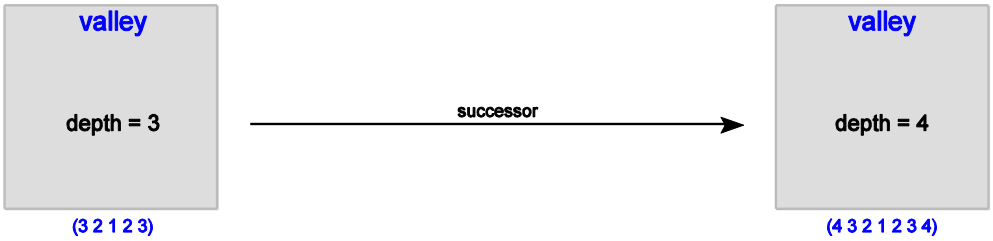


Figure 6.6 Analogy between “3 2 1 2 3” and “4 3 2 1 2 3 4”

The catch here is that only *if* two groups are seen as valleys can Seqsee draw analogies between them, extend analogies between valleys, and even understand moderately complex sequences such as Sequence 106, which consists of valleys alternating with “5”s. If Seqsee were to ask itself whether “4 3 2 1 2 3 4” is a valley, it would give an affirmative answer. However, in the extension to Seqsee to deal with valleys that I have described so far, I have added no pressure whatsoever to ask such a question. No codelet having anything to do with valleys ever gets generated, and consequently no valleys are ever seen. The next subsection describes how Seqsee discovers the presence of interlaced groups, and how it needs to be extended to see valleys.

Sequence 106. 2 1 2 | 5 | 3 2 1 2 3 | 5 | 4 3 2 1 2 3 4 | 5

6.2.3 Discovery

Before I discuss how the discovery of categories proceeds in Seqsee, I wish to describe an undesirable way to achieve the desired results. Let’s direct our attention to Sequence 107 below.

Sequence 107. 1 7 19 2 8 20 3 9 21

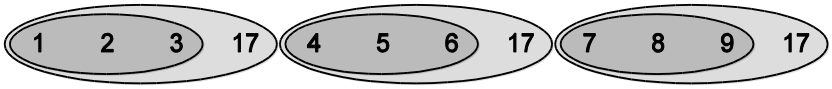
The following must have happened, I predict, as you looked at and understood the sequence:

- The *three-ness* of the sequence immediately jumped out at you. It dawned upon you, I imagine, that you were looking at three intertwined sequences.
- You made absolutely no attempt to group items into chunks of size 2 or size 4. That is, you did not try to see whether the grouping "(1 7) (19 2) (8 20)" or the grouping "(1 7 19 2) (8 20 3 9)…" made any sense.

There was no pressure created by observations to invoke the notion of two or four intertwined sequences, and reasonable people generally do not make random moves in the absence of any pressure to do so; neither should Seqsee. This is a core tenet of FARG architectures that we have already discussed at length (Section 1.2).

We can use the category "Interlaced 2" to illustrate the trouble with brute-force methods. I have repeated a sequence that we have already seen where this category is relevant as Sequence 108 below. The two interlaced sequences in the sequence are "17 17 17" and "(1 2 3) (4 5 6) (7 8 9)".

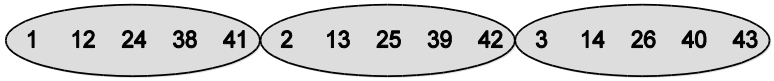
Sequence 108.



"Interlaced-2" is an example of a *generated* category, by which I mean that I have not directly coded this category into Seqsee but have added the ability to generate it should the need arise. This is one category from an infinite family of categories, and this family includes the category relevant to Sequence 107 ("Interlaced 3"), and also, to pick another example at random, the category "Interlaced 523", which is useful in understanding sequences consisting of 523 interlaced strands (which is to say, it is utterly useless).

An unacceptable way of trying to assign categories to groups would be to try various categories one by one, either systematically or randomly. That is, for a particular group, pick a category at random and check if the group is an

instance of that category. Such an approach of aimlessly trying many things to see what works — apart from being inconsistent with human cognition (which we are attempting to model here) — is also impractical for sequence extrapolation. A simple demonstration of this is that a category such as "Interlaced 5" must be one of the categories that are potentially worth trying, since some relatively simple sequences rely on it for their comprehension (for example, Sequence 109), and yet, in other sequences, even the briefest consideration of this rare category will almost always be a total waste of time.

Sequence 109. 

It will *not* be a waste of time, however, if the sequence's pattern *does* involve this category. If there is a reasonable way to invoke only potentially relevant categories, the number of failed attempts at checking whether a group belongs to a category goes down significantly. In the context of Sequence 107, we may ask how Seqsee can know that there are three interleaved sequences present without actually first checking whether the sequence can be split into three simpler strands. This question can be rephrased again as "How can Seqsee smell the three-ness?", or, more grandiosely, as "What are the mechanisms of intuition in Seqsee?"

There is a vast gulf between, on the one hand, knowing and using a category and, on the other hand, being able to recognize it in the wild. This was brought home to me recently when I decided to improve my chess skills. I picked up a book containing chess problems of the "white to play and mate in three" variety and realized that I am pretty good at solving these. When I am explicitly *told* that a mate in three exists, I can usually find it without trouble. The catch, of course, is that this skill does not carry over to actual chess games I play, where I have occasionally missed recognizing mate-in-three situations. Few real chess positions are just three steps away from definite victory or loss, and no reasonable players ask themselves before *every single move* if the current configuration is of this type. And, in any case, if I do not strongly believe that a mate in three is possible in a given situation, my attempts to discover those moves will be half-hearted, thereby making it less likely that I will discover the

right moves. I staunchly believe that the statements “this problem is easy” and “this problem has a solution” are excellent hints, even though they appear to convey absolutely no information pointing to the solution.

To take another example of the same distinction between knowing and recognizing, consider the difference in abilities of a student to solve two nearly identical problems, one of which reads “Prove using mathematical induction that …”, and the other of which reads the same except that the phrase “using mathematical induction” is absent. In my experience with students, I have noticed — unsurprisingly — that the former is much easier than the latter for many students. These students have difficulty in smelling the category “problems amenable to a solution using mathematical induction”.

Without further delay, let’s look at Seqsee’s sense of smell for certain categories. We begin with how Seqsee can sense the presence of three interleaved sequences. Triply interleaved sequences (such as Sequence 107, repeated below) tend to have relations between objects at distance 3.

Sequence 10. 1 7 19 2 8 20 3 9 21

Here, Seqsee usually notices quickly that the “8” is a numerical successor of the “7”. These two are at distance 3 from each other, and this fact alone is very mildly suggestive of the existence of three interlaced sequences. But all by itself, this single distance-3 relationship is nearly meaningless. However, if Seqsee notices such relationships over and over again, the case for hypothesizing that there are three distinct strands within the sequence gets much stronger.

Once any distance-3 relatioship is seen, a node representing the notion of “three interleaved sequences” is added to the long-term memory unless such a node already exists from prior runs. Seqsee’s long-term memory (LTM) is described in Chapter 7. Briefly, the following facts about LTM are relevant here. Each node in LTM has a number associated with it called its *activation,* and this represents the strength of Seqsee’s belief that the concept is relevant to the

problem at hand. Various events may increase the node's activation, and nodes also spread activation to neighboring nodes. Further, in the absence of a new influx of activation, activation decays over time.

Each time an analogy at distance 3 is seen, a tiny bit of activation is sent to the node representing "three interleaved sequences". That is, each distance-3 relation seen makes Seqsee believe a tiny bit more strongly that the sequence may in fact consist of three interleaved sequences. The strength of this belief does not increase linearly: it is a sigmoidal function as shown in Figure 6.7, (the idea of using a sigmoid in this fashion is quite standard in FARG models, but this particular figure has been reproduced from the recent dissertation of Harry Foundalis (2006)). In this figure, reinterpreted for the current situation, the x-axis represents the number of times a distance-3 relation has been spotted, and the y-axis displays the activation of the concept. Thus, the first few times that such relations are seen, the activation stays near zero. The activation starts climbing rapidly when enough instances have been seen, and after a few of these, subsequent instances do not have much impact.

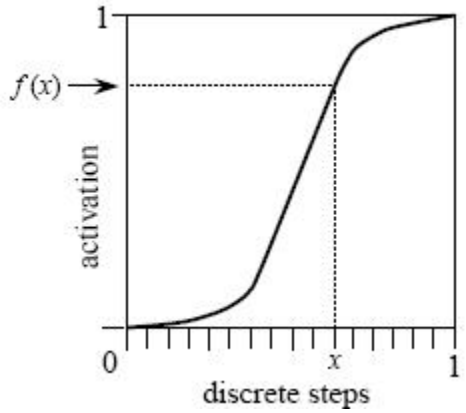


Figure 6.7 Activation function

Even though this activation may decay over time, the abundant distance-3 relations in Sequence 107 ensure that the node quickly becomes activated and stays highly activated, as can be seen in Figure 6.8. In this figure, the x-axis represents the number of codelets run and the y-axis shows the activation level. As can be seen, there are occasional tiny dips (caused by decaying activation),

but for the most part activation keeps on rising because Seqsee regularly notices distance-3 relations, as these are plentiful.

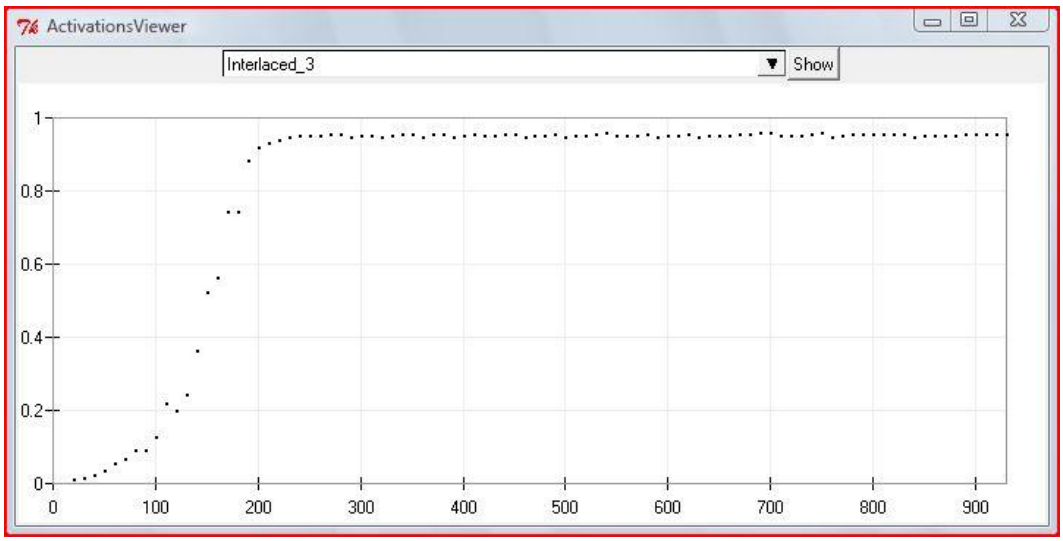


Figure 6.8 Activation of "Interlaced 3" while solving Sequence 107

A high activation of this node strongly influences what Seqsee does when a distance-3 analogy is seen. Because of the high activation, Seqsee is willing to "believe in" the presence of three interleaved sequences, and thus it expects the sequence to be composed of groups based on a template of length 3. If, in Sequence 107, Seqsee sees the "3" as the successor of the "2", it is willing to believe that these two objects are corresponding objects in successive length-3 groups. In this situation, Seqsee is likely to create a group starting at the "2" and ending just before the "3", expecting that it would later see a similar length-3 group starting at the "3" (here, it will form a "2 8 20" group). How likely Seqsee is to form such a group depends on how active the node is. It is also worth noting that even if the two nodes "three interleaved sequences" and "eight interleaved sequences" are somehow equally active, Seqsee is much less likely to consider grouping together eight elements. Seqsee can see sequences made up of eight interleaved sequences, but it resists the idea unless the evidence is overwhelming, and unless there is no simpler way of seeing the sequence.

It is natural for the reader to wonder at this stage if Seqsee's long-term memory has a node for every possible number of interleaved sequences. That is,

is there a node corresponding to a sequence consisting of, say, 71 interlaced sequences? The answer to that question is: Seqsee would create such a node if it saw a distance-71 relation, and not otherwise. It does not start out with such a node. Most nodes in Seqsee's long-term memory are generated on the fly, as need for them arises. This only-as-required technique makes the outlandish node that we are discussing a tiny bit more palatable, but it is still problematic, since Seqsee is too generous in what it considers *needed*. If such a long-distance relation were ever seen, no person would jump from this alone to the conclusion that this is even the tiniest bit of evidence for the presence of 71 interlaced groups. The situation is made somewhat worse because of another deficiency in Seqsee that I discuss in Chapter 7 — the notion of *forgetting* has not been implemented, and therefore this node would stick around — inertly at zero activation — until the memory was reset.

If a sequence really did consist of 71 interlaced sequences, how would a (mind-numbingly tenacious) person catch on to this fact? The person would certainly need to see thousands of terms. Before this concept sprang into existence in the person's mind, other more general concepts would surely arise, including perhaps "something is repeating", "this seems to be a collection of dozens of interlaced sequences", and so forth. The creation of such an unlikely node requires much more than the flimsy evidence provided by a single relation.

One might conceivably say the following in defense of Seqsee's creation of this outlandish node: the activation of this node, even if it had been created, would stay near zero if it were not seen as a relevant concept, and the presence of that node in the LTM would have absolutely no impact. However, that is not a good enough excuse to do something cognitively implausible. This is a serious shortcoming in Seqsee.

Let's return to the discussion about recognizing the presence of three interleaved sequences. There is considerable subtlety in the apparently crisp notion of "distance 3" that muddies up the waters, and that makes it somewhat harder both for Seqsee and for people to see interleaved sequences. Consider Sequence 110 below. This is later shown with ovals (as Sequence 111).

Sequence 110. **1 17 1 2 1 18 19 1 2 3 2 1 20 21 22**

In this sequence, what is the distance between "1 2 1" and "1 2 3 2 1"? Is it 2 or is it 3? The two-ness of this sequence is more hidden[23], and it takes a while to become apparent to Seqsee. Figure 6.9 shows how the activation of "Interlaced 2" changes over time. It takes Seqsee a long time to take serious notice of this concept (over 9000 codelets in this particular run, as opposed to only about 300 for the more obviously interlaced Sequence 107, activation for which was shown in Figure 6.8).

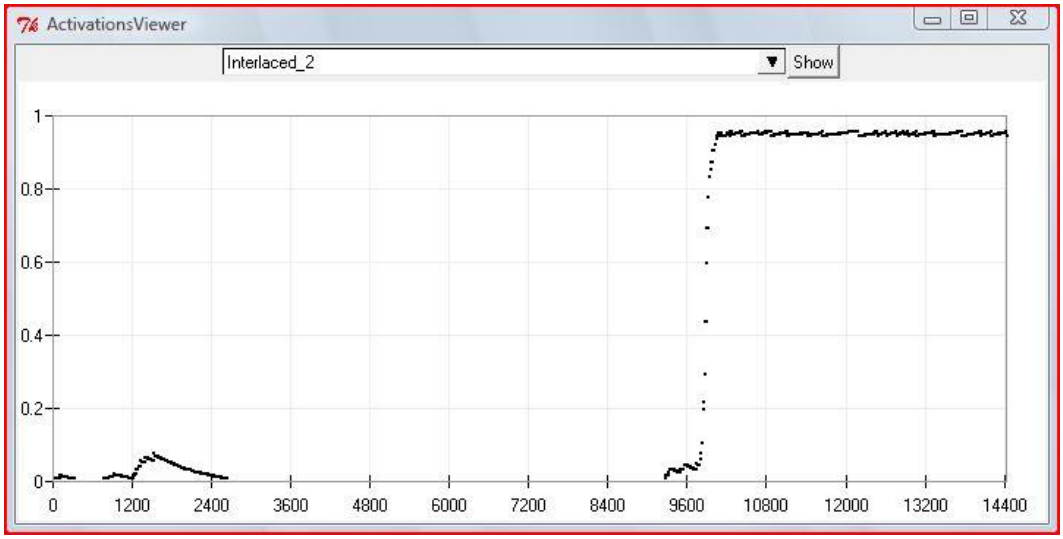
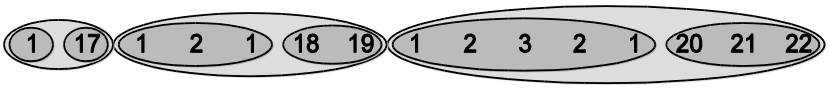


Figure 6.9 Activation of "Interlaced 2" in Sequence 110, in a particular run

Sequence 111. 

Let us now turn to another example of smelling a category. The activation of the node "mountain" goes up when an upslope (that is, an ascending fragment such as "3 4 5") followed by a downslope (that is, a descending fragment) is seen. When such a pattern is seen frequently, Seqsee is likely to

[23]People can see this much more easily than Seqsee can. The numbers 17 through 22 are clearly far bigger than 1, 2, and 3, and occupy a different zone of the number line, so to speak, and being double-digit numbers, they also stand out visually. Seqsee does not notice such differences, however, and so it struggles harder to understand this sequence.

convert subsequent spottings of such upslope/downslope combinations into groups labeled by the category *mountain*. Something similar would need to be done to make Seqsee aware of the presence of valleys.

6.2.4 Gradedness of Category Membership

In the current version of Seqsee, at any stage during a run, an object either *has* been labeled with a particular category or it has *not* been. The existence of the label is black-and-white. Nonetheless, category membership in Seqsee is still graded in the following sense.

An object like the oval "4 5 (6 6) 7" in Sequence 101 can be seen as an ascending group, but it takes Seqsee longer than for it to see "4 5 6 7" as ascending. Longer also implies less likely. Statistically speaking, at each step, Seqsee takes the action that seems most promising, and if something will take long to accomplish, it needs to seem highly promising for a longer period. Thus, category membership *is* graded in terms of how likely Seqsee is to spot the membership.

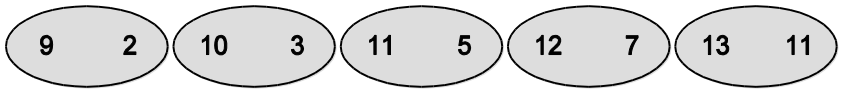
Category membership is graded in another sense as well. As in the case of Copycat, each object in Seqsee's Workspace has a time-varying *strength* associated with it — a number between 0 and 100 that represents Seqsee's belief that this is a relevant group to construct. Simple groups tend to be strong, as do groups that are similar to many other groups that Seqsee has seen. Weaker groups are more likely to be destroyed, and therefore have a shorter life expectancy than stronger groups. The argument made in the previous paragraph can also be reused here to show that weaker groups are less likely to be labeled. Thus, category membership is also graded in that, other things being equal, simple, strong groups are more central instances of categories than complex, weak groups are.

Future versions of Seqsee will have explicitly graded category membership.

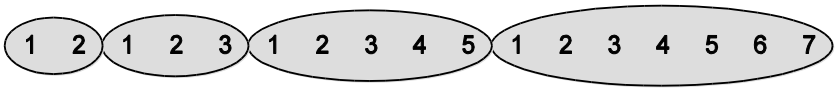
Section 6.3 THE CATEGORY "PRIME NUMBER"

Prime numbers, strictly speaking, are not a part of the Seek-Whence domain. In general, Seqsee has no ability to recognize such numbers, and as a result, it cannot extrapolate sequences such as "2 3 5 7 11". However, if it could label numbers appropriately as primes, and if it also had the ability to describe "successor" relationships among primes, this sequence would be trivially solved, as would slightly more involved sequences, such as Sequence 112 (where the second term in each block is a prime) or even Sequence 113 (where each oval is an ascending group whose end is a prime number). As we saw in Chapter 3, Seqsee has an optional switch that can be flipped on to force labeling of primes below 100 and "successor" and "predecessor" relationships among them.

Sequence 112.



Sequence 113.



Simple analogies based on this category are of three types: *sameness* (for example, between "7" and "7"), *prime successor* ("7" and "11"), and *prime predecessor* ("7" and "5"). The analogy between the last two ovals in Sequence 113 is shown below.

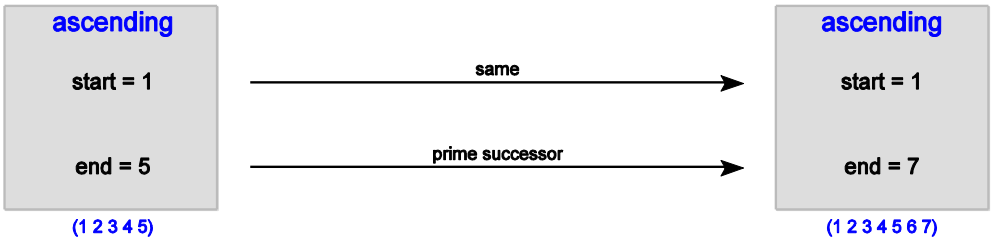
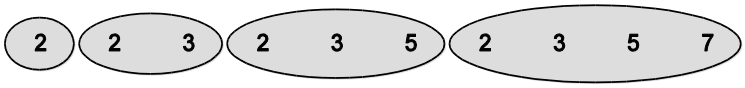


Figure 6.10 Analogy between "1 2 3 4 5" and "1 2 3 4 5 6 7"

Analogies based on the categories "even number" and "odd number" similarly consist of "even successor", "odd predecessor", and so forth.

Successive elements of a group such as “3 5 7 11” are seen to be related by being prime successors, but the group as a whole is not seen to be an *ascending group*, or even as a *prime ascending group* (no such category exists in Seqsee). However, we will see in the next section that Seqsee can manufacture that category based on the notion “prime successor”, allowing it to see sequences such as:

Sequence 114.



Before I leave the topic of primes, it is important to remember that although Seqsee can solve a large number of sequences involving primes, its implementation of the concept prime number is very shallow. A prime number is recognized exclusively by its presence in a list that I typed (and in fact, for a few days, Seqsee believed 91 to be a prime because of confusion in my mind). Every number is automatically checked for primality — Seqsee has no intuitions about what situations require the use of prime numbers. The current way of detecting primality is, of course, pure brute force — it was used because it was easier to implement and because I was willing to tolerate this shortcut, since primes are not a part of the Seek-Whence domain.

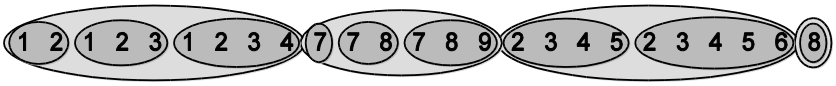
In defense of Seqsee, I should point out that the *higher-level* structures that it creates are in fact motivated by what it sees in the sequence, even if they are partly based on prime numbers (which are labeled as such in an unmotivated or brute-force way by Seqsee). For example, when the grouping “(9 2) (10 3) (11 5)” is constructed, there are plenty of cognitively plausible hints for Seqsee to believe that groups of length two make sense in the current sequence, even though that belief is dependent upon its having labeled “2”, “3”, and “5” as primes in a brute-force manner.

Section 6.4 A Family of Generated Categories

Groups can be based on arbitrary analogies, and each analogy defines a category. In this section, I will describe a large family of categories whose

instances are fragments of sequences, and I will show how these categories enable Seqsee to understand a wide variety of sequences.

Consider the following four groups, all of which are similar in some way. (For brevity, I have put them all in a single sequence. The overall sequence does *not* make sense.)

Sequence 115. 

These four groups appear to be similar in some fashion. Each contains sub-ovals that are ascending groups. Notice how the sub-ovals within a single outer oval are related: the mapping that takes “1 2 3” to “1 2 3 4” also takes “7 8” to “7 8 9” and also “7” to “7 8”. The rightmost outer oval is a degenerate example, containing a single sub-oval, which is itself a degenerate ascending group.

Two successive sub-ovals within a single oval in Sequence 115 are related by the mapping shown in Figure 6.11.

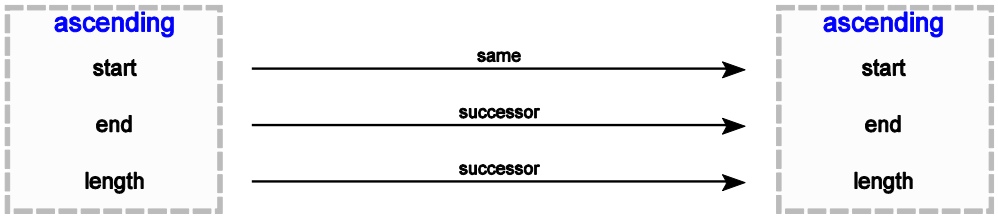
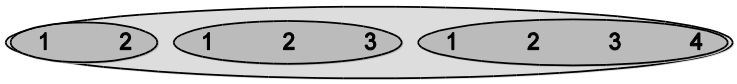


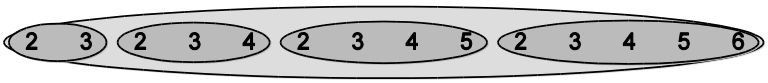
Figure 6.11 Analogy between successive sub-ovals in Sequence 115

I now wish to describe the category whose instances are things like the outer ovals in Sequence 115 — that is, groups whose pieces are connected by the mapping shown in Figure 6.11. For want of a better name, let’s just call this “the new category”.

Given this mapping connecting successive sub-ovals, we can reconstruct any of the outer ovals given two pieces of information: the first sub-oval and the number of sub-ovals. This opens up the possibility of describing relationships

between groups of this new category. Consider two members of this category and how one can be transformed into the next.

Sequence 116. 

Sequence 117. 

Both groups above are instances of the new category, and an analogy between them is shown below:

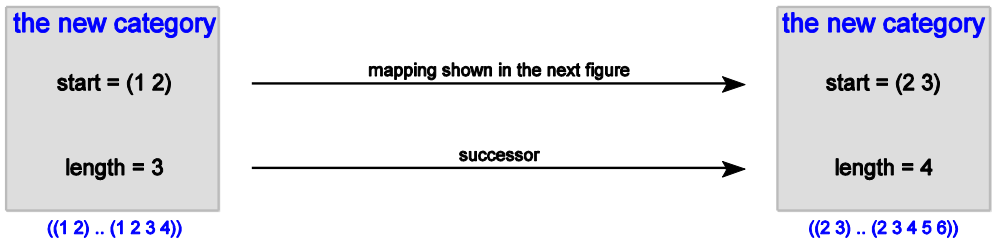


Figure 6.12 Analogy between the two groups shown above

and the mapping between the start of each group is shown below

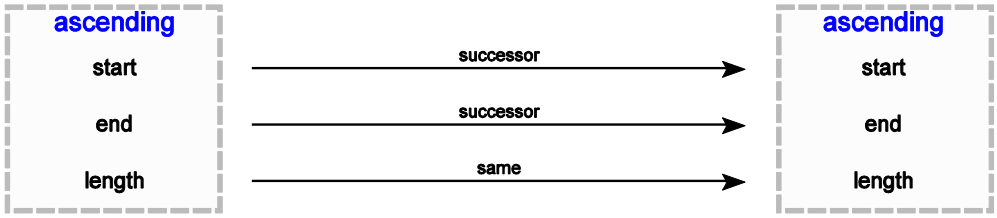
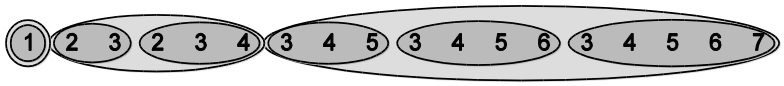


Figure 6.13 Analogy between the starts of the two groups shown above

Being able to see this category allows Seqsee to successfully extrapolate complex sequences like Sequence 118. If we describe the transformation of the first sub-ovals in the outer ovals (i.e., the relationships between “1” and “2 3” or between “2 3” and “3 4 5”) as “shift and elongate”, then the relation between the outer ovals is as shown in Figure 6.14.

Sequence 118.



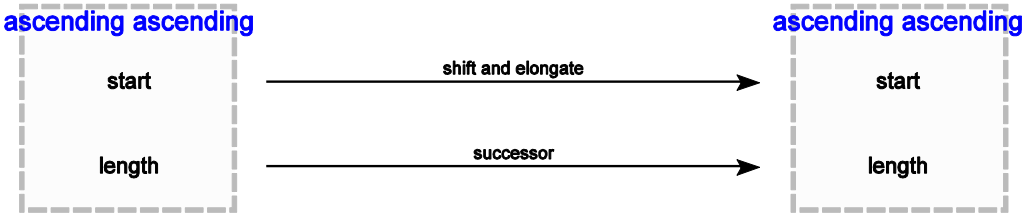
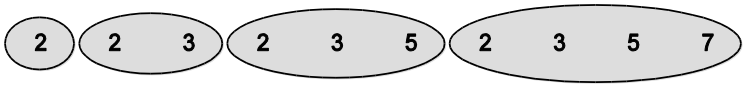


Figure 6.14 Analogy between successive groups in Sequence 118

Many sequences are conveniently described in this fashion, including Sequence 114 (repeated below). The analogy between successive ovals in that sequence is shown in Figure 6.15.

Sequence 114.



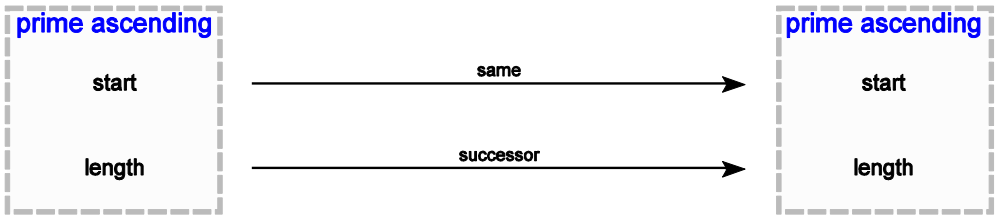


Figure 6.15 Analogy between "(2 3 5)" and "(2 3 5 7)"

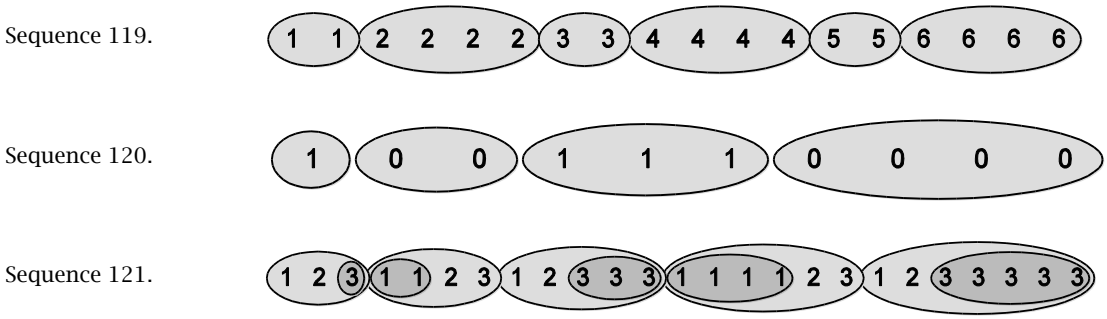
It may appear that we are straying from being true to how people think by allowing groups of such complexity. The unsettling quality about the sequence perhaps stems from the lack of names for these categories — "ascending ascending" and "prime ascending" are names that I manufactured for ease of description. Even though these categories are not extremely complex, they are slippery, and if we were to be distracted by some extraneous task (such

as the telephone ringing), we would probably have to make a significant effort in order to reconstruct what the sequence was.

However, I think that the description that Seqsee has arrived at is similar to the way a person might describe it. All that Figure 6.14 says is that each successive group is getting longer and starts off in a slightly different way than the previous group did, and that is close to the way a person might describe it.

Section 6.5 A CATEGORY WITH A FAINT ODOR

A recent addition to Seqsee's category bestiary is the generated category "alternating". Our first order of business here should of course be to give a few examples.

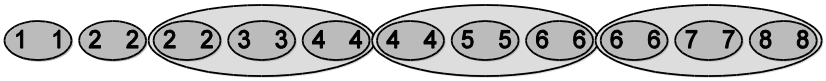


In these sequences, different types of things are alternating: the length of ovals in Sequence 119 is alternately "2" or "4"; in Sequence 120, "0" and "1" take turns; and finally, in Sequence 121, the location of the "blemish" flips between *first* and *last*. All these are, despite their differences, instances of the same abstract category, "alternation".

I had a hard time figuring out how Seqsee should smell the presence of this category. Seqsee can create the category "alternate between 0 and 1" or even "alternate between '1 2 3' and '5 6 7'" if it seems called for. But these categories are difficult to smell: a huge number of false positives may result unless care is taken. For example, during an attempt to solve Sequence 119, if "2 2 2 2" is seen as followed by "3 3", just how strong is that as a piece of evidence for the presence of the category "alternation between 4 and 2"? Not very strong

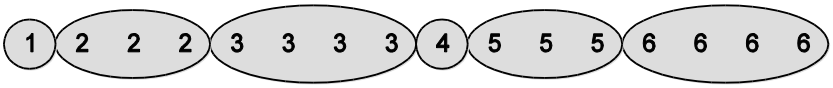
at all. Depending on what groups Seqsee had already constructed, it might just as easily have seen the "2 2 2 2" followed by a single "3". That would then suggest the idea "alternating between 4 and 1" or perhaps "alternating between 2 and 3". It could also be that the long group "2 2 2 2" is just a misleading garden-path, and that the "real" interpretation of the sequence involves splitting that group into two smaller groups, as shown in Sequence 122, which is, except for the initial blemish, a very simple sequence.

Sequence 122.



If Seqsee cannot form a clear idea of the categories involved — as happens here because of tiny bits of (mostly spurious) evidence for a wide variety of categories — it is unlikely to make progress. The way Seqsee smells the category *alternation* is, instead, by the presence of *three* consecutive objects in which some aspect is alternating. Thus, in Sequence 120, when the three consecutive groups "0 0", "1 1 1", and "0 0 0 0" are seen, a tiny bit of evidence for a "0"/"1" alternation is recorded. If a few other instances of "0"/"1" alternation are seen, Seqsee will come to see alternation between these two entities as a relevant concept and will try to use it to understand the whole sequence. With three objects involved, there will be far fewer false positives to confuse Seqsee. The price that must be paid is that this is a specific solution that works only for alternation of *two* things, and not for the closely related notion of cycling among three things, as happens in Sequence 123.

Sequence 123.



In this chapter, I have repeatedly used the phrase "evidence for (the presence of) some category", and have described how nodes in long-term memory accumulate such evidence. This way of phrasing things is also used frequently by George Pólya in his books *Mathematics and Plausible Inference* (Pólya, 1954a, 1954b). Pólya makes a strong case for the idea that mathematics proceeds by educated guesses, and he suggests ways of estimating the strength of a guess. Imagine that a mathematician has guessed a new theorem based on

some observation, but has not yet proved the guess to be correct. If the guess leads to some prediction that turns out to be true, the guess is rendered more palatable, more plausible. If the guess predicts something *surprising* that also turns out to be true, this confers a higher degree of credibility on the guess. If, furthermore, the guess is seen to be analogous to some well-known result, it becomes yet more trustworthy. Pólya offers dozens of detailed examples (including a translation of some of Euler's writings to the same effect), showing how a multitude of tiny facts can add up to a fair amount of trust or distrust in an initial guess, and how these kinds of instincts guide mathematicians at all stages. In its far humbler pursuit of extrapolating sequences, I like to think, Seqsee acts in a similar way.

Section 6.6 DERIVATIVE SEQUENCES

In this section and the next, we return to the sequences mentioned in Section 6.1 as being beyond Seqsee's current abilities, and we consider how Seqsee could and should be extended.

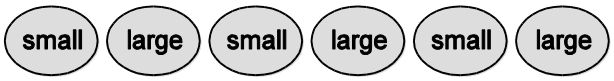
Sequence 124. **1 2 1 2 2 1 2 1 2 2 1 2 2 1 2 1 2 2 1 2 1 2 2 1**

My thesis advisor, Douglas Hofstadter, has long wanted Seqsee to be able to see sequences like Sequence 124. This sequence consists of groups of "2"s sandwiched between solitary "1"s. We may create another sequence from this by counting the lengths of the "2"-groups, and it turns out that this derived sequence is Sequence 124 again.

Seqsee does not create derivative sequences at all, and cannot figure out what's going on in the sequence above. For a long time, I resisted adding the ability to derive sequences, arguing that this would be needed only for rather exotic sequences (such as Sequence 124) and for mathematical sequences (the reader may recall the discussion of Superseeker from Chapter 1), and not for the sort of sequences we have been looking at. However, I have recently realized how wrong I was.

We humans use derivative sequences all the time in understanding even simple sequences. These are usually not made up of numbers, though. The sequences in Section 6.1 can be converted to the sequences shown below, and in any case, this is how we describe these sequences.

Sequence 125.



Sequence 126.

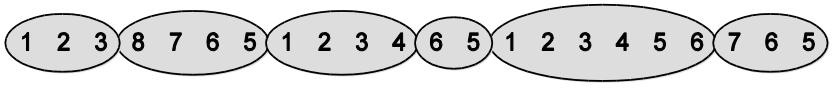
If Seqsee is extended so that it can generate such sequences, not only would the simplicity of the derivative sequence allow it to see more complex patterns, it would be able to extrapolate Sequence 100 and many other like it correctly.

Section 6.7 THE CATEGORY "THINGS LIKE X"

Since Seqsee cannot yet create and use categories like "things like '1 2 3'", this section is more speculative than the rest of the dissertation. I suggest a path that we might follow to equip Seqsee with such elusive categories.

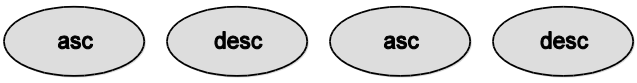
To keep things concrete, let's look at Sequence 127. To a human, the appropriate categories to create are obvious, and one of the two categories is "things like '1 2 3', '1 2 3 4', or '1 2 3 4 5 6'". How can Seqsee hit upon this category without falling prey to the category "things like '1 2 3' or '6 5'"?

Sequence 127.

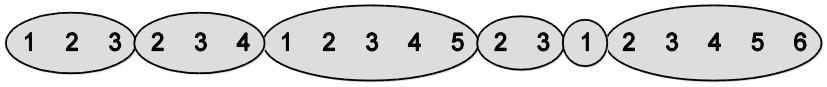


There are two types of groups here: ascending groups and descending groups. These are easy to tell apart, and the sequence can be seen as

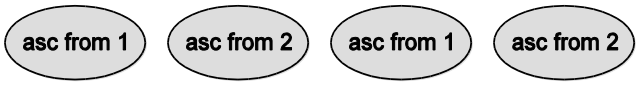
Sequence 128.



Not all alternations of two different categories in this fashion can be told apart so readily, though, and it is easy to mix groups from two categories that are harder to tell apart, thereby making the problem of how to discover the right categories more challenging:

Sequence 129. 

The issue that we need to resolve in order for Seqsee to be able to see such sequences is the following. Seqsee easily discovers ascending groups. The categories of relevance here are, however, more specific subcategories, namely, ascending groups starting at "1" and ascending groups starting at "2". If Seqsee labels all of these groups appropriately, it might be able to discover the following derivative sequence:

Sequence 130. 

However, this is an oversimplification, since the processes of group formation and of labeling are inextricably interlinked. The last two groups shown in Sequence 129 are the result of an unnatural splitting-up of "1 2 3 4 5 6" in this *a priori* very bizarre fashion. Such an unnatural perception is justified only when the presence of ascending groups starting at "2" has been seen to be relevant for the sequence.

Seqsee might be able to create novel categories by adapting methods from Phaeaco (Foundalis, 2006, Chapter 8). Foundalis motivates his technique with a figure that has been reproduced here as Figure 6.16.

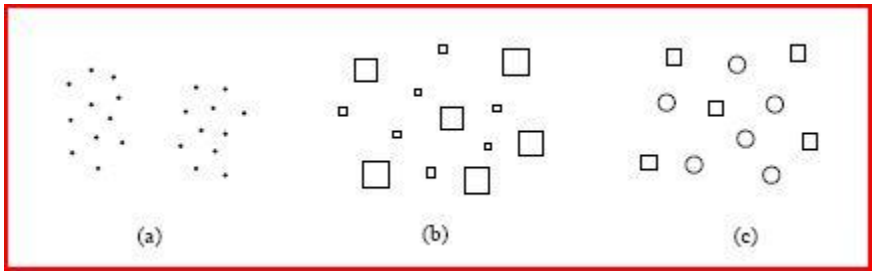


Figure 6.16 Example of grouping from Phaeaco

Phaeaco is able to cluster objects according to various features. In the figure above, objects may be clustered based on, respectively, their position, their size and their shape. It might be possible to adapt Phaeaco's algorithm to allow Seqsee to cluster groups in Sequence 127. The algorithm is described in Foundalis (2006, p. 228 to 231). If all goes well, we would end up with exactly the two clusters of groups that we need — the ascending groups and the descending groups.

Seqsee sorely requires something along these lines. The canonical sequence that we wished Seqsee would solve faithfully has been shown without ovals as Sequence 131.

Sequence 131. **2 1 2 2 2 2 2 3 2 2 4 2**

People generally come quite quickly to the idea of paying attention to "things that are not '2'" in the sequence above, since these items seem to hold the key to understanding the "2"s. The category "things that are not '2'" can also emerge from applying a clustering algorithm, as the "2"s will form a large cluster containing all but three numbers in the input.

What types of objects should we apply clustering to? As we have just seen, applying the clustering algorithm to the set of groups and elements might be useful. Likewise, applying the clustering algorithm to the set of analogies might be useful if it helps Seqsee discover that all analogies seen in a particular sequence are of only two types.

When should such a clustering algorithm be applied? All the time! Phaeaco's clusters are incremental, meaning that they can be updated as more data arrive. It should be easy to maintain clusters of the input elements, groups, and analogies, and just to update the clusters when, for example, a new group is formed.

I wish to stress once again that Seqsee does not yet do this, and as always, when I eventually come around to implementing this, many unanticipated problems and issues will surely arise.

In this chapter, we have seen how categories are implemented in Seqsee, and how the hardest part of the use of categories is discovering which categories are relevant without explicitly checking a large number of categories for relevance — in other words, *smelling* the presence of a category is the hardest part. I have also shown how easy it is to add categories to Seqsee, and easier still if one takes cognitively unrealistic shortcuts (as I did for primes). These shortcuts highlight the ever-present danger of the Eliza effect, since they can be added effortlessly to Seqsee, thereby enabling it to solve impressive-looking sequences involving primes, but more effort is required to enable Seqsee to understand less impressive-looking sequences involving valleys, although the latter are understood more honestly.

Chapter 7 LONG-TERM MEMORY

This chapter is short, as befits a chapter that represents only a tiny amount of work. These ideas are at an early stage of exploration, but I do not wish to imply that I consider the ideas I present here unimportant — on the contrary, I believe some to be crucial candidates for more thorough investigation.

The implementation of the long-term memory in the current incarnation of Seqsee borrows liberally from two sources: Copycat's Slipnet (Mitchell, 1990), and extensions to it made by Harry Foundalis for Phaeaco (Foundalis, 2006). I will describe the Slipnet and modifications to it only briefly, and then will point out the ways in which long-term memory in Seqsee significantly departs from these systems.

I mentioned Copycat's Slipnet a few times in Chapters 2 and 5. Section 5.2 describes the crucial a role that it plays. To recap, the Slipnet is a graph where each node stands for the core of a specific concept, and at each moment of a run, the activation of a particular node represents the strength of Copycat's belief that the corresponding concept is relevant to the current problem. Moreover, highly active concepts spread activation to nearby concepts. Figure 7.1 shows the entire Copycat Slipnet.

Seqsee's long-term memory differs from Copycat's in two significant ways: in how activation is spread, and in the way that different types of links are used.

Section 7.1 A DIFFERENT METHOD FOR SPREADING ACTIVATION

In Copycat, after every 15 codelets are run, activation spreads from every sufficiently active node to its neighbors. Spreading activation is an expensive process, computationally speaking, and although the desire was to do it after every codelet, that was deemed to be too resource-heavy.

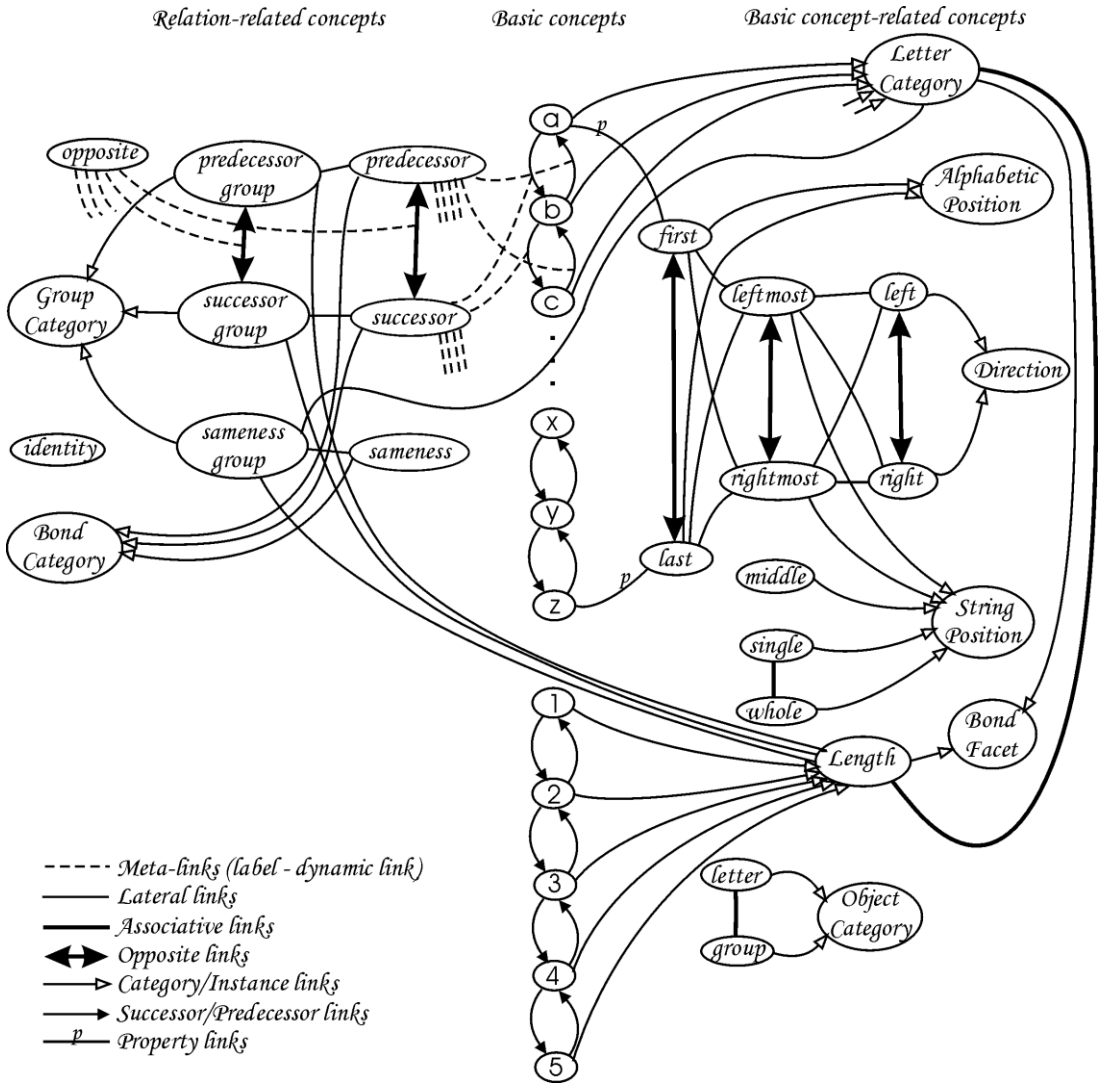


Figure 7.1 Copycat's Slipnet[24]

Seqsee handles activation-spreading quite differently. I made this change in order to fix the problem of overstimulation of the network, as described by the following thought experiment. Imagine, for a moment, an embodied descendant of Copycat that is smart enough to pass the Turing test. It will certainly need to have the concept "grocery-store checkout stand", which, since the program can pass the Turing test, would likely be connected to the concept "tabloid newspaper", which in turn would be connected to "paparazzi" and so on, through "Princess Diana's accident". If activation spreads from all nodes at

[24] This figure has been borrowed from the master's thesis "Fluid Concept Architecture: A Critical Evaluation" by Joaquin Vanschoren located at http://www2.cs.kuleuven.be/~joaquin/zsp/fluidconcepts/.

all times, then standing long enough in a grocery-store checkout line will unfailingly result in at least a tiny bit of activation reaching the concept "Princess Diana's accident", every single time — a ridiculous notion. In fact, if every node is connected to every other node along some (possibly long) path, the entire Slipnet will always be at least slightly active.

To counter this, Copycat spreads activation only from fully active nodes. This safeguard prevents the entire Slipnet from becoming active in every single run. However, this itself causes other problems. Having too many nodes that are fully excited cheapens and dilutes the value of activation, in my opinion. Also, it is easy to find legitimate cases where we would like activation to spread from two different nodes that we do not wish to be equally active. One such example can be found when trying to understand the sequence "1 2 3 4 5 5 6 7 8 9 10 10 11 12 13 14 15 15", which is full of successor relationships but only a few sameness relationships. It makes no sense for both of these concepts to be equally active — and both should not, therefore, be fully active — but it does make perfect sense for activation to spread from both nodes, albeit to differing degrees.

Seqsee's method of spreading activation is intimately tied to its Workspace and to its focus of attention, and I wish to explain that connection carefully. Imagine that Seqsee is solving the sequence shown above. As it does so, it creates groups and bonds. As the reader may recall from Chapter 4, a "Read from Workspace" codelet creates codelets of families such as "Focus on Group" and "Focus on Category". It may create a codelet to focus on the group "1 2 3", for instance. This group has the category label *ascending group*, and activation is pumped into that node. Moreover, activation is spread from that node to nearby nodes (those that are separated by two links or fewer). This method (spreading activation outward a small distance from the one concept currently being focused on) is the *only* way that activation spreads in Seqsee. In this sequence, this has the following effect. Sameness groups, being less abundant, are less frequently chosen by "Read from Workspace" codelets than the more abundant successor groups are, and thus activation spreads more slowly from the *sameness group* node. In fact, both nodes (for the concepts

sameness group and *successor group*) spread activation in proportion to how common and significant they are in the Workspace. This method of spreading activation has the advantage of nodes not having to be fully active in order to spread activation, and as a bonus, it is computationally inexpensive, and can therefore be performed after every codelet.

One possible objection that may be raised to this limited way of spreading activation is that, by requiring an object to be focused on before it can spread activation, it makes thinking *too conscious*, and does not allow those magical "subterranean" connections to emerge, seemingly out of nowhere, that result in marvelous bon mots, sparks of genius, and errors of great hilarity. However, I believe that the model just described (pump activation into into the focused-upon node and spread it a short distance), in combination with the notion that overlapping fringes cause attention to be focused in a particular way, has ample possibilities for subterranean serendipitous connections to emerge.

I discovered a nice example of such a connection this morning while I was in the shower. A short flashback. Last night, I printed out a map. I found it strange that the printed image was somewhat skewed — the edges of the map were not parallel to the edges of the paper it was printed on. This morning, in the shower, I noticed that the door of the shower was a bit skewed — its edges were not parallel to the edges of the surrounding metal frame. I was reminded momentarily of the printed map, and immediately my thoughts shifted to the fact that I had been procrastinating buying ink for the printer.

The concepts of "door" and "map" are not generally close to each other. Spreading activation alone cannot explain this reminding. The fact that the map episode was fresh in my mind is certainly important — had a week elapsed between the two episodes, the reminding would almost certainly not have happened. The fringe of the door and that of the map had something in common — perhaps a mental image of two slightly-misaligned rectangles — and it was enough to make me think of the similarity between them. I did *not* consciously think of this mental image at that time, however.

Currently, Seqsee stores the fringes of the 10 objects that it has most recently focused on. This is obviously simplistic — I had focused for a split second on thousands of objects in the intervening hours between the two episodes, and most of these more recent episodes were gone from memory while the map episode had lingered. This can be attributed to my finding it strange that a printer could produce a skewed image. Items in memory do not fade at a uniform rate, and I should find a more intelligent way of storing fringes.

The method of spreading activation, however, seems capable enough of explaining unexpected connections. Indeed, these ideas are at the very core of how Seqsee works. As we saw in Chapter 5, things that have just happened bias perception in a way that generates a steady stream of serendipitous connections.

Section 7.2 A Feature Missing from Seqsee: Forgetting

Unlike the case of Copycat, the nodes and links in Seqsee's long-term memory increase in number, over the course of a single run (and therefore across multiple runs), as new nodes and links are added by a process described in the next section. In this regard, Seqsee is like Phaeaco.

Unlike Phaeaco, however, Seqsee does not (yet) have a notion of forgetting. If, in the process of solving some problem, a very unusual concept or an unusual link between two concepts is added, it will stay in long-term memory forever (although at near-zero activation). This significant shortcoming should be remedied.

Section 7.3 How New Links are Created

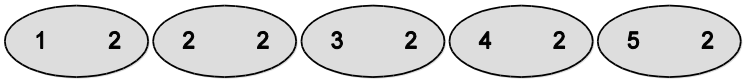
Long-term memory in Seqsee has three types of links among nodes, and all of these are automatically created. This results in a rudimentary form of learning.

The three types of links are “is-a” (connecting an instance to a category), “follows” (connecting a group with another group that often follows it), and “can be seen as” (connecting a group with another group that it can be seen as).

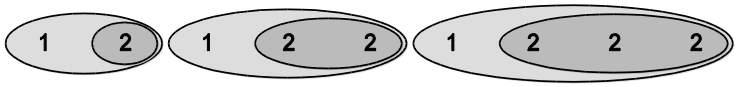
When Seqsee focuses on any group (say, on the group “(1) (1 2) (1 2 3)” in the Workspace), there is some chance that new links will be created. A “follows” link between “(1 2)” and “(1 2 3)” may get created, and an “is-a” link between “(1 2)” and the node corresponding to *ascending group* may get created if, as in this case, the relation between the three pieces of the large group being focused upon is based on each being an ascending group. If any of the links just mentioned is already present, it will be made stronger.

I should explain why I have added the requirement for the creation or strengthening of an “is-a” link. An instance of “1 2”, it is true, is itself intrinsically an ascending group and does not require other groups to legitimize this fact. However, it is not *just* an ascending group — there are many ways of seeing any group and, in a different context, this group can play a different role, as can be seen in the sequences below.

Sequence 132.



Sequence 133.



A link between “1 2” and the category *1 followed by a sameness group consisting of ‘2’s* should be created only if there is pressure to do so. If “1 2” is part of a group whose existence depends on it being seen as an instance of this category, Seqsee creates this link. If “1 2” is never again seen in this way, the link will stay weak. On the other hand, over the course of dozens of runs, in all likelihood, “1 2” will be seen more frequently as an ascending group and the link between the two will therefore be strong.

In a similar vein, if “1 2 3” routinely follows “1 2” in many sequences that Seqsee has seen, the “follows” link between them will be strong. In subsequent runs, when Seqsee is focusing on a “1 2” group, it may launch codelets to

actively seek a “1 2 3” to the right of the “1 2” group. This is how Seqsee is able to suggest possible extrapolations of a sequence when only a single term has been provided to it.

Actually, what I have just described is a simplification. Seqsee does not learn that “1 2 3” often follows a “1 2”. What it learns, instead, is more general. It learns that an ascending group is often followed by another ascending group that starts out the same but which is longer by 1. That is, it associates ascending groups with the mapping shown in Figure 7.2.

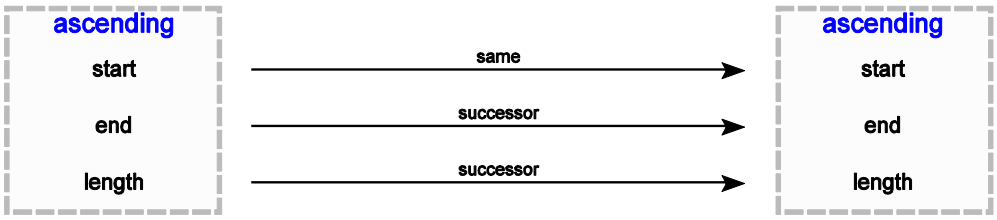


Figure 7.2 Mapping between “1 2” and “1 2 3”

This explains why Seqsee is quicker at extrapolating the target sequence in Figure 7.3 when it has previously seen either sequence a or sequence b. In all three sequences, the mapping connecting one block to the next is exactly the same (shown in Figure 7.5). Since successive items in sequence c are not connected by the same mapping, that sequence will have a much weaker influence on how quickly Seqsee figures out the target sequence.

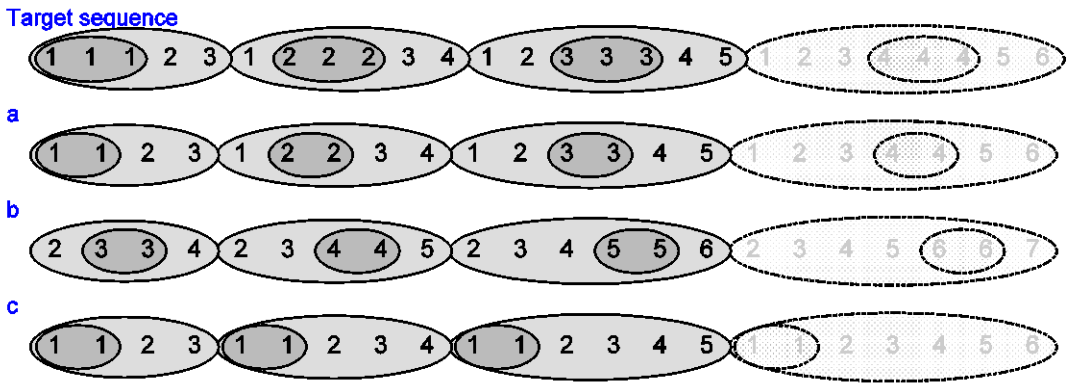


Figure 7.3 Sequences to demonstrate the effects of long-term memory

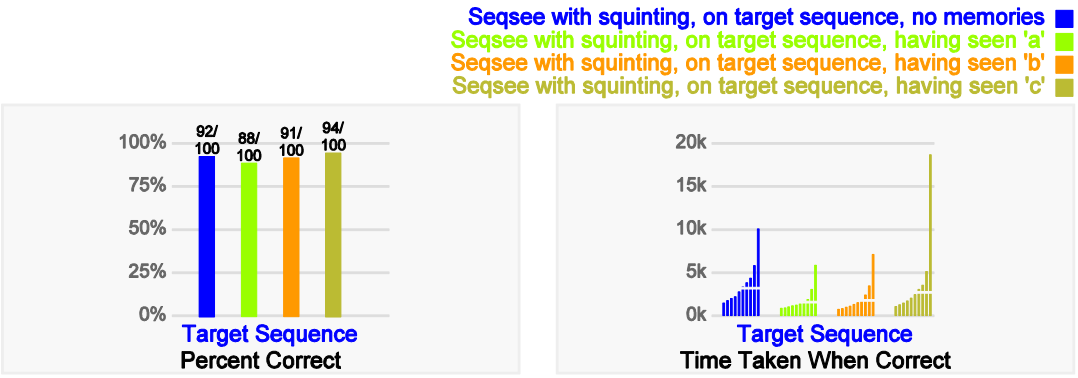


Figure 7.4 Effects of having seen a similar sequence

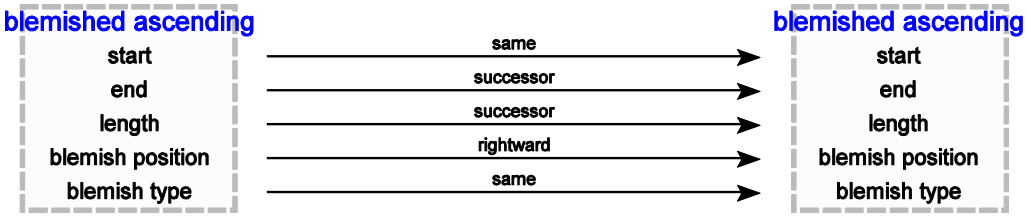


Figure 7.5 Mapping between successive terms in target sequence

This concludes the description of the novel features in Seqsee's long-term memory.

Chapter 8 CONCLUSIONS AND NEXT STEPS

We come now to the final chapter, which is simultaneously a discussion of the shortcomings and deficiencies in Seqsee, a description of how one might improve the program, and a proposal for the next version of Seqsee.

The chapter is organized as follows. Section 8.1 describes a few deficiencies in the current implementation of Seqsee. Section 8.2 explores the notion of granularity of codelets in Seqsee. Section 8.3 suggests a strategy that might be used to achieve finer granularity, and Section 8.4 shows how this will result in better solutions for problems in the Seek-Whence domain.

Section 8.1 A FEW DEFICIENCIES IN SEQSEE

In this section, I touch upon some shortcomings in Seqsee. Ways in which Seqsee falls short range from the very specific (such as a particular codelet that can be improved) to the very general. I will look only at the general end of the spectrum here, since issues there tend to eclipse the more specific lacunas. In the next few sections, I will make suggestions about how some of the general problems may be overcome, and if these suggestions get implemented, the end product will undoubtedly be a system that has a *different* set of specific problems.

A few problems were mentioned in preceding chapters, and were discussed in some depth there:

- In Section 3.1, I described how Seqsee makes overhasty guesses;
- In Section 3.8, I described various sequences that are within the Seek-Whence domain but which Seqsee is unable to extrapolate successfully, and I also discussed sequences that it rarely extrapolates;
- In Section 6.1, I described important types of categories that Seqsee does not have, and explained how categories such as “prime number” in Seqsee are extremely shallow;
- In Section 7.2, I described Seqsee’s inability to forget.

In this section, I will discuss two deeper issues with Seqsee's architecture.

8.1.1 Features in Seqsee Sometimes Interfere With Each Other

A glaring problem with Seqsee is that various features can interfere with each other, as we have already seen in Section 3.4, where an example was presented showing that turning on the squinting feature of Seqsee makes some sequences harder to see. I described in that section why that specific sequence suffers when squinting is turned on.

Each optional feature makes Seqsee sensitive to some aspect of situations that it is otherwise completely blind to. For instance, when the feature "prime" is turned on, but not otherwise, relationships between successive elements in "5 7 11 13" become apparent. The more things Seqsee notices, however, the more its attention is spread thin. If all the optional features are turned on, Seqsee is inundated with details and finds it difficult to separate the wheat from the chaff. This is an important problem to address, since it prevents Seqsee from scaling well.

8.1.2 Granularity of Codelets

A second problem has to do with the granularity of codelets in Seqsee. To describe the issue, I will focus on the process of construction of groups.

In Seqsee, a few different types of codelets create groups. For instance, a codelet that extends groups has to be able to construct groups, since extending the group "(1) (1 2) (1 2 3)" rightward requires the ability to produce "(1 2 3 4)".

Here is how this construction currently happens at the source code level. Every category has a subroutine that, given a description, constructs the appropriate group. For pre-existing categories (such as ascending groups and sameness groups), I have written this subroutine, and for generated categories, I have written code to construct such a subroutine on the fly. It could be said that group construction happens at the subcognitive level in Seqsee, even below the level of codelets, since a single codelet creates a group and since this creation is only one piece of what that codelet does. It is as if the construction of such a group (even an arbitrarily complex group) takes less than a single codelet.

Having a single codelet do so much runs very deeply against the grain of FARG projects. It is desirable to make each codelet do less, but that would result in far more codelet families, and would aggravate the issues mentioned in the previous subsection.

It must further be noted that even if I were to split the above-mentioned operation into many codelets, some codelet in this new system would still, in all likelihood, be doing too much.

In the next two sections, I discuss some nascent ideas, and then I suggest in Section 8.4 how these might be used to improve Seqsee and specifically to address the two problems just mentioned.

Section 8.2 A Hard Look at Granularity

Once Seqsee discovers a solution, many codelets participate in describing it. By contrast, when Seqsee discovers a description of the next group (say, that it is an ascending group from 3 to 7), a single codelet converts this description into an actual group — in this case, into "(3 4 5 6 7)". There is a certain amount of arbitrariness in the decision of which tasks get multiple codelets and which tasks are handled by a single codelet. Many types of practical considerations influence the decision. Each of the two extremes — the fine granularity achieved by distributing a task to a set of codelets and the rough granularity resulting from a single codelet doing a lot — has benefits and drawbacks.

A finer granularity may be more cognitively realistic (as I discuss below in Subsection 8.2.1), but a coarser granularity is easier to implement (although it is accompanied by its own problems, as is discussed below in Subsection 8.2.3). Practical considerations such as "it is easier to implement" should not be taken lightly. An analogy comes to mind. Various computer programming languages differ not in what they make possible — most modern programming languages are Turing-complete and can solve exactly the same set of problems — but rather, in what they make easy. Some types of solutions are more easily implemented in certain languages, and these languages therefore gently push (not force) the programmer into implementing those types of solutions. If we look upon Seqsee not as a completed project but as a work in progress (which in

a sense it is), these considerations about what is easy and what is hard will certainly influence (though not fully determine) where it goes in the future. It is therefore desirable, to the extent it is possible, to make the cognitively more realistic finer granularity nearly as easy to implement as coarser granularity.

8.2.1 Benefits of a Finer Granularity

To show that finer granularity is cognitively more realistic, I begin with a silly thought experiment, but what I discuss applies just as well to more mundane, down-to-earth examples. Imagine, if you will, a ridiculously hypothetical future embodied descendant of Seqsee that is capable of solving math problems and of playing chess, and which, being embodied, occasionally needs to see its dentist.

How is this program's ability to visit the dentist implemented? It obviously makes no sense for this hypothetical program to possess a single codelet for an entire dentist visit. Many things are wrong with such a mammoth codelet.

Such a codelet is inflexible, and it is not reusable for other nearly identical tasks such as a visit to the doctor, which is in most ways identical to a dentist visit. Both involve driving to their office, parking, spending inordinately long times in the waiting room, describing what is hurting, paying, and so on. If the dentist visit were composed of dozens (or thousands, or millions) of codelets, most could be reused in other overlapping tasks. A large number of tasks overlap with a dentist visit at least to some extent: anything that involves making appointments, or paying, or driving, or describing something has *something* in common with visiting dentists.

A codelet in Seqsee represents one single action. When larger actions are composed of several codelets (such as in discovering the presence of an ascending group), the component codelets can also participate in different fashions in other actions (the most related example being in discovering the presence of a descending group). A single codelet cannot be similarly analyzed.

A single codelet is less flexible than an equivalent collection of smaller codelets. In Seqsee, a codelet is a piece of code that I have written. It is made up

of a few lines of Perl code that run in a particular order, or, in the presence of if-then statements, the codelet may follow one of a handful of different paths through its code. Splitting the same task into multiple codelets immensely increases the number of trajectories that the program can potentially follow. The task effectively gets split into several smaller chunks and (unlike in the case of the single codelet) Seqsee can simultaneously work on different chunks (since the Coderack may contain codelets that tackle different pieces).

Section 8.2.4 ("A Codelet for Multiplication?") shows how implementing multiplication using a single codelet is unrealistic and rigid (although utterly trivial to implement), and how a collection of codelets is needed to achieve a much higher flexibility.

8.2.2 Roger Schank on Finer Granularity

The description above is similar in spirit (and in the examples chosen) to that used by Roger Schank in *Dynamic Memory Revisited* (1999) where he argues against his own notion of a script from 22 years earlier, suggesting its replacement by more fine-grained structures that would allow greater sharing among overlapping tasks such as doctor and dentist visits.

In *Dynamic Memory Revisited* (1999, p. 8), Schank describes the original idea of scripts (original italics):

> Our initial definition of a script was *a structure that describes an appropriate sequence of events in a particular context* or *a predetermined stereotyped sequence of actions that defines a well-known situation* (Schank and Abelson, 1977).

The best-known example of a script from Schank's work is the restaurant script. Schank used this script to create a program called SAM whose goal was to understand stories about a restaurant and to answer questions about such stories. A trivial story might be "John ordered salmon. He left a big tip." Much is left unsaid that we can nevertheless correctly infer, since a visit to a restaurant usually follows a standard template: entering, being shown to a table, sitting down, reading the menu, ordering, eating, and paying all typically occur in that sequence. Schank's program could answer such questions as "Did John sit down?" based on its knowledge of the typical sequence. Visiting a different type

of restaurant (such as a sushi bar) might contain a slightly different order of events, and this was represented in a script using different "tracks".

It is instructive to look at Schank's evolving position. I agree with Schank that something like scripts (in some form) is necessary for human-level cognition, and I also agree with his diagnosis about what is wrong with his earlier position. Schank's description of how this initial notion of scripts later felt inadequate is relevant to our discussion, and I quote three paragraphs from it (pp. 8-9):

> Restaurant stories being neither plentiful nor very interesting, we began to look for new domains after we had initially demonstrated the power of script-based processing in our computer programs. We chose car accidents because of their ubiquity in newspapers and their essential simplicity, and we began to alter SAM to handle these. Immediately we ran into the problem of what exactly a script was. Is there a car accident script? A computer could certainly use one to help it process such stories, but that would not imply that most people naturally would have acquired such a script. People who have never been in a car accident would not have a car accident script in the same sense that they might be said to have a restaurant script. Certainly, the method of acquisition would be vastly different. Furthermore, the ordered, step-by-step nature of the script, that is, its essential stereotypical nature due to common cultural convention, was different.
>
> This was emphasized in the way that a car accident script actually could be used to handle newspaper stories. Whenever a car accident occurred, we had to expect at least an ambulance script, a hospital emergency room script, a police report, possibly a subsequent trial script, and perhaps others as well, to be present. Were all these things really scripts? And, if they were, why was it that they seemed so different from the restaurant script in acquisition, use, and predictive power? To put it another way, it seemed all right to say that people know that in the restaurant you can either read a menu and order, or stand in line for your food. We felt justified in saying that there were many different *tracks* to a restaurant script, but that each of these tracks was essentially a form of the larger script. That is, they were like each other in important ways and might be expected to be stored with each other within the same overall outer structure in memory.
>
> But what of accidents? Was there a general accident script of which collisions, accidental shootings, and falling out of windows were different tracks? Alternatively, was there a vehicle accident script of which those involving cars, trucks, and motorcycles were different tracks? Or was there a car accident script of which one car hitting an obstruction, two cars colliding, and chain reactions were different tracks?

Later in the book (p. 111), Schank suggests how to move away from scripts by using structures having finer granularity:

> What I am suggesting, then, is that a lot of knowledge previously theorized to have been stored as part of the dentist script is, in reality, part of other memory structures used in understanding a story involving a dentist visit. [….] Although it may be possible to collect all the expectations we have about a complex event into one complex structure, such a structure does not actually exist in memory. Instead, the expectations are distributed in smaller, shareable units.

8.2.3 Challenges of Implementing at a Finer Granularity

Implementing at a finer granularity, though appealing, has its own problems. The main problem is that it is harder to achieve, both programmatically and conceptually. As a case study, we will consider Seqsee's implementation of the task of describing a solution once it has been discovered. Although this implementation works, it is a hack, and as hacks always are, it is inflexible and inelegant.

I should point out, however, that this task is somewhat atypical of tasks handled in Seqsee by a collection of codelets. Here, much more so than in other parts of Seqsee, it is important that codelets run in a particular order. However, I should also point out that if other codelets that are currently doing too much are split into a collection of codelets, relative order would become important for these as well. Take, for example, the single codelet that, given the description, creates the group "(1) (1 2)…(1 2 3 4 5)". If each successive piece of that group were generated by a different codelet, these codelets would need to run in a highly coordinated fashion.

In broad brushstrokes, here is how a solution is currently described. A codelet of the family "Describe Solution" orchestrates the many codelets involved. Let us call this codelet "Codelet 1". It adds to the Coderack the codelet that it wants to run next (for example, it may add a codelet that lists the initial blemish — initial terms that Seqsee has not been able to make sense of). It sets the urgency of this new codelet to be extremely high (hundreds of times larger than the largest "regular" urgency), thereby virtually guaranteeing that it is chosen next. This codelet, in turn, may add another codelet with an equally high

urgency, thereby nearly ensuring that *that* codelet is run next. In this fashion, the correct relative order is maintained. However, this is achieved at a high cost, since we are effectively suppressing the benefits that having multiple codelets in the Coderack gives us. Henceforth, I will refer to this technique of using high-urgency codelets to ensure seriality by the uninspired name "Hack 1".

This method suffers from another problem as well. After describing the initial blemish, Seqsee also needs to describe the rest of the solution, and Codelet 1 should launch other codelets to describe, for instance, the blocks that the sequence consists of, categories that these blocks belong to, and how one block can be changed into the next. Thus, after the codelet that described the initial blemish finishes its task, Codelet 1 should launch other codelets to continue the description of the solution. Codelet 1, however, cannot stick around to do this additional task: codelets in Seqsee run one after the other, and Codelet 1 is not present any longer. What I have programmed instead is that another codelet of the same family ("Describe Solution") is created that knows exactly where the previous "Describe Solution" codelet left off. I refer to this technique of using multiple codelets to simulate a codelet that needs to launch a series of codelets — where each codelet in the series needs to be launched after the previous has run to completion — as "Hack 2".

It hardly needs to be pointed out that these hacks are cognitively implausible. It is desirable to split the large task of describing an entire solution into many codelets, but ensuring the correct order for these Codelets to run is a nontrivial task. And correct order is needed here: if the dozen or so Codelets were to run in any other order, the resulting description would be unintelligible. If, further, we had made the granularity of describing the solution even finer, the task of rendering a single English sentence would have been carried out by many codelets, and their being out of order would have generated ungrammatical garbage.

In a nutshell, what we are trying to achieve here is the ability to do sequential tasks in a massively parallel system. Although Seqsee's architecture is excellent for distributed tasks, it seems lacking for serial tasks. Although this parallel system is itself implemented on serial hardware, this is a difficult task

and is reminiscent of the following central paragraph from *Consciousness Explained* (Dennett, 1991, p. 210):

> Human consciousness is itself a huge complex of memes that can be best understood as the operation of a "*von Neumannesque*" virtual machine implemented in the parallel architecture of the brain that was not designed for any such activities. The powers of this virtual machine vastly enhance the underlying powers of the organic hardware on which it runs, but at the same time many of its most curious features, and especially its limitations, can be explained as the byproducts of the kludges that make possible this curious but effective use of an existing organ for novel purposes.

As Dennett goes on to explain, *von Neumannesque* refers to a serial architecture such as the one that exists in a typical present-day computer. In a sense, that entire book is an attempt at describing how the highly parallel human mind is at all able to carry out serial tasks that take months or years to complete (such as building bridges or completing Ph.D.'s).

How to impart to Seqsee the ability to do sequential tasks in the right way (that is, without using hacks such as the one above) is an important question to ask and to try to answer.

8.2.4 A Codelet for Multiplication?

Let's look more closely at multiplication, and think about whether multiplication is a script — that is, do we have an algorithm that we rigidly follow in order to multiply? If people do follow such an algorithm, then it is scientifically acceptable to implement it as a single codelet.

It may appear at first blush that we do follow a predetermined recipe for multiplication. Certainly, multiplication is *taught* as a recipe, and children who are just learning the concept will usually follow this method. However, with experience, multiplication begins to look less and less rigid. To multiply by 100, for instance, we merely add a couple of zeros at the end, and this already departs from the "standard" procedure. Multiplying by 50 may similarly be achieved by appending a pair of zeros and halving the result. With more experience, we may even get shortcuts for such things as squaring a number ending in "5" (such as 75). Here, the shortcut is to drop the "5", multiply what is

left ("7") by its successor ("8"), and tack on a "25" at the end (here, we will end up with 5625). This technique can more generally be used if the two numbers are identical except for the last digit, and the last digits add up to 10 (instead of appending "25", append the product of the last digits — 72 times 78 is 5616). The more frequently we need to do a certain task, it seems, the larger our repertoire of shortcuts, and the steps that we actually take become less rigid and less formulaic.

How can we make the system carry out multiplication in a more human-like way? My view (undoubtedly biased) is that a Seqsee-like system should be used — with all the usual trappings, including a Coderack, a Workspace, a stream of thought, and so forth. This might seem to be overkill, especially since the ability to multiply is hardwired into digital computers and can be accessed using a single line of assembly code. However, that shortcut is unavailable to us if we want to emulate the often plodding (but occasionally swift) way in which we humans multiply numbers. It is a mistake to underestimate the complex cognitive acrobatics involved even in such a humble operation as multiplication. Multiplication, as carried out by people, may involve sensitivity to such facts as "124 is nearly one eighth of 1000", a fact that leads to a shortcut (generated on the fly, no less!) for multiplying by 124.

I will not describe how a Seqsee-like system could implement multiplication, but I *will* describe (in Section 8.4) how a "shortcut-rich" problem in the Seek-Whence domain can be handled.

Section 8.3 MICRO-SEQSEE

As was already mentioned in Section 8.1.1, Seqsee has several optional features that are kept turned off by default because these features step on each other's toes. People have a repertoire of categories that is many orders of magnitude larger than Seqsee's, and yet this seems to cause them no trouble. It is worth pondering how people manage to deal with millions of categories effortlessly, and how Seqsee might be scaled up to beyond just a few dozen categories.

8.3.1 Relevant Knowledge

In order to explore this question, let us continue our silly example from the previous section, talking about the embodied sequence-extrapolating chess-playing dentist-visiting descendant of Seqsee, and let us watch it play a game of chess where four moves have already been made by each side. Because of its broad expertise, our program has access to a very large number of categories. Which of these does it really need at this stage?

Certainly, much of its knowledge of sequences of integers is irrelevant here. *Ascending group* is a notion virtually absent from chess. The notion of a *group* is extremely general and is definitely present, but the things that one does with groups in understanding sequences are different from what one does with them in chess. In chess, seeing three white pawns on adjacent squares does not set up an expectation that a fourth one would also be present — patterns that are important in the two domains are distinct. Many of the patterns that Seqsee must perceive in order to understand sequences have no analogue in chess. A chess player must be sensitive to different patterns.

Most of the program's knowledge of chess is also, I will contend, unneeded at this point in the game. For instance, much of what it knows about the endgame (such as how to mate a solitary king using just a bishop and a knight) plays no role whatsoever at this early stage of the game. What might be relevant after quite a few more moves is the fact that it is *possible* to force a mate using only those two pieces, but precisely how to do that is knowhow that need not be wheeled out and dusted off just yet. In fact, not even all knowledge of chess openings is needed: the first few moves that have occurred have already ruled out many of the openings, and yet more are ruled out by factors such as whether it wants to play an attacking or a strategic game.

8.3.2 Mental Spaces

One possible way in which knowledge may be organized is suggested by Gilles Fauconnier's notion of *mental spaces* (Fauconnier, 1985), which, to my mind, is a notion that is closely related to a host of others, such as Ervin Goffman's notion of frameworks (Goffman, 1986).

Let us begin with a simple example in order to ground this discussion. Fauconnier is mostly concerned with mental spaces set up by linguistic expressions such as "In *Lord of the Rings*, dwarves and elves do not like each other." Depending on what the listener knows about the book, the preamble in that sentence ("In *Lord of the Rings*") sets up a world in which the rest of the sentence can be interpreted. Such a preamble is called a *space builder*. Other examples of linguistic space builders — phrases that set the stage and provide a context in which to interpret what follows — include "John says", "If I were a millionaire", and "Maybe".

I am not particularly interested here in mental spaces set up by linguistic expressions, however. Mental spaces exist also in how we think and act. Here is a rather trivial example of the sort of space I have in mind. When I go to work each day, I take the shuttle. I need to check the schedule, walk to the stop, show the driver my ID, and get off at Google. However, the shuttle does not run on weekends, and if I need to get to Google on a Saturday, I might take the city bus. I need to check the schedule on a *different* website, walk to a *different* stop, purchase the ticket, and get off at a *different* location. I have done both of these enough times not to have to give any conscious thought to what I'm doing. Getting to work on a weekday and on the weekend are two slightly different spaces, and in each space what I do and how I do it are slightly different. The space builder here is figuring out whether or not it's a weekday.

If this example felt too trivial, here is one from the world of geometry. When one is working on a problem in plane geometry, there are several techniques of attacking the problem — coordinate geometry (which imposes a coordinate system onto the figure, and which enables the use of numerical calculations), trigonometry (which makes use of trigonometric relations between various parts of the figure, and which makes it tempting and often easy to use trigonometric identities to complete the proof), pure geometry (which for the most part eschews numerical calculations, instead using geometric theorems), or even the kinematic method (which proves the theorem for a special case, such as an equilateral triangle, and then shows that deforming the triangle maintains the validity of the theorem). Each technique here really consists of a

number of related sub-techniques (in the case of coordinate geometry, for example, one may use Cartesian coordinates, polar coordinates, or yet other coordinate systems). Each of these systems has its own biases — what aspects of the problem to pay attention to, what subproblems to set up, how to identify which subtechniques to employ, and even how to write down the solution once it is discovered. Each technique is a mental space.

I cannot resist giving one final example of mental spaces, although I recognize that it will doubtless seem esoteric to some readers, but this is a type of activity that, as a software engineer, I have to engage in every single day: reading computer programs. I include this example, despite its esotericness, because the subspaces involved are demonstrably quite different from each other. Consider the following line of Perl code, which I urge the reader to try and read (I did not even use the word *understand*, just the word *read*):

```
$string =~ s#([\(\)\[\]\<\>\{\}\|])# $1 #g;
```

Unless you have considerable experience with reading Perl code, you doubtless struggled with even splitting that line into "words", just as somebody learning a new language has difficulty in finding word boundaries in a rapid conversation. What that line of code does is add spaces around all instances of any of the eight characters "**()[]<>{}**" within the variable "$string". Although I will not explain fully what each part of that line means, I will describe enough to show how, in understanding that line, mental spaces are created.

Concepts that are involved in understanding an entire Perl program include *modules*, *functions,* and *variables*. When this line is encountered in reading the program, and the two symbols "=~" are seen, an experienced programmer will know that what follows is a regular expression[25]. Within a regular expression, the three concepts mentioned above play no role whatsoever; other notions such as *captures*, *escaping*, and *character classes* become relevant. Even though regular expressions form a subset of Perl, reading a regular expression feels very different from reading the rest of the program. Individual characters take on a different meaning within the regular expression.

[25] Regular expressions are compact ways of describing patterns within strings.

Blanks become significant (that is, blanks change the meaning of what is being read, unlike in most other parts of Perl, where changing a single space into ten spaces has no effect whatsoever). Yet another aspect of regular expressions is different: in most parts of Perl, the symbol "#" denotes the start of a comment, and the compiler will ignore the remainder of the line. Not so in a regular expression. In the line above, what is to be replaced is present between the first pair of "#"s, and what it is to be replaced by is found between the second and the third "#", and thus the "#"s are front and center to an understanding of the line — a far cry from being unimportant stuff to be ignored. One needs to put on different-colored glasses in order to read regular expressions, as it were.

It is as if on seeing "=~", we were suddenly (although temporarily) transported into a different world (that of regular expressions), which works differently and where different concepts and strategies are required for understanding.

There is much more to this nesting of worlds (the world of regular expressions within the world of a Perl program). Within a regular expression, the presence of a left square bracket signals the beginning of a character class, and character classes have their own internal logic, which is different from that of regular expressions or from regular Perl. Individual characters take on a different meaning within a character class. When reading a Perl program, we have to keep in mind what world we are in, so to speak, or the program won't make sense to us.

Recast in terms of mental spaces, the above can be described as saying that "=~" is a *space builder* (that is, a word or a phrase that signals the construction of a particular type of mental space), and the expression is understood within this space. Within that space, a left square bracket is another space builder creating a yet different type of space. I should point out that, outside of a regular expression, a left square bracket is a space builder for a different type of entity (an array literal) instead of a character class. This is an example of how deeply a mental space alters meanings of entities within it — in the mental space "regular expressions", not only do characters such as "#" take

on a different level of significance, but also what entities act as space builders and what spaces they build changes.

Understanding that dense line of code requires a good understanding of what is and is not important, and where. A newcomer to Perl — even somebody who knows all the rules — will be overwhelmed. There is just too much to know in understanding a program, and keeping all that knowledge in the head at once is hard. Of course, the trick is to not have to keep it all there at once, and mental spaces make this possible.

Only a tiny subset of all that is known about a particular domain is actually relevant (or used) at a given time. Being able to zoom in on (or summon up) the right subset is crucial to intelligence. Even though my description of the Perl line above was long, not all the details were needed simultaneously for me to understand that line.

Not only was just a small fraction of my Perl knowledge needed at a given time, the subproblems that I had to solve (such as understanding the fragment between the first pair of "**#**"s) were nearly isolated from the rest of the task. At each moment, I was dealing with only a small fragment of the problem, and could therefore temporarily ignore most of what I knew.

This made the task much easier. This is a theme sure to be repeated in any act of problem-solving. When Seqsee is solving some sequence, it may need to construct an ascending group starting at 1 and ending at 7. Once the need for this group has been identified, then during the actual construction, most of its knowledge of sequences is unneeded, and what is happening in the rest of the sequence can be temporarily ignored.

Mental spaces are ubiquitous in thought. Let us revisit the Ramanujan episode from Chapter 5 and view it in this new light. I quote from that chapter:

> Once, Ramanujan was cooking when an elated friend stopped by, wishing to share the solution of a problem that he had seen in a magazine and had solved. When the friend read out the problem, Ramanujan said — still stirring his curry — "please take down the solution", and dictated a continued fraction that gave infinitely many solutions to the problem. On being asked how he had so quickly solved the problem, Ramanujan said, "It was clear

that a continued fraction was needed. I just had to figure out which one."

By the time Ramanujan had concluded that a continued fraction was required, the most difficult part was done. Trying to find the specific continued fraction relevant to the given problem is a much simpler subproblem — one can safely ignore dozens of competing theories about how to solve the problem. Moreover, experience with finding other continued fractions can be brought to bear, and the solving process can be nearly mechanical (for Ramanujan, in any case).

Seqsee does not currently have any sort of mental spaces. Or, put differently, Seqsee has just a single mental space in which all operations are carried out. Seqsee would greatly benefit from being able to create different spaces for different types of subproblems, as we shall see in the next section.

Section 8.4 Mental Spaces in Extrapolating Sequences

I will now present my vision for the next version of Seqsee. The crucial idea here is to be able to replace a large codelet with a more flexible, Seqsee-like system. For this discussion, I will try to show how construction of groups in Seqsee could be done more cleanly using mental spaces.

Consider the following eight descriptions of groups, the last five of which stipulate that the successive terms are connected by the following mapping:

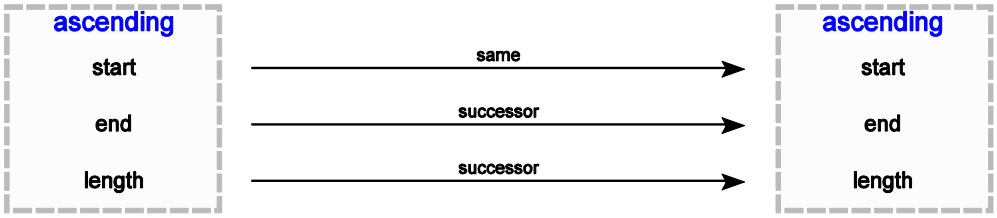


Figure 8.1 Mapping between successive terms of groups #4 through #8

1. An ascending group of length 4, starting at "1" (that is, the group "1 2 3 4")

2. An ascending group of length 4, ending at “7” (i.e., “4 5 6 7”)
3. An ascending group starting at “3” and ending at “7” (i.e., “3 4 5 6 7”)
4. A group containing three top-level elements that is based on the mapping shown in Figure 8.1, starting at “(2 3 4)”. That description represents the group “(2 3 4) (2 3 4 5) (2 3 4 5 6)”.
5. A group containing four top-level elements that is based on the mapping shown in Figure 8.1, ending at “(3 4 5 6 7)”. This description stands for the group “(3 4) (3 4 5) (3 4 5 6) (3 4 5 6 7)”.
6. A group based on the same mapping as above, starting at “(2 3 4)” and ending at “(2 3 4 5 6)”. This is “(2 3 4) (2 3 4 5) (2 3 4 5 6)”.
7. A group based on the same mapping as above, starting at “(2 3 4)” and ending at “(3 4 5 6)”. No such group exists.
8. A group containing four top-level elements that is based on the mapping shown in Figure 8.1, ending at “(4 5)”. Again, there is no such group.

Here is how Seqsee currently constructs the first three groups described above. Given any two of *start*, *end*, or *length* of an ascending group, the *start* and the *end* can be calculated if they are not already provided. This involves addition or subtraction (since, if the starting point is not given, it is one more than the end minus the length), and Seqsee uses such operations, even though they are outside the Seek-Whence domain.

The five other groups (i.e., Groups 4 through 8) are a different story, and of these, Seqsee can construct only Group #4. To do so, it uses the starting point as the first item of the group being constructed, and uses the mapping to construct subsequent items until enough have been collected. For Groups #5 and #8, where the *length* and the *end* have been provided, Seqsee would need to construct the inverse of the mapping and work backward. Groups #6 and #7, where the *start* and *end* are known, require a far greater sophistication: the naïve way of successively applying the mapping beginning at the starting point until the end is reached may sometimes never halt, as happens in Group #7.

Implementing a cognitively realistic ability to construct such groups is nontrivial. In fact, as in the case of multiplication above, people discover and use shortcuts. There is no fixed recipe for detecting the impossibility of a description. However, a person may notice that this particular mapping always increases the length by one. In Group #7, after the mapping has been applied once (to "(2 3 4)"), a length-four group is obtained. This group, "(2 3 4 5)", is of the same size without being the same group as the end (which is "(3 4 5 6)"). It is possible to realize that no matter how many times the mapping is applied repeatedly, "(3 4 5 6)" will never show up, as each subsequent group will be longer. Similarly, it is possible to discover that the first item in Group #8 must be of length -1, an outlandish notion.

Seqsee manages only the most trivial case, represented by Group #4. A single subroutine does the construction of groups and, needless to say, squeezing all this complex machinery (such as being able to invent shortcuts on the fly) within a single subroutine is difficult and foolhardy. I will now attempt to sketch an alternative.

I begin by describing an independent Seqsee-like system that creates a group given a description, independently of any sequence being extrapolated, and then I look at how this can be incorporated into a system that extrapolates sequences.

The input to our system will include the mapping between consecutive elements and some subset of the following: the start, the end, and the length. The codelet families include the following: "Start and Length Scout" (which checks for the presence of both the start and the length, and if these are present, launches a codelet that will repeatedly apply the mapping enough times to produce the group), and "Start and End Scout" (which checks for the presence of both the start and the end, and launches a codelet that can specifically handle this case).

We have not achieved much yet: a few paragraphs back I described the start-end case as requiring more than a single codelet, but just now I described it as using a single codelet. Let us rectify the situation by creating a mental

space in which to solve group-creation problems where start and end are given. Again, we will delay the discussion of how this fits in relative to the previous subsystem.

This new subsystem — whose job is to create a group given the start, the end, and the mapping between successive elements — includes several codelet families, including "Find Next Subgroup" (which applies the mapping to the most recently calculated subgroup), "Have We Reached the End?" (which checks if the most recent subgroup is the same as the end, and therefore whether the entire group has been constructed), and "Are We Closer to the End?" (which checks if the application of the mapping has taken us closer to the end).

Let us step back momentarily and see what this carving-out of a subproblem has given us. In designing Seqsee, I have had to strike a balance between generality of codelets and tuning them specifically to the domain. It makes no sense in the current version of Seqsee to have a codelet family "In the Process of Constructing a Group Given Start and End, Are We Closer to the End?". If such highly specific codelets were present, we would have hundreds of codelet families, and figuring out their relative urgencies and trying to ensure the right mix of codelets in the Coderack at various points in the run would be an impossible task. However, in this much smaller domain (i.e., in the mental space where Seqsee is focusing, to the exclusion of all else, on the task of constructing a group given its end-points), it is possible for us to have codelets that are far better targeted. Since we will have only a small number of codelet families in this mental space, setting their relative urgencies is much simpler.

A second benefit is that implementing sequential actions is easier. The hack ("Hack 1") described in Section 8.2.3 of launching codelets with extremely high urgency in order to ensure that they would run in a specific order was required because of the presence of other codelets unrelated to the task of describing the solution that could interrupt the desired process by being run midway through a process of description of the solution. As the mental subspace has its own Coderack that contains only codelets specific to this task, no such hack is needed. Furthermore, since the mental subspace will have its own Workspace, we can add a small "state" object to this Workspace, which will

track what has already been done and which can therefore coordinate various codelets. Given the tininess of the task tackled by a mental space, this object will be small. We will therefore not require "Hack 2" mentioned in Section 8.2.3.

There is one other benefit, which may be the most important: a better version of long-term memory can now be implemented. Since a subspace solves a more specific problem, and since the same type of subspace would be generated for similar subproblems, such subspaces are prime locations in which to store strategies related to solving that specific problem. I have not thought through the matter of exactly how such learning of shortcuts could be implemented, but it certainly seems an easier challenge, given the greater uniformity in problems solved and the limited size of each subspace.

I have left unanswered the question of how the different mental spaces would fit together, and this is because I am not sure what the answer will look like. I have some ideas, but, like everything else in this chapter, they have not been tested in a working program. One possibility is that Seqsee will start out in the default space. Depending on what types of structures it perceives, it may create other spaces that would coexist with the original space, and each of them would get a different-sized slice of Seqsee's attention, much as multiple programs take turns while running together on a serial computer. Each subspace would need to have a way to evaluate when its task was done (or to realize that it is not doable, or is not worth the effort), and each one would also be able to decide when to stop. When a subspace stopped, it would be able to add a codelet to the space that generated it, which would allow the parent space to take appropriate action. I know this is all very blurry, but which of the many conceivable ways of having many mental spaces interact smoothly works best is at least partly an empirical question.

Using mental spaces, it seems to me, has the potential of making Seqsee more flexible, more easily extensible, and more capable in its task — although one would also need to be continuously on the lookout for the expert-systems trap. Super-expertise has never been the primary goal of Seqsee, and it should not be allowed to overshadow the goal of creating a cognitively realistic system.

Section 8.5 CONTRIBUTIONS OF THIS RESEARCH

This dissertation has described a set of ideas about perception, situational pressures, contexts, labeling, concepts, categorization, memory, fringes, and the stream of thought. The computer program that this dissertation has produced is able to extend a wide range of sequences in a humanlike way.

Seqsee's architecture is based on those of Copycat and other FARG programs. Ideas such as codelets, the Coderack, the Workspace, and the Slipnet (here called the long-term memory) were already present in Copycat and in earlier programs. There are, however, a number of novel features in Seqsee:

1. The most significant is the incorporation of William James' notions of the fringe and the stream of thought. These ideas contribute temporality to Seqsee — what happened in the recent past becomes significant in a way that was not present in earlier programs. It also makes serendipitous occurrences more likely, and in a fashion very similar to how taking advantage of serendipity happens in the case of people.
2. Seqsee can generate very specific pressures, an ability that I have argued to be crucial to cognition. This is not just a quantitative improvement, but represents a qualitatively different type of focusing, enabled by the stream of thought, as described in Section 5.7.
3. Seqsee can create more complex groups, can see more complex analogies between objects, can see longer-distance relationships, and has a larger and more diverse set of categories than either Copycat or Metacat did.
4. Seqsee is relatively easily extensible, and small, local changes can be made in one specific part of the source code that will allow Seqsee to perceive many new sequences (without modifying the deep principles of the underlying architecture in any way).
5. Seqsee can see an entity as something else, and it can use this ability in extending sequences.

6. Seqsee's long-term memory is organized in such a way that having previously seen a sequence makes Seqsee quicker on other *similar* sequences. Copycat and Metacat do not make use of prior runs when solving a problem. The spreading of activation in long-term memory is also different in Seqsee, and this helps to resolve some problems faced by earlier models.
7. Not least, a rich set of visualization tools has been built into Seqsee, allowing the designer (or the user) a detailed look into what Seqsee is doing, or what it did in its most recent run. It can be used to see what structures Seqsee has created, what categories it believes are relevant, how strongly it holds such beliefs, how such beliefs changed over time in a previous run, where the program's attention was distributed, and so forth. Metacat also has a rich set of such tools, and it was a major inspiration.

Apart from the ideas already implemented, I am excited about starting work on the next set of features that I have described earlier in this chapter.

This work represents, to my mind, at least a small step forward in understanding how human cognition can faithfully be modeled on a computer.

Appendix A READING

I have frequently used the phrase “Seqsee reads a group”. The verb “read” is clearly used metaphorically, since Seqsee has no visual system with which to read. Also, the word may seem ill-fitted to describe Seqsee looking at a group that the program itself created. I wish therefore to spell out the imagery behind that word.

Imagine a person standing in front of a very long blackboard trying to extrapolate a sequence whose one thousand initial terms are written — with an indelible chalk — on the board. Imagine further that the sequence is not so trivial to understand that a single reading would suffice, and that the person has colorful chalks with which to make notes, to draw ovals, or to annotate the sequence in any fashion. Since it is all but impossible to keep this many terms in the head at once, one may expect the solver to read some terms multiple times, to make copious notes, and to re-read the notes as well. Andy Clark (2001, p. 42) describes the process of writing an academic paper in a similar way:

> Take, for example, the process of writing an academic paper. You work long and hard and at day’s end you are happy. Being a good physicalist, you assume that all the credit for the final intellectual product belongs to your brain: the seat of human reason. But you are too generous by far. For what really happened was (perhaps) more like this. The brain supported some rereading of old texts, materials and notes. While rereading these, it responded by generating a few fragmentary ideas and criticisms. These ideas and criticisms were then stored as more marks on paper, in margins, on computer discs, etc. The brain then played a role in reorganizing these data on clean sheets, adding new on-line reactions and ideas. The cycle of reading, responding and external reorganization is repeated, again and again. Finally, there is a product. A story, argument or theory. But this intellectual product owes a lot to those repeated loops out into the environment. Credit belongs to the embodied, embedded agent in the world. The naked biological brain is just a part (albeit a crucial and special part) of a spatially and temporally extended process, involving lots of extraneural operations, whose joint action creates the intellectual product. There is thus a real sense (or so I would argue) in which the notion of the ‘problem-solving engine’ is really the notion of the whole caboodle: the brain and body operating within an environmental setting.

Seqsee is a much simpler system, and it cannot make use of external physical scaffoldings the way we humans do. However, it is still useful to

consider Seqsee as containing something akin to a blackboard on which to make notes. When Seqsee is given the first few terms of a sequence as input, in a sense it "knows" what these terms are — after all, it was just told what these are. However, it is not yet *aware* of these terms. It has the terms stored internally — in a variable — and this is very roughly analogous to the person having *seen* but not *read* the terms on the board. Seqsee does not really "remember" all the groups that it has created (though, in a trivial sense, it does have the information stored internally, and it does use this information for display purposes). Seqsee can therefore be said to "read" some of the information stored and to become aware of it — and soon to forget it.

Although the exact details of what it has read remain with Seqsee for a very short time (only for the duration of a single codelet), as was described in Chapter 5, an abstract notion, a gist, or a summary of what it has read lingers on as the fringe.

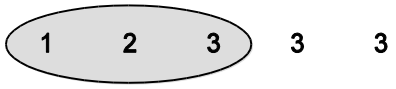
Appendix B RESTRICTIONS ON GROUPS IN SEQSEE

Groups are shown by the ovals that we have seen in most sequences. Grouping allows us to treat several objects as a single unit, and it is indispensable in the understanding of a sequence. There are certain restrictions on what kinds of groups the current version of Seqsee can construct, and in this section we shall describe these constraints, along with their justification, consequences, and workarounds.

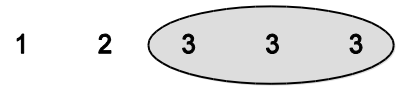
There are three restrictions on groups: “no holes allowed”, “subgroups should not overlap”, and “at most one squint per group”. The “no holes” restriction can be elaborated as stating that the elements of a group should form a contiguous chunk of the sequence. Thus, in the sequence “1 2 3 4 5”, (the current version of) Seqsee is not able to create a group comprised of “1”, “2”, and “4” without also including the “3”.

Now let us look at the second constraint. As can be seen in many sequences, an oval may contain another oval embedded within it. This inner oval is a subgroup of the outer oval. In its current incarnation, Seqsee cannot compose two overlapping groups to form a larger group, as shown in the three sequences below. Seqsee will be able to create the group shown in Sequence 134 and the group shown in Sequence 135, but not the outer group in Sequence 136.

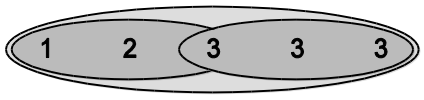
Sequence 134.



Sequence 135.

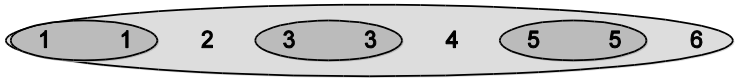


Sequence 136.



Finally, the third constraint states that Seqsee cannot see the group in Sequence 137 as being a funny version of "1 2 3 4 5 6".

Sequence 137.



These constraints simplify the actions of several types of codelets, and thereby simplify the implementation. For example, disallowing holes greatly simplifies the process of extending a group. If holes were allowed, a rightward extension could be located *anywhere* to the right, as opposed to *immediately* to the right, as is illustrated in Sequence 138. In that sequence, imagine the hole-containing group composed of the two ovals. What is its rightward extension? Perhaps, it seems, we need a "3 3 3" next. But where? There may or may not be a hole after the "2 2". Even if we are somehow sure that there must be another hole at that location, should that hole be of size 1 (as the previous hole was), or of size 2 (as might be imagined, since the ovals in our group are also elongating)?

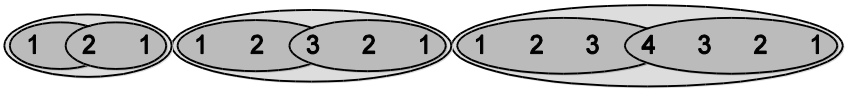
Sequence 138.

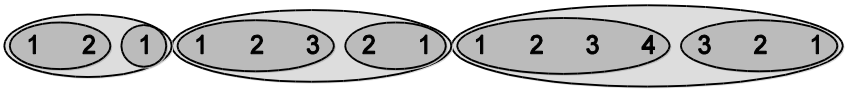
Lifting the second constraint creates almost identical obstacles as allowing holes does. When an analogy between two overlapping groups is extended, the next group (i.e., the extension) should probably overlap the second, but again, it is not obvious how much overlap there should be.

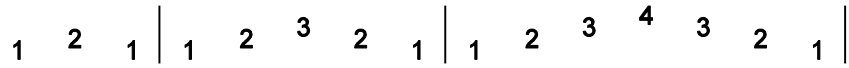
These difficulties are not insurmountable, but they are by no means trivial. A future version of Seqsee may lift these restrictions. However, even in the current version, there are workarounds. These workarounds were not added to address this particular issue of restrictions on groups, but are instead a result of Seqsee's ability to see a single sequence in multiple ways. Seqsee cannot see the sequence "1 7 2 8 3 9" as consisting of the two derived (infinite) groups "1 2 3…" and "7 8 9…" (since both contain holes). However, it is able to see this as Sequence 139.

Sequence 139. 

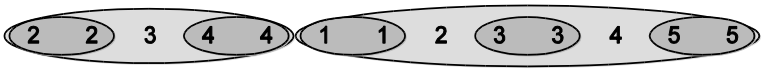
Each successive oval here is obtained from the previous oval by applying the rule "take the successor of the first item and of the second item". Similarly, Seqsee cannot see Sequence 140 as shown below (because overlapping subgroups are disallowed), but it *can* see it as shown in Sequence 141. Sequence 141 is a shallower interpretation, in that it misses the relatedness of the ascending and the descending pieces. However, Seqsee may also see the same sequence via the category *mountain*, as is shown in Sequence 142.

Sequence 140. 

Sequence 141. 

Sequence 142. 

Removing the third restriction also complicates the implementation. For instance, how may one describe what changed from the first outer oval to the second outer oval in the sequence below?

Sequence 143. 

How can one describe how the doubled elements moved and changed? Changes are easier to describe if a squint may occur in at most one spot per group. For the current version of Seqsee, only this relatively simpler problem has been tackled.

Appendix C SIMON AND KOTOVSKY'S WORK

Herbert Simon and Kenneth Kotovsky (1963) described a model of human acquisition of concepts for "sequential patterns", as they called them. In terms of its goals, their work was similar to Seqsee — both projects were aimed at describing how concepts relating to patterns may be formed, and both attempt to achieve this end by the implementation of a computer program that, in some qualitative ways, behaves as people do.

The differences between the two approaches are immense, however. The most obvious difference is the complexity of the sequences used: Simon and Kotovsky used only sequences that I have called quasi-periodic, and even within that restricted set, they limited themselves to sequences where the lengths of components were unchanging. Although this restriction was nowhere made explicit, each of the 25 sequences they presented is of this type. Thus, Sequence 144, but not Sequence 145, is a part of the domain that they considered:

Sequence 144. 1 7 19 2 6 20 3 5 21

Sequence 145. 1 7 1 2 8 1 2 3 9

Now that I have pointed out the most obvious difference between the two (apart from the even more obvious difference that they used letters of the alphabet instead of numbers), let us turn to deeper issues separating the two explorations.

In Simon and Kotovsky's own words, an important component of their work was the creation of a simple *language* for characterizing serial patterns. They wished to reduce the sequence to a simple specification that a computer (or a person) could use to mechanically churn out indefinitely many terms of it. There is no perception involved, once this specification has been generated. Although this is quite easy to achieve for sequences of such simplicity as they considered, it is possible to think of sequences whose rule for extrapolation requires creative perception and thinking. In other words, a machine to *extend*

such a sequence would be of comparable complexity to a machine that could *discover the rule* behind the sequence (i.e., that could "seek whence" the sequence came).

In Seqsee, the processes of discovery and of extrapolation go hand in hand. It would be very difficult to tease the two apart in this architecture, and it would be the wrong thing to attempt. What processes in our daily cognitive life, after all, can be expressed as declarative, easily mechanizable sets of instructions? Although books such as *How to Think Like Leonardo da Vinci* (Gelb, 1998) attempt to simplify the working of the Master so that anybody can follow in his footsteps, or so it is claimed, even the author of that book would hesitate before claiming that a computer program could read the book and be enlightened. I fail to see why sequences are necessarily so much simpler that their extrapolation can be reduced to a simple computer program. The sequences chosen by Simon and Kotovsky (and even many of the sequences that Seqsee currently solves) can indeed be reduced in this fashion, but that argument does not extend to all sequential patterns that a person might be able to extend, and this desire to construct cut-and-dried formulas constitutes an unbridgeable chasm between our philosophical positions.

The second serious difference between the two approaches concerns the process of discovery of relevant categories. The most relevant category for the sequences considered by Simon and Kotovsky was the periodicity of the sequence. This was discovered by trying out different periods until some period could explain the sequence, a strategy that seems cognitively extremely unrealistic to me, since I believe that in a sequence such as Sequence 146 below, no person would try to see whether the sequence can profitably be split into chunks of size 2. This technique of detecting the period, moreover, makes it impossible for the computer program to understand, for example, Sequence 145. This approach is diametrically opposed to what has been done in Seqsee, where a huge amount of effort has been expended in modeling mechanisms for detecting categories *without using brute-force tests* for all sorts of categories. Seqsee does a parallel terraced scan and "sniffs" or "smells" that certain

categories may be relevant, and only then does it explore those categories more deeply.

Sequence 146. **1 9 19 2 10 20 3 11 21**

I end this appendix with a short Perl program that I created to solve *all* of the sequences used as targets by Simon and Kotovsky. I created this program — which certainly cannot be called well-written — immediately after reading their paper and being baffled by the quote (original italics): "We postulate: *normal adult human beings have stored in memory a program capable of interpreting and executing descriptions of serial patterns. In its essential structure, the program is like the one we have just described*." I strongly felt that there is certainly far more to cognition in the world of sequences than this.

```
sub seq{join('', map { chr(65 +($_[0] - 1+ ($_-1) * $_[2])%26)} (1..$_[1]));}
my $ARGC = scalar(@ARGV);
my @nums = map { ord($_) - 64 } @ARGV;
sub check_periodicity{
    my ( $p ) = @_; my @cont;
  LOOP: for my $i (0..$p-1) {
        my $s = int( ($ARGC-$i)/$p);
        my $substr = join('',@ARGV[map {$i+$p*$_} (0..$s-1)]);
        for (-3..3) {
            if ($substr eq seq($nums[$i],$s,$_)) {
                push @cont, $_; next LOOP;
            }
        }
        return;
    }
    return \@cont;
}
sub print_out{
    my ( $period, $cont_ref ) = @_;
    for (0..50) {
        print chr(65+ ($nums[$_%$period]+
                       $cont_ref->[$_%$period]*int($_/$period)-1)%26);
    }
    print "\n";
}
for (1..4) {
    if (my $cont_ref = check_periodicity($_)) {
        print_out($_, $cont_ref); exit;
    }
}
print "Failed!\n";
```

Figure C.1 My program to solve all the sequences in Simon and Kotovsky (1963)

Appendix D HUMAN PERFORMANCE

In the fall of 2007, in Robert Goldstone's laboratory, I presented 48 psychology undergraduate students with the first few terms of a number of integer sequences, and their task was to provide at least the five next terms.

Every student was shown a total of 40 sequences. Here is a series of screenshots indicating what they saw. Initially, they saw a button labeled "click to view sequence".

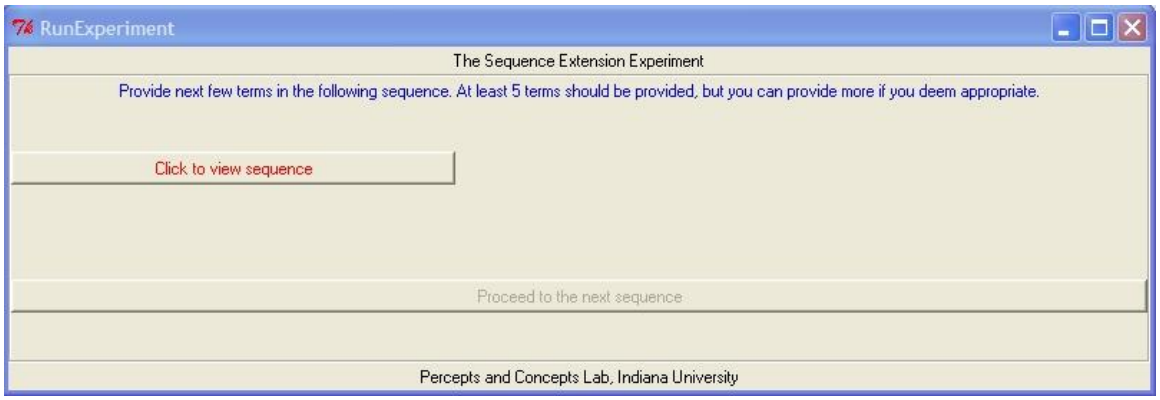


Figure D.1 Initial screen for each sequence

After they had clicked this button, the next screen containing the initial terms of the sequence became visible, and this screen had a button labeled "I am ready to enter the answer".

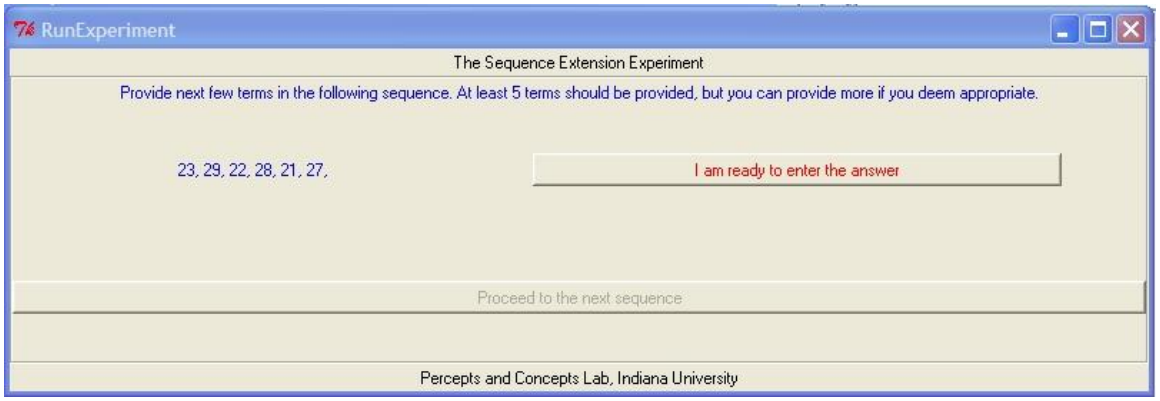


Figure D.2 Screen shown after clicking "Click to view sequence"

When this second button was clicked, 10 boxes in which to enter the next terms became visible, but only the first was active (the other 9 were grayed out). The time elapsed between the clicking of the button on the first screen shown above and the typing of first digit of the first new term was treated as "the time needed to understand the sequence". As each new term was entered, the next box became active. When at least five terms had been entered, the subject was allowed to proceed to the next sequence.

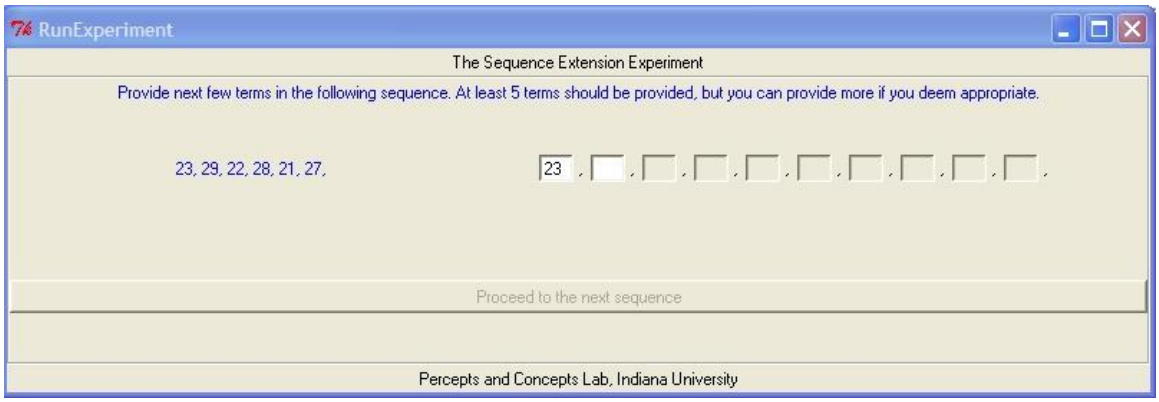


Figure D.3 Entering a term opens up next box

The instruction sheet given to the students before the start of the experiment provided an escape hatch for any sequence: they could just enter five zeros to indicate that they could not figure out how to continue the sequence.

The table below lists all the sequences shown to the participants. These have been grouped into sets of related sequences. From each set, either precisely one sequence was shown to each student, or all of them were shown to each student. The order of sequences was randomized.

For each sequence, the table below shows how many students the sequence was shown to. It also shows what fraction of students got the wrong answer — the assumption being that all the sequences had a unique good continuation. This assumption is patently false for Set #12.2, and therefore that set does not contain data about correctness. The time shown in the last column is the average time in seconds, averaged only over the correct responses.

The sequence “2 2 3 3 3 4 4 4 4” is present in two sets — #9 and #13 — and therefore the numbers of times that sequences in those sets were shown do not add up to 48.

Table D.1 Summary of subjects’ performance

Set	Sequence	# times seen	% Wrong	Time
1.1				
	1 4 1 5 1 6	24	4.17	5.99
	1 1 1 2 1 3	24	70.83	10.58
1.2				
	1 7 2 7 3 7 4 7	25	16.00	5.13
	6 7 7 7 8 7 9 7	23	43.48	16.87
2				
	7 1 7 7 2 7 7 3 7 7 4 7	13	38.46	8.12
	2 1 2 2 2 2 2 3 2 2 4 2 2 5 2	20	50.00	32.58
	2 1 2 2 2 2 2 3 2 2 4 2	15	60.00	44.63
3				
	21 19 22 20 23 21	48	41.67	10.98
	23 29 22 28 21 27	48	10.42	13.50
	11 19 12 18 13 17	48	41.67	9.44
	17 10 16 11 15 12	48	47.92	8.49
4				
	1 2 3 1 2 3	48	2.08	2.54
	1 1 2 1 2 3	48	70.83	11.24
5				
	24 24 25 24 25 26	48	77.08	11.89
	19 19 18 19 18 17	48	79.17	13.68
	17 18 17 19 18 17	48	56.25	15.07
	15 14 15 13 14 15	48	60.42	11.88
6				
	1 2 2 3 4 5 6 1 2 3 3 3 4 5 6 1 2 3 4 4 4 4 5 6	20	25.00	17.28
	1 2 3 3 4 5 6 1 2 3 4 4 4 5 6 1 2 3 4 5 5 5 5 6	28	14.29	17.56
7				
	1 18 17 1 2 18 17 16 1 2 3 18 17 16 15	48	20.83	13.48
	1 18 1 17 2 18 1 17 2 16 3 18 1 17 2 16	48	87.50	45.76

Table D.2 Performance, continued

Set	Sequence	# times seen	% Wrong	Time
8				
	4 7 10 13 16	48	16.67	5.30
	1 4 9 16 25 36	48	31.25	17.90
	1 9 25 49 81	48	52.08	26.35
	2 5 10 17 26	48	43.75	22.29
9				
	3 3 3 4 4 4 4 5 5 5 5 5	10	10.00	3.73
	2 2 3 3 3 4 4 4 4	30	3.33	4.81
	3 3 4 4 4 5 5 5 5	18	22.22	3.62
	2 2 2 3 3 3 3 4 4 4 4 4	9	0.00	5.26
10				
	5 5 5 6 6 6 6 7 7 7 7 7	10	10.00	4.48
	5 5 5 5 5 5 4 4 4 4 4 3 3 3 3	15	6.67	9.35
	5 5 5 4 4 4 4 3 3 3 3 3	10	10.00	7.81
	5 5 5 5 5 5 6 6 6 6 6 7 7 7 7	13	23.08	11.82
11				
	6 2 6 7 2 7 8 2 8	14	14.29	14.14
	6 2 6 5 2 5 4 2 4	10	30.00	12.22
	6 2 6 7 2 5 8 2 4	11	63.64	19.69
	6 2 6 5 2 7 4 2 8	13	61.54	29.98
12.1				
	17 1 2 3 4 5 6	27	62.96	8.85
	1 1 2 2 3 3 3 4 4 5 5	21	95.24	4.86
12.2				
	1 2 3 4 18 5 6 7 8	20		
	1 2 3 4 18 6 7 8 9	28		
13				
	1 2 2 3 3 3	25	16.00	2.63
	2 2 3 3 3 4 4 4 4	30	3.33	4.81
14				
	1 1 1 2 3 1 1 1 2 3 4 1 1 1 2 3 4 5	13	15.38	8.99
	1 1 1 2 3 1 2 2 2 3 4 1 2 3 3 3 4 5	14	7.14	17.65
	1 1 2 3 1 1 1 2 3 4 1 1 1 1 2 3 4 5	10	10.00	18.41
	1 1 2 3 1 2 2 2 3 4 1 2 3 3 3 3 4 5	11	27.27	15.13

Table D.3 Performance, continued

Set	Sequence	# times seen	% Wrong	Time
15				
	1 7 2 8 3 9	15	20.00	4.64
	1 7 19 2 8 20 3 9 21	14	21.43	25.52
	1 7 1 2 8 1 2 3 9	19	31.58	18.96
16				
	1 7 19 2 8 19 20 3 9 19 20 21	25	40.00	22.53
	1 7 19 2 8 20 21 3 9 22 23 24	23	52.17	29.16
17				
	1 2 3 4 5 6	48	14.58	2.80
	1 7 1 7 1 7 1 7 1 7	48	2.08	3.70
	1 7 19 1 7 19 1 7 19 1 7 19	48	6.25	7.67
	1 0 0 1 1 1 0 0 0 0	48	16.67	8.65
	1 2 3 3 4 4 5 5 5 6 6 6	48	22.92	12.50
	5 4 3 4 5 4 3 4 5	48	18.75	10.20
	1 2 3 17 3 4 5 17 5 6 7 17	48	33.33	12.35
	1 2 1 1 2 3 2 1 1 2 3 4 3 2 1	48	33.33	28.46
18				
	1 1 2 3 1 2 2 3 1 2 3 3 1 1 2 3 1 2 2 3	48	20.83	20.74
	1 1 2 3 1 2 2 3 1 2 3 3 1 2 2 3 1 1 2 3	48	35.42	23.62
	1 2 2 3 3 1 1 2 3 3 4 4 1 1 2 2 3 4 4 5 5	48	33.33	23.74

BIBLIOGRAPHY

Andel, Pekvan (1994). Anatomy of the Unsought Finding. Serendipity: Origin, History, Domains, Traditions, Appearances, Patterns and Programmability. *The British Journal for the Philosophy of Science, 45*(2), 631.

Ariely, Dan (2008). *Predictably Irrational: The Hidden Forces that Shape our Decisions*. New York: Harper.

Austin, James H. (1978). *Chase, Chance, and Creativity: The Lucky Art of Novelty*. New York: Columbia University Press.

Baars, Bernard J. (1988). *A Cognitive Theory of Consciousness*. New York: Cambridge University Press.

Baratz, Robert S. (2001). Hearing on Swindlers, Hucksters and Snake Oil Salesmen: The Hype and Hope of Marketing Anti-Aging Products to Seniors. *United States Senate Special Committee on Aging*.

Borges, Jorge Luis (1962). Pierre Menard, Author of the Quixote. *Labyrinths: Selected Stories and Other Writings*, 36–44.

Campbell, Murray, A. Joseph Hoane, and Feng-hsiung Hsu (2002). Deep Blue. *Artificial Intelligence, 134* (1-2), 57-83.

Cannon, Walter B. (1940). The Role of Chance in Discovery. *The Scientific Monthly, 50*(3), 204-209.

Cannon, Walter B. (1945). *The Way of an Investigator: A Scientist's Experiences in Medical Research*. New York: W. W. Norton & Company.

Clark, Andy (2001). *Mindware: An Introduction to the Philosophy of Cognitive Science*. New York: Oxford University Press.

Conklin, E. Jeffrey and William Weil (1997). Wicked Problems: Naming the Pain in Organizations, from http://www.leanconstruction.org/pdf/wicked.pdf

Defays, Daniel (1988). *L'esprit en friche: Les foisonnements de l'intelligence artificielle*. Liege, Belgium: Pierre Mardaga.

DeGrace, Peter and Leslie H. Stahl (1990). *Wicked Problems, Righteous Solutions*. Upper Saddle River, N.J.: Yourdon Press.

Dennett, Daniel C. (1991). *Consciousness Explained*. Boston, Mass.: Little, Brown, and Co.

Einstein, Albert (1949). Autobiographical Notes. In Paul Arthur Schilpp (Ed.), *Albert Einstein: Philosopher-Scientist*. New York: Harper & Row.

Ellis, John M. (1993). *Language, Thought, and Logic*. Evanston, Illinois: Northwestern University Press.

Esar, Evan (1978). *The Comic Encyclopedia: A Library of the Literature and History of Humor Containing Thousands of Gags, Sayings, and Stories*. Garden City, N.Y.: Doubleday.

Fauconnier, Gilles (1985). *Mental Spaces: Aspects of Meaning Construction in Natural Language*. Cambridge, Mass.: MIT Press.

Feynman, Richard P. (1965). *The Character of Physical Law*. Cambridge, Mass.: MIT Press.

Fitzpatrick, Geraldine (2003). *The Locales Framework: Understanding and Designing for Wicked Problems*. Dordrecht, The Netherlands: Kluwer Academic Publishers.

Foundalis, Harry E. (2006). *Phaeaco: A Cognitive Architecture Inspired by Bongard's Problems.* Doctoral dissertation, Computer Science Department, Indiana University, Bloomington, Indiana.

French, Robert M. (1995). *The Subtlety of Sameness: A Theory and Computer Model of Analogy-Making*. Cambridge, Mass.: MIT Press.

Freud, Sigmund (2003). *The Joke and Its Relation to the Unconscious*: Penguin Classics.

Gelb, Michael (1998). *How to Think Like Leonardo Da Vinci: Seven Steps to Genius Every Day*. New York: Delacorte Press.

Goffman, Erving (1986). *Frame Analysis: An Essay on the Organization of Experience*. Boston, Mass.: Northeastern University Press.

Hofstadter, Douglas R. (1983). The Architecture of Jumbo. In Ryszard Michalski, Jaime Carbonell, and Thomas Mitchell (Eds.), *Proceedings of the International Machine Learning Workshop* (pp. 161-170). Urbana, Illinois: University of Illinois.

Hofstadter, Douglas R. (1985). *Metamagical Themas: Questing for the Essence of Mind and Pattern*. New York: Basic Books.

Hofstadter, Douglas R. (1995). On Seeing A's and Seeing As. *Stanford Humanities Review, 4*(2), 109-121.

Hofstadter, Douglas R., Marsha J. Meredith, and Gary A. Clossman (1982). Seek-Whence: A Project in Pattern Understanding. CRCC Report No. 3, Center for Research on Concepts and Cognition, Indiana University, Bloomington, IN.

Hofstadter, Douglas R. and the Members of the Fluid Analogies Research Group (1995). *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. New York: Basic Books.

James, William (1890). *The Principles of Psychology*. New York: Dover Publications.

Lakoff, George (1987). *Women, Fire, and Dangerous Things*. Chicago: University of Chicago Press.

Marshall, James B. (1999). *Metacat: A Self-watching Cognitive Architecture for Analogy-making and High-level Perception.* Doctoral dissertation, Computer Science Department, Indiana University, Bloomington, Indiana.

McGraw, Gary (1995). *Letter Spirit (Part One): Emergent High-level Perception of Letters Using Fluid Concepts.* Doctoral dissertation, Computer Science Department, Indiana University, Bloomington, Indiana.

Meredith, Marsha J. (1986). *Seek-Whence: A Model of Pattern Perception.* Doctoral dissertation, Computer Science Department, Indiana University, Bloomington, Indiana.

Meyers, Morton A. (2007). *Happy Accidents: Serendipity in Modern Medical Breakthroughs* (1st ed.). New York: Arcade Publications.

Mitchell, Melanie (1990). *Copycat: A Computer Model of High-level Perception and Conceptual Slippage in Analogy-making.* Doctoral dissertation, Computer Science Department, University of Michigan, Ann Arbor, Michigan.

Mitchell, Melanie (1993). *Analogy Making as Perception*. Cambridge, Mass.: Bradford Books/MIT Press.

Norman, Donald A. (2002). *The Design of Everyday Things*. New York: Basic Books.

Pólya, George (1954a). *Induction and Analogy in Mathematics*. Princeton, N.J.: Princeton University Press.

Pólya, George (1954b). *Mathematics and Plausible Reasoning* (Vol. 2: Patterns of Plausible Inference). Princeton, N.J.: Princeton University Press.

Rehling, John A. (2001). *Letter Spirit (Part Two): Modeling Creativity in a Visual Domain.* Doctoral dissertation, Computer Science Department, Indiana University, Bloomington, Indiana.

Rittel, Horst W. J. and Melvin M. Webber (1972). *Dilemmas in a General Theory of Planning*. Berkeley: Institute of Urban & Regional Development.

Roberts, Royston M. (1989). *Serendipity: Accidental Discoveries in Science*. New York: John Wiley.

Schank, Roger C. (1999). *Dynamic Memory Revisited*. New York: Cambridge University Press.

Schank, Roger C. and Robert P. Abelson (1977). *Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures*. Hillsdale, N.J.: Lawrence Erlbaum.

Sewell, William H. and J. Michael Armer (1966). Neighborhood Context and College Plans. *American Sociological Review, 31*(2), 159-168.

Shannon, Claude E. (1950). Programming a Computer for Playing Chess. *Philosophical Magazine, 41*(7), 256-275.

de Sigura, Ruy López (1561). *Libro de la invención liberal y arte del juego del axedrez* [The Book of Liberal Invention and the Art of the Game of Chess]: Alcalá.

Silman, Jeremy (1998). *The Complete Book of Chess Strategy*. Los Angeles: Siles Press.

Silman, Jeremy (1999). *The Amateur's Mind: Turning Chess Misconceptions into Chess Mastery*. Los Angeles: Siles Press.

Simon, Herbert A. and Kenneth Kotovsky (1963). Human Acquisition of Concepts for Sequential Patterns. *Psychological Review, 70*(6), 534-546.

Sloane, N. J. A. (2007a). The help file for Superseeker, from http://www.research.att.com/~njas/sequences/superhelp.txt

Sloane, N. J. A. (2007b). The On-Line Encyclopedia of Integer Sequences, from http://www.research.att.com/~njas/sequences/

Sloane, N. J. A. and Simon Plouffe (1995). *Encyclopedia of Integer Sequences*. San Diego: Academic Press.

Sobel, Kenith V. and Randolph Blake (2002). How Context Influences Predominance During Binocular Rivalry. *Perception, 31*(7), 813-824.

Villalba, Juan J., Frederick D. Provenza, and R. E. Banner (2002). Influence of Macronutrients and Polyethylene Glycol on Intake of a Quebracho Tannin Diet by Sheep and Goats. *Journal of Animal Science, 80*(12), 3154.

Wells, Herbert G. (1915). *The Research Magnificent*: New York: The Macmillan Company.

Wells, Herman B (1980). *Being Lucky: Reminiscences and Reflections*. Bloomington, Indiana: Indiana University Press.

Wittgenstein, Ludwig and G. E. M. Anscombe (1953/2001). *Philosophical Investigations* (the German text, with a revised English translation). Malden, Mass.: Blackwell.

Curriculum Vitae

Abhijit Mahabal
amahabal@gmail.com

Education

- Integrated M.Sc. in Mathematics and Computer Applications, IIT Delhi, 1995-2000
- M.S. in Computer Science, Indiana University, 2007
- Ph.D. in Computer Science and Cognitive Science, Indiana University, 2010

Teaching Experience

- Associate Instructor and Research Assistant, 2002-2008. Assisted in the teaching of
    - Graduate classes: “Theory of Computation”, “Analysis and Design of Algorithms”, and “Specification and Verification”.
    - Undergraduate classes: “Scheme”, “Discrete Structures”, and “Artificial Intelligence”.

Publications

- Mahabal, Abhijit (2007). Benefits of Incorporating a Stream of Thought: A Case Study In D.S. McNamara and J.G. Trafton (Eds.), *Proceedings of the 29th Annual Cognitive Science Society* (pp. 1259-1264). Nashville, TN.

Awards

- National Talent Scholarship, 1993.
- Top 25 in national physics exam (IAPT), 1995.
- Bronze medal at the International Math Olympiad, Hong Kong, 1994, and in Toronto, 1995.