Letter Spirit (part one): Emergent High-Level Perception of Letters Using Fluid Concepts

Gary E. McGraw, Jr.

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

September 1995

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

Dr. Douglas R. Hofstadter
(Principal Adviser)

Dr. Michael Gasser

Dr. Robert Goldstone

Dr. Robert F. Port

Bloomington, Indiana
September 1995.

Dr. Gregory J.E. Rawlins

For Amy and Jack.

ABSTRACT

This thesis presents initial work on the Letter Spirit project, with a cognitive model of letter perception as its centerpiece. The Letter Spirit project aims to model central aspects of human high-level perception and creativity on a computer, focusing on the creative act of artistic letter-design. The aim is to model the process of rendering the 26 lowercase letters of the roman alphabet in many different, internally coherent styles. Two orthogonal aspects of letterforms are basic to the project: the *categorical sameness* possessed by instances of a single *letter* in various styles (*e.g.*, the letter 'a' in Times, Palatino, and Helvetica), and the *stylistic sameness* possessed by instances of various letters in a single style, or *spirit* (*e.g.*, the letters 'a', 'k', and 'q' in Times alone). Starting with one or more seed letters representing the beginnings of a style, the full Letter Spirit program will attempt to create the rest of the alphabet in such a way that all 26 letters share the same style. Letters in the domain are formed exclusively from straight segments defined by a sparse grid in order to make decisions smaller in number and more discrete. This restriction allows much of low-level vision to be bypassed and forces concentration on higher-level cognitive processing, particularly the abstract and context-dependent nature of letter concepts.

The overall architecture of Letter Spirit, based on the principles of emergent computation and fluid concepts, has been carefully developed and is presented in Part I. Creating a gridfont is an interwoven process of guesswork and evaluation — the "central feedback loop of creativity". The notion of continual evaluation of a creative product throughout the design process, and its necessary foundation in perception,

have been largely overlooked in cognitive models of creativity. In Letter Spirit, we address this issue head-on.

In order to be truly and autonomously creative, a program must do its own *assessment* and *adjustment* of any large-scale work-in-progress throughout the course of design. To this end, we have focused initial Letter Spirit work on a sophisticated and subtle model of gridletter perception. Our emergent model of letter perception is based on the hypothesis that letter categories are made up of conceptual constituents called *roles*. Roles exert clear top-down influence on the segmentation of letterforms into structural components. Part II fleshes out the role hypothesis, discusses a model of letter recognition implementing it, and presents psychological evidence supporting the existence of roles in human perception of letters. Empirical results support the applicability of our emergent, cognitive architecture to the task of letter recognition. Because we are ultimately interested in the design of letters (and the creative process as a whole), an effort has been made to develop a model rich enough to recognize and understand in terms of constituent roles a large range of letterforms, including letters that push at the fringes of their categories.

ACKNOWLEDGMENTS

In 1985 I came across Douglas Hofstadter's book called *Gödel, Escher, Bach* at my friend Steve Wassel's house. Unaware of the fact that I would not return it for two years (and only then under threat of severe penalty), Steve let me borrow it. I'm glad he did. Around the same time, I was enrolled in a Philosophy course taught by Paul Humphreys at UVa called "Computers, Minds, and Brains". Arguing at the philosophical level about Searle's ubiquitous Chinese Room and whether or not Artificial Intelligence was even possible was an interesting and at the same time frustrating experience. Much to the dismay of some of the Philosophy faculty at UVa, Doug Hofstadter's writings influenced my thinking about machines and cognition very deeply. Reading *GEB*, along with the "systems reply" article in another of Hofstadter's books, *The Mind's I*, made me want to leave Philosophy, get my hands dirty, and do some modeling for myself. I decided to seek out a graduate program in AI.

I was fortunate to end up in Bloomington and become enrolled in both the Computer Science Department and the brand new Cognitive Science program. I am grateful to the Computer Science department for betting on a Philosophy major and admitting me into the CS PhD program. As fate would have it, Doug Hofstadter had recently been lured back from the University of Michigan, the Fluid Analogies Research Group (FARG) had been transplanted to "the lab" at 510 North Fess, and the Cognitive Science program was off to a promising beginning.

Doug Hofstadter's influence on this thesis is visible at all levels, from its conceptual core to the clarity of its prose. Many thanks to Doug for being a truly great mentor, a generous academic provider, a friend, and a constant source of ideas and opinions. Doug dreamed up Letter Spirit many years ago — I am glad he encouraged me to bite off a piece of it and get the ball rolling.

It was Doug who gathered together the FARGonauts. If I were to invent a research group, FARG would be it. The FARG environment is outstanding in many ways: interesting and fun people constantly bubbling with ideas, state-of-the-art machines with infinite disk space, Doug's inimitable and deeply insightful thoughts about cognition, a relaxed and fun environment where working on side-projects is encouraged, etc. My time at Indiana has been intellectually stimulating, thought provoking, challenging, and, above all, fun, mostly thanks to FARG. In the early days, FARG included Melanie Mitchell (author of Copycat who inspired all us junior FARGonauts with her outstanding work), David Moser (wordmaster supreme and constant source of intelligent humor) and Bob French (ringmaster of the FARGlets with a connectionist bent). Many thanks to them for the encouragement and for blazing the trial.

I joined FARG around the same time as my favorite deluded zombie, Dave Chalmers. I'll never forget arguing philosophy with Dave in between bouts of "major action". Thanks for being wrong, Dave. Others who have made FARG good include Jim Marshall (a perfect person to consult for a reality check on crazy ideas, and a source of well-considered thoughts and opinions) and John Rehling (who will soon carry the Letter Spirit torch himself). Jim made several important suggestions during the development of the program and read several chapters of this thesis. John and I worked together closely designing, implementing, and analyzing the psych experiments.

Many others have passed through FARG over the years, leaving their own marks on my thinking. Thanks to: David Leake, Peter Suber, Morten Christianson, Eric Dietrich, Terry Jones, Wang Pei, Don Byrd, Jim Herriot, Liane Gabora, and my undergraduate slave Dan Drasin. Also special thanks to Helga Keller who has kept FARG running and pointed in the right direction administratively for years.

The ragazzi — Jim, John, and I — imported FARG to Italy for a year during Doug's sabbatical in Trento. While at the Istituto per la Ricerca Scientifica et Tecnologica (IRST), we were joined in spirit by Achille Varzi. Special thanks to Achille for keeping us sane at IRST. Also thanks to Alex Cozzi, Alessandro Cimatti, Elena Not, Enrico Franconi, Freddie Oursin, and Francesco Ricci.

The saddest time in my graduate career occurred in Italy when Carol Hofstadter suddenly died in a coma after emergency brain surgery. Though Carol's loss will always leave a hole in the fabric of Danny, Monica, and Doug's lives, they have coped admirably well since losing her. My heart is with them. Carol and Italy will always remain intertwined for me.

Lisa Meeden and Doug Blank, two of my graduate school colleagues (both die-hard connectionists) made life fun in Bloomington and kept my symbolic tendencies in check. We once built a robot car together for fun and then turned it into research. Doug Blank has been my closest friend here, conversing with me for hours about cognition, learning, connectionism and analogy. Thanks to Blank for reading many of the chapters and helping me hone the argumentation.

There are many others who made graduate school great. They include: Rich Wyckoff and Sushil Louis (my partners in GA/CBR crime), Dedaimia Whitney, Bob Montante, Charles Daffinger, Eric Jeschke (fellow Brown-countian), John Nienart, Jon Rossie, Laura Blankenship, Michael Chui, and Bill Dueber. Also thanks to the Ultimate crowd for the exercise and for supporting Amy's addiction. And special thanks to "Guitar" John Blair, Brian Ridgley, Dennis Scoville, and Judy Augsburg for the music that made life sweet.

The other members of my committee both challenged me and guided me throughout my thesis work. Gregory Rawlins taught me how to do research and encouraged me to begin publishing early in my career. Rob Goldstone kept me abreast of current work in Psychology and helped put this work on a strong psychological foundation.

Mike Gasser and Bob Port influenced my thinking about cognitive science, learning, and connectionism. Other professors at Indiana and at other places have influenced my work. They include: David Leake, Jonathan Mills, Dan Friedman (who taught the best CS course I ever took), Giles Fauconnier, and Dave Touretsky.

Of course, none of this would have been possible without my parents. Mom and dad's loving encouragement will serve as a model for how to raise baby Jack. My dad's PhD and his title "Dr. Gary E. McGraw" inspired me at a young age to be a scientist. (Now there're two of us in the world!) My brothers Walt and Chris recharge my batteries every time I see them at the Stick.

Once Amy stopped supporting my school habit in order to adventure with me in Italy (meanwhile deciding to become a mom), our extended family helped us get by. Thanks to the McGraw's, the Kiczales's, and the Barley's for underwriting the writing.

Finally, I want to thank Amy, the best mommy in the world and the love of my life. Amy made possible our luxurious Bean Blossom lifestyle. Then she gave up her career to come to Italy. She has been waiting for me to get out of "gradual school" for a long time now. I'm glad she waited.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

PREFACE

This thesis is organized in two parts. Part I provides an overview of the entire Letter Spirit project — a cognitive model of typeface design focusing on creativity. Part II presents an implemented computer model of letter recognition that will serve as Letter Spirit's letter-perception system. Though the solid core of this thesis is certainly the letter-recognition work presented in Part II, Part I serves to put this work in perspective, providing a justification of our[1] emphasis on perception in early work on Letter Spirit.

Chapter 1 provides an overview of the Letter Spirit project. After the orthogonal categories of *Letter* (corresponding to categorical sameness) and *Spirit* (corresponding to stylistic sameness) are introduced, the gridfont microdomain is carefully explained. Special care was taken during the development of the grid to ensure that the deep, conceptual-level issues raised by the problem of typeface design were kept intact while surface-level details were left by the wayside. The grid serves to focus attention away from aspects of letter design that tend to bog down letter-design programs in domain-specific details, such as exact curvature of lines, line thicknesses, tapering serifs, and the like. Constraining letters to a grid brings the cognitive issues to the forefront — especially the nature of fluid concepts and the conceptual nature of style. A theory of letter concepts, based on the idea that letter concepts are made up of constituent

[1]Throughout this thesis the first person will always be written as "we" (possessive "our"). This usage reflects the fact that some parts of this research are the result of a group effort between myself, Doug Hofstadter, John Rehling, and Rob Goldstone. I chose to use "we" in all cases (even those where the work was performed solely by me) for the sake of stylistic consistency.

roles, is proposed. Chapter 1 also introduces the architecture of Letter Spirit and discusses its four main "agents": the Imaginer, the Drafter, the Examiner, and the Abstractor. The "central feedback loop of creativity" is introduced, and the crucial role of perception in creative models is discussed.

The aims of Letter Spirit can be viewed both narrowly and broadly, as the program can be seen both as a typeface design system (the narrow view) and as a model of creativity (the broad view). Chapters 2 and 3 take first one, then the other of these two perspectives on the project, respectively. Chapter 2 surveys some typeface-design programs, discussing their creative and non-creative aspects. Comparisons between Letter Spirit and several other typeface-design programs serve to emphasize the fact that what we are after is more than an engineering solution to typeface design. What we are after is a model of human creativity.

Chapter 3 delves more deeply into the idea of automating the creative process. Hard questions of *autonomy* — that is, of a program *making its own decisions* — provide a framework for understanding models of creativity in the AI literature. We propose that the sort of autonomy that we want to capture arises from the following five attributes: a rich, highly-interconnected knowledge base; active, dynamic concepts; conceptual-level exploration; the ability to perceive and assess tentative output in order to adjust it; and gradual convergence on a large-scale creative product through an interleaved process of suggestion and assessment. Special emphasis is placed on the idea of high-level perception, without which a program would be unable to *know what it is doing.* A related thread that runs through Chapter 3 involves the tradeoff between flexibility on the one hand and control on the other. Balancing these two opposing forces in a creative program turns out to be a hard problem. The parallel terraced scan, a way of implementing the emergence of perceptual structure under top-down pressure from the conceptual level, provides an elegant solution to this problem. Chapter 3 serves to justify our emphasis on perception in early implementation work on Letter Spirit.

Without a sophisticated understanding of letters at the conceptual level, Letter Spirit would end up as just another "blind" program, relying on its human programmer/user to assess and judge its output.

Part II introduces the centerpiece of this thesis — a computer model of letter perception based on the idea that conceptual roles are the fundamental building blocks of letter concepts. We call this model the "role model". The most general aim of Part II is to provide a thorough understanding of the role model.

Chapter 4 begins with a discussion of the role hypothesis, restating and refining ideas about letter concepts introduced in Chapter 1. The role model, implemented as Letter Spirit's Examiner agent, serves as a concrete test of the role hypothesis. The role model is based on an approach to cognitive modeling in which dynamic, associative, active concepts play a major role. Our role-model program not only categorizes letterforms, but more importantly segments letterforms into natural constituent parts that correspond to the conceptual roles of a letter conceptualization. Roles provide a conceptual handle on the deepest level of style in letterforms. At its deepest level, style has to do with *how roles are filled* and *how roles are inter-related*. Thus a role-based model of letter perception that can uncover this deep-style information and make it available for use in the analogical design of other letterforms is a necessary first step in the implementation of a full-blown Letter Spirit program.

The role model is an emergent, stochastic model in which the processing of each run is done by hundreds of micro-agents (called codelets) that are instantiations of sixteen codelet types. The asynchronous, parallel, local processing done by the codelets implements a parallel terraced scan (as in the role model's predecessor, Copycat) that is applied to our categorization task. Role provide top-down pressures on the kind of emergent perceptual structure built up by codelets. The resulting mix of top-down pressures and bottom-up structure-building makes for a surprisingly flexible model of letter perception.

The three main architectural features of the role model are: 1) a Conceptual Memory made up of a set of highly interconnected, flexible, norm-based concepts; 2) a Visual Focus serving as a subcognitive workspace in which codelet processing — building, destroying and re-arranging perceptual structure — occurs in parallel; and 3) a stochastic Coderack that implements (simulated) codelet parallelism by picking codelets to run nondeterministically, but in a probabilistically-biased way, according to estimates of their relevance to the current situation. The implementation of these three architectural features is the main subject of Chapter 4.

Chapter 5 concerns itself with the performance of the role model. Performance is analyzed at multiple levels, from a "microscopic" view of codelet-level processing on six particular runs, to a "galactic" view of thousands of runs considered together at once. This chapter makes no attempt to hide the weaknesses of the role model, and in fact discusses some of them in detail. The last section of Chapter 5 relates experiments in which main parameters in the role model are systematically varied.

A discussion of competing models of letter recognition makes up the contents of Chapter 6. First, a framework for understanding the widely varying approaches to modeling letter-perception is introduced. This framework posits a continuum with two opposing approaches at either end — a "flat" approach based on low-level features and template matching, and a "structured" approach based on the use of built-in category models having both hierarchical (part/whole) and relational (between-part) structure. The role model falls at the latter end of the spectrum. By contrast, most optical character recognition (OCR) models lie at the former end, since they tend to be based on template matching. Throughout Chapter 6, both OCR models and psychological models of letter recognition are fit into our framework. Special attention is paid to two models in particular: Sanocki's model, which has "attributes" that in some ways are related to our thinking about roles, takes a psychological approach to letter perception; Williams and Hinton's statistically complex deformable-models system takes an engineering approach to the problem. In addition to considering models developed outside our research group, we have developed two "in-house" models to act as competition: Dumrec, a "scruffy" symbolic-AI recognizer based on weighted property-list matching, and Netrec, a three-layer backpropagation network that learns to categorize letterforms based on associative training. These two rudimentary models provide contrasting views of letter perception that we have tested against the role model using the same gridletter datasets.

Chapter 7 concerns itself with human perception of letters and presents the results of two psychological experiments. The first of these explores the gridletter recognition problem. Several subjects were run on a dataset of hundreds of gridletters that included letterforms of various styles (ranging from standard to eccentric). The resulting confusion matrix is richer than those usually found in letter-perception work, since acute variations in style cause more interesting and less artificial errors than do the conditions usually used in such experiments (such as very short stimulus-display times or the use of letterforms displayed far away from a subject). One interesting side issue is to what extent recognition of standard, prototypical letters differs from the recognition of eccentric, highly-stylized letters. The second psychological experiment investigates letterform parsing, probing how people break letterforms into "natural" parts. Together, these two experiments suggest that human letter perception (of single characters, presented alone) is strongly influenced by role-level representations.

Behavior of the role model, Dumrec, Netrec, and humans is compared and contrasted in Chapter 8. The role model shows strong and consistent correlation with our human data, especially when error-making is taken into account. Analysis of Dumrec and Netrec errors suggests that these simpler models are closely based on template matching and low-level feature extraction (two common approaches taken by most OCR programs). Theoretical error matrices developed according to these two particular representational theories — low-level features and template matching

— confirm this view, and shed more light on the kind of processing done by Netrec and Dumrec. The role model is thus empirically shown to be a very strong model of letter recognition. Our results point out fundamental shortcomings in approaches to letter recognition that do not deal with letters as realizations of roles in the same way as people do.

In the final chapter of the thesis, the contributions of this research are brought together in one place. A summary of major results is presented, some minor problems with the role model are covered, and the future of the Letter Spirit project is discussed.

As an intellectual project, Letter Spirit is quite old, the domain having been initially proposed and a computational architecture sketched out as far back as 1980, but as an implemented computer program it is in its infancy. (A skeptic might therefore say that it exists only “in fancy”.) Right now, the program can recognize and understand many letters, but is still very far from being able to create any. We hope this will change over the next few years.

Prior Sources

Some of the work in this thesis has been adapted from previously published articles and chapters. Chapter 1 is drawn almost verbatim from Chapter 10 of *Fluid Concepts and Creative Analogies* [Hofstadter and McGraw, 1995]. Parts of *FCCA*'s Chapter 10 also turn up in Chapters 2 and 3, though in altered form. Also interwoven into Part I is text from several published articles on Letter Spirit including [McGraw and Hofstadter, 1993b], [McGraw and Hofstadter, 1993a], and [McGraw, 1992]. The "Copycat section" of Chapter 3 was adapted from [Hofstadter, 1994] with the permission of its author.

Original versions of the chapters in Part II dealing with human letter perception, Dumrec, and Netrec have also appeared elsewhere in print. Dumrec and Netrec were introduced in an article co-authored with Dan Drasin [McGraw and Drasin, 1993]. The human letter-recognition experiment was originally reported at the 1994 Cognitive Science Conference in Atlanta [McGraw et al., 1994a]. A longer article dealing with the human studies was published as a CRCC technical report [McGraw et al., 1994b]. In addition, parts of Chapter 4 appeared as a conference paper at the 1995 Midwest AI and Cognitive Science Society conference [McGraw, 1995].

Part I

The Letter Spirit Project

CHAPTER ONE

Overview of Letter Spirit

We love letters for their minds, not their bodies.

— Jim Marshall

The Letter Spirit project is an attempt to model central aspects of human creativity on a computer. It is based on the belief that creativity is an automatic outcome of the existence of sufficiently flexible and context-sensitive concepts — what are referred to herein as *fluid concepts*. Accordingly, our goal is to implement a model of fluid concepts in a challenging domain. Not surprisingly, this is a very complex undertaking and requires several types of dynamic memory structures, as well as a sophisticated control structure involving an intimate mixture of bottom-up and top-down processing. Letter Spirit involves a blend of high-level perception[1] and conceptual play that will allow it to *create* in a cognitively plausible fashion. The full realization of such a model will, we believe, shed light on the mechanisms of human creativity.

The specific focus of Letter Spirit is the creative act of artistic letter-design. The aim is to model how the 26 lowercase letters of the Roman alphabet can be rendered in many different but internally coherent styles. Starting with one or more seed letters

[1]High-level perception is the perceptual process whereby sensory data activate concepts at various levels of abstraction. For a discussion of high-level perception see [Chalmers, French, and Hofstadter, 1992] and [Hofstadter and FARG, 1995].

representing the beginnings of a style, the full Letter Spirit program will attempt to create the rest of the alphabet in such a way that all 26 letters share that same style, or *spirit*.

Two important and orthogonal aspects of letterforms are basic to the Letter Spirit project and, in fact, serve as inspiration for its name: the *categorical sameness* possessed by instances of a single **letter** in various styles (*e.g.,* the letter 'a' in Times, Palatino, and Helvetica) and the *stylistic sameness* possessed by instances of various letters in a single style, or **spirit** (*e.g.,* the letters 'a', 'k', and 'q' in Times).

1.1 The motivation of Letter Spirit

1.1.1 Letters as concepts

Letter Spirit focuses on a fundamental question of cognitive science and artificial intelligence: *What are the mechanisms underlying the fluidity of human concepts?* One specific form this question takes in the Letter Spirit research is: *What is the conceptual essence of the letter* 'a'*?*

Letterforms are subtler than people generally realize. Most literate people take letters for granted, and think of them as simple categories. Indeed, most people, if asked, would say that the letter 'a' is a *shape*, not a concept. A closer look reveals an interconnected web of abstractions that make up the idea of 'a' itself. Figure 1.1 shows a number of wildly differing lowercase 'a's, behind all of which lies a single highly abstract idea. Certainly there is nothing like a fixed shape here.

The concept of lowercase 'a' can be thought of as a marriage of two smaller ideas: (1) the idea of a curved umbrella-handle-like curve touching the baseline on the right, and (2) the idea of a smaller c-like curve nestled below the umbrella-handle. (See the first 'a' in Figure 1.3.) These two interrelated conceptual components — what we call

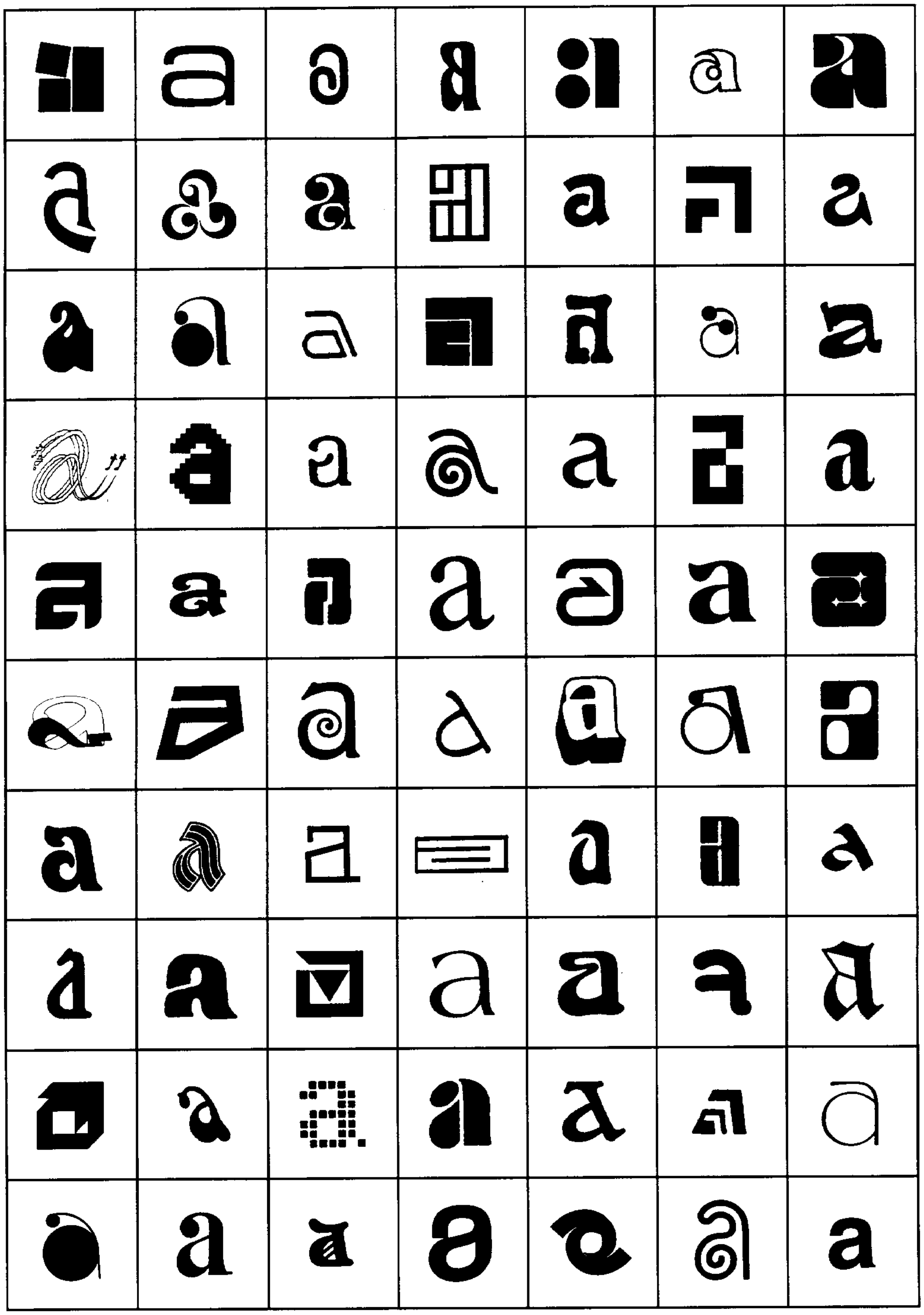


Figure 1.1: Seventy 'a's made for display faces and logos by professional designers. The collection as a whole reveals the abstractness and richness of the concept 'a'. What do all of these decorative and fanciful 'a's have in common? What is the *idea* behind lowercase 'a'?

roles — are not explicit shapes *per se* but are *ideas* supplying norms and constraints on what letter parts should be like, what their acceptable bounds are, how far they can be stretched before they no longer work as letter parts, and how they relate to each other to define an entire letterform.[2]

In order to better distinguish the concept of a letter from the various geometric shapes that may instantiate it, we introduce some terminology. We distinguish four levels of abstraction, starting at a highly abstract level and becoming more concrete as we move towards the actual graphic letterform.

- A *letter concept* is the abstract, shapeless notion of a letter, with which there is no particular style associated whatsoever. This does not mean, of course, that '*a*' and 'a', simply because they are pronounced alike, are represented by one and the same letter concept. Instead, there are two letter concepts here, as different from one another as are the concepts 'a' and 'z', and for the same reason — namely, they implicitly define collections of graphic shapes that have essentially no overlap.
- A *letter conceptualization* is a breakdown of a letter concept in terms of particular roles. For instance, one conceptualization of 'b' might state that 'b' consists of a *post* attached in two particular spots to a left-facing *open bowl* sitting to the right of the post, and on the baseline. A related but different conceptualization for 'b' might state that 'b' consists of a *post* attached in just a single spot to a *closed loop* sitting to its right on the baseline. These two conceptualizations, along with any number of other ways of thinking about what 'b' is, constitute the full letter concept of 'b'. When it comes to making a concrete 'b' in a given style, either some already-known conceptualization may be selected, or else a

[2] The work of Blesser [1973], presented in Chapter 6, discusses a related concept. Psychological evidence supporting the role hypothesis is introduced in Chapter 7. Also see [McGraw et al., 1994a] and [McGraw et al., 1994b].

fresh conceptualization may be spontaneously invented by dissecting and recombining in some new way the roles in an already-known conceptualization. In any case, settling on a letter conceptualization, whether new or old, is a decision at the very deepest level of style.

- A *letter-plan* starts with a specific letter conceptualization and fills in details characterizing *how each role should be realized.* (For example, should a post be tall or short? Should a bowl be fat or thin? Roundish or squarish? Should a crossbar be high or low? And so on.) Since every role has many aspects, a letter-plan is a very complex entity, full of information of many sorts. Although it is far more specific than the previous two levels, a letter-plan is still purely mental, and as such it has an infinite number of different ways to be realized on paper. Decisions made at this level are still at a fairly deep level of style, but not as deep as ones involved in coming up with new conceptualizations.
- A *letterform* is an actual graphic shape drawn on paper to realize a given letter-plan, thus also realizing a particular letter conceptualization and, ultimately, a letter concept.

Letter Spirit is concerned with all of these levels: the interaction between letter concepts, the selection and invention of letter conceptualizations, the devising of letter-plans, and the instantiation of letter-plans as physical, visible letterforms.

Letter categories have flexible but strong boundaries that separate members from non-members. Some of the 'a's in Figure 1.1 are better than others. The weaker ones are found near the boundaries of the concept 'a', and help to define what being an 'a' is. Boundary lines between categories are flexible to the extent that they shift under contextual pressures. What is clearly a non-'a' in some situations may be a passable 'a' in others. Often the drawn shapes deviate from one or more norms, realizing the constituent roles in highly stretched ways.

1.2 Wholes and roles

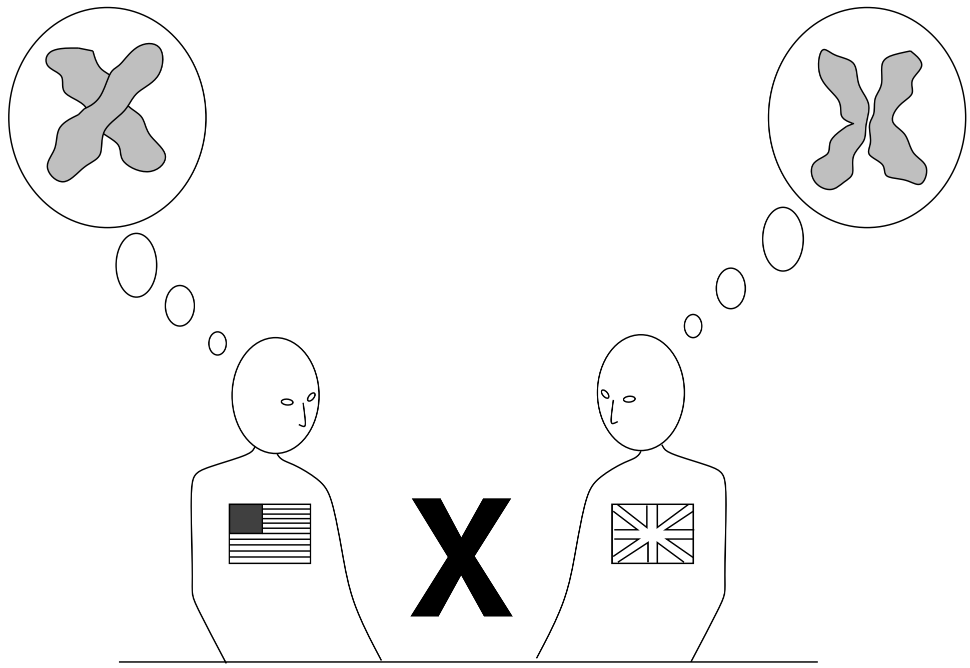


Figure 1.2: The letter 'x' can be conceptualized in many different ways. Here, two people from different cultures who are looking at the same shape perceive it (*i.e.*, conceptualize it) in two different ways. Both conceptualizations shown are common ways of "seeing 'x' ".

A vivid example of the shape/concept distinction involves lowercase 'x'. For most people who learned to write in the United States, the letter concept for 'x' consists of just a single conceptualization, whereby the letter is thought of as consisting of a forward slash and a backward slash of the same size that cross somewhere near the middle. This conceptual breakdown remains the same even when one learns cursive writing. English schoolchildren, by contrast, are taught to draw a lowercase cursive 'x' as a pair of small crescents facing away from each other and "kissing" somewhere in the middle. If we look at a printed 'x' in this way, we are suddenly struck by this new conceptualization. The shape on the paper has not changed, of course, but our inner way of conceiving it is completely fresh and intriguing, and to anyone who is curious about letters, this new vision of 'x'-ness implicitly but quite clearly makes a host of potential suggestions about *other* letters as well.

The conceptual pieces that are critical for 'x'-ness include not just the "slashes" or "crescents", but also such things as the location of the four *tips*, as well as the central *crossing-point* or *kissing-point*. We call these inter-role relational structures *r-roles*, and they are just as important as roles in determining the category membership of a shape. For example, a graphic shape might have a strong exemplar of *post* to the left of a strong exemplar of *loop* and yet the way they interact might still make them form a bad 'b' (think of the fact that if they do not touch, the shape will look more like 'lo' than 'b').

As Figure 1.2 shows, there is more than one way of conceptualizing an 'x'. In fact, there are are many more: 'x' can be thought of as a lambda with a small forward slash on the top right, or as a 'v' sitting on top of an upside-down 'v', and so on. An important question is, are some of these conceptualizations better than others? Is one of them the *right* way to think about 'x'? Our research suggests that some conceptualizations *are* better than others. In most cases, people tend to use fairly normal, default conceptualizations.

Central to the Letter Spirit project is the notion that the internal structure of a category consists of a collection of roles and r-roles, and that category membership at the whole-letter level is determined by membership at this lower level. Each role or r-role has a different degree of *importance* to the letter — the degree to which its presence or absence matters. Of course, different graphic items instantiate a given role more strongly or more weakly than others. In other words, roles, just like *wholes* (complete letters), are concepts in their own right, with somewhat nebulous boundaries. The difference is that membership in a role is easier to characterize than membership in a whole, so that reducing wholes to collections of interacting roles is a step forward in

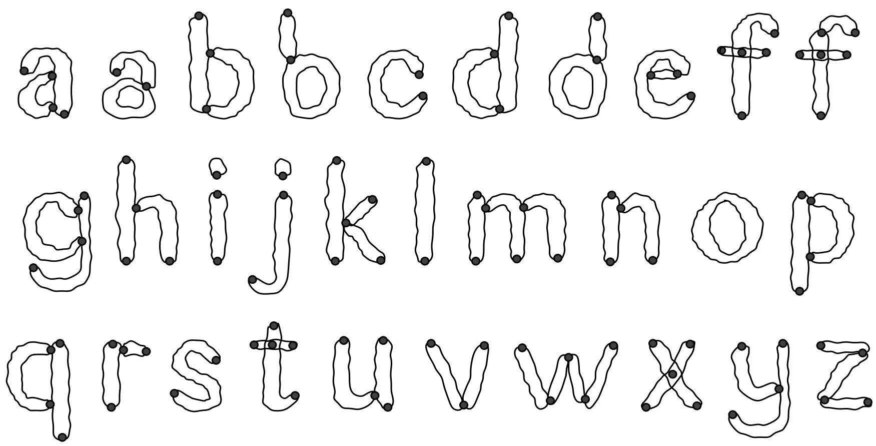


Figure 1.3: A graphical representation of some of the most standard sets of roles for the 26 lowercase letters. Only letters on the first line are shown with multiple sets of roles. There are other possible standard ways (not shown here) of conceptualizing many of the letters. Roles are not themselves shapes; instead, they are abstract sets of norms at the conceptual level, into which physical parts of letterforms can be fit. These pictures are meant to show how roles work together to produce letter conceptualizations. The black dots shown represent some of the between-role relations (r-roles). (A more complete picture of common role-sets can be found in Chapter 4 as Figure 4.2.)

simplification.[3] Figure 1.3 is a graphical representation of some, but by no means all, of the most standard sets of roles for each of the 26 lowercase Roman letters.

We come now to the question of how style and roles are related. In generating a new letter, first of all comes the choice of conceptualization — in other words, which set of roles will be instantiated? Thus at this deepest level of a letter — a purely mental level, before any shapes are on paper — roles are essentially all there is to style. One could turn this around and equally well say that at this deepest level of style, roles are essentially all there is to a letter.

Next, once a particular conceptualization has been picked or formed, the remaining stylistic aspects of a letterform are a function of *how the various roles in the letter conceptualization are filled.* Perhaps a more vivid and accurate way of saying this

[3]For related ideas on perceptual hierarchies see, for example [Palmer, 1977], [Sanocki, 1986] and [Erman et al., 1980]. More discussion of the role hypothesis can be found in Chapter 7.

is that this medium-deep level of style has to do with *how norms are violated.* For instance, consider the role *crossbar*, which belongs to conceptualizations of 'e', 'f', and 't' (at least). Possible norm violations are many but certainly include the following: "crossbar too high", "crossbar tilted upwards", "unusually short crossbar", "crossbar missing", etc. Any such violation is a stylistic hallmark that must be respected and propagated (via analogy) to other letters, even, if possible, to letters that lack the crossbar role!

How can a letter whose abstract characterization does not involve the notion of "crossbar" be influenced by a style in which crossbars are too high (say)? The key to this is *analogy.* To carry such a stylistic trend across, one might make some *other* role be realized in an unusually high way, as long as that role can be seen as analogically close to the concept of "crossbar", and as long as the distortion does not make the resultant letterform unrecognizable as a member of its desired letter category.

The name of the game in alphabet design is the constant power struggle between letter-category pressures, which involve just the one letter category being worked on at the moment, and stylistic pressures, which involve the alphabet as a whole. We characterize this give-and-take as a fight between two forces: an *incentric* or *centripetal* force, which tends to pull the shape being created *close to the center* of the intended letter-category (so that it is a strong instance of the *letter*), and an *eccentric* or *centrifugal* force, which tends to push the shape *away from the center* of the letter category (so that it better fits the desired *style*). The modeling of this conflict between incentric and eccentric forces lies at the heart of the Letter Spirit project.

1.2.1 Letter and Spirit as orthogonal categories

Although much work has been done on character recognition and the reading of handwriting, AI has had little to say about letters *as concepts.* So far, attempts at letter

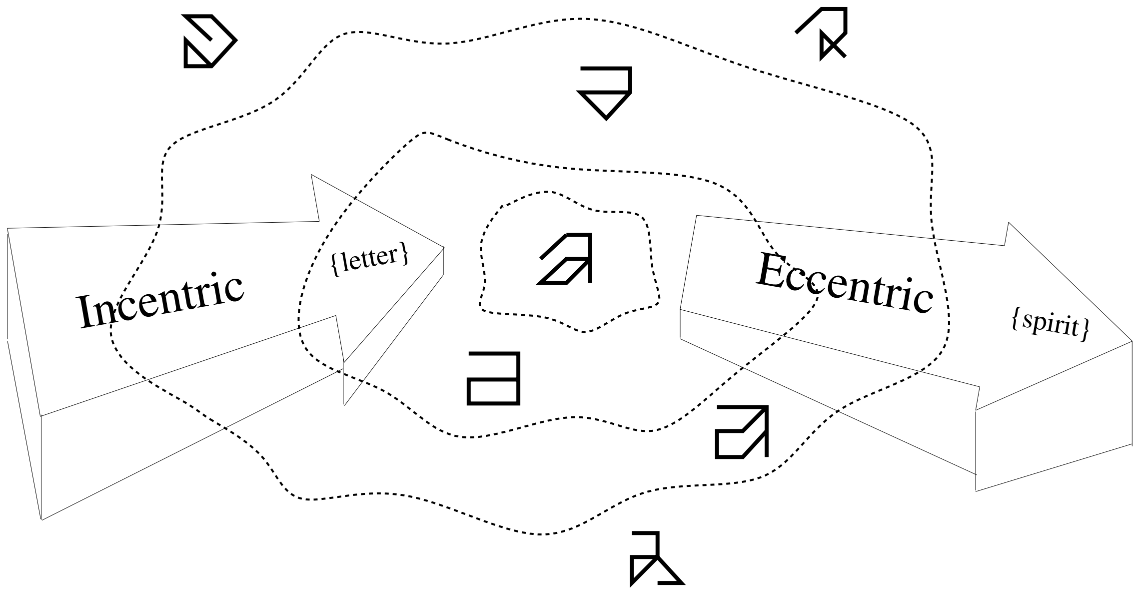


Figure 1.4: Letter versus Spirit. The incentric force pulls letters towards the center of the category, where strong letters reside. The eccentric force pushes the letter away from the center of the category, in order to make style more conspicuous. A related Figure, showing our model's actual performance in terms of this distinction, can be seen in Chapter 5 as Figure 5.25.

recognition and letter creation in AI have mostly been based on the notion that letters are basically syntactic variants of a single underlying *shape* (for instance, see [McClelland and Rumelhart, 1981] and [Hinton et al., 1992]).[4] Such an approach ignores the cognitively and philosophically central question [Hofstadter, 1985, Chapter 26]: *What is* 'a'*-ness all about conceptually?*

Although this is already a very challenging question, an even more challenging question regarding letterforms is: *How are the letters in a given style related to one another?* This can be cast in terms of an analogy problem. Given an 'a' in a certain style, how would you make an 'e' or a 'k' in the same style? Of course the problem

[4]Chapter 6 discusses both of these models (among a host of others) in detail and places them within a framework we have developed for better understanding the relationship between recognition models, including our own.

here is figuring out what “the same style” means. Transferring stylistic aspects from one letter to another involves the sort of “slippability” discussed in [Hofstadter, 1979], [Hofstadter, 1987], [Mitchell, 1993], and [Hofstadter and FARG, 1995]. Stylistic qualities of one letter cannot usually be *directly* transferred to another; rather, they must be “slipped” into reasonable variants so that they fit within the conceptual framework of the new letter without bending it beyond recognition.

As mentioned above, the Letter Spirit project is based on two orthogonal aspects of letterforms — categorical sameness and stylistic sameness. Figure 1.5 illustrates these two ideas and shows their relationship to one another. What do all items in any column have in common? **Letter.** What do all items in any row have in common? **Spirit.** Letters can thus be thought of as “vertical categories”, spirits as “horizontal categories”. In this sense, the two types of category are orthogonal. Part II of this thesis investigates the notion of categorical sameness in great detail — the *letter* of Letter Spirit.

A frequent preconception, mentioned earlier, is that all members of a given letter category have “the same shape”. Figure 1.5, like Figure 1.1, shows how naïve that assumption is. There is no fixed skeletal shape lurking within or behind the diverse manifestations of each of the letterforms constituting a given column. Each letter category is, rather, an abstraction of high order.

In contrast to what people naïvely say about letters, no one would ever think of suggesting that all the diverse carriers of a given style or spirit share “the same shape” — it is intuitively obvious that such a claim would make no sense, and that what binds together all the items in any given row is, instead, some set of very abstract qualities. One of the purposes of Figure 1.5 is to underscore the fact that the deep issues about perception and shared essence that are so self-evidently raised by the “horizontal” question (whose answer is “spirit”) are already all raised, though in a subtler manner, by the simpler-seeming “vertical” question (whose answer is “letter”).

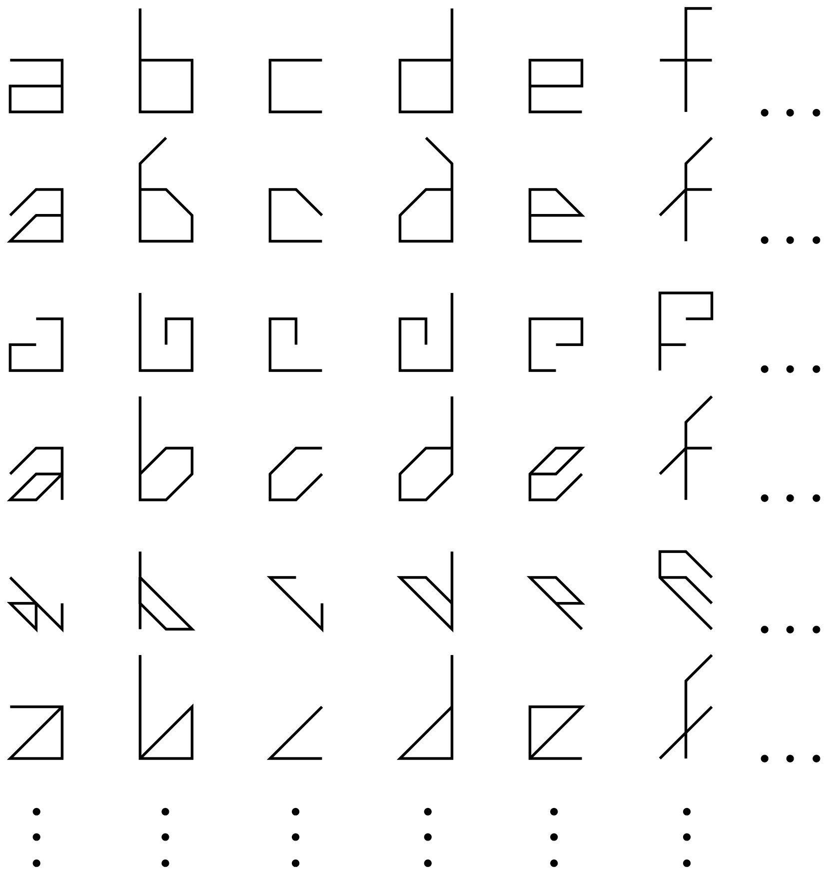


Figure 1.5: The Letter/Spirit matrix, illustrating the two orthogonal notions on which the Letter Spirit project is based. Items in any column have *letter* in common. Items in any row have *spirit* in common.

It would follow that a full and final model of the human perception of letters would necessarily include a deep model of the perception of style. This provocative claim flies in the face of the assumptions underlying many ongoing attempts to mechanize the reading of handwriting and other types of written input, which are based on highly pragmatic assumptions about the nature of perception and which pay no attention to questions of esthetics or style.

Given a few letters in a *gridfont* (our name for the highly constrained, gridbound alphabets used in the Letter Spirit project), the challenge to Letter Spirit is to figure out what category a given seed letter belongs to and what the stylistic tendencies suggested by the seed letter(s) are, and then to design the remaining letters in what it considers to be the same style. Of course, there is never just one good answer, since seed letters do not uniquely specify a style, nor does a style uniquely specify its constituent letterforms.

1.3 The Letter Spirit microdomain

1.3.1 From real-world to microworld typeface design

Letter perception and creation provide an elegant window onto the workings of the mind. If through working in the domain of letter design we can gain general insight into the workings of concepts and their fluid interrelations, we feel we will have made important headway on the problem of intelligence. It is, in fact, our belief that such headway can *only* be made via the detailed study of a small aspect of the world carefully selected for its generality.

Like music composition or novel writing, letter design is an art form that requires years of practice. We could not possibly aspire to make a model that operates at the level of skill of an experienced human letter-designer. Instead we have drastically simplified the domain. Nonetheless, we hope to have preserved the *deep* aspects of the art of letter design while removing many of the shallower artistic qualities that, for the most part, rely on well-practiced, refined motor skills, such as: the use of elegant curves and tapering lines, knowledge of typographic conventions, and the need for a well-trained hand. Our microdomain, by contrast, involves very stripped-down letterforms, which are much closer to the concepts behind them than are fully fleshed-out letterforms, as we shall see below.

Actually, the initial ambition of the Letter Spirit project was to write a computer program to design full-fledged typefaces having the same order of complexity as conventional typefaces such as Helvetica, Times, Baskerville, and so on. However, this quickly revealed itself to be far too hard and, ironically, of little interest to cognitive science. The problem is that real-world typeface design involves far too many nuances to allow practical simulation in a computer model. Moreover, full-scale simulation of this human ability would involve the modeling of so many domain-specific details that the cognitive issues meant to be the core of the project — the nature of fluid concepts and the conceptual nature of style — would be lost among a welter of irrelevant noncognitive concerns.

For this reason we deliberately decided to eschew all the complexities involved in curves, stroke-taperings, and other aspects of curvilinearity that seem to involve low-level or intermediate-level vision rather than the high conceptual level that was our main focus. The idea was that by forbidding the manipulation of surface-level aspects of letterforms, we would simultaneously reduce the complexity of the programming task and force concentration on deeper, more *conceptual* levels of the design process — *deep style*, in short.

At its deepest levels, style involves playing with the conceptual foundations of letters (*e.g.,* conceptualizing an 'x' as *a 'v'-ish entity perched on top of an inverted 'v'*, instead of the more conventional way of seeing 'x' as *two intersecting slashes*). This distinction between deep and shallow style is of course not a black-and-white one. Instead there is a continuum, with shallow style at one end and deep style at the other. Letter Spirit is focused on the deep-style end of the continuum.

1.3.2 Gridletters and gridfonts

To avoid the need for modeling low-level vision and to focus attention on the deeper aspects of letter design, we eliminated all continuous variables, leaving only a small number of discrete decisions affecting each letterform. Specifically, letterforms are restricted to short line segments on a fixed grid having 21 points arranged in a 3×7 array [Hofstadter, 1985, chapter 24]. Legal line segments, called *quanta*, are those that connect any point to any of its nearest neighbors horizontally, vertically, or diagonally. There are 56 possible quanta on the grid, as shown in Figure 1.6 — 14 horizontal, 18 vertical, and 24 diagonal.

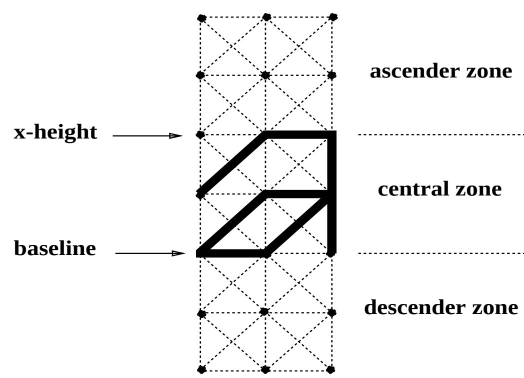


Figure 1.6: The Letter Spirit grid, with one of the possible sets of quanta instantiating an 'a' turned on.

The grid can be divided into three zones corresponding to locations where parts of letters usually fall. The *central* zone is where the letters 'a', 'c', 'e', 'm', 'n', 'o', 'r', 's', 'u', 'v', 'w', 'x', and 'z' are found in their entirety. The other letters — 'b', 'd', 'f', 'g', 'h', 'i', 'j', 'k', 'l', 'p', 'q', 't', and 'y' — have ascenders and/or descenders that jut into one of the other two zones, called, naturally enough, the *ascender* and *descender* zones. The bottom of the central zone is called the *baseline*, and the top is called the *x-height* line.

Letters created on the grid might be thought of as "skeletal" geometric representations of letter concepts. To some people, this would suggest "fleshing them out" into "true" letterforms by adding curvature, thickness, and so on. Figure 1.7, for instance, shows the relationship between a gridfont called *Benzene Right* and a derivative "dressed up" alphabet. Using straightforward and totally mechanical shape-for-shape substitutions, any gridfont can be converted into a more conventional typeface. A

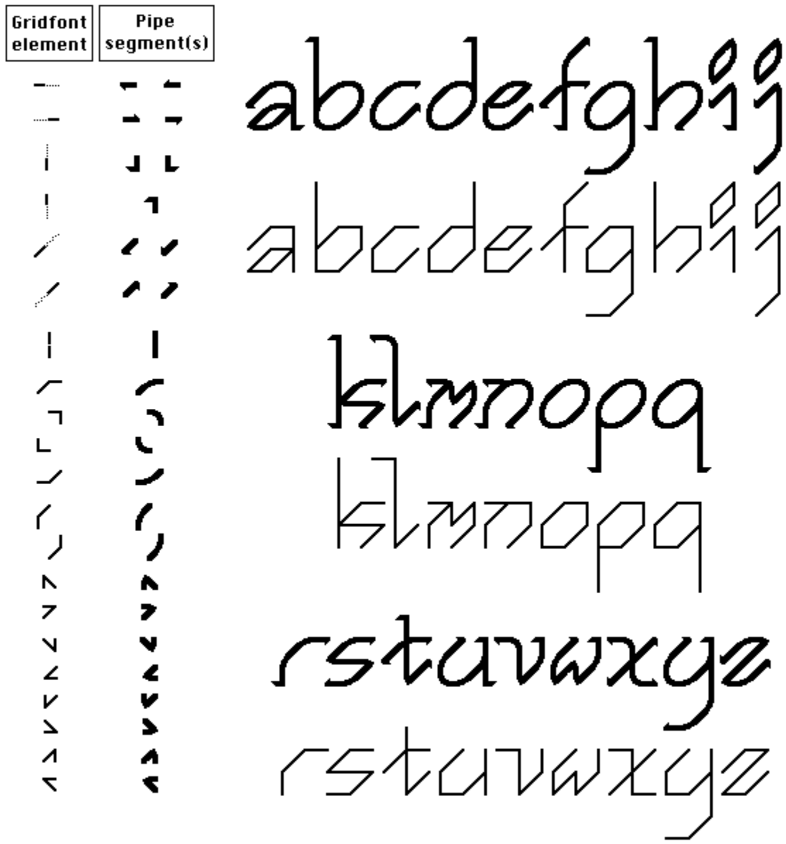


Figure 1.7: The gridfont *Benzene Right* is displayed below one of its many possible fleshed-out pipeface cousins.

chart of the substitutions involved is shown on the left side of Figure 1.7.

Although this is possible, we have chosen to leave out the step of adding “flesh” to the “bones” of a gridfont, and to work at the more basic level of the “skeletons”. Basically, we choose to think of the gridbound letterforms as full-fledged letters. Seen as stand-alone typefaces, some gridfonts are visually fascinating — even beautiful (see Figures 1.9 and 1.11, as well as Appendix A) — and might make acceptable typefaces for certain purposes, but most of their beauty is more intellectual and flows from the ways in which they illustrate and explore the conceptual essences of letters, as opposed to the more sensual or “retinal” beauty of fully fleshed-out, curvilinear typefaces.

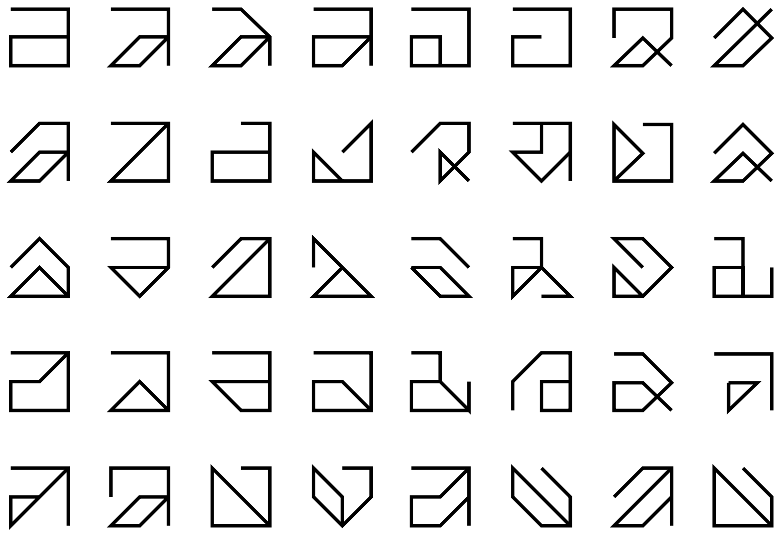


Figure 1.8: Forty renditions of the letter 'a' on the grid. They are by no means all equally strong members of the category 'a' — in fact, some are rather weak.

Since all there is to do in designing a gridfont is to turn quanta either on or off, decisions on the grid are coarse. Surprisingly, the variety among letters of a given category is still huge — hundreds of versions of each letter have been designed by humans. Of course, people are generally much less interested in sitting around exploring the space of possible gridbound 'k's or 'q's, say, than in designing a small number of full gridfonts. So far, several hundred complete gridfonts have been designed.[5] Surveying them all, one is struck by the diversity present in each letter category.

Almost paradoxically, the Letter Spirit domain's severe limitations seem to engender this diversity. As an example of the richness the grid allows (even within one letter category), see Figure 1.8, which shows 40 of the approximately $1,500$ different 'a's, ranging from very weak to very strong, that have been created by people.

It is impossible to make tiny changes on the grid, so any two instantiations of a given letter are significantly different from each other. Decisions to add or subtract

[5]A number of representative gridfonts are included in Appendix A. A large database of gridfonts is also available by **ftp** from ftp.cogsci.indiana.edu (more information about Internet availability of articles, source code, and data can be found in Appendix B).

Figure 1.9: This figure can be thought of as illustrating both the vertical problem (categorical sameness) and the horizontal problem (stylistic sameness) for which Letter Spirit is named.

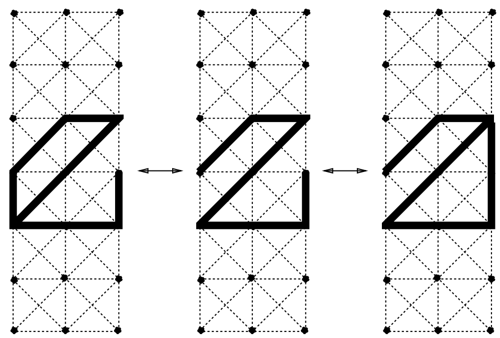


Figure 1.10: Rapid metamorphosis of an 'e' into an 'a' via 'z', with merely a one-quantum change in each step.

even one quantum often fundamentally affect category membership. Figure 1.10 illustrates the large distance in letter-space one can travel with minimal changes on the grid. A "small" change in the letterform — the erasure of just one quantum — changes the first shape from a strong 'e' to a strong 'z'. A similar one-quantum addition transforms the 'z' into an 'a'. So category membership in the Letter Spirit domain is a tricky matter. In fact, modeling the process of deciding which of the 26 categories a given shape on the grid belongs to (if any) is one of the hardest aspects of the project and represents the major focus of this thesis (see Part II).

The Letter Spirit project involves coming up with a letter-plan for a shape, then physically rendering that plan in some manner on the grid. This rendering — a geometric shape — must then be perceived and critiqued by a perceptual agent. In Letter Spirit, the program's criteria for what is acceptable as an 'a' are not fixed and static, but rather are dependent on the *style currently being developed.* The essence of 'a'-ness is thus not completely specified in advance in a rigid pre-packaged structure; rather, it emerges as letterforms are created and perceived. Instead of there being 26 platonic *shapes* that get "fleshed out" with fanciful (but human-designed) curves (as in the pipeface shown in Figure 1.7 or the DAFFODIL program, discussed in Chapter 2), there are 26 *fluid abstractions*, one for each letter, which respond in a flexible way to stylistic pressures. Each such abstraction must be capable of adapting under pressure in such a way that its spectrum of instantiations has members that simultaneously fit in the style being designed and remain within the space of their intended letter category.

Despite the eccentricity encouraged by the grid, most (but not all) gridletters are easily recognized by humans. One important feature of the grid is that it can both force letters to their categorical extremes because of its sparseness and yet still contain many easily identifiable letterforms. This makes it an excellent vehicle for letter-perception research — one that allows attention to be focused on issues of style.

As was mentioned above, the constraints defined by the grid actually encourage tampering with letter concepts at a deep level. Because there are no fine-grained features to manipulate, one soon finds oneself playing at the boundaries of the 26 categories. Consequently, many gridfonts are wild, sometimes having angular, blocky, spiky, sparse, or otherwise bizarre letters, as Appendix A makes abundantly clear. We must stress that we are not striving for typefaces with high readability or letterforms beautiful at the surface level. Rather, we are attempting to probe the conceptual nature of letterforms and thereby gain insight into what imbues concepts in general with their fluidity. Pushing the 26 categories to their edges in a coordinated way often results in an intellectual beauty not localized in any single letterform of a gridfont, but spread out over the gridfont as a whole — the beauty of *spirit* rather than of *letter*.

While at first glance, the Letter Spirit domain might be shrugged off by some cognitive scientists as merely another “toy domain”, this would be a gross underestimate of its subtlety. In spite of — or more accurately, *because* of — the reduction to the grid, the Letter Spirit challenge is, in terms of cognitive-science issues, extremely rich. The cognitive issues are magnified, not reduced, by the act of simplifying the domain. All that has been thrown out is the need for many years of prior experience designing letters. One does not need to be a professional typeface designer or lifelong student of letterforms to appreciate the consistency of a well-designed gridfont. Even a novice can design a passable gridfont, although doing a sophisticated one is very difficult. (Were it not, creativity would be a simple thing to model!)

1.4 The principal varieties of stylistic determiners

Numerous types of stylistic pattern characterize a gridfont as a whole, including the following:

- A *role trait* characterizes how a specific role tends to be instantiated, independently of the specific letters it belongs to. In other words, a role trait is a "portable norm violation" — a norm violation attached to a specific role and thus capable of affecting a number of different letters. For instance, in a particular style, the roles of *post* and *tail* might be realized several times over in a zigzaggy manner; in some other style, the role of *bowl* might be realized in various letters in a tall and narrow manner; in yet another style, the role of *crossbar* might be realized in one or more letters by passing through a "hole" or "gap" in the associated vertical stroke; and so on. Each of these stylistic features is a role trait: a description of how a specific role has been (or might become) instantiated in various letters.
- A *motif* is a geometric shape used over and over again in many letters. If it is very simple (*e.g.,* a two-quantum backslash crossing the central zone), it may be required to appear in complete form in every single letter. If the motif is a more complicated shape (*e.g.,* a perfect two-quantum-by-two-quantum square, or a hexagon that looks like a tilted benzene ring), then parts of it may be allowed to be absent from various letters, so long as a reasonably substantial portion of it remains. Some styles allow a motif to appear in reflected, rotated, and/or translated form; others allow translation but no reflection or rotation; yet others

allow reflection and/or rotation but no translation; and so on.

- An *abstract rule* is a systematic self-imposed constraint such as: allowing no diagonal quanta; allowing only diagonal quanta; requiring each letter to consist of precisely two disjoint parts; forbidding any straight section of more than one quantum in length; and so on. It is not that there is some particular shape that is repeated, but rather that a rule of some sort is globally enforced.
- *Levels of enforcement.* The three preceding types of stylistic attribute pertain directly to shapes on the grid. A much more abstract determiner of style — in fact, a kind of "meta-level" aspect of style — is the degree to which any such constraint is considered "unslippable" (*i.e.,* absolute or inviolable), as opposed to being "slippable" (*i.e.,* allowed to be disrespected under sufficiently great pressure). The level of enforcement of a stylistic constraint — strict, lax, or somewhere in between — sets an abstract, almost intangible tone for the entire gridfont.

1.4.1 The spirits behind four sample gridfonts

The four human-designed gridfonts named *Benzene*, *Square Curl*, *Checkmark*, and *Funtnip*, displayed in Figure 1.11, are excellent for illustrating these stylistic-consistency devices. These gridfonts also demonstrate how deeply style can pervade, and occasionally torture, the 26 alphabetic categories.

We shall briefly comment on these four fonts, one by one.

- In *Benzene*, the six-sided "benzene-ring" motif is found in full or fragmentary form in nearly every letter. Ironically, the first letter designed in *Benzene* — the letter that sparked the entire font — was the 'a', whose ring has only four sides. The full ring came into being only when the attempt was made to export this motif to the category 'b', at which point the constraints of the grid, interacting

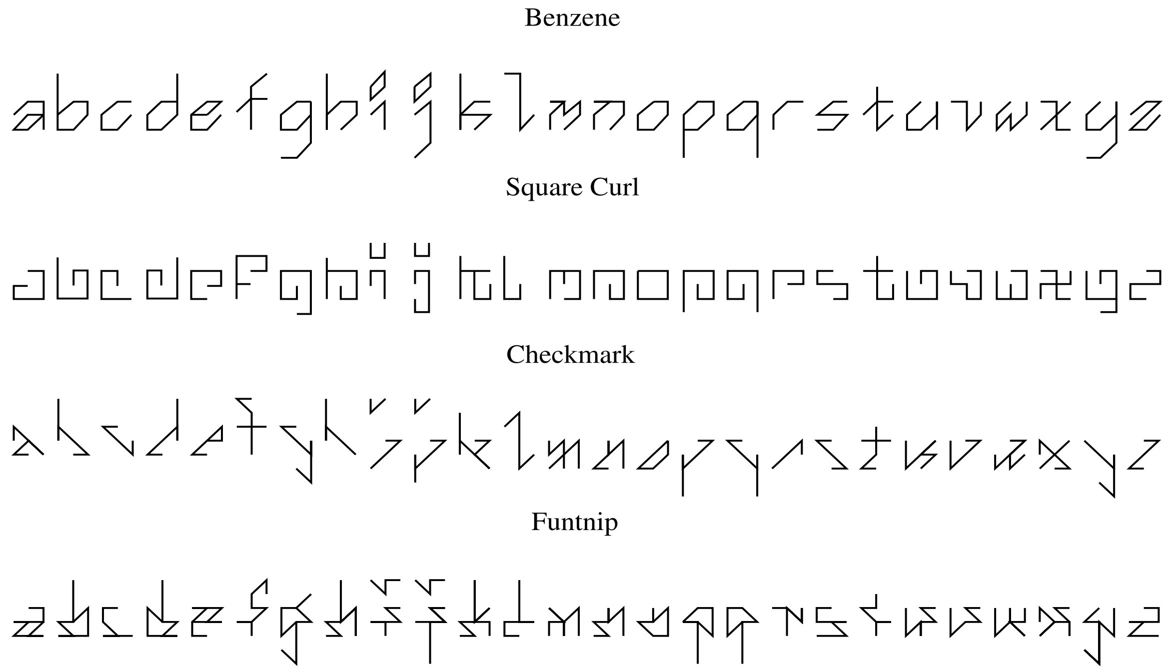


Figure 1.11: Four human-designed gridfonts illustrating various mechanisms, including rigid and free motifs, abstract rules, and norm violations, for propagating style or spirit back and forth along the length of the alphabet. See the text for detailed comments on these four styles.

with the norms of that category, caused the six-sided benzene shape to materialize (see the upper half of Figure 1.12).[6] But the smaller original diamond motif was not abandoned; indeed it is taken up again in the 'e' and, in rotated form, as the dot on the 'i' and 'j'. One even sees it a little bit in the 'm', 'w', and 'z'.

Note that the full-sized benzene ring (namely, the 'o') is a *rigid* motif, in the sense that it is never — or almost never — rotated or reflected. The biggest exception to this rigidity is the *Benzene* 'x', which was constructed on the English kissing-crescents model. To do this, the 'o' was sliced down its middle to make two crescents, and then those crescents were swapped and put back

[6]Interestingly, the same 'a' appears in the gridfonts *Stately Right* and *The Tale of 'q'* as well. See Appendix A.

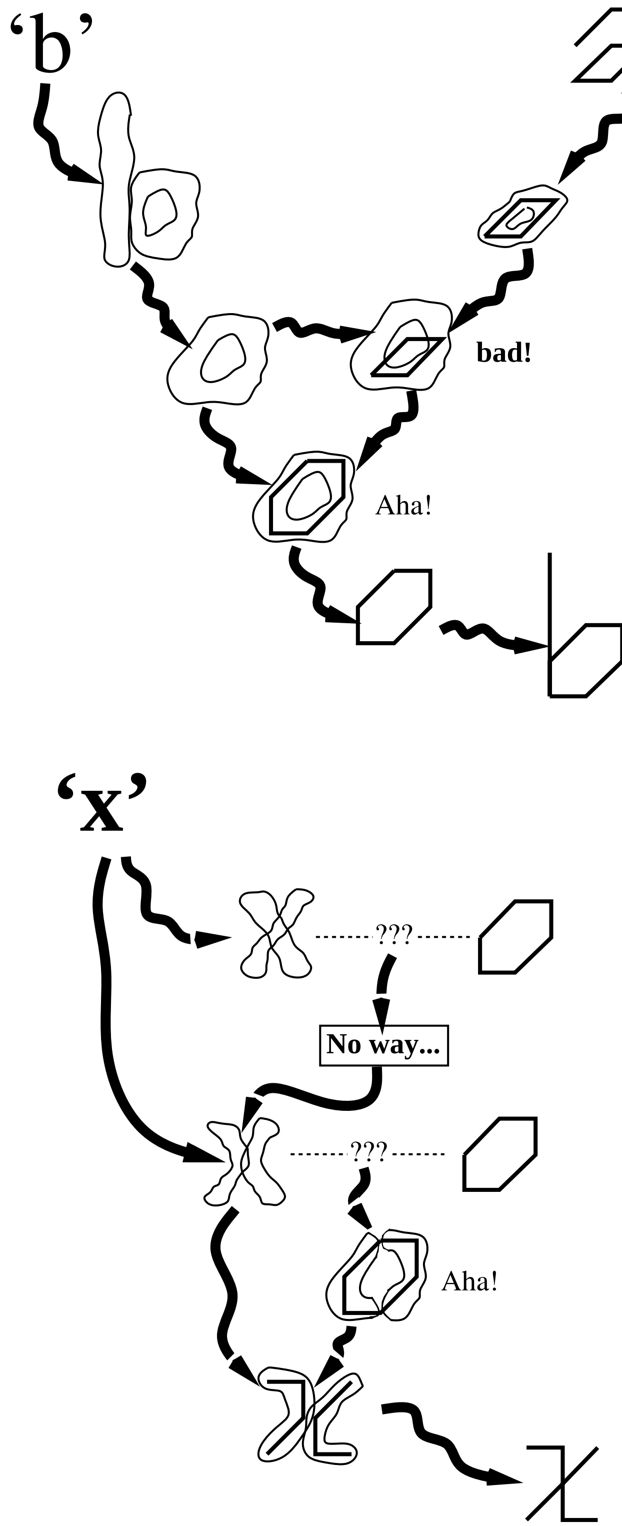


Figure 1.12: Schematic representations of the stories behind two *Benzene Right* letters, 'b' and 'x'.

together, so that they meet back to back (see the lower half of Figure 1.12). What is particularly interesting about this 'x' is that despite its origin in the kissing-crescents conceptualization, the eye far more readily perceives it as consisting of two crossing components that fill the roles in the rival "crossing-slashes" conceptualization. In other words, the subtle conceptual origin of this letterform is deeply hidden. Such "covering of one's tracks" is a surprisingly common occurrence in creative acts, and often makes them seem more magical than they actually are.

- The gridfont named *Square Curl* was a consequence of the interaction of a strict abstract rule (forbiddenness of diagonal quanta) with a *free* motif — the "curl" seen most clearly in the 'n' and 'u' ("free" meaning it is free to rotate or flip). *Square Curl* also involves a norm violation — namely, its ascenders and descenders are shorter than usual, in the sense of not reaching the top or bottom of the grid's ascender and descender zones. This norm violation and the curl motif have a certain mutual resonance, in that the curl motif as manifested in letters like 'a' and 'e' likewise involves a role that falls short of its usual destination, which gently suggests that it would be fitting for yet other roles to "fall short" in some sense — as in the horizontal tails of the descenders of the '*g*' and 'y', or the tips of the 's' and 'z', and so on.

  The 'i' and 'j' playfully violate the tried-and-true convention — the norm — of letter designers, according to which those letters' dots should be lower than or as high as, but certainly not higher than, the tops of letters with ascenders (such as 'b' and 'k'). Despite this norm violation, these dots do not damage the font but add to its verve. This is an example of a general principle in design, which says that when norms are violated *ignorantly* (*i.e.,* by a novice), the results are likely to be bad, whereas when they are violated *knowingly* (*i.e.,* by an expert), the result is likely to be fine. Experts have a keen sense for when they can get away with something peculiar or daring, whereas novices don't even realize that there is anything unusual going on.

- The free motif that gives the style called *Checkmark* its name is the shape of the 'r', which in rotated, reflected, translated, and/or truncated form is visible throughout the alphabet. Another of the key ideas underlying this font is that of *gaps* or *absences*, closely related to the notion of *incompleteness*. In the 'a' and 'b', for instance, there are little gaps caused by lines being shorter than one would expect. This theme of absence or incompleteness is even stronger in the 'h' and 'y', among others. On the other hand, sometimes one gets a bit *more* than one expects, as in the 'j', 'n', and 'u'. Clearly the 'j' was borrowing something from the descenders of letters like '*g*' and 'p'. Perhaps it did so too literal-mindedly, resulting in a shape whose category membership is maybe just a tad too teetery. That's a very subjective judgment call, of course. The 'k' exhibits a curious and rather pleasant combination of "too much here" and "too little there".
- *Funtnip* is certainly the most complex of these four fonts. One of its multiple themes is the idea of a horizontal stroke passing either *all* the way across the middle of the central zone (as in 'a', 'i', and 'k') or just *partway* across (as in the 'f' and 'm'). Only the 'c' and 'r' violate this rule. Another crucial theme is the "lopsided arrowhead", so to speak — a free motif most clearly seen in the 'o', the 'p' and 'q', and in part in many other letters.

  Yet another key theme is the notion that ascenders and descenders should be located halfway across the letter, rather than on the left or right side; this, of course, is a deeply norm-violating suggestion, which almost seems absurd. For letters like 'p' and 'q', for instance, whose distinction usually depends on little other than the location of their descenders, such an idea could only be made to work by making the bowls very asymmetric, so that *they* now carry the burden of identity that the tails usually carry. Fortunately, this can be done successfully.

Then there are other minor themes in *Funtnip*, such as the idea of a full or partial horizontal line on the baseline (seen in the first four letters and then only occasionally thereafter — *e.g.,* 'k' and 'n'); the idea of a perky little diagonal fillip (as in the '*g*' and 'm' and even the 'w'); the extra little "stingers" in the 'a', 'e', 's', and 'z'; and finally, just to liven things up a bit more, the square-root-sign dots on the 'i' and 'j'. This odd potpourri of seemingly unrelated ideas somehow manages to come together and produce a strong, well-integrated, and distinctive visual style. How does this kind of thing happen?

Disclaimer: We have no genuine theory as to how a gridfont as complex and daring and multi-themed as *Funtnip* or *Checkmark* is generated. We would aim for something far more modest, such as *Standard Square* (see Figure 1.9). Even building up *Benzene* from its 'a' alone would be considerably beyond our expectations for the Letter Spirit program.

1.5 Creating a gridfont — a broad view

For a person, designing a gridfont usually takes between ten minutes and three hours (after which the font still remains potentially subject to scrutiny and minor revision). The process involves myriad small operations, ranging from the highly mechanical to the highly inventive. A typical action (whether as large as creating a 'k' entirely from scratch or as small as changing a single quantum in an already-designed letter) sets off repercussions that echo throughout the continuing design process, all over the gridfont, and at all levels of abstraction. This reverberation up and down the alphabet is largely guided by *a priori* notions of the interrelatedness of letter categories.

This is worth spelling out in some detail. Suppose one is given the 'o' as a seed letter. This 'o' might easily suggest a 'c' (just open up a little gap on the right), and the ‘c’ might in turn suggest an ‘e’ (just add a crossbar). The ‘c’ might also suggest a ‘d’ (just add a post on the right side). The ‘d’ could then suggest a ‘b’, a ‘p’, and a ‘q’, by rotation and reflection. The ‘q’ in turn could suggest a ‘*g*’ (just bend its tail a bit), and the ‘*g*’ could suggest a ‘y’ (open up the loop and straighten out the tail), which may be rotated to give a first stab at an ‘h’. Obviously you can cut off the ‘h’’s ascender and thus get an ‘n’. Add a bit in the middle to the ‘n’ and get an ‘m’, then turn that upside-down and see if the resulting shape passes muster as a ‘w’. Of course, the ‘w’ may well suggest a ‘v’ and a ‘u’ (although the act of rotating the ‘n’ provides an independent source of ideas for ‘u’) — and on and on it goes. Out of many such inter-letter suggestions and influences, together with a vast number of intra-letter refinements, a stable gridfont eventually emerges.

For a human designer, *dissatisfaction* and *contradiction*, either within a given letterform or between letters, are among the prime movers of the design process. If Letter Spirit is to be faithful to how humans create, it must be capable of recognizing and resolving such conflicts, in order to create a coherent style. Sometimes the conflicts are subtle, and therefore require refined artistic judgment-calls. Modeling the ability to find conflicts, diagnose them, and convert diagnoses of problems into reasonable suggestions for solutions is an indispensable aspect of the project.

Any design decision will affect not only the letter currently under consideration but also conceptually close letters. For example, a specific decision as to how to instantiate the role of *post* in ‘b’ is likely to have an enormous effect on the post of the ‘d’, as well as on many other letters with posts, and may also noticeably influence the *tails* of the ‘p’ and the ‘q’. Of course, the extent and type of this influence are highly dependent on the particular style and are in no way mechanical or formulaic. In most conventional typefaces, for instance, tails and posts are closely linked by tight analogy, but there are certainly typefaces in which tails and posts are conceptually independent. The conscious decision that takes place in a designer’s head to “unlink” tails from posts — that is, not to respect the "natural" or default analogy between them — is in itself a major stylistic decision.

Most aspects of letterforms are likely to propagate their influence to varying extents through the entire gridfont. The propagating wave will probably cause many retroactive adjustments (both major and minor) to some "already finished" letters, and give rise to ideas for letters not yet designed. One design decision will typically spark others, which in turn will engender others, and so on.

Eventually, when enough "decision waves" have washed over the entire gridfont, all the letterforms begin to have a high degree of internal consistency, and a clear style begins to emerge. Once the tension of inconsistency eases up, no more large-scale changes are required. Minor adjustments may continue, but for the most part, the large-scale creative act will be finished.

This temporally-extended, serial process of integration and gradual tightening of internal consistency is an indispensable part of true creativity, and is a well-known property of such creative acts as musical composition, the writing of poetry and prose, the activities of painting and sculpture, the evolution of scientific theories, and even the design of cognitive models!

The architecture required to model any dynamic creative process is necessarily complex. We now move on to a description of the architecture that we are currently designing, refining, and implementing in order to make a workable Letter Spirit program.

1.6 The implementation of emergent processing

The architecture for Letter Spirit draws on ideas developed in earlier research projects of the Fluid Analogies Research Group (FARG) — most of all Jumbo [Hofstadter, 1983], Copycat [Mitchell, 1993], and Tabletop [French, 1995].[7] Although the Letter Spirit architecture is closely related to their architectures, there are some major divergences. Perhaps the most significant one is the complex internal structure of Letter Spirit categories — the breakdown of wholes into roles, roles into weighted sets of abstract features, and so on. This means that the concepts in Letter Spirit — in particular, the 26 letter concepts — will have a much richer structure and behavior than concepts in earlier work in our group.

Copycat and Tabletop solve analogy problems in microdomains using psychologically plausible methods. Both are models of high-level perception making use of a novel type of architecture that falls somewhere between the symbolic and connectionist paradigms. Top-level behavior emerges from many low-level stochastic computational actions that occur in parallel. Copycat and Tabletop model the way in which a small number of relevant concepts can be awakened from dormancy in order to understand and reason about a situation. Letter Spirit will have a similar architecture, but will focus on a larger-scale creative process that unfolds over an extended period of time. Jumbo, though smaller and earlier than Copycat and Tabletop, pioneered the parallel-terraced-scan architecture (explained below). Its aim was to model the hierarchical perceptual clustering and fluid regrouping-operations that occur in the activity of looking for anagrams. This work is especially relevant to the creative process and bears much kinship to operations in Letter Spirit.

All perceptual and creative processes in the Letter Spirit model are emergent, in the sense that they result from the actions of a large number of independent *codelets* — computational micro-agents that create, examine, and modify structures representing parts, roles, letters, stylistic traits, and so forth, and that do so in (simulated) parallel.

[7] *Fluid Concepts and Creative Analogies*, a book about our work at the Center for Research on Concepts and Cognition, has recently been published [Hofstadter and FARG, 1995]. It explains the relationship between older projects and the projects currently under way at CRCC.

Codelets continually come into being and wait to be run in a structure called the *Coderack*, which can be thought of as a stochastic waiting room. In contrast to a standard operating-systems queue, where processes wait before being deterministically given their slice of CPU time, the Coderack features stochastic selection of actions. To each codelet is attached an *urgency* value — a number that determines its probability of being chosen next. This value is based on how important and useful the codelet's possible effect is gauged to be at the time the codelet is created.

The Coderack is the stochastic control center of Letter Spirit. Actions of every sort — gluing, labeling, scanning, matching, adjusting, regrouping, destroying, and so on — are carried out by codelets.[8] The effect of each codelet considered by itself is very slight; however, as many codelets run, their independent effects build upon one another into a coherent collective behavior.

A useful image is that of a large structure (like a bridge) being built by a colony of hundreds of ants or termites. The ants work semi-independently but cooperatively. Codelets correspond to the ants in this metaphor, and perceptual structures to the bridge. Perceptual structures, like the ant-built bridge, develop nondeterministically but not haphazardly. From a large number of very small probabilistic decisions, each on its own insignificant and dispensable, coherent perceptions are built up.

As old codelets run and "die", new codelets are created and placed on the Coderack. New codelets are created in two ways: some are posted as *follow-ups* by codelets as the latter are completing their tasks; others are placed on the Coderack *automatically* as certain stages of processing are reached. The population of the Coderack thus dynamically adjusts to the system's needs, somewhat mimicking the way a cell's population of enzymes adjusts to the current "needs" or "desires" of the cell, as expressed in terms of molecular scarcities or abundancies in the cell. (The

[8]Exactly what all these operations amount to is described in Chapter 4.

functioning of a biological cell was in fact a major inspiring metaphor in the architecture of the CRCC cognitive models [Hofstadter, 1984; Hofstadter and FARG, 1995]. See also Section 9.3 of this thesis.)

Each new codelet is assigned an urgency representing an estimate (usually quite crude) of the importance of its possible action. Codelets that seem most likely to enhance the evolving perceptual structures and to dovetail with the emerging view will be assigned high urgencies and thus will have a good chance of getting to run. Codelets that seem less promising will be assigned low urgencies and will probably have to wait a long time to run. The biased-random nature of codelet-picking ensures that low-urgency codelets have at least some chance of running, while ensuring that aimless processing is avoided.

Over a long period of time, *processes* are interleaved in a manner reminiscent of time-sharing. (A process consists of many codelets, which *ex post facto* can be seen to have been acting in concert.) One notable difference between this and conventional time-sharing is that the biased nondeterministic selection of codelets amounts to having different processes run at different speeds. Moreover, the speeds themselves can be regulated over time to favor more-promising directions over less-promising ones. Since codelets have very small effects, it is never critical that any particular codelet get selected. What does matter is that certain broad-stroked courses of action as a whole run faster than others. Probabilistic selection based on urgencies allows this to happen.

Emergent behavior of this type constitutes a *parallel terraced scan*. The parallel terraced scan can be thought of as the allocation of processing power as a function of the degree of promise of an evolving exploratory pathway. This strategy for parallel exploration has much in common with genetic-algorithm search methods and the solution to the k-armed-bandit problem [Holland et al., 1986].[9]

Though Letter Spirit is being implemented on a serial computer, so that only one

[9] For more on this relationship, see Chapter 4, Section 4.2.3.

codelet runs at a time, the system is in principle a parallel system in which many different activities take place in parallel at different speeds. Viewing Letter Spirit as a parallel-processing system is justified since codelets work in local and independent ways. Hence the "parallel" in "parallel terraced scan".

Note that the notion of "process" in Letter Spirit is very different from the notion of the same name in classical time-sharing architecture. In the latter, a process is determined even before it is run, and can thus be conceptually broken into pieces of any desired grain size in advance of being run. In Letter Spirit, a process comes into being only as a result of certain codelets having been run — it could not have been anticipated or described beforehand. In fact, processes in Letter Spirit are in the eye of the beholder — they are not objective entities.

1.7 Four global memory structures

The Letter Spirit program, as it is now materializing, contains four dynamic types of memory, each concerned with a particular level of concreteness and abstraction of shapes (and concepts pertaining to shapes). These memories are:

- The **Scratchpad**, which can be thought of as *a virtual piece of paper* on which all the different letters of a given font are drawn and modified; as such, it is more a type of external memory than an aspect of mental activity.
- The **Conceptual Memory**, which can be thought of as the program's *locus of permanent knowledge and understanding of its domain.* This consists in, for each concept, the following three facets: (1) a set of *category-membership criteria*, whose purpose is, roughly, to specify the recognition requirements for instances of the concept in terms of more primitive concepts (roles); (2) a set of *explicit norms attached to roles*, which serve to distinguish very prototypical instances of the concept from more eccentric ones; and (3) an *associative halo*,

consisting of links having time-varying lengths connecting the concept with various other related concepts, essentially giving a sense of where the concept is located in "conceptual space" by saying what it most resembles.

- The **Visual Focus**, which can be thought of as *the site where perception of a given letterform occurs*. In it, perceptual structures representing a given gridletter are built up and converge to stable categorical and stylistic interpretations. It is critical that when a letterform has been designed, some agency inside the program be able to take a fresh and hopefully unbiased look at that shape, in an effort to judge whether an outsider would recognize it easily as a member of the intended category, and also whether an outsider would see it as fitting in stylistically with other already-present letters. This is what takes place in the Visual Focus. Incidentally, the ability to step back and take a somewhat objective, detached view of something that one has just produced is absolutely indispensable for a creator — but is enormously hard to acquire.
- The **Thematic Focus**, which can be thought of as the program's *dynamically changing set of ideas about the stylistic essence of the gridfont under way*. In it are recorded stylistic observations of all sorts concerning letters already designed. If and when some of these passive observations start to be perceived as falling into patterns, those patterns can be taken as determinant of the style, meaning that they can be elevated to the status of explicit *themes* — ideas that play an active role in guiding further design decisions, in the sense of serving as "pressures" on the construction of further letters.

A useful perspective on these four structures is afforded by the following set of rough equivalences with various familiar types of human and computer memories. The Scratchpad can be thought of as an *external memory device*; the Conceptual Memory can be thought of as a *permanent semantic memory* containing both procedural and declarative knowledge of the system's repertoire of concepts; the Visual Focus as a *subcognitive workspace* (that is, a very-short-term cache-like working memory in which parallel perceptual processes, mostly occurring below the system's threshold of awareness, collectively give rise to rapid visual classification of a shape, whose final category assignment is made cognitively accessible); and finally, the Thematic Focus as a *cognitive workspace* (that is, a much slower, and therefore more conscious, level of working memory in which abstractions derived from more concrete and primary perceptions are stored, compared, and modified). We now describe each of these memory structures in more detail.

1.7.1 The Scratchpad

The Scratchpad is the place where experimental letterforms are created and critically examined. At the beginning of a run it is empty; by the end of a run it contains at least 26 completed letterforms. We say "at least" since it is certainly possible for the program to have created and stored a number of alternate forms for a given letter. This is a frequent occurrence in design by humans, and we see no reason to force the program to completely discard ideas that it creates, even if it winds up preferring other ones in the end. Such near-miss letterforms are in fact very useful windows onto the creative process, and often can serve as seed letterforms for another gridfont.

It is amusing to note that in this way, at least theoretically, the program could generate its own seed letters rather than needing human input for each new gridfont. Hypothetically, one can envision the program finishing one gridfont and then being "eager" to use rejected ideas as the starting point for new gridfonts, thus becoming a self-driving creator!

Little needs to be said about how the Scratchpad is structured; it simply contains an arbitrary number of grids, each of which is a 56-bit data structure telling which of the 56 quanta in the grid are turned on and which are off, together with an optional category label pointing off to the intended category (effectively saying, for example, "This is supposed to be a 'k'."). The judgment as to whether the given set of quanta actually succeed in their collective ambition to embody 'k'-ness or 'g'-ness or whatever is of course carried out in the Visual Focus.

1.7.2 The Conceptual Memory

The Conceptual Memory provides each concept in the domain with an internal definitional structure and a local conceptual neighborhood. Roughly speaking, a concept's "internal definitional structure" consists of its specification in terms of simpler concepts, and a concept's "local conceptual neighborhood" consists of its links to peer concepts in conceptual space. The internal definitional structure itself breaks down into two facets: *category-membership criteria* and *explicit norms*. Finally, a concept's local conceptual neighborhood is known as its *associative halo*, and one of its main purposes is to serve as the *source of conceptual slippability*. What this means is that, in times of severe pressure, the possibility arises that the concept itself might "slip to" some concept in its associative halo, with closer concepts of course being more likely slippages. This means that the nearby concept is tried out, and possibly accepted, as a substitute for the concept itself. We now say a bit more about each of these three aspects of a concept.

Category-membership criteria specify perceptual criteria that contribute toward membership in the category, with different weights attached to the various criteria, reflecting their level of importance. The purpose of this weighted set of criteria is, roughly, to "reduce" the concept to a collection of more primitive, more syntactic notions (*e.g.,* a letter is "reduced" to a set of interacting roles, and likewise, a role is "reduced" to a weighted set of desired properties of a set of bonded-together quanta).

By contrast, a set of *explicit norms* exists (at least for roles and wholes, which are sufficiently "semantic" concepts), the purpose of which is to make the "core" of the concept explicitly accessible to agents seeking ways to make a weak instance of the concept stronger, or conversely, to make a strong instance somewhat weaker without casting its category membership into complete limbo. Norms represent some of the program's declarative (*i.e.,* cognitively accessible) knowledge about the internal structure of each category in its domain.

A concept's *associative halo* consists of time-varying links of varying lengths connecting the concept with various other concepts. The links encode such information as: standard resemblances between letters (*e.g.,* between 'n' and 'u', after rotation through 180 degrees), standard analogical relationships between different types of roles (*e.g.,* between *post* and *tail*), conceptual proximity of various descriptors used in specifying norm violations (*e.g.,* "high" is the opposite of "low"), and so on.

Knowledge of letter–letter resemblances serves a dual function. It not only helps in designing new letters (*e.g.,* a good heuristic for a first stab at 'y' is to rotate an already-designed 'h'), but also serves as a warning that one letter has a tendency to be confused with another (*e.g.,* 'b' and 'h' are close enough categories that a shape intended for one of them could easily turn out to lie inside the territory of the other).

The set of links emanating outward from any concept effectively gives that concept a halo of conceptually close, potential-substitute concepts — concepts to which it might slip under sufficiently great contextual pressures. We will see the critical role that conceptual slippage plays when we go through an example of the designing of one letter under the stylistic influence of another.

1.7.3 The Visual Focus

The Visual Focus is where recognition of a single letterform takes place in a series of stages. This process is schematically summarized in Figure 1.13 and presented in further detail in Chapter 4. At the beginning, quanta belonging to a letterform are chunked together in a syntactic (*i.e.,* purely bottom-up) manner into small, simple structures having *a priori* perceptual plausibility (how this is done is fully explained

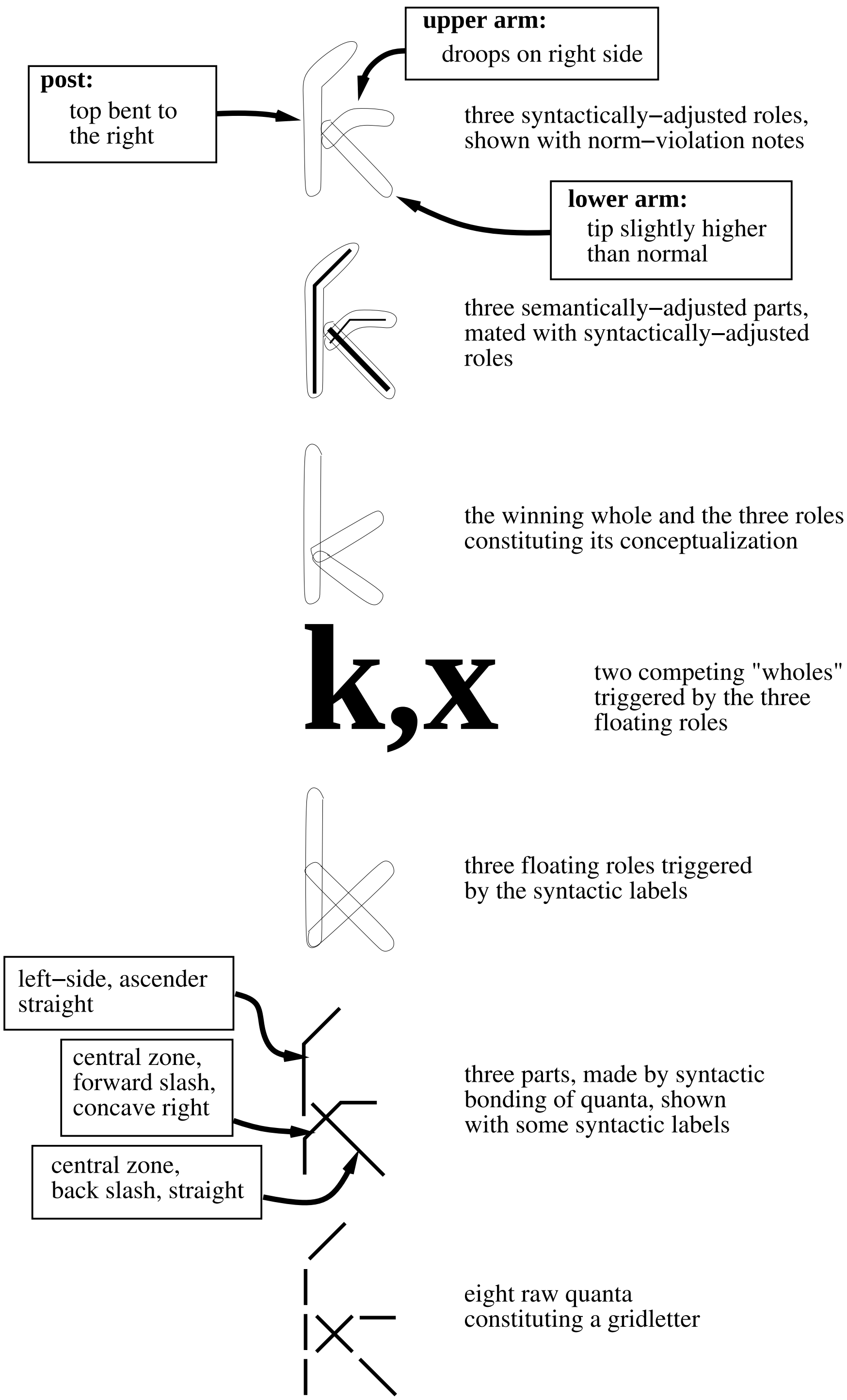


Figure 1.13: What happens in the Visual Focus. This figure starts at the bottom, with a set of unattached quanta to be seen as some letter of the alphabet. Processing proceeds from glomming quanta into parts, through labeling, role-activation, and semantic adjustment, to recognition of a complete whole with norm violations noted. This process is explained in detail in Chapter 4.

in Chapter 4).

In a sense, the next stage is the subtlest, because it is where syntax meets semantics — that is, where the specific nature of the categories into which the perceptual processing is feeding starts to make itself felt. To be concrete, these initial natural-seeming components of the letterform — called *parts* — are explored as possible *fillers of roles*. The intermediaries or "matchmakers" between parts and roles are *syntactic labels*, which describe simple geometric properties of the parts that have been built. These labels serve to selectively evoke a few roles that are promising mates for the parts. Once some roles have been activated, they in turn suggest one or more *letter concepts*. At this point, a top-down phase of perception is launched. Specifically, the most strongly-invoked letter concept pushes for one of its *conceptualizations* (coordinated role-sets) to be instantiated. This means that the roles involved therein try to exert top-down pressure on one or more parts to *adjust*, which means either for one part to break into two, or for two parts to fuse together or to swap quanta, in such a way that the resulting *semantically-adjusted parts* might better fit into roles.

The final stage of the recognition process occurs when parts and roles have mutually accommodated to yield a satisfactory realization of exactly one whole, without any strong rivals. (If the rivals do not go away, then the letterform under scrutiny is judged a poor one.)

Once a letterform's category has been identified, attributes of its *style* must also be extracted from it. Perhaps the most fundamental component of such stylistic perception involves looking at how the given parts fill their roles, and creating "stylistic notes" that summarize any norm violations found (*e.g.,* "bent crossbar", "unattached bowl", "post not touching baseline", etc.).

A norm violation essentially tells how a role has "bent" — that is, compromised its ideals — in order to accommodate to what is out there in the real world. In this sense, there is a systematic duality between the semantic adjustment of parts and the syntactic adjustment of roles.

Two other components of stylistic perception involve “filtering” and “focusing”. *Filtering* means ignoring most aspects of a shape and looking solely at abstract global qualities that it might have, such as lacking diagonals, or being filled with 135-degree angles, or being narrow, and so on. *Focusing* means concentrating on some particular local visual area, such as the top of the ascender zone, and extracting a shape from that zone. The extracted shape can then be posited as a possible motif. There is much more to the story of perception of stylistic attributes of a letter, of course, but this gives a feeling for what must go on.

1.7.4 The Thematic Focus

The Thematic Focus is both the court that judges, and the warehouse that stores, stylistic ideas. More poetically, the Thematic Focus is the site where spirit gradually crystallizes. Whenever any stylistic attribute is observed inside an individual letter in the Visual Focus, it has some chance of being transferred into the Thematic Focus for consideration as a candidate for potential propagation throughout the entire alphabet. Roughly speaking, the more often a specific stylistic attribute has been noticed as a systematic pattern in different letters, the more chance it has of making an upward shift in status — rising, that is, from being merely a casual quirky observation that has no influence to the status of an officially sanctioned force or guiding principle — a *theme*.

Whenever a new stylistic attribute is elevated to themehood during the creation of a gridfont, the default assumption is made that it is unslippable (*i.e.,* that its level of enforcement is maximal). By definition, default assumptions are not noticed at the moment they are made, and so the new theme will start out being obeyed stringently without question. However, if down the road some letter/spirit conflict gets sufficiently severe, pressures may arise that call that automatic assumption into question, at which point the level of enforcement will emerge from hiding and become an explicit variable that can be adjusted. Although it takes considerable pressure to "unbury" any of them, all levels of enforcement are ultimately under the control of the program, and can thus, in principle, be tampered with during the design process. The ability to recognize and manipulate these levels of enforcement represents a type of higher-level self-awareness on the part of the system, and will hopefully make the styles it comes up with more intellectual and sophisticated.

Every stylistic attribute, including levels of enforcement, is explicitly represented in the Thematic Focus and is thus globally accessible, not just in the sense of being *observable* by agents in the program but also in the sense of being *alterable* by agents in the program. Thus thanks to the Thematic Focus, all aspects of style are under complete control of the program itself.

1.8 The predictable unpredictability of the creative process

The notion of "elevation to themehood" is a little-appreciated but pervasive aspect of creative acts. In working on a gridfont, a human designer starts by being inspired by (let us suppose) a single seed letter. Aspects of this first letter, borrowed analogically, suggest further letters. But each of these secondary letters, when looked at by a style-conscious eye, will be seen to have stylistic attributes that are entirely its own and that were not implicit in or implied by the seed letter alone. These unexpected attributes are a result of the serendipitous interaction of two entirely unrelated sets of constraints: one set of constraints that together specify style (as so far perceived), and another set of constraints that together define the letter category currently being worked on. Some kind of compromise is found that accommodates both sets of constraints as well as possible.

The resulting physical shape cannot help but have some aspects — motifs or other little patterns — that simply were not present in any previous letters. Some of these will be noticed, often by sheer chance, in the act of visually scanning the letter, and added to the growing sense of the gridfont's style.

The beauty of this process is that it gives rise to what might be called "predictable unpredictability". That is, one can rely on unpredictable new stylistic attributes to emerge, simply as by-products of the creation of new letters. And then, once any such attribute is recognized and explicitly elevated to themehood, it becomes an active force in shaping yet further letters. Of course this means that the process is in some sense recursive — new letterforms give rise to new emergent attributes, which in turn give rise to new letterforms, and so on. (It is not even so uncommon for a letterform to suggest a stylistic attribute that goes out, affects some other letters, and then comes back like a boomerang and causes the letterform itself to be modified, thus undermining the very shape that gave rise to it!) The upshot is that new, very unexpected stylistic attributes are continually emerging — ideas that the designer never had in mind at the outset and never would have dreamt of, had this serendipitous interaction of complementary constraints, iterated over and over again, not taken place. This highly unpredictable meandering in "style space" reflects the extreme subtlety of the creative act.

1.9 Four interacting emergent agents

Emerging from the many small actions of codelets are four conceptually separable types of large-scale activities:

1. the high-level conceptual activity of *selecting or inventing a letter conceptualization, and then devising a new letter-plan* (*i.e.,* either an idea for an as-yet

undesigned letter or a possibility for improving an already-designed letter);

2. the intermediary activity of *translating a fresh letter-plan into a concrete shape* on the Scratchpad;

3. the relatively concrete perceptual activity of *examining a newly-drawn shape and categorizing it* (*i.e.,* deciding which letter of the alphabet it is, and how unambiguously so);

4. the more abstract perceptual activity of *recognizing the stylistic attributes of a newly-drawn letter, and deciding how to treat them* (*i.e.,* selecting pieces of a letterform that might become motifs, noticing aspects of the letterform that might be described by a simple rule, or finding norm violations and describing them in "exportable" ways, as well as taking note of whether a newly-observed stylistic attribute supports or opposes stylistic attributes previously observed in other letters.

It is often convenient to speak as if these emergent activities were carried out by four explicit and cleanly separable modules, together comprising the totality of the program. (These high-level agents could be likened to the agents referred to in [Minsky, 1985].) The names we shall apply to these hypothetical modules or agents are, respectively, the *Imaginer*, the *Drafter*, the *Examiner*, and the *Abstractor*, and we shall briefly describe each of them in turn. However, it must be borne in mind that these modules are in some sense convenient descriptive fictions, in that each one is simply an emergent by-product of the actions of many codelets, and their activities are so intertwined that they cannot be disentangled from each other in a clean way.

1.9.1 The Imaginer

The Imaginer does not deal with, or even know anything about, the constraints defined by the Letter Spirit grid (*i.e.,* the fact that letterforms are made up of discrete quanta); rather, it functions exclusively at the abstract level of *roles*. Its job is to make suggestions regarding roles, which it then hands over to the Drafter (which will attempt to implement them concretely in conformity with the constraints of the grid — that is, as parts composed of quanta). The Imaginer can make suggestions of two distinct types — *norm-violation suggestions* and *role-regrouping suggestions*. Although both types can lead to highly novel instantiations of letters, suggestions of the first type tend to be tamer than ones of the latter type, since they do not cut quite so close to the core of the concept.

A *norm-violation suggestion* starts with an already-existing set of interacting roles (*i.e.,* a particular conceptualization for the letter); then, for one or more roles in that set, it suggests one or more ways of violating associated norms. For instance, suggestions such as "Bend the tip of the ascender to the right", "Use a short crossbar", "Don't let the bowl touch the post at the x-height", "Make the bowl narrow", and so on (suitably expressed in an internal formalism) would be typical norm-violation suggestions. Though fairly specific, such suggestions still require more fleshing-out to be realized on the grid itself.

A *role-regrouping suggestion* is more radical and deep, in that it involves tampering with the very essence of the letter — in other words, coming up with a new conceptualization for the letter. This means taking apart one or more roles and making new roles that combine aspects of the old roles. An easy-to-grasp example is the earlier-described conceptual move from imagining 'x' as two intersecting slashes to imagining it as two "kissing" angle-brackets. Role-regrouping is very subtle because it takes place completely at an abstract level. That is, *no shapes are involved at any time*; rather, the Imaginer deals exclusively with abstractions — abstractions that, to be sure, have the general "feel" of shapes, in that they are associated with spatial locations and have spatial functionalities — but such abstractions are not shapes.

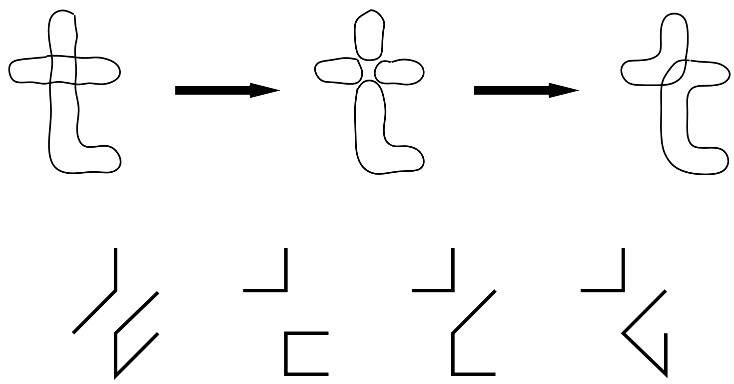


Figure 1.14: A schematic illustration of the process of conceptual regrouping of the default roles in the letter concept 't'. Below the new conceptualization are shown some sample gridbound 't's that would naturally follow from it but not from the default conceptualization.

For the sake of concreteness, let us consider the conversion of one conceptualization of 't' into another (see Figure 1.14). The idea of a "linear unit" (shorthand for the concept "long thin shape that does not bend much") that passes through some known point (more precisely, an r-role) suggests in a fairly obvious manner the idea of breaking the linear unit into two *shorter* linear units, using the point as the divider. Applying this "breakup" idea both to the *ascender* and to the *crossbar* role in a 't', we obtain four shorter roles that converge like spokes to a hub (one of the r-roles, of course). Now comes the regrouping operation. Whereas the "west" and "east" sub-units had formed one conceptual linear unit (the crossbar), and the "south" and "north" sub-units another (the ascender), we now pull each of these units apart and regroup their pieces into two new conceptual units — one made up of the "west" and "north" sub-units, and the other of the "south" and "east" sub-units. Despite the picture, one has to keep in mind that the pulling-apart and reattaching operations are actually just *mental*, as opposed to being splicing-operations on graphic shapes.

We now have come up with two completely new roles (and of course the unperturbed *central-point* r-role), which give us a novel way of thinking about what 't' is — that is, a novel starting-point for creating new 't's, some of which are also found in Figure 1.14. Norms for the roles belonging to this new conceptualization have to be derived from the norms associated with the roles in the old conceptualization.

Once a new conceptualization has been produced, it can be handed over directly as a suggestion to the Drafter, or norm-violation suggestions can be made in addition, and then the whole thing handed over as a package to the Drafter.

1.9.2 The Drafter

The Drafter, unlike the Imaginer, does know about the grid; indeed, its main function is to take the Imaginer's high-flown, grid-independent suggestions and adapt them to the down-to-paper and severe constraints of the grid. In other words, the Drafter must convert the Imaginer's abstract suggestions into concrete instructions for drawing shapes on a grid on the Scratchpad.

Here is an example that could easily come up in designing a 't' or an 'f'. A norm-violation suggestion like "Make the crossbar short", which in real-life circumstances would offer a letter-designer a full continuum of possibilities, offers much less freedom to a designer restricted to the grid. For a gridbound 't' or 'f', a conventional (*i.e.,* norm-respecting) crossbar would be a horizontal line segment composed of two quanta. Obeying the suggestion to make it short would thus seem to offer just three alternatives: dropping the left quantum, dropping the right one, or dropping both. Of course, dropping both quanta would seem very drastic, if not outrageous (although possibly the right solution for some very far-out styles); thus in all likelihood, the Drafter should opt for drawing just a single quantum, probably a horizontal one. Then the question is whether it should draw it to the left or to the right of the ascender. This choice will certainly depend in part on precedents in other letters in the developing gridfont (*e.g.,* if the decision "Draw a quantum on the *left* side of the ascender" had already been made in the case of the 'f', then the 't' might want to follow suit), but it will also depend on how strong the potential letter's category membership will be.

The Drafter will generally be uncritical of the suggestions it receives from "on high", but it is conceivable that it will encounter so much difficulty converting them into reasonable instructions for a grid-shape that it will send back a "complaint" to the Imaginer, asking it to revise its suggestion in some way. Generally, however, feedback to the Imaginer comes from the perceptual agents — the Examiner and the Abstractor. This all-important feedback loop will be described shortly.

Note that, because of the localization of grid-knowledge in the Drafter, it would be possible, at least theoretically, to use the very same Imaginer with a totally different grid (*e.g.,* a hexagonal or triangular grid, or even no grid at all). The Drafter would of course have to be altered to reflect the specific properties of the grid involved.

1.9.3 The Examiner

The Examiner is responsible for taking the specification of a gridletter in terms of its quanta and determining which (if any) of the 26 letter categories it belongs to, and how strongly and unambiguously so. All processing in the Examiner takes place in the Visual Focus. The Examiner is the only agent of the four that has been developed at this point in the project. It is described in detail in Chapter 4, and more briefly here.

It is useful to cast the Examiner's work in terms of *syntactic* and *semantic* operations.[10] Syntactic operations are purely bottom-up chunking operations. They

[10]Here, use of the term "syntactic" has nothing to do with grammars, but with the idea of manipulating shapes *without regard to concepts*, whereas "semantic" operations involve the influence of concepts in permanent memory (*i.e.,* in the Conceptual Memory).

serve to put quanta together in a way that would be perceptually plausible no matter what type of shape was being recognized — whether the domain was roman letters, Arabic letters, Bengali letters, Chinese characters, or even pictures. In other words, syntactic operations yield *context-free chunkings* that would presumably arise as the result of any naturally-evolved visual system.

Semantic operations, on the other hand, depend on the set of categories into which the shapes are being channeled. In the case of letters, this set of categories will be a writing-system-specific repertoire that in a person has been acquired through years of experience. Thus the semantic chunking operations that would occur in the visual system of a Chinese person looking at Chinese characters would tend to result in the extraction of shapes quite different from those that would occur if an English speaker looked at the same characters. Semantic operations take the output of syntactic actions (which occur earlier in perceptual processing) and adjust it. Adjustment consists in making parts bigger or smaller by chopping and borrowing subparts. The end result is a set of *semantically-adjusted parts* that conform sufficiently well to expected or known abstract (*i.e.,* semantic) structures — in other words, a "marriage" of bottom-up structures coming from current sensory stimuli and top-down expectations defined by prior knowledge of letter concepts.

Processing begins at the level of quanta and starts out being completely bottom-up. Quanta get syntactically chunked into parts, which are then assigned any number of syntactic labels (*e.g.,* "straight", "tall", "central-zone", "zigzag", "left-side", etc.). Top-down semantic influence enters the picture as the labeled parts are matched up with conceptual roles. As the interpretation rises towards the level of letters (or, wholes), even more top-down influence is brought to bear.

The Examiner has been fully implemented. For a detailed description of how it works and how it compares with other models of letter recognition, see Part II.

1.9.4 The Abstractor

The Abstractor is concerned with a more abstract type of category membership — namely, *stylistic consistency*. It is not enough for a candidate letterform to be perceived by the Examiner as a strong member of its intended letter-category; that letter must also be judged by the Abstractor as embodying the same stylistic qualities as the seed letter(s) and any already-generated letters. This requires a set of high-level descriptions of stylistic qualities to be manufactured as the alphabet develops. Obviously, no single letter contains all the information about the style it belongs to, and so stylistic attributes from various letters, as they come into existence, must be assembled in a global list belonging to the gridfont as a whole. This list is of course the Thematic Focus. Thus whereas the 26 *letter* categories exist in the Conceptual Memory *prior* to any run, a single *spirit* category gradually comes into existence in the Thematic Focus *during* a run.

The types of stylistic attributes that the Abstractor looks for in order to judge a candidate letterform for its consistency with the style (which, as it is constantly evolving, constitutes a moving target) include those mentioned earlier — *role traits*, *motifs*, *abstract rules*, and *levels of enforcement*. The new letter is then assigned a "stylistic-coherency rating" that reflects how many prior themes are echoed in it, and how strongly they are present.

The Abstractor also tries to extract from any new letter *new* stylistic attributes, in order to extend the set of themes that collectively define the emerging style. Figure 1.15 illustrates three of many possible stylistic attributes that might be noticed when a particular 't' is looked at by the Abstractor.

An attribute discovered in a single new letter may not be considered strong enough to be elevated to themehood, but if that observation is reinforced by finding the attribute echoed in other new letters, then it stands a good chance of becoming a new theme and, as such, helping to drive the design of further letters and even the

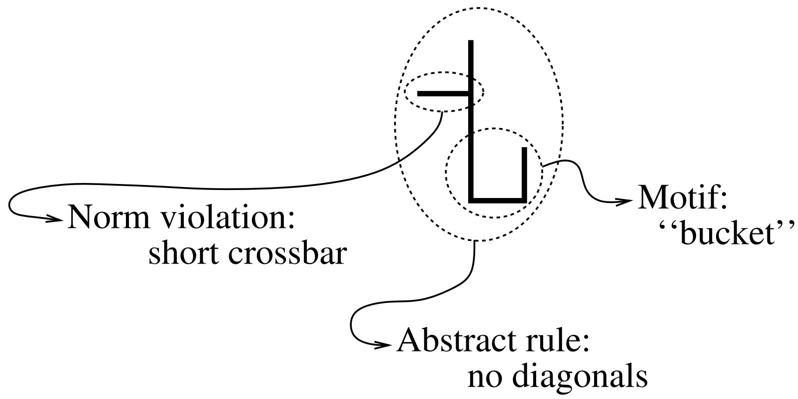


Figure 1.15: Three stylistic attributes that might be posited by the Abstractor in its scanning of a particular 't': (1) an interdiction of diagonals (an abstract rule); (2) a "bucket" (a motif); (3) abnormally short crossbars (a norm violation).

retroactive modification of older letters.

As was pointed out before, one completely predictable thing is that stylistic attributes will emerge in a completely unpredictable fashion. Here is a very simple but very typical example. Suppose the Abstractor happens to notice that neither the seed letter nor the first new letter generated contains any forward slashes. This could well be simply a coincidence, not at all intended. Nonetheless, the Abstractor may then generalize from these two examples and thereafter strictly forbid forward slashes. Such a decision, quite unforeseeable, will of course have global ramifications. On a different run where the same two letters existed, a totally different course could be taken if that observation were not made, or even if it were, if the interdiction of slashes were taken loosely.

As this brief survey shows, recognizing stylistic characteristics and formulating them appropriately (neither too specific nor too general; neither too rigid nor too loose; etc.) is most complicated, and the Abstractor's job is therefore a very subtle one.

1.10 The central feedback loop of creativity

A major part of the Letter Spirit challenge is the problem of *attempting to do in one framework something that has already been done in a significantly different framework.* Here, the two frameworks are different letter-categories, such as 'd' and 'b'. A designer is handed a somewhat off-center member of the first category, such as the 'd' shown in Figure 1.16, and the challenge is to transport its eccentricity — its stylistic essence — into the other category, or in other words, to reproduce the *spirit* that it embodies in the second framework, despite the fact that the second framework is by no means isomorphic to the first. (Many people think that 'b' and 'd' actually *are* isomorphic categories, exactly identical but for spatial reflection. This is a fair first approximation to reality, but far from the truth, as we are about to see.) What would be an appropriate 'd' in the same spirit as this 'b'? This is a typical, if small, Letter Spirit challenge, which we hope the reader will take seriously.

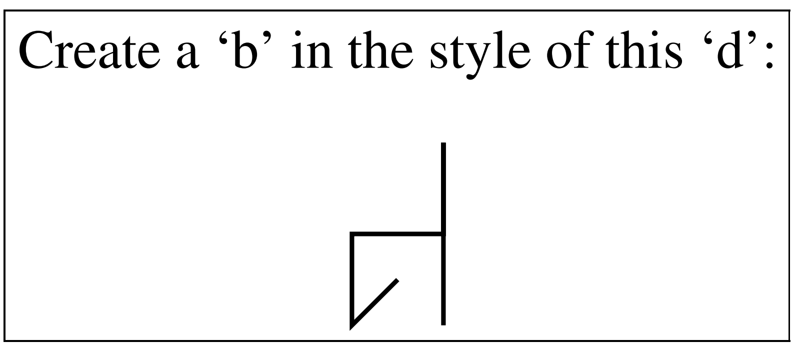


Figure 1.16: An analogy puzzle typical of those that crop up all the time in the creation of uniform alphabetic styles.

Such transport of spirit cannot be done in a reliable manner except by trial and error. Guesses must be made and their results evaluated, then refined and evaluated again, and so on, until something satisfactory emerges in the end. We refer to this necessarily iterative process of guesswork and evaluation as the *central feedback loop of creativity*. (Of course, it is not anything close to being a "loop" in the strict computer-science sense of the term — it is, rather, a back-and-forth flow of intensity of activity in different types of tasks.)

In the case of the above challenge, the most obvious move — namely, mirror-reflecting the given 'd' — yields an attractive candidate shape (see the left side of Figure 1.17), but one whose category membership is unfortunately quite ambiguous, since it is not fully closed at the bottom. Although this shape might well be seen as a 'b', it might just as easily be seen as an 'h'. Such ambiguity is certainly not a desirable property for a letterform to have. What can be done?

This is an analogy problem featuring a *snag*. Fortunately, an escape hatch is provided by a *conceptual slippage*. The idea in this case is to slip from one concept to its opposite. Here, the description of the norm-violating bowl slips from "unattached at *bottom*" into "unattached at *top*". This yields the solution shown in the right side of Figure 1.17. Note that the new shape's category membership is quite unambiguous, and yet it manages to capture, at a high level of abstraction, much of the spirit of the seed letter. (Precisely *how much* it captures is, of course, debatable. Different designers have different sets of unconscious criteria for making such judgments, and those differences give rise to very different styles of style-making.)

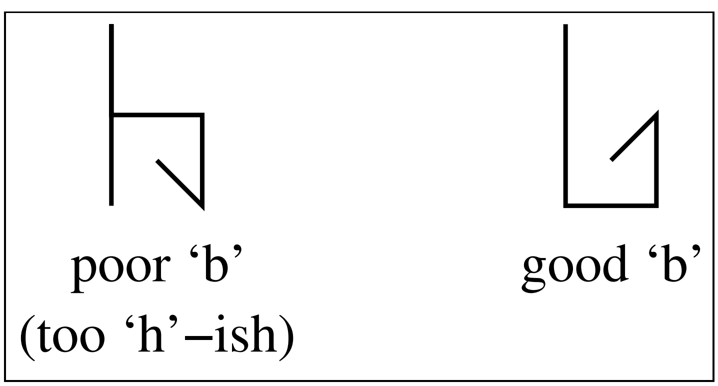


Figure 1.17: Two possible solutions to the preceding puzzle, one weak and one strong.

Taken together, this 'b' and the original 'd' suggest all sorts of rich new ideas for the rest of a hypothetical alphabet. This little example gives the flavor of the central loop of the creative process, at least as it occurs in the human mind. It is, incidentally, gratifying to note that in the design of Friz Quadrata, a relatively common display typeface, the analogous solution to the analogous problem was found.

1.11 The strangely circular origin of a style

Although single-letter-to-single-letter analogy problems of the sort just shown capture a good deal of the Letter Spirit challenge, the full challenge is actually quite a bit more complicated. There are two main ways in which the 'd'-gives-rise-to-'b' example is misleading. Firstly, it seems to imply that each new letter comes directly from a tight analogy based on precisely *one* previously-designed letter, whereas in fact the final design of any given letter is far more typically an amalgam of influences from any number of different letters. While it is true that the *first draft* of a new letter commonly springs from just one other letter, once the draft is "on the table", there comes a period of examination, comparison, critiquing, and revision in which other letters' influences get blended in to various degrees until there is no simple pedigree at all. Even when one finally comes to feel happy with the design for the given letter and goes on to the next one, one often finds that a few letters down the road, one is drawn back to that letterform and feels a need to tamper with it further. Indeed, it is often very hard to decide when one has reached the end of the design process, either of a single letter or of a gridfont as a whole.

This brings us to the second way in which the 'd'-to-'b' example is misleading: namely, it suggests a clean distinction between "parent" letters and "child" letters — that is, between fully-finished letters and still-evolving letters. In this very example, for instance, the clever discovery of the "flipped" solution for the 'b' could perhaps have retroactive influence on the very 'd' that inspired it (one could simply mirror-reflect the new 'b', for instance). This act would overturn the supposedly stable source for the analogy and thus, to some extent, destabilize the entire alphabet. After all, if *every* letter is at all moments potentially subject to revision, then where on earth is *terra firma*? What can be relied on? Where lies the ultimate source of the gridfont?

One way out of this confusion would simply be to declare any and all human-provided seed letters to be immutable and inviolate; this would establish a fixed grounding for all the remaining letters. However, this is certainly not the way that a human designer works. A more faithful imitation of true human design practice would require each designed letter on the Scratchpad and each stylistic theme in the Thematic Focus to have a numerical *stability value* suggesting how deeply it should be respected at any given moment. Letters with low stability values would of course be likely to take their cues from ones having high stability values, rather than the reverse. But stability itself would be subject to change: each stability value would be periodically updated, and the calculation would give credit for various factors, such as being a seed letter (this should, after all, count for *something*!), being a strong member of a category, and being "consistent" with other letters and themes.

What makes this whole stability notion rather convoluted is the fact that credit for consistency with other letters and themes would have to be *weighted* rather than egalitarian: in particular, more points should be awarded for being consistent with a letter or theme that *itself* has a high stability value. That is, stabilities themselves should serve as the weights in determining stabilities. There is thus a deep — though not paradoxical — circularity to the whole notion.

Where, then, does the ultimate source of a gridfont lie? The answer is not simple; it depends on the stage at which the question is asked. At the very outset, of course, the source can lie in only one place: the set of seed letters. There, it is totally implicit. As the style spreads to other letters, it becomes more explicit and abstracted. It also becomes less and less localized. Gradually, however, as *themes* congeal, the gridfont's source becomes somewhat "institutionalized" in them, and thus more localized after all. Nonetheless, a set of abstract themes without any concrete examples does not contain enough information to determine a style. For this reason, once a gridfont is fully created, the answer to the question about its ultimate source is that it is spread out among a small set of themes and a few key letters that embody them — not necessarily including any of the seed letters!

These kinds of considerations help to reveal the somewhat frightening complexity of the full Letter Spirit project. However, the core of the design part of the project, or at least the design aspect that has to be tackled first, is without any question the challenge posed by single-letter-to-single-letter analogy-making, as illustrated by the previous puzzle. We therefore turn to a consideration of how that type of core challenge could be implemented.

1.11.1 Towards the implementation of the central feedback loop

We now run through another letter-to-letter analogy, and a rather simple one at that, in order to give a sense of how the central feedback loop of creativity will be implemented in Letter Spirit, featuring an interaction of all four emergent agents. We suppose that Letter Spirit is given as its lone seed letter an 'f' with a conventional ascender (*i.e.,* a tall vertical stroke that curves over to the right and then back down at the top), but with *no crossbar at all* (see Figure 1.18). What kind of gridletters might this seed inspire? What kind of overall style? Readers might think about how they would move from this 'f' to other letters. What letter would *you* design next, and how would you go about it?

To begin with, the seed's letter category must be identified (in itself a nontrivial

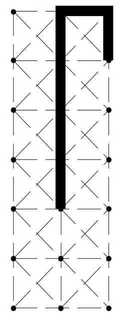


Figure 1.18: A possible seed-letter 'f' featuring a very striking norm violation.

task, given the eccentricity of the letterform). To this end, the Examiner is invoked. The quanta in the seed letter are quickly glued together, and because there is a fairly weak juncture near the top — namely, where the ascending line bends to the right to form an "overhang" — two distinct syntactic parts are made. After being suitably labeled, these parts wake up two roles: *post* and *hook*. Since there is nothing else to see, no other roles are strongly activated. This pair of activated roles now activates two wholes — the letter-categories 'f' and 'l'. Counting against 'f' is the lack of crossbar, but counting against 'l' is the hook at the top. This is a small dilemma. However, let us suppose that the power of two strongly-filled roles overwhelms the pressures for seeing 'l', and the shape winds up being seen as an 'f' whose primary stylistic attribute is the lack of anything to fill the *crossbar* role. Thus "crossbar suppressed" — a norm violation — is the main stylistic note attached to the given seed letter. (Of course, in principle, "no diagonals" could be a tentative abstract rule extracted from this letterform, but let's leave that aside.)

We now move from the perceptual to the generative phase. Given that 'f' and 't' are linked *a priori* as similar letters in the Conceptual Memory, there is a high probability that 't' would be the next letter tackled. Let us assume, for simplicity's sake, that the 't's ascender would be a conventional one, so that all we are concerned with is figuring out the nature of its crossbar. An obvious idea for the 't' would be to suppress *its* crossbar. Like any good copycat, the Imaginer would have little trouble coming up with that analogy, since the role *crossbar* exists in both ‘f’ and ‘t’; all it would need to do is take the norm-violation that describes ‘f’ (“crossbar suppressed”) and copy it literally into a suggestion for the ‘t’. Upon receiving this norm-violation suggestion, the Drafter would have no problem converting it into a grid-oriented instruction saying, in effect, “Draw no horizontal quanta at the x-height” (the height of the top of a lowercase ‘x’). So far, so good. The Drafter now easily renders this suggestion for a ‘t’ on the Scratchpad (see Figure 1.19), leaving it up to the Examiner to look at it and make of it what it can.

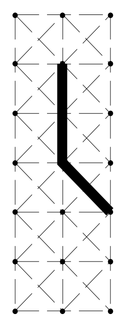


Figure 1.19: One quite far-out idea for realizing the concept ‘t’ in the same style. The spirit is willing, but the letter is weak.

The Examiner sends out codelets that quickly glue the quanta together into into a single perceptual part — namely, a vertical line rising from the baseline to somewhere above the x-height. After labeling, this wakes up the role *ascender*, and since there are no other parts, no other roles are activated. This rather sparse “combination” of activated roles now sharply activates one and only one whole — namely, the category ‘l’. There might in addition be a rather feeble activation of the category ‘t’, due to two facts: (1) this part ascends but does not reach as high as the ‘f’ did; and (2) the sole letter category featuring an ascender whose default height is lower than that of ‘f’ is ‘t’. Despite this weak competition from ‘t’, the category ‘l’ is almost sure to be found the overwhelming winner. At this point, the Examiner, knowing that ‘t’ was intended, pronounces the attempt at 't' a failure, and provides what is hopefully an accurate diagnosis: the fact that the role *crossbar* never got awakened at all.

This information goes back to the Imaginer, which was, after all, the source of the idea of suppressing the crossbar entirely. So the Imaginer is now caught in the crossfire of Letter and Spirit pressures: on the one hand, it has just learned that suppressing the crossbar leads to disaster (this is Letter pressure), but on the other hand, it wants to follow the stylistic lead of the 'f' (Spirit pressure). Something has to give!

Luckily, there is a way out, provided by *conceptual slippage*, which involves consulting the Conceptual Memory for potential substitutes provided by conceptual halos. In the halo of the concept "suppress", the Imaginer finds such close neighbor-concepts as "austerity", "minimality", "sparsity" — or if not all of those poetic concepts, then at least the simpler concept "underdo" (or a more formal structure representing that idea). Thus, under the pressure created by the failure of using the concept "suppress", it is quite likely that the Imaginer will make a *slippage* — namely, it will take the nearby idea "underdo" and try it on for size. In other words, the Imaginer supposes that "underdoing" the 't's crossbar is the next-best thing to all-out suppression of it. This slippage is of course the key creative breakthrough. It now just needs some fleshing-out, still to be done by the Imaginer.

In order to translate the vaguish "underdo" into a more specific operation, the Imaginer must have information about the *meaning* of "underdo". This is available through its internal definition, which (in a suitable formalism) is given as "reduce the key dimension of". Now the Imaginer must consult the norms attached to "crossbar" in order to find out if a crossbar has a key dimension, and if so, what it is. It finds that for "crossbar", there is only one norm involving size, and that is horizontal length. This fortunate discovery allows the vague "underdo" hint to be straightforwardly translated into a norm-violation suggestion that says, in effect, "Make a short crossbar". The Imaginer thus hands to the Drafter a modified letter-plan that incorporates this suggestion.

If all goes well, this can lead to a 't' with a one-quantum crossbar — in other words, a perfectly acceptable and style-loaded 't' (see Figure 1.20). It is, of course, debatable how faithfully the 't''s in Figure 1.20 preserve the truly austere spirit of the seed letter 'f', but no one could dispute that either of them constitutes a pretty reasonable attempt.

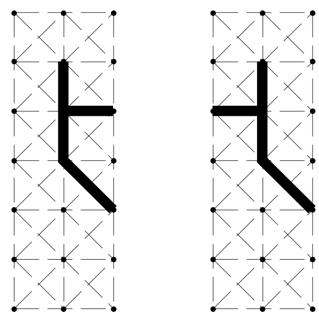


Figure 1.20: Two refinements of the shape introduced in Figure 1.19. In each of them, the spirit of the barless 'f' is somewhat diluted, but in compensation, the letter is stronger.

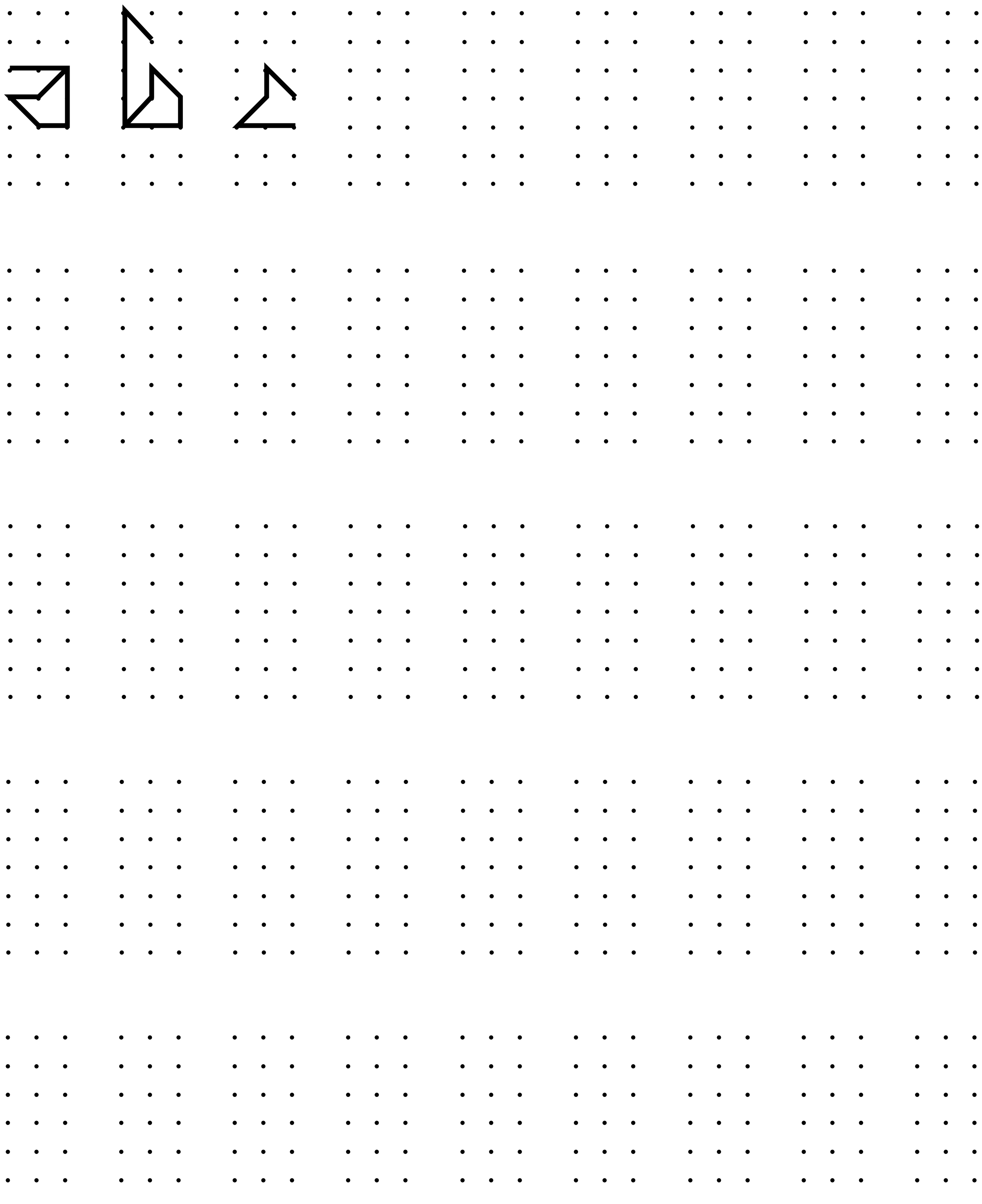
Not all feedback in the central loop of creativity originates in the Examiner; it can originate in the Abstractor, too. In contrast to the example just described, it is possible that the Examiner will be satisfied with a given letterform but the Abstractor will be dissatisfied. In such a case, the letterform is rejected and a message is sent to the Imaginer explaining how the letterform fell short of one or more *stylistic* criteria (as opposed to *letter-category* criteria, as in the scenario just given). The Imaginer must then modify its suggestions accordingly.

The hypothetical interaction of these four agents, wherein ideas are suggested, critiqued, revised, possibly even abandoned and then regenerated, and so on, meshes with our intuitive sense of what human creativity really is. It seems to us fair to say that this kind of emergent, unpredictable processing constitutes a program's *making its own decisions*. This concurs with the views expressed by [Johnson-Laird, 1988] that free will and creativity are closely related. It is our hope to realize a program of this degree of complexity and subtlety. This thesis represents the first strides in that direction. It remains to be seen how far we can actually carry it.

A LETTER SPIRIT CHALLENGE

We have found that the best way to get a deep understanding of the creative process of gridfont design is to attempt designing a gridfont yourself. At this point, we invite the reader to try to finish designing the gridfont that we have started on the next page. Remember, the seed letters are not immutably set in stone, but they should provide at least some momentum for the budding style. There are more than 26 grids provided. The extras can be used as scratch-space or for alternative letters. One of the many possible solutions to this challenge appears at the end of Part I.

It is very informative to try and consciously track the design process as it unfolds. For instance, note which letters are designed when. Also think about the role that inter-letter analogies play as the full-scale gridfont emerges. Another thing to note is at what points a whole alphabet scan is made, and how often "finished" letters get re-examined and changed.



CHAPTER TWO

Competing Models of Typeface Design

...an enormous amount of subtlety lies behind the seemingly simple letter shapes that we see every day, and the designers of high-quality typefaces have done their work so well that we don't notice the underlying complexity. — Donald Knuth

The Letter Spirit microdomain is meant as an experimental testbed for research on creativity. Put another way: we are mostly interested in *creativity*, and we just happen to be using typeface design as a convenient vehicle for study. We believe that typeface design is an inherently creative activity that shares a host of fundamental mechanisms with other temporally-extended, large-scale creative acts such as composing music, designing buildings, writing novels, and designing and implementing cognitive models.

So what, then, counts as related work? It depends on what one considers the focus of Letter Spirit to be. If one conceives of the aims of Letter Spirit broadly, as we do, considering it to be a project whose aim is to model and thereby deepen our understanding of the creative process, then any work on creativity, discovery, or design counts as related. We consider models of creativity in the next chapter.

Considered narrowly, the project could be construed as primarily concerned with *letters* — the perception and generation of letters, to be specific. In that case, related work includes all prior research done on recognition of handwriting and printed documents, and of course any programs that purport to create consistent typefaces. There are literally hundreds of approaches to letter recognition and word recognition (many of which are considered in detail in Chapter 6), but only a handful of projects concerned with letter *design*. In this chapter we address just those models — models concerned with the automation of letter design. Curiously, the last model we discuss is a program deliberately designed to be competition to Letter Spirit. This is the connectionist GridFont model of Stork, Grebert, and co-workers [Grebert et al., 1992].

Our discussion of competing models of typeface design will necessarily involve consideration of each model's creative potential. As we have emphasized, we believe that typeface design is an inherently creative activity. Thus any model that "does it right" will have some important things to teach us about the creative process. Discussing the relationship between existing models of typeface design and creativity provides a context within which Letter Spirit can be better understood.

2.1 I thought Metafont did that!?

Not infrequently, when we state our goal as "Making a program that can design letters of the alphabet in many different styles", intelligent and well-informed listeners ask, "But doesn't Donald Knuth's Metafont program already do precisely that?" This all-too-common misconception merits a careful reply.

Metafont [Knuth, 1982] is a tool that allows a human user to design parameterized letters and letter-parts, and thus to fashion an entire alphabet (not only our own, but also Greek, Hebrew, Hindi, Japanese, etc.) in a uniform way. However, all design decisions, from first to last, are made by the human user. For instance, if the 'h' is to be made from the 'n' by making the vertical post on the left side taller, the user must tell the program precisely that, since the program itself has no prior representation of the concepts 'h' and 'n'. Lacking any knowledge whatsoever of letter concepts, Metafont no more designs letters than a word processor composes poems or a piano composes waltzes. Metafont merely *facilitates* letterform design for a human designer (and can therefore enhance a human designer's creativity). Of course, Knuth himself never made any claims that Metafont is an autonomous letter-designer, but a nodding acquaintance with it can easily give rise to such an impression.

We bring this up because it helps put the goals of Letter Spirit in perspective. In particular, it is enlightening to analyze what is lacking in Metafont and many other programs commonly but erroneously thought of as "models of creativity".[1]

In a nutshell, the problem is that most such programs *don't know anything about what they are doing*. This phrase actually has two quite distinct meanings, both of which usually apply to the programs under discussion. The first meaning is "the program has no knowledge about the *domain* in which it is working". The other is "the program has no internal representation of the *actions* that it is itself taking, and no awareness of the *products* that it is creating". Both of these gaps are serious defects in anything that purports to be a model of creativity.

2.2 DAFFODIL: Another non-creative typeface designer

Along these lines, a quintessential, though little-known, example is the DAFFODIL project [Nanard et al., 1989].[2] From our experiences in talking to people about our

[1] More musings on Metafont may be found in *Metamagical Themas*, Chapter 13 [Hofstadter, 1985].

[2] DAFFODIL is an acronym standing for **D**eclarative **A**pproach **F**or **FO**nt **D**esign by **I**ncremental **L**earning.

project, we suspect that many people, if they were familiar with DAFFODIL, would point to it as a program that has already achieved our stated goals. It is not clear to us whether the designers of DAFFODIL intended their work to be a model of human creativity; but as we have argued above, we cannot imagine how a font-designing program could possibly work *without* creativity. In any case, thinking that DAFFODIL had somehow beaten us to the punch would be a serious misconception, and for many of the same reasons that we gave above in discussing Metafont.

DAFFODIL is a program that combines three types of input:

1. a set of 26 *backbones* (one for each capital letter), where a backbone, rather than being a shape, is an immutable description of the given letter in terms of a dozen or so *a priori* stroke-types, which are simply phrases such as "horizontal segment", "vertical segment", "left arc", and so on;

2. a set of *decorations*, a decoration being a rigid graphical shape concocted by a human, usually but not necessarily being an ornate or florid design;

3. a *mapping* that associates stroke-types with decorations.

Given these three ingredients, the program executes the mapping on all 26 letters — that is, it carries out a systematic *stroke-to-decoration substitution operation*, which converts the set of 26 abstract backbones into a set of 26 actual graphic shapes, each one being some particular arrangement of the given decorations (this process is illustrated in Figure 2.1). *Et voilà* — a new typeface has apparently been "created" by the program.

To be sure, a new typeface has been *generated*; however, to think of what DAFFODIL does as in any way resembling creation by humans would be an egregious error, for the following reasons:

- a tacit assumption behind the DAFFODIL project is that style does not involve

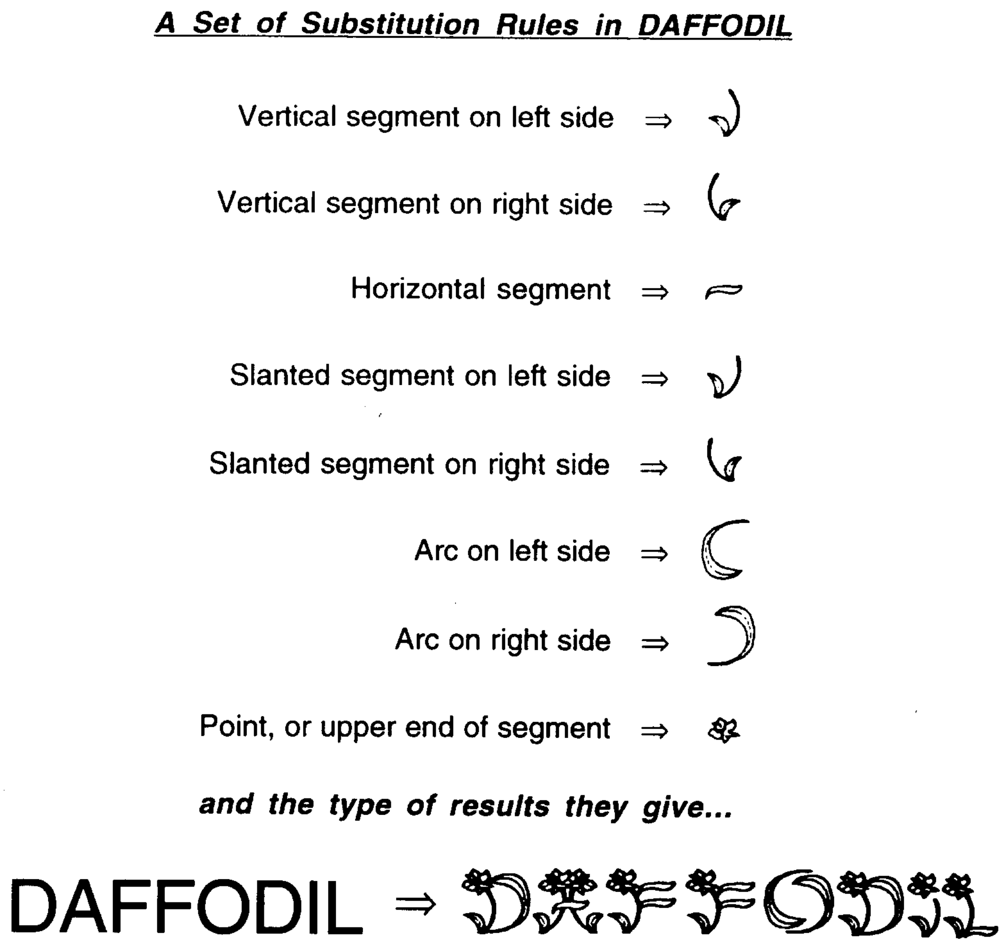


Figure 2.1: Transformation rules and sample output from the DAFFODIL program, developed by Nanard *et al.*.

manipulation of abstract concepts behind the scenes, but merely involves playing with how surface-level aspects are to be rendered;

- DAFFODIL's 26 backbones are all designed by a human and fed to it from the outside, and once a backbone has been fed in, it remains absolutely fixed, so that any creativity involved in this deep-lying aspect of the letterforms is external, not internal;

- DAFFODIL's knowledge of each letter-concept (*i.e.,* the letter's backbone) is very impoverished, and it has no knowledge of how letter-concepts are interrelated, other than the fact that two backbones might share one or more stroke-types;

- the decorations to be "plugged in" by DAFFODIL are all designed by a human and supplied to the program, and once a decoration has been fed in, it remains absolutely fixed, so that any creativity involved in this shallow aspect of the letterforms is external, not internal;

- no decisions are made by DAFFODIL — it simply plugs in decorations as required, occasionally asking a human being for approval or disapproval;

- DAFFODIL has no ability to perceive or judge anything it has produced — that is, it cannot tell whether a shape it has made actually looks like an 'A', whether it is attractive or ugly, or whether it fits well or poorly with other already-produced letters.

Taken alone, any of these points speaks tellingly against DAFFODIL as a model of creativity; taken together, they pretty much destroy any claims that might be made for it as such a model.

Like Metafont, DAFFODIL is at best a sophisticated tool. People design the backbones and the ornate shapes with which to flesh them out. People specify the heuristics for associating a particular curlicue with a section of the backbone. People look at the final product to find out if it is acceptable as a letterform. The program barely does anything interesting at all. Its job amounts to syntactic substitution and graphical display. So in the end, any resulting font is not created by the *program* but by a *person*, using the computer as a helpful tool.

Another somewhat deeper complaint: DAFFODIL may well be a useful tool enhancing certain aspects of letter design, but we disagree with the philosophy of design that it tacitly espouses. DAFFODIL is based on the assumption that style amounts to no more than simple ornamentation at the surface level. The message pervading the DAFFODIL approach is that a single alphabetic template (*i.e.,* a set of geometrical backbones for all the letters), some nice flowery shapes, and a set of substitution rules are all there is to letter design. Very little is said about deep stylistic manipulation and transformation at the conceptual level. This sort of thing *is* possible within the system, since a human DAFFODIL user can tamper with the backbones, but on its own, the program does no work at this level.

DAFFODIL's approach to letter design is diametrically opposed to ours. In Letter Spirit, the program, not a human, will be in control of the design process. The program will decide whether a letterform it is working on is acceptable as an 'a'. As sketched in Chapter 1, there will not be 26 explicitly defined skeletal backbones; instead, Letter Spirit will work with conceptual abstractions of letters, made up of roles able to adjust and respond to stylistic pressures.

2.3 Debra Adams' abcdefg

In her master's thesis, Debra Adams compiled a thorough survey of software tools developed for font design [Adams, 1986]. Her survey provides the context for the presentation of preliminary research on an *automatic* letter-generation program called **abcdefg** [3]. Adams' in-depth survey includes reviews of both commercial and experimental programs intended to aid designers in typeface creation. All of the programs discussed have one essential "module" in common — the human user. That is, just as in DAFFODIL and Metafont, all aspects of perception and design are delegated to the human user of the system. In the end, each piece of software is essentially

[3] **abcdefg** is pronounced "ab-kuh-def-gee", as all three-year-old readers will know from watching Sesame Street.

just a fancy and powerful time-saving design tool. Of course, none of the models that Adams discusses (**abcdefg** aside) are touted as anything more than design tools built to make a human designer's task easier. Systems discussed include ELF, IMP [Carter, 1984], and Paragon [Ruggles, 1983], with special emphasis placed on Metafont (see Section 2.1).

In contrast to the standard software-tool approach, Adams introduces the idea of having a program automate some aspects of letterform *creation*. This automation is to be embedded into a graphical design system much like the programs just listed — a sophisticated drawing program with built-in letter-drawing tools such as click-and-drag splines, resizing mechanisms, and so on. The **abcdefg** program works according to the hypothesis that at least some aspects of all lowercase roman letters can be automatically derived from a set of forms (or parts) taken from four control characters ('o', 'h', 'p', and 'v'). The catch is that for every font, these four control characters *must* be designed by a human. Furthermore, the system produces at best "preliminary designs of the remaining characters" based on a straightforward, mechanical recycling of parts from the control characters [Adams, 1986, page 6]. (In fact, the entire derivation process is only implemented in a preliminary fashion for a handful of letters, as we shall see.)

The relationship of the control characters to partially-derived characters is shown in Figure 2.2. Regarding the figure, Adams states:

> The 'o' contains the primary round letter characteristics, the 'h' the primary square letter characteristics, the 'p' the round and square letter characteristics combined, and the 'v' the diagonal letter information. Attributes and values extrapolated from the control characters are used to constrain the design of other letterforms. *[Adams, 1986, page 59]*

Obviously, many of the attributes of non-control letters can be found nowhere in the

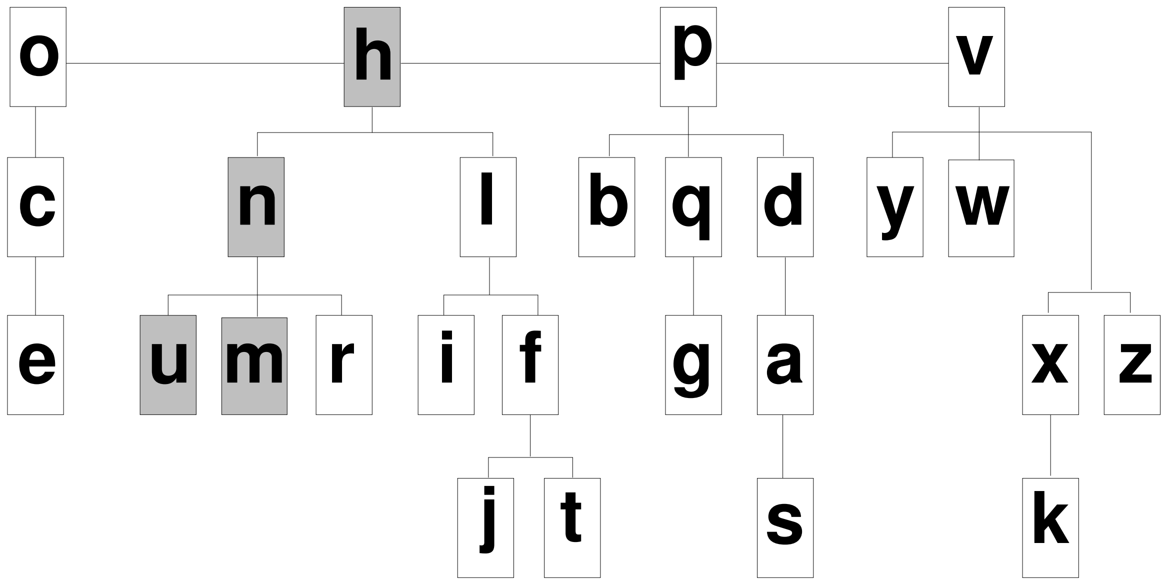


Figure 2.2: This tree shows how derived characters are related to the control characters in **abcdefg**. Distance from the control characters represents the degree of similarity in shape or structure, as well as the complexity of the transformation procedures involved in derivation. The only transformations actually implemented in **abcdefg** are those from 'h' to 'n', 'u', and 'm' (shaded in gray).

control characters (*e.g.,* the slope and joining characteristics of 'k', or curve gradations on the right side of 'c' that are underspecified by the 'o', to name just two). These additional features must eventually be supplied by the human user, as the program has no basis on which to create them. This is because the program makes no use of the analogical relationships between letter attributes. In contrast, Letter Spirit's generational capabilities will be entirely based on analogical connections between letters.

The preliminary version of **abcdefg** was designed and tested using the *sans serif* typeface Bell Centennial (by Matthew Carter), and was able to automatically generate the letters 'n', 'u', and 'm', given the 'h'. Note that these transformations work only in one direction. That is, 'h' can give rise to 'n', but not vice versa. No perception is involved in the system at all. The program blindly follows a set of transformation rules, producing an unseen, untested product at the end of a chain of transformations.

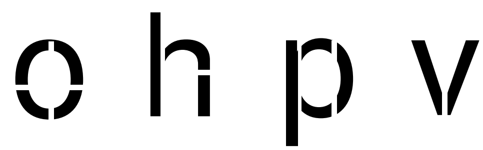


Figure 2.3: The four control characters of **abcdefg** between them have thirteen parts. These parts are recycled for use in derived characters, sometimes taken "as is" and sometimes slightly altered.

As in DAFFODIL, a human must ensure the quality of any resulting letterforms.

Adams' attempt at automation of typeface design was a brave one, in that she used full-fledged (though very plain) letterforms in her system. This opened an industrial-sized can of worms, making the already very hard problem of typeface design even more complicated, and causing too much of the program's focus to be diverted away from conceptual issues and toward what we consider tangential issues, such as how to control exact curvature and line thickness. Letters in **abcdefg** were represented as collections of parts (see Figure 2.3) that were in turn stored as discrete point coordinates connected with cubic spline curves. Points and curves together define a shape outline for each part. Automating design boiled down to reusing sometimes-transformed (but often unrevised) parts of one letter as parts of another. As was pointed out above, the parts made available to the system at the beginning of a run (the parts of the control characters) constrain only partially the design of most remaining letters. Only three of the transformations suggested in the chart shown in Figure 2.2 were implemented, and it must be said that implementation of the remaining transformations seems less than straightforward. Coding up the 'h'→'n', 'h'→'u', and 'h'→'m' rules and testing them on a single typeface is a start, but only a small one. This preliminary work does not test the "letter design by rule-based part recycling" idea thoroughly enough to prove that it is fundamentally correct for even one typeface, much less multiple typefaces.

Even given transformation rules that work for one typeface (something that may well be impossible in the first place), it seems extremely unlikely that any of the resulting transformation functions will be directly applicable to other typefaces. (A clear discussion arguing for the inherent impossibility of parameterizing or "knobbifying" style in this way can be found in [Hofstadter, 1985, Chapter 13].) This renders the idea of automatic design via Adams-style transformation suspect. It is clear that vast amounts of research remain to be done before the control-letter/part-transformation hypothesis is borne out.

The crux of the problem lies in the attempt to encode part transformations as completely specified rules. Creating a rule that is specific enough for use with a particular typeface like Bell Centenial and at the same time general enough to apply to more than one typeface is a tricky business since style often reaches deep into letterforms. The same sorts of issues often arise in the expert systems approach to AI. The answer to this dilemma is *analogy*. Instead of transforming one instantiation of a part into another in a brittle, mechanical, algorithmic way, the design of one part should influence the design of other *analogous* parts. The idea is to do the "same thing" for the new part as was done when designing the old one. Discovering just what this "thing" is involves perception. That is, capturing the designed-in, essential aspect of a part — its gist — necessarily involves *perception*. It naturally follows that a model of perception must be in place before any design work can begin in earnest. This characterizes our approach to Letter Spirit, and explains why this thesis is centrally concerned with a letter-perception system.

Of course we agree with Adams that the idea of automating typeface design is an intruiging and very challenging puzzle. We simply disagree with the methods proposed and partially implemented in **abcdefg**. Nevertheless, throughout her work, Adams introduces many of the critical issues involved, providing interesting background information on the challenge of automating letter design.

2.4 GridFont: A connectionist approach to the Letter Spirit challenge

We now turn to a program created deliberately as a rival to our own approach. Grebert, Stork, and colleagues, inspired by the domain and the creation task of Letter Spirit, decided to try a purely connectionist approach and see how well it could do [Grebert et al., 1991a].[4] The underlying motivation of their work seems to be to demonstrate that connectionist systems are good at generalizing from examples not only for the purposes of *recognition* (*e.g.,* after having been exposed to many pieces of cursive handwriting by presidents, being able to recognize that a particular set of marks is an instance of the word "seven" and moreover that it was penned by Abraham Lincoln) but also for the purposes of *production*. To illustrate what they mean by this, and to suggest the scope of this type of task, they give the following examples of "generalizing for production", in addition to the Letter Spirit challenge itself:

- painting a portrait of George Bush in the style of Vincent van Gogh;
- singing "I wanna hold your hand" in the style of Elvis Presley;
- shopping for computer software in the style of Imelda Marcos;
- playing Monopoly in the style of Mother Teresa.

The model designed by Grebert *et al.* is called "GridFont", and is a three-layer feedforward network that learns by backpropagation. Input to the network consists of a localist representation of both the letter and the font (this amounts to 32 units — 26 for the letter code and 6 for the font code). The hidden layer has 88 hidden units and

[4] See also [Grebert et al., 1991b], [Grebert et al., 1991c], and [Grebert et al., 1992].

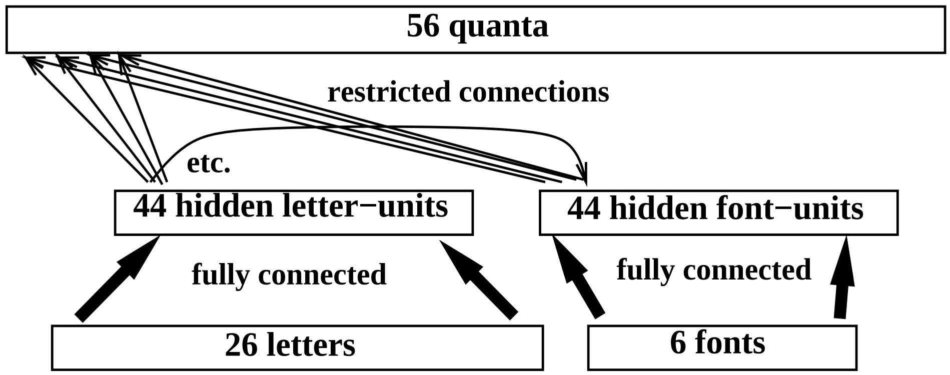


Figure 2.4: The GridFont network architecture. Each of the two input layer sections (representing "letter" and "font" in a localist way, respectively) is fully connected only to its associated hidden layer. Output from the "letter" hidden units is fed sideways into the "font" hidden units. The projections from the hidden layers to the output layer are restricted.

is divided into two modules of 44 units each. One of the hidden-layer modules receives input from the letter nodes and the other from the font nodes. Activation from the "letter" hidden units feeds sideways into the "font" hidden units (making the network in some senses a four-layer network). Both sets of hidden units are specially connected to the output layer, which contains 56 units — one for each quantum of the Letter Spirit grid. The output units are grouped together in sections that correspond to the zones of the grid (see Figure 1.6 for a picture of the grid), and connections between the hidden layer and the output layer are restricted according to these zone-based sections. Figure 2.4 shows the network architecture.

Connections between the hidden layer and the output layer are restricted for three reasons. (1) Restriction reduces the total number of connections, thereby decreasing simulation time and improving generalization. (2) Meaningless correlations between strokes in widely separated zones are prevented. Associations between quanta in the descender zone with quanta in the ascender zone are probably unnecessary and can be removed from the network. (3) Dividing resources between zones allows greater computational resources (*i.e.*, more weights) to be given to the central zone, which is the most important zone. Restricting the connectivity helps the network considerably in its creation task, since a number of the possible mappings have been eliminated *a priori*.

Aside from its carefully-designed connectivity, the GridFont network came with zero built-in knowledge about letters of the alphabet. It was therefore trained on letters belonging to six different gridfonts, all of them human-designed. (All of the complete training fonts used in the GridFont network research are presented together in Appendix A as Figure 9.1.) In the case of five of these styles, the network saw all 26 letters, but in the sixth style, called *Hunt Four*, it was exposed to only fourteen different letters (shown in Figure 2.5).

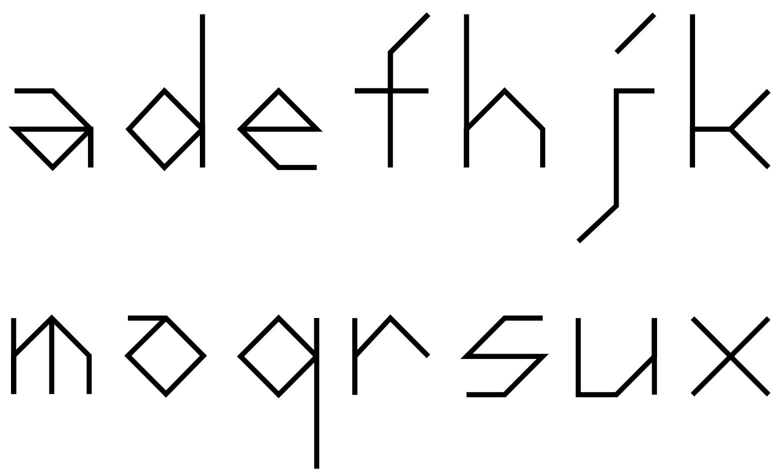


Figure 2.5: Fourteen letters fed to the GridFont network in order to suggest the human-designed style called *Hunt Four*.

To train it on, for example, *Benzene Right* 'y', exactly one of the 26 "letter nodes" (namely, the 25th), as well as a particular one of the six "style nodes", was activated on its input layer; then backpropagation was used to reinforce the activation, on the output layer, of those nine nodes that represent the nine quanta that actually make up *Benzene Right* 'y'. Some $10,000$ trials of this sort were carried out, at the end of which the GridFont network was able, on request, to reproduce quite reliably on its output layer all 26 letters of the five full gridfonts it had been trained on, and also the fourteen letters it had seen of *Hunt Four*.

Now, having been trained on five full gridfonts and one partial one, the network's task was, of course, to extrapolate the *Hunt Four* spirit to the remaining twelve letter categories. We suggest that readers consider this a challenge for themselves as well.

Actually, referring to this as "the network's task" is misleading, because the word "task" implies that some *further* processing was needed at this stage (as would indeed be the case if a human being — such as yourself — were in an analogous position). But further processing was not needed. As soon as the GridFont network had been trained on the fourteen letters of *Hunt Four*, its opinions about the remaining twelve were all implicitly present in the inter-node connection strengths, with no need for further calculation or computation. The answers could simply be read right out, one by one, by activating the twelve remaining input-letter nodes, along with the one *Hunt Four* input-style node, and reading the quanta off the output layer. Those answers are shown in Figure 2.6, which shows also, for purposes of comparison, the actual choices for the same letters made by the human designer of *Hunt Four*. (We remind readers that these particular stylistic-extrapolation choices are by no means absolute and incontrovertible; they are simply the choices that were made at a particular point in time by a relatively sophisticated and experienced human designer.)

Each reader will of course form their own personal impression of these results, which is fine, but we would like to supplement that with our own commentary. Firstly, we comment on the "vertical" dimension — that of *letter*. Most of the shapes produced by the network are recognizable as members of their intended categories, although all but the 'i', 'l', 't', and 'w' are fairly mediocre ones, in the sense of having strange extra lines that bring to mind, at least weakly, unintended categories. Aside from the 'z', which is of course a complete failure, the 'c', the 'n', and the 'v' are the shakiest, since they activate, respectively, the unintended categories 'e', 'm', and 'w', at least a

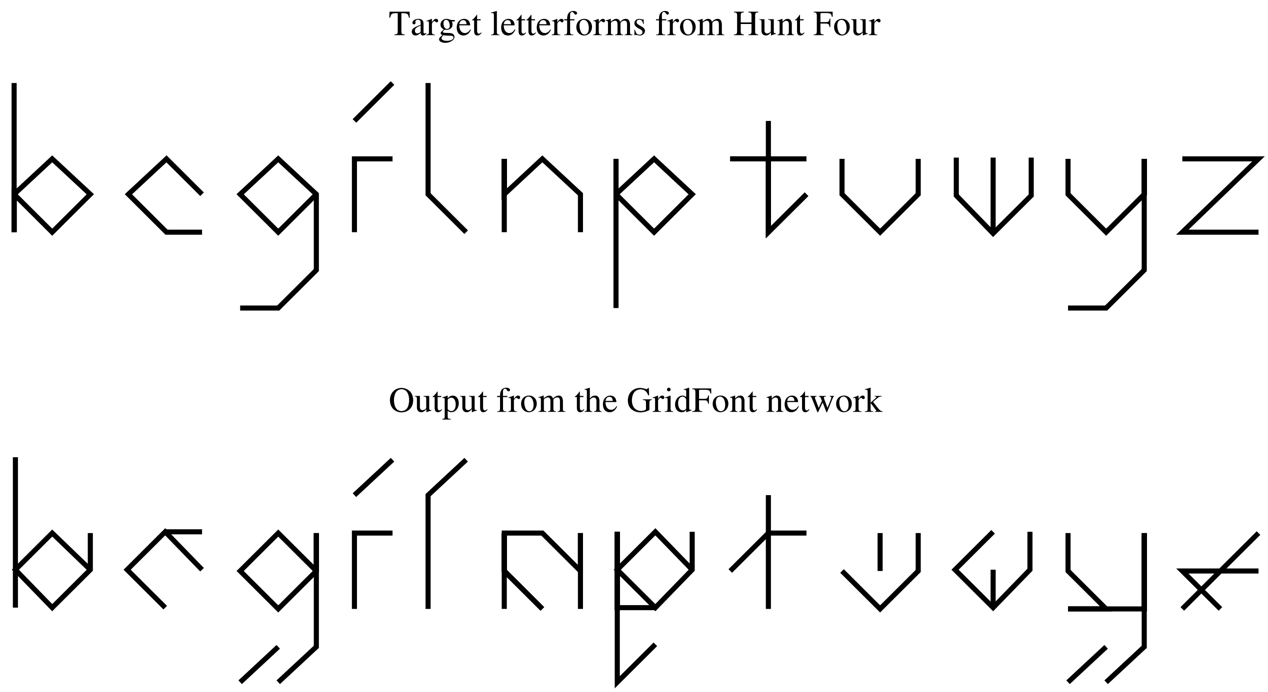


Figure 2.6: The remaining twelve letters of *Hunt Four*, above, as designed by the human inventor of the above-shown fourteen letters, and below, as produced by Grebert *et al.*'s GridFont network.

bit.

One of the many weird things the GridFont network does is to put an extra quantum "jog" on its 'b'. This is something it is hard to imagine a human designer ever doing (especially in the stylistic context of *Hunt Four*). The network, however, lacking a realistic concept of 'b'-ness, can do nothing but consult the chart of associations it has coded into its weights. As it turns out, most of the 'b's that the network was trained on happen to have in common the vertical quantum in the upper-right part of the central zone. So the GridFont network readily adds what in this case turns out to be an "extra" quantum to its version of the *Hunt Four* 'b'. Since the network is doing nothing more than low-level pattern association, it has no *conceptual* foundation for design. It simply creates letters based on statistical similarities that it has "generalized out" of its training set. If concepts were merely statistically defined, this might work, but letter concepts are in fact much more complex than this.

We now turn to the orthogonal dimension — that of *spirit*. There are some recognizable stylistic elements, particularly the diamond motif in the central zone, that seem to give the results a reasonable degree of credibility. Undoubtedly, the network's biggest triumph along the spirit dimension was its precise reproduction of the human designer's solution for the 'i'. This at first seems quite stunning, but when one looks back at the five fonts the network was trained on and sees that in every single one of them, the 'j' completely and perfectly subsumes the 'i', the result is considerably less impressive. One also notices that the central diamond motif was present in full form in three of the input letters, which likewise reduces the achievement somewhat.

What these examples reveal is that if two or more letter categories resemble each other in exactly the same way in one training font after another, then the statistically-generated distributed representations in the hidden layer will take advantage of that fact and encode them in an overlapping manner. Thus, as we said, across the five input fonts, there was a completely regular, standard 'i'↔'j' relationship, and because it was so reliable, it became embedded as an implicit rule, which was of course then respected in the test font. The same kind of absolutely reliable repetition held true, in all five input fonts, for the bowl shape as a component in all six of the letters 'b', 'd', 'p', 'q', 'g', and 'o'. In that respect, the very carefully chosen input fonts were at one and the same time extremely *useful* (in that they supplied the program with stereotyped inter-letter connection information) and extremely *misleading* (in that such stereotyped information is of use only in the most stereotyped of situations).

A human designer of course exploits this kind of standard, highly formulaic inter-letter relationship at times, but that has little if anything to do with what creative design is really about, which is far above the level of rigid shapes and absolutely predictable, clichéd overlaps. It is about unpredictable, highly abstract, one-of-a-kind connections that are made spontaneously on the basis of subtle analogies evoked by unique constellations of context-dependent pressures. This kind of thing is orders of magnitude away from what GridFont was doing. Distributed representations in the hidden layers of connectionist systems can do some pretty clever things, but they are not *that* clever.

The GridFont network results strengthen our claim that a conceptual basis is required for esthetic gridfont design. Although the designers of GridFont might argue that their network has a conceptual basis (in the sense that it has a set of associations that roughly correspond to 'a'-ness), the problem is that the network has no *explicit* concepts that it can access in order to manipulate them. GridFont has no awareness of the internal structures and boundaries of its "concepts". In order to create new 'a's and 'e's, one needs to have accessible, explicit knowledge of *what it means* to be 'a's and 'e's, what 'a's and 'e's have in common with other letters, and what sets them apart. At the very least, some sort of perceptual filter should be applied after each candidate letter is created, so that too-bizarre letters get thrown away or get cleaned up.

No matter how long one stares at the twelve letters generated by GridFont, one does not feel that they truly share a style — especially when one combines them with the original fourteen, to make a full alphabet (see Figure 2.7). There is something awkward, confused, and discomfiting about the network-produced letters that simply does not go away with time. (Perhaps it is at this very abstract meta-level that the twelve new letters *do* constitute a style, after all!)

2.4.1 The inescapable temporality of the act of creation

Let us leave aside the evaluation of the GridFont network's output, since that is, of course, a somewhat subjective matter, and instead turn to the question of what the point of the work by the Grebert group might be.

First of all, recall that the idea of Letter Spirit is to take just a *single* letter, or

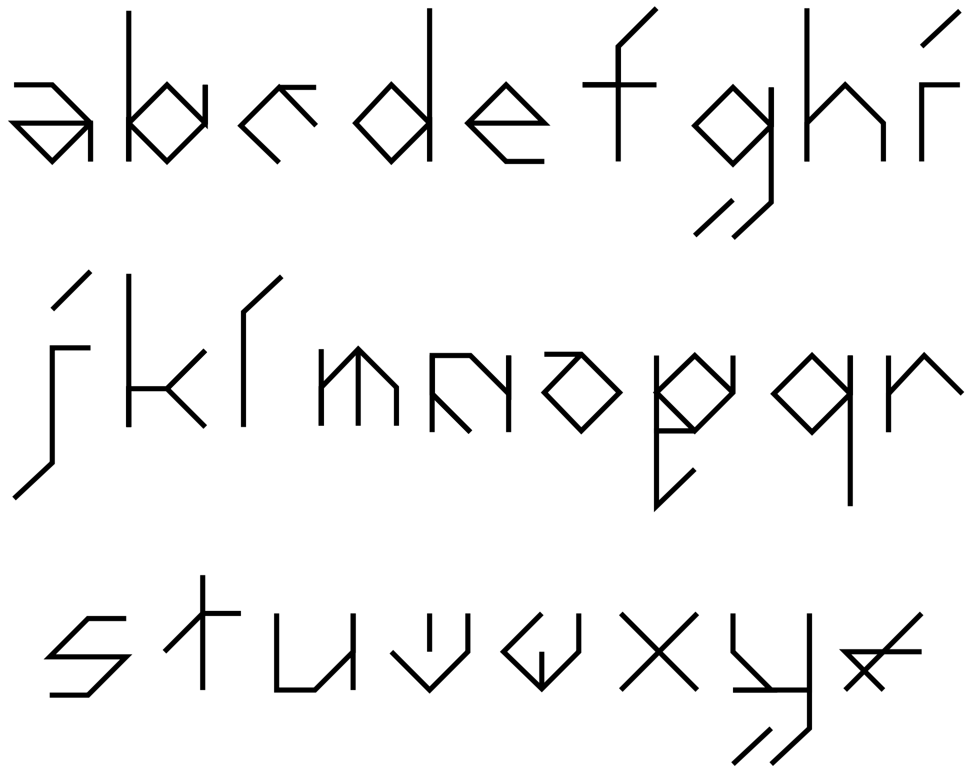


Figure 2.7: The twelve "test" letters of *Hunt Four*, as produced by Grebert *et al.*'s GridFont network, combined with the fourteen human-designed training letters. Together they create a rather disturbing gridfont that might be called *Hunt and Peck*.

perhaps two or three, but certainly not half an alphabet, and to extrapolate it into a full style. When most of the solution is handed to the program for free, what is its performance supposed to show? GridFont was trained on so many letters from its target gridfont (specifically, 'a', 'd', 'e', 'f', 'h', 'j', 'k', 'm', 'o', 'q', 'r', 's', 'u', and 'x') that most of the key stylistic features had already been designed by a person.[5] For example, the network had seen 'q', which has a right descender; 'f', to present the crossbar; and so on. Certainly GridFont does not carry out the style-creation challenge that we set as a goal for our own program. But even this is a minor quibble

[5]In fact, Grebert *et al.* selected their training set specifically in order to have this characteristic [Grebert et al., 1992, page 704].

compared with our main criticism.

The real problem we see with this project is that it completely bypasses the cognitive issues that we are concerned with: what it is that happens in the mind when human beings engage themselves in tasks that require concentration over an extended period of time, tasks that require going back and looking at what one has done, judging it, adjusting it, abstracting from it, and so forth — in short, the wealth of cognitive and subcognitive activities that were subsumed under our term "central feedback loop of creativity" (see Section 1.10). In genuine human creation (and in our emerging Letter Spirit architecture), any design decision made during the creation of one letter has some chance of influencing the design of all the other letters in the gridfont — even supposedly "finished" ones. Indeed, one single tiny decision, made just when one thought one was finally approaching the end of the process, can trigger a cascade of related decisions throughout the alphabet and thus lead to an avalanche, forcing total reconsideration of every decision made, and the possible reconstruction of every single letter. This inherent unpredictability and instability of what one is doing is precisely the excitement and magic of the genuine creative act. It is closely related to the idea we describe in Chapter 3 of a system's *making its own decisions.*

In stark contrast to this dynamic, vibrant image of mental activity, the GridFont network's letters are all pushed out in one single feedforward gush, without any process that maps onto *working it out.* (Technically speaking, the various letters are created one at a time, since only one letter-node can be clamped during a cycle, but the act of outputting one letterform has no subsequent effect on the outputting of any others. On a conceptual level, a full font is produced all at once, without any perception of the results, let alone back-and-forth propagating of the effects of such perceptions.) Even supposing that the results of GridFont had been superb, it would still be the case that *this kind of thing is not even remotely close to what goes on in actual human minds.*

The philosophy behind GridFont reminds us of someone trying to compose a piece of music by producing all of its different measures at once without any specific regard for how they fit together. Even if all the measures are very competent on their own, even if they are all in the style of a given composer, something essential is missing. For a piece of music to be good, there simply has to be a great deal of back-and-forth activity during the compositional process, serving to weave all of its parts tightly together. The same could be said of a poem, a painting, or an alphabetic style.

Suppose GridFont, or some four- or five-layer improvement thereupon, were some day to come up with one impeccable gridfont after another, triggered each time by just one seed letter or two (a most unlikely scenario, but never mind). This would be a bit like a clever chess program running on superfast custom chess hardware and overwhelming the human world champion, but doing so via pure brute-force lookahead, a billion board positions a second — the antithesis of how chess masters play. In a sense, such a triumph of computational chess-playing (and it may well come in the next few years) would tell us that, sadly, the domain of chess is not quite as interesting or as challenging as we had once presumed it was — but certainly it would not in the slightest deny the interest of the mental processes that go on in the minds of international grandmasters! And a comparable triumph by GridFont or its progeny would perhaps reduce our interest in the microworld of gridletters, but certainly not our curiosity about what goes on in the mind of a skilled human gridfont designer! In any case, such a success by GridFont itself or by Great-Great-GridFont is sheer fantasy at this point.

2.4.2 Genuine cognition is not a free lunch

The hubris of the authors of GridFont is surprising to us. Consider the paragraph with which they conclude [Grebert et al., 1992]:

> The approach discussed here could be applicable to a range of generalization for production problems. For instance, one might eventually be able to form a network to generate speech in the style of a particular talker, and hence have a network speak George Bush's inaugural address in the style of John F. Kennedy. Of course the training corpus and the hidden unit representations would be far more extensive than those described above. Nevertheless, our success here gives us confidence that the architectural considerations discussed above — most notably the form and connection of hidden layers — will be applicable in other domains.

Leaving aside the question of whether the phrase "our success here" is warranted, we find their extrapolation from the GridFont network to such complex intangibles as John Kennedy's oratorical style or Vincent van Gogh's painting style simply fantastic. This attitude conjures up the image of a skier who, after stumbling all the way down the 100-meter descent at the so-called "Nashville Alps" in Indiana, gets up, brushes all the snow off their outfit, and in a serious, un-self-conscious tone proclaims, "Hey, man, I'm off to Aspen to tackle those black-diamond slopes — and next year when you see me take the gold in the Olympic giant slalom, you can say you once rode the rope tow with me way back in Indiana!"

The reason we do not feel that the very same caricature argument boomerangs back on us, making us question the value of our own Letter Spirit project, is this: in developing our architecture, we have striven hard to be faithful to what we believe to be the underlying principles of creativity, rather than relying on the naïve blanket faith that the hidden layers of a feedforward backprop connectionist network, when suitably trained, can do anything and everything cognitive. Such a belief is simply an excuse for not thinking about cognition itself and trying to get so-called *learning* (actually, just the most boring type of learning — learning by endless repetition) to do everything for you: it is a belief in the notion of cognition as a free lunch, courtesy of distributed representations. This is the connectionist analogue of the Boolean Dream [Hofstadter, 1985, Chapter 26], and is equally implausible.

We believe, contrarily, 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?[6]

The Letter Spirit domain — at least when construed in the way that we intended it — forces one to pay attention to the aspects of concepts that count: their blurry boundaries and elusive essences, their multiple levels of abstraction, and most of all, their strange and unpredictable connections that suddenly jump to mind as a function of some unique, unanticipated set of pressures, and that thereby open up whole new pathways of unsuspected potential in the space of ideas. It is our firm belief that only an architecture explicitly focused on understanding and modeling these types of mental, not neural, phenomena will wind up shedding light on them.

[6] More thoughts on current attempts to get some such intermediate structure into connectionist networks is included in Chapter 6.

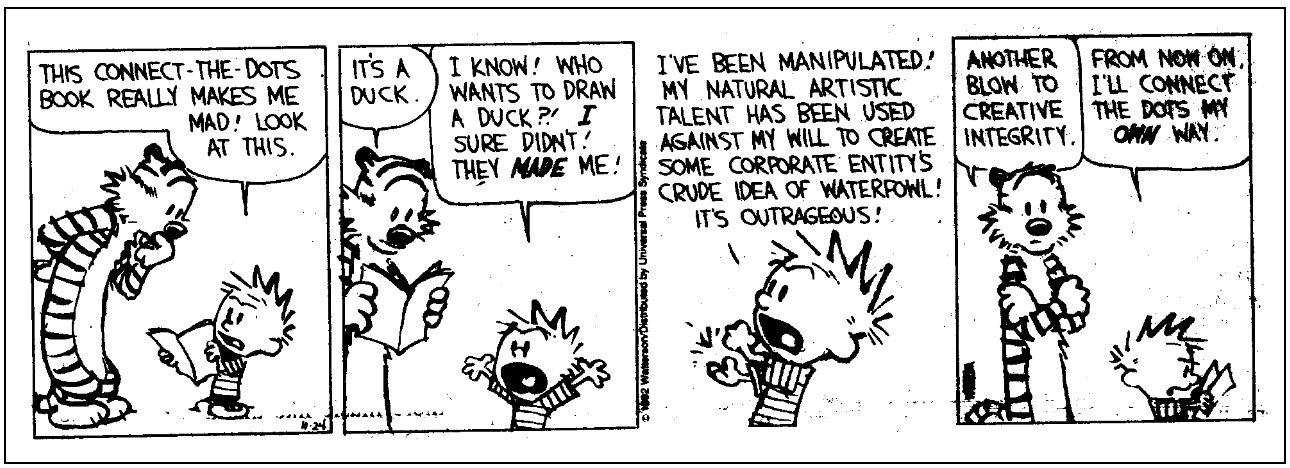


Figure 2.8: Deep statements on creativity by Calvin.

CHAPTER THREE

Creativity and Artificial Intelligence

The Letter Spirit project's fundamental aim is to model, and thereby give rise to a deeper understanding of, the creative process. As we have shown, our approach to creativity involves a complex intertwining of analogical thinking and high-level perception in an internal feedback loop of imagining, drafting, examining, and adjusting. In order to clarify the novelty of our approach, we must first consider other models of creativity that have been proposed and developed during the brief history of artificial-intelligence modeling. Thus the time has come to plunge headlong into the briarpatch of creativity and AI.

The central idea of this chapter can be summarized in a pithy one-liner:

High-level perception is the key to creativity.

It surprises us that AI modelers have paid so little attention to the act of internal perception and judgment that goes on in large-scale creative activities such as creative writing, music composition, or large-scale design. Human designers move from an overall idea for something to a finished product through a long series of *assessments and adjustments*. For convenience, we will call this the "A&A loop". It is precisely the existence and pervasiveness of the A&A loop that leads to the aphorism: "Creativity is 2% inspiration and 98% perspiration." Although the numbers may not be right on the money, there is a certain ring of truth to the statement.

Both of the 'A's of A&A — assessment and adjustment — are grounded in perception. It is simply not possible to create a high-quality, truly novel, large-scale product (like a gridfont, a mathematical theorem, a drawing, a poem, an architectural blueprint, a computer program, a piece of music, a joke, a movie, a scientific experiment, a better mousetrap, etc.) without being able to look at it afresh and understand it anew throughout the process of design. Assessment and re-assessment are critical and unavoidable steps. It is even more impossible to make useful adjustments of a product without first having assessed its merit (lucky hits resulting from random "adjusting" aside). For just this reason, many existing models of creativity in AI rely on humans to do their perceiving and assessing for them, as we shall see later in this chapter. While this may be a clever way to sidestep the very tricky and complicated problem of properly modeling perception, it casts the true creative abilities of some of these programs into doubt. One wonders what such programs would be capable of if their "human modules" were taken away.

We argue that all models of creativity must necessarily *know something about what they are doing.* Such understanding has two distinct and equally important facets. First, each creative program should have vast knowledge about the *domain* in which it is working. Second, each program should have internal representations of the *actions* that it is itself taking, and awareness of the *products* that it is creating. Each of the letter-design models discussed in the previous chapter is lacking in these two fundamental areas. Of course, some of them — like Metafont — were never intended as models of creative design, though others — like **abcdefg** — were.

3.1 True creativity implies autonomy

We propose that for a design program to be called "creative", it should meet the following requirements:

- the program itself must arguably *make its own decisions* rather than simply carrying out a set of design decisions all of which have already been made, directly or indirectly, by a human;
- the program's knowledge must be rich — that is, each concept must on its own be a nontrivial representation of some category with flexible criteria for judging degrees of membership, and among diverse concepts there must be multiple explicit connections;
- the program's concepts and their interrelations must not be static, but must be flexible and context-dependent;
- the program must experiment and explore at a deep conceptual level rather than at a shallow surface level;
- the program must be able to perceive and judge its own tentative output and be able to accept it, reject it, or come up with plausible ideas for improving it;
- the program must gradually converge on a satisfactory solution through a continual process in which suggestions coming from one part of the system and judgments coming from another part are continually interleaved.

We would in fact argue that the deep (and philosophically controversial) question raised implicitly by the first point — "What would it mean for a program to make its own decisions?" — is answered by the last five points taken together.

The Letter Spirit architecture is an attempt to model all these crucial aspects of creativity, albeit in a rudimentary way. As was stated in Chapter 1, several of its features — nondeterminism, parallelism, and especially their consequence, statistical emergence — are key elements in allowing it to achieve these goals.

3.2 Other models of creativity

AI's historical record is speckled with attempts to build creative programs in all sorts of domains ranging from the humble to the grandiose. It should come as little surprise that these programs differ widely not only in *what* they attempt to do but also in *how* they go about trying to do it. We now survey a wide range of computer programs designed to exhibit creative behavior, trying to get across a general flavor of the many varied approaches that have been tried.

It is impossible to touch on every last one of the many attempts at automating creativity in this thesis. Instead we take the easy way out and point the reader elsewhere. A useful survey of many recent creativity projects can be found in Margaret Boden's book *The Creative Mind: Myths and Mechanisms* [Boden, 1990]. Another insightful discussion of artificial creativity, this time focusing on emergent computation, is provided in a survey article by Jon Rowe and Derek Partridge [Rowe and Partridge, 1991b]. Closer to home (ours anyway) are several chapters of Hofstadter's *Metamagical Themas* (Chapters 10, 12, and 13, to be precise) [Hofstadter, 1985]. Last, but not least, the reader is referred to the recent book *Artificial Intelligence and Creativity*, an intriguing collection of papers edited by Terry Dartnall [Dartnall, 1994].

The nine projects we discuss below range from ancient (twenty years old is fairly ancient in AI terms) to state-of-the-art. Where space (and suitable output) permits, we will begin each discussion of a program with an actual example of its craft. We have no reservations about presenting a mixed bag of "discovery" programs and "artistic" programs, since all make claims about their relationship to creativity.

3.2.1 Racter: A computer raconteur

> "War," chanted Benton, "war strangely is happiness to Diane." He was expectant but he speedily started to cry again. "Assault also is her happiness." Coldly they began to enrage and revile each other during the time that they hungrily swallowed their chicken. Suddenly Lisa sang of her desire for Diane. She crooned quickly. Her singing was inciting to Benton. He wished to assassinate her yet he sang, "Lisa, chant your valuable and interesting awareness." Lisa speedily replied. She desired possessing her own consciousness. "Benton," she spoke, "you cry that war and assault are a joy to Diane, but your consciousness is a tragedy as is your infatuation. My spirit cleverly recognizes the critical dreams of Benton. That is my pleasure." Benton saw Lisa, then began to revile her. He yodeled that Lisa possessed an infatuation for Diane, that her spirit was nervous, that she could thoughtfully murder her and she would determinedly know nothing. Lisa briskly spoke that Benton possessed a contract, an affair, and a story of that affair would give happiness to Diane. They chanted sloppily for months. At all events I quickly will stop chanting now.

Racter, the program that wrote the above selection, was itself written by William Chamberlain and Thomas Etter [RACTER et al., 1984]. Racter exploits the raw power of generative grammars, as it is a program comprised almost entirely of a computerized complex grammar acting in concert with a repertoire of semantically-tagged words. Racter constructs phrases, sentences, poems, and stories by metaphorically throwing dice and probabilistically following an augmented-transition-network-like structure through a grammar, picking words out of its semantically organized lexicon as it goes. This method for generating grammatical sentences is enhanced by an ability to adapt and re-use previously uttered phrases, thus providing a (deceptive) sense of continuity to the prose.

The entire process of story creation by Racter is mindless. There is no conceptual imagery behind a story, no intent to say anything, and no reflective word choice. Racter simply churns out perfectly-formed syntactically-complex sentences made from a very interesting and appealing collection of words and longer phrases, sometimes repeating itself subtly as it goes along. So why are we so easily taken in by this approach? Or if not by Racter in particular (since it was honestly billed as a droll parlor game), other programs very much like in in spirit? Simple. The Eliza effect.

People have a susceptibility for reading far more understanding into a program's symbolic output (especially if that output consists of words) than a program is actually capable of modeling. The Eliza effect is so powerful that even though numerous articles have been written warning of its perils (see, for example [Weizenbaum, 1976; Boden, 1977; McDermott, 1981][1]), AI researchers still walk straight into its trap seemingly with their eyes wide open. Thus the Eliza effect poses an ominous danger for AI, causing many otherwise properly skeptical and wary intellectuals to let down their guard and be duped by non-intelligent but nonetheless clever hacks. Perhaps there should be a Surgeon General's warning on every AI program that reads: *"Warning: The Eliza effect can be dangerous to your comprehension of cognitive modeling. Please examine all symbols carefully before use."*

There is no doubt that Racter produces something new each time it runs. Obviously, then, there is much more to creativity than simply producing something *new*. Racter's prose is disturbingly superficial — something that becomes apparent only after its initial shock value has worn off. Any interest that a passage by Racter *does* provoke is due to the incredible flexibility and fluidity of each human reader's mind. People are astoundingly good at constructing a conceptual justification (seemingly out of thin air) for vaguely related strings of words. This propensity towards reading meaning out of hints that meaning *might* be lurking somewhere is subtly encouraged by Racter's habit of slightly repeating itself.

Regrettably, it is unclear exactly how Racter produced its book *The Policeman's Beard is Half-Constructed* [RACTER et al., 1984]. Too little has been said of Racter's mechanisms: where its sometimes surprising, low-frequency words come from; how

[1] For a more recent treatment, see [Hofstadter and FARG, 1995, especially page 157].

some of its more interesting phrases were produced (*e.g.*, Racter's seemingly self-referential phrase, "I am silicon and epoxy energy enlightened by line current"); how much weaker material was waded though by Chamberlain and Etter when selecting passages for the book; whether entire paragraphs or works were produced coherently all at once or cut and spliced together by Chamberlain and Etter; and so forth.

What we *can* learn from Racter is that there is much more to creativity than a generative grammar hooked up with a random-number generator. It is not enough to generate seemingly flexible and ambiguous behavior at the surface level by flipping coins and relying on humans to glean meaning that is not really there from empty prose. What is missing in Racter is a coherent set of top-down semantic constraints so that it can create according to a plan or goal, avoid mundanity, and assess its own output. The moral: randomness alone does not creativity make.

3.2.2 TALE-SPIN: Another computational teller of tales

The Fox and the Crow

> Once upon a time, there was a dishonest fox named Henry who lived in a cave, and a vain and trusting crow named Joe who lived in an elm-tree. Joe had gotten a piece of cheese and was holding it in his mouth. One day, Henry walked from his cave, across a meadow to the elm-tree. He saw Joe Crow and the cheese and became hungry. He decided that he might get the cheese if Joe Crow spoke, so he told Joe that he liked his singing very much and wanted to hear him sing. Joe was very pleased with Henry and began to sing. The cheese fell out of his mouth, down to the ground. Henry picked up the cheese and told Joe Crow that he was stupid. Joe was angry, and didn't trust Henry any more. Henry returned to his cave. THE END.

A completely different approach to story telling is taken by Meehan in his program TALE-SPIN — the approach of top-down, goal-directed story telling [Meehan, 1976; 1978].[2] In TALE-SPIN, characters are set up with goals that they try to meet by formulating plans and interacting in their imaginary world. A TALE-SPIN story amounts to a natural-language trace of the subgoal-solving process — something similar in nature to what a story might look like if based entirely on processing done by the GPS approach to goal resolution [Newell and Simon, 1963]. Notice how this approach is diametrically opposite to Racter's. The program is chock full of knowledge. It knows about the characters, how they move about the world, about their goals, about their social relationships, and so on.

This approach yields stories that are often conceptually quite coherent, although in the many "mis-spun" tales that Meehan, to his credit, quoted and analyzed, the program's brittleness and lack of knowledge come to the surface, often producing ludicrous results. Unfortunately, even TALE-SPIN's best efforts are boring. Any flexibility in the stories is sacrificed to the recursive subgoal operation of the planner. So in the case of TALE-SPIN we end up with too much top-down constraint and not enough freedom. Artistic story-telling creativity gets quashed from above. What is needed is both control *and* flexibility.

3.2.3 MINSTREL: A state-of-the-art computer Æsop

The Vengeful Princess

> Once upon a time there was a Lady of the Court named Jennifer. Jennifer loved a knight named Grunfeld. Grunfeld loved Jennifer.
>
> Jennifer wanted revenge on a lady of the court named Darlene because she had the berries which she picked in the woods and Jennifer wanted to have the berries. Jennifer wanted to scare Darlene. Jennifer wanted a dragon to move towards Darlene so that Darlene believed it would eat her. Jennifer wanted to appear to be a dragon so that a dragon would move towards Darlene. Jennifer drank a magic potion. Jennifer transformed

[2] Also see [Schank and Riesbeck, 1981, chapter 2].

> into a dragon. A dragon moved towards Darlene. A dragon was near Darlene.
>
> Grunfeld wanted to impress the king. Grunfeld wanted to move towards the woods so that he could fight a dragon. Grunfeld moved towards the woods. Grunfeld was near the woods. Grunfeld fought a dragon. The dragon died. The dragon was Jennifer. Jennifer wanted to live. Jennifer tried to drink a magic potion but failed. Grunfeld was filled with grief.
>
> Jennifer was buried in the woods. Grunfeld became a hermit.
>
> MORAL: Deception is a weapon difficult to aim.

The latest program in the tradition of story-telling software is Scott Turner's MINSTREL program [Turner, 1992; Turner, 1994]. Coming from the same knowledge-based camp as TALE-SPIN, MINSTREL is imbued with all sorts of knowledge about its domain of damsels, knights, and dragons. MINSTREL makes use of problem-solving, planning, and goal resolution to tell a coherent story.

MINSTREL's approach is case-based. As a case-based reasoning system [Kolodner, 1993], MINSTREL solves problems by recalling similar past problem situations and applying the solutions of the recalled situations to the current problem. MINSTREL discovers new, "creative" solutions to a problem by finding a related problem that it can solve, solving the slightly-different related problem, and adapting that solution to the original problem. This process is implemented as a three-part Transform-Recall-Adapt Method (TRAM) [Turner, 1991].

A weak type of creative recall, deemed "imaginary recall" by Turner, can be implemented with TRAMs by a process of mutating recall features until at least one case is recalled, and then using the information about the mutation(s) to adapt the recalled case. For example, at one point MINSTREL is asked to tell a story about a knight who commits suicide. Being unable to recall any such stories from its memory, MINSTREL applies the TRAMs, *Intention-Switch* and *Similar-Outcomes-Partial-Change*, to its recall parameters: "knight who purposefully kills self". Application of these two TRAMs allows "kills" to become "injures", and "purposely" to become "accidentally". These tweaks allow MINSTREL to recall a story about a knight who accidentally injures himself while killing a troll. Once this story has been recalled, it is subjected to "backwards" adjustments that undo the changes made to the recall parameters. This results in a scenario in which a knight purposefully kills himself by losing a fight with a troll — a newly-invented story-fragment.

Like TALE-SPIN, MINSTREL produces sensible stories that are conceptually coherent. Although they are not quite as boring, there is still something very clunky and mechanical about them. Some flexibility in the story-telling process is gained through the use of TRAMs, and the conceptual flow of the story is certainly smooth, but MINSTREL does not seem to be able to tell the interesting parts of the story ("The dragon was Jennifer.") from the boring parts ("Grunfeld moved towards the woods. Grunfeld was near the woods.") in order to be able to adjust its prose accordingly.

Worse yet, by its very nature, the case-based reasoning methodology severely limits the range of possible creative products, since all new things must be reasonably close to already-known cases so that they can be reached through a series of simple tweaks. Small-scale adaptations and tweaks are not powerful enough (even when chained together) to move interestingly far beyond what is in the system's database in the form of cases to begin with.[3] Once again, too much flexibility is sacrificed to the iron grips of control, although this time significantly less than was sacrificed in TALE-SPIN.

3.2.4 AM/Eurisko: Silicon set-theorists

We now move to discussion of a "discovery" program. Lenat's "Automatic Mathematician", better known as "AM", generates and explores ideas of elementary arithmetic

[3] A criticism acknowledged by Turner [Turner, 1994].

[Lenat, 1982]. Beginning with a set of 100 fundamental and very simple mathematical concepts drawn from set theory (including sets, lists, ordered pairs, and some simple operations including union and intersection), the program uses its 300 heuristics to examine, transform, and combine its original concepts in order to build compound, higher-level concepts as it runs. What gives this project its deep interest is its ability to explore, transform, and extend its own conceptual space in this way. For example, as it runs, AM first builds up the notion of *number*, then addition, then subtraction, and so on.

AM is able to generate the highly abstract concept of an *inverse function* and later use this new concept to define division once it has defined multiplication, or to define square root once it has devised perfect squares. Eventually, AM is able to discover the concept of "prime number" and even use it as a stepping-stone to the formulation of the Goldbach conjecture, which states, "Every even number greater than 2 is the sum of two primes". This is roughly as far as AM is able to go, probably because of the nature of its heuristics, which are applicable to general concepts such as "set" but apply much less powerfully (if at all) to more specialized concepts such as "number". This is not to say that they do not apply *at all.* In fact, AM's ability to get all the way from "number" to $+$, $-$, $\times$, $\div$, to primes, to the Goldbach conjecture is a stunning accomplishment in view of the fact that the Goldbach conjecture was only made by humans 200 years ago. But AM runs out of gas at the Goldbach conjecture and is unable to generate anything further.

AM's processing is guided throughout a run by the numerical values of an "interest" parameter associated with each concept. This allows it to come up with "hunches" and follow them through to possible fruition. However, it is not clear exactly how these critical interest parameters are changed throughout a run. This has opened the door to a fair amount of criticism of AM [Rowe and Partridge, 1991b; Ritchie and Hanna, 1990]. When a user renames an AM-discovered concept, for instance, the concept's interest level is automatically given a jolt. Just as Chamberlain must pore over Racter's output separating the wheat from the chaff, Lenat did so for AM, covertly guiding its search from outside.

Such human–machine collaborations are common in many approaches to AI and creativity, and led Hofstadter to pose the question: "At what point does the seemingly innocent act of *selection* by a human turn into *direction* of the program?" [Hofstadter and FARG, 1995, page 476]. This criticism is directly tied to the idea of a program *knowing what it is doing* and being able to *perceive and assess its own products*, something we are devoting much effort to in Letter Spirit.

AM hits a processing wall shortly after it discovers primes because its built-in heuristics no longer hold any purchase. Perhaps, thought Lenat, AM could *process its own heuristics*, adjusting them along with everything else as its concepts and knowledge expand and evolve. This was attempted by allowing AM to work on its own heuristics during a run. Unfortunately the first attempts to allow AM to work on its own heuristics failed entirely, with AM generating completely useless new heuristics [Lenat, 1983b]. This happened because of the way AM represented its original heuristics in Lisp. As Lenat himself points out, syntactic, mechanical tweaking of the poorly-represented heuristics often yielded new but ridiculous results [Lenat, op cit.]. A new representation language allowing syntactic mutations to give rise to meaningful semantic changes is apparently required.

A closely related program, Eurisko, addressed this idea of heuristics for heuristics. This change was accomplished by adding meta-level heuristics for adjusting the program's lower-level heuristics [Lenat, 1983a]. In addition to set theory and number theory, Eurisko can run in other domains. VLSI design and naval fleet design are two that have been reported on. Once again, though, Eurisko works best when it interacts with a user who can evaluate and filter program output. Such a human user serves as a key perceptual boost to the system. Interestingly, Eurisko was able to get no further than AM did in the original set-theory/number-theory domain, a fact that calls into question exactly how much power the addition of heuristic-changing heuristics (at least of the sort attempted) gives to a system.

3.2.5 BACON, et al.: Data-crunching discoverers

The BACON family of programs uses a different, data-driven approach to discovery [Langley et al., 1987]. Starting with a body of numerical data, these programs attempt to find a hypothesis that fully explains the data. Rediscoveries by BACON include: Boyle's law of idea gases, Kepler's third law of planetary motion, Galileo's law of uniform acceleration, and Ohm's law of electrical resistance. The BACON programs, in the final analysis, amount to good *describers* that are completely reliant on their data being represented in near-optimal form ahead of time. Before any processing is done, the data are compiled, processed, and organized by BACON's programmers, using knowledge of the eventual solution to substantially bias the structure and contents of the data. Fortunately for BACON, such knowledge is available to its programmers, since the "discoveries" it makes have long since been made. A completely fleshed-out criticism based on this line of reasoning can be found in [Chalmers et al., 1992].[4] Basically, BACON plays syntactic number games until it has completely summarized the data. This amounts to an *ad hoc* curve-fitting exercise.

A set of related programs — GLAUBER, STAHL, and DALTON — attempt to build simple models of their input data in order to express results in terms of general laws. But they too suffer from the fact that *the idea of what to look for is completely specified in their heuristics*, making their behavior rigid to the extreme and entirely predictable. As Rowe and Partridge point out, "it seems that when the heuristics are tightened to avoid generation of trivialities (as both AM and Eurisko were prone to do

[4] The article is reprinted in [Hofstadter and FARG, 1995]).

in the long run), all flexibility of representation is lost." [Rowe and Partridge, 1991b, page 12].

3.2.6 A computer jazz-improviser

Another set of computational approaches to creativity work in more "artistic" domains. One such program is the jazz improviser of Philip Johnson-Laird [Johnson-Laird, 1989]. According to its author, himself an amateur jazz musician, Johnson-Laird's program performs at the level of a moderately competent beginner. Its task is to first create a simple yet harmonically sensible chord sequence (defined by a built-in grammar) that is then embellished in certain ways to produce a nested hierarchical melody structure. The embellishment is all made according to a set of rules about musical form devised by Johnson-Laird and Christopher Longuet-Higgins.

The program improvises a performance in real time, sometimes choosing among alternative plausible chords in a progression by flipping a biased coin. Constraints on the system provided by harmony, melody, and tempo make make the task possible.

In improvisation, assessing must occur in real time — *as a piece is being performed* — and must not interrupt the flow of the music. Most improvisers, myself included, develop a set of standard "riffs" making up a kind of bag of tricks that can be applied throughout a live performance. For example, one such riff is: if a mistake is made, do it again so that through repetition, the initial mistake is mitigated and does not appear to have been a mistake at all. Of course, improvisation is very different in nature from large-scale design of the sort we are interested in modeling. Improvisation happens much more spontaneously and seems to involve less assessment and adjustment than other sorts of creative tasks. Nevertheless, improvisation is certainly not devoid of the entire assessment procedure. After all, a real-time variety of assessment is required for the riff mentioned above to work. Without noticing a "mistake" in the first place, how could it be corrected through the use of the riff?

Johnson-Laird's program seems to lack even this sort of perception, however. It relies instead on a "skeleton" (defined by a grammar) which is then embellished by the application of simple rules. His program thus unfortunately seems to have much in common with DAFFODIL.

3.2.7 Aaron: A computer artist

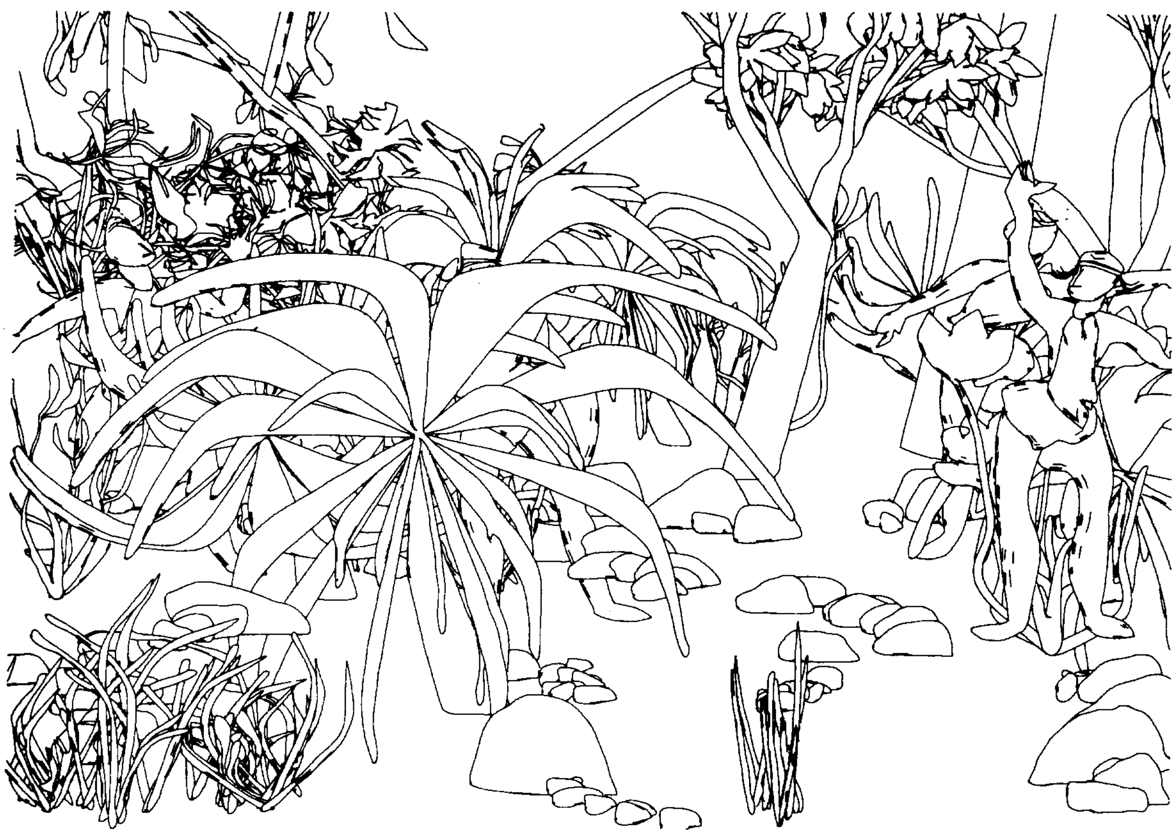


Figure 3.1: One of Aaron's 1987 Eden pictures. Note the couple on the right side of the picture who appear to have startled the person hidden in the undergrowth on the left side. However, there is no such social-level understanding or planning in the program. Instead, the placement of figures, plants, and rocks was arbitrary.

One of the most provocative of all "artistic" programs is Aaron [McCorduck, 1991]. Harold Cohen, an established artist in his own right, created Aaron to draw pleasing and unpredictable pictures. Aaron makes use of a large amount of knowledge about everyday physical laws, balance, and ocular occlusion to draw pictures of humanlike figures doing things like dancing, holding giant balls, and balancing (sometimes in jungle-like settings, as in Figure 3.1). Its finished products have been exhibited at famous galleries around the world.

Unfortunately, no technical articles about the internal workings of Aaron have been published.[5] The best source (cited above) is Pamela McCorduck's 1991 book *Aaron's Code*. It seems that Aaron is an expert system of sorts, with many hard-and-fast rules working together to define and constrain a drawing. Aaron's probabilistic component — which aids in its decision-making about placement, composition, and the like — allows it to make novel drawings each time it is run. On each run, Aaron draws its pictures in its coded-in, easily-recognized style. This style depends entirely on Aaron's current set of rules and can only be changed by a programmer who has access to the rules in coded form. Aaron has no capability of reflecting on its own drawings or its own stylistic tendencies, ergo it cannot by itself change its style. Its expert-system-like rules can be changed only by Cohen himself. Even more disturbing is the fact that Aaron cannot perceive its own work at all in the way that humans can perceive their own work throughout the coming-into-being of a picture. Aaron has no internal perceptual system.

Like Racter, Aaron is another program that hooks deeply into the Eliza effect; however, it is not clear that Aaron's benefiting from the effect is as intentional on Cohen's part as in the case of Racter. Because Aaron's pictures are pictures of humanlike figures, viewers tend to read far more into them than is actually there. This is compounded by a built-in "wiggly line" heuristic that provides a simple surface-level way of making Aaron's drawings seem less "mechanical". (A perfect example of the

[5]There is a WWW cite for the Center for Research in Computing and the Arts (Cohen's lab), at <http://cra-www.ucsd.edu/index.html>.

provocative dangers of shallow style.) After all, using a state-of-the-art Computer Aided Design program (like AutoCAD) to draw polygonal people in various locations is certainly feasible these days. Writing a CAD script to do so would be a reasonably simple project. But what would such a script tell us about human creativity?

3.2.8 GENESIS: A computer card-shark

Rowe and Partridge's GENESIS model explores the possibility of modeling creativity using a multi-agent system with an emergent memory mechanism [Rowe and Partridge, 1991a; Rowe, 1991]. The model is based both on Minsky's *Society of Mind* ideas about distributed control and multi-agent systems [Minsky, 1985] (especially with regard to k-lines as memory-organization tools), and on the ideas of emergent, high-level perception found in the work of Hofstadter [Hofstadter and FARG, 1995]. Central to the philosophy of Rowe's approach is the idea that low-level activity produces emergent, high-level structures that in turn serve to guide further low-level search. This kind of feedback loop in which bottom-up and top-down processes interact is closely based on the approach developed by Hofstadter and his students.

Rowe's GENESIS model is applied to the sequence-prediction card game Eleusis (actually a limited version called "micro-Eleusis"). The program makes use of a perceptual model, so that various patterns that develop in the course of a game can be recognized and properly predicted. Winning the game requires ferreting out the dealer's rule based on an analysis of the sequence of cards played up to a certain point. Although the hidden rule is fixed, the pattern of cards that develops according to this rule may take on various complex guises. Discovery of the rule requires analogical thinking in which theories regarding the pattern are constantly revised and abandoned as play continues. Sometimes acting as dealer, the program must also be able to invent a rule that will dupe some players while being apparent to others. This is because the dealer's score is tied to a function involving minimum and maximum scores of the other players.

Various agent-control strategies were compared and contrasted in the research. Results showed that emergent systems having no central controller exhibit very flexible (and fortunately not very random) behavior. The GENESIS program addresses the need for flexible representation and control in models of creativity. Rowe's work, like Letter Spirit, stresses the importance of emergent structure and the interaction of autonomous agents in modeling creativity.

3.2.9 Copycat: Creative analogical thinking

As noted in Chapter 1, Letter Spirit's approach to creative modeling is based on the architecture developed over the last fifteen years by Hofstadter's research group [Hofstadter and FARG, 1995]. Jumbo, a program for working on anagram puzzles (or "jumbles"), introduced the idea of temperature-controlled probabilistic processing [Hofstadter, 1983]. Seek-Whence extended the ideas sketched in Jumbo to the sequence-perception task [Meredith, 1986].

Copycat, the most advanced of the fully implemented CRCC models, constructs analogies between strings of letters.[6] Here is an example of one of the simplest and most mundane analogy problems in the Copycat domain: If **abc** changes to **abd**, what happens to **ijk** if you do the "same thing" to it? Or graphically:

Problem 1
abc $\Longrightarrow$ **abd**
ijk $\Longrightarrow$?

Most people answer **ijl** (seeing the initial change as a replacement of the rightmost letter by its successor). Sometimes people answer **ijd** (seeing the change as a

[6] The overall argument in this section is lifted from Hofstadter's Epilogue in [Dartnall, 1994]. For more on Copycat and creativity, see that chapter as well as [Mitchell, 1993].

replacement of the rightmost letter by **d**). These two answers are objectively the most common. Since Copycat is nondeterministic — following different pathways on different runs — it is capable of finding both of these answers. In one set of 1000 runs, it produced **ijl** 980 times and **ijd** 19 times, also getting the strange answer **ijj** one time. A statistical evaluation of Copycat's performance of this sort exhibits the phenomenon by which nearly-deterministic behavior emerges from microscopic non-determinism. This means that if Copycat were re-run 1000 more times, roughly the same global behavioral pattern would emerge.

Copycat also evaluates the answers it gets in terms of their *depth* or *quality* by returning a final temperature value. Temperature ranges from 0 to 100, with lower values reflecting higher structural goodness and internal coherence. In the sample problem above, the **ijl** answer had a final average temperature of 17 (meaning it was considered very deep), **ijd** was somewhat less deep with 23, and **ijj** was far shallower with 48.

Another, more interesting, Copycat problem is:

Problem 2
abc $\Longrightarrow$ **abd**
xyz $\Longrightarrow$ **?**

The focus in this problem is on the **z**, the problem being that **z** has no obvious successor. Copycat's non-circular alphabet rules out the possible answer **xya**, as it stops dead at **z**. In response to this "**z**-snag", people come up with a variety of answers: replace **z** by nothing, getting **xy**; **xyd** (justified as in problem 1 above); **xyz** — do nothing; and **xyy** — take the *predecessor* of **z** since its *successor* cannot be taken.

The deepest and most satisfying solution to the **xyz** problem involves recasting the relationship between **abc** and **xyz** in order to see them as mirror images of each other. The possibility of mapping the two strings in this manner is reinforced by the fact that **a** is first and **z** is last in the alphabet. So the door is opened for mapping the "successor string" **abc** onto the "predecessor string" **xyz** (now considered as running from right to left). This new perception results in three major conceptual slippages — from "first" to "last"; from "right" to "left"; and from "successor" to "predecessor". When these conceptual slippages are applied to the initial, naïve idea of taking the successor of the rightmost letter, one obtains a new rule: *replace the* **leftmost** *letter by its* **predecessor**. Thus the highly satisfying answer **wyz** can be found. This answer is a *creative* answer, and is a miniature example of a "paradigm shift" [Kuhn, 1970].

1000 runs of Copycat on the problem 2 show the following pattern (listed in order of prevalence): **xyd** (77.1%; temp 22), **wyz** (13.7%; temp 14), **yyz** (7.8%; temp 44), **dyz** (0.7%; temp 33), **xyy** (0.3%; temp 33), **xyz** (0.3%; temp 74), and **yzz** (0.1%; temp 42). Notice that although the creative answer **wyz** is found less often than the most prevalent answer **xyd**, its final temperature is lower. This type of discrepancy between prevalence and depth is intrinsically associated with creative answers. Broadly interpreted, this means that creative answers, though not the easiest answers to arrive at, are among the best.

Copycat's discovery of the **wyz** answer to the second problem is related to the so-called "Aha!" phenomenon, whereby the human mind is able to radically and instantly transform its perception of a situation under extreme pressures. Such shifts in perception are common to creative acts. See Mitchell's recent book about Copycat for a detailed account of how Copycat arrives at the **wyz** answer [Mitchell, 1993].[7] The successful modeling of the entire process of snag-hitting, restructuring, deep slippage, and locking-in of the new view, provides an interesting and important way to think about how paradigm shifts occur in the human mind.

As we mentioned above, Copycat's behavior is nondeterministic. Are we suggesting, then, that randomness plays a role in creativity? And if so, what sort of

[7]Also see [Hofstadter and FARG, 1995, chapter 5].

randomness? Isn't making making *random* decisions much weaker (and sillier) than making *systematic* ones? The answers to these three questions are: "yes"; "we shall see"; and "it depends", respectively.

The key to figuring this all out is to make sure not to confuse levels. If all we do is flip coins to pick words out of semantically pre-arranged piles, à la Racter, then randomness is indeed a nonsensical and counterintuitive mechanism. Where would global coherence come from? What provides the top-down structure? These are legitimate and important questions. Fortunately, they do not apply to Copycat's use of nondeterminism, as this use is a horse of a very different color.

In Copycat, randomness controls early micro-exploration (at least at first when things are foggy and there is little perceptual structure to go on). Once structure begins to emerge from the local structure-building activity of many codelets, more reasonable and "macroscopic" pathways appear, and more and more system resources are devoted to the exploration of these pathways (as long as they keep paying off, that is). As structure is built up, the temperature (which controls the amount of randomness in the system) drops. This causes the overall randomness in the system to drop too. So, the clearer things get — as the initial fog is burnt off — the less randomness is required by the system.

Emergent behavior of this type constitutes a *parallel terraced scan*, as introduced in Chapter 1. The parallel terraced scan can be thought of as allocating processing power as a function of the degree of promise of a pathway. This strategy for parallel exploration has much in common with genetic-algorithm search methods and solutions to the problem of exploration versus exploitation [Holland, 1975; Holland et al., 1986].

One interesting side effect of this approach to exploration and search in a cognitive model is the potential of getting terrible answers. The mechanisms required to act creatively inevitably open a system up to this possibility. One such fringe answer is the **ijj** answer to problem 1. Two more are the **dyz** and **yzz** answers to problem 2.

We believe that this is the price that has to be paid for flexibility [Hofstadter and FARG, 1995; Mitchell, 1993]. Bad answers should occur only rarely, however, and fortunately this is the case in Copycat.

Copycat's creativity stems from its ability to balance stochastic flexibility with top-down conceptual control. This allows it to have both guided behavior (like TALE-SPIN, only less rigid) and flexibility (like Racter, only less empty). The parallel terraced scan plays a critical role in this balancing act between flexibility and control.

3.3 Deep and shallow design

Letterform design is an intricate and beautiful art making use of a large number of *letter-concepts* behind the scenes. Visual arts seem to require some type of conceptual design to go on before any physical drawing or model-building is done. Many architecture schools offer entire classes on design conceptualization alone [Vickery, 1983]. Even engineering schools (like Georgia Tech) recognize the importance of teaching students how to conceptualize during the process of design [Janet Kolodner, personal communication]. The key to creativity, we argue, lies in understanding how this conceptualization works.

The grid was invented in order to get to the heart of letters and styles as abstract concepts. We are interested in studying what we call *deep style*. Deep style involves manipulating, adjusting, and assessing stylistic elements at the conceptual level, while *shallow style* involves surface-level (usually geometric) feature manipulation. As such, shallow style is usually concerned with *embellishment*[8] or simple surface-level manipulation of the sort available in MacPaint's pull-down menus.

We argue that this sort of surface play, though often fun, is not truly creative.

[8]The DAFFODIL program provides a prime example of a system relying solely on shallow style of this sort [Nanard et al., 1989]. (See our commentary in Chapter 2).

Real style is more than just skin deep. Work at the level of deep style requires a strong conceptual grounding and a perceptual system with close ties to the conceptual level. Perception guides creative play, leading the search for a plausible design along pathways that seem fruitful and helping to avoid blind alleys.

At the deepest levels in our domain, style involves working with the conceptual foundations of letters (*e.g.*, re-conceptualizing a 'd' as "a post and a loop (o l)" instead of as "a post and a bowl (c l)" in order to relieve the tension caused by some particular style). The distinction between deep and shallow style is of course not a black-and-white one. Instead, a continuum exists between the deep and shallow ends. Every imaginable shade of gray is to be found in between.

3.4 High-level perception and creativity

Our brief survey of nine models of creativity leads us to two main conclusions. First, there is a trade-off between the *flexibility* that randomness gives us (Racterness) and the *control* that the top-down constraints of knowledge give us (Tale-spinitude). Finding a fulcrum on which to balance these two opposing forces is a hard but very critical business. We address this problem with the parallel terraced scan — a decentralized mode of processing in which statistical emergence and conceptual-level perception play important roles. What results is a system that is both flexible *and* robust at the same time.

The second lesson we learn is that in order for a system to be truly creative it must be able to *perceive and assess its own products.* In the case of Letter Spirit, this translates into the ability of the system to be able to deeply understand letterforms at the conceptual level, so that it can "see" letter parts that fill roles, as well as analogies between letter categories as a whole.

We call this second ability "high-level perception". High-level perception ("the process of making sense of complex data at an abstract conceptual level"[9]) is a crucial feature of all creative activity. This type of perception involves processing information up to the level of *meaning* by accessing concepts and interpreting sensory data at a *conceptual level*. Creativity flourishes in the human mind precisely because of the fluid, associative, malleable, context-sensitive nature of human concepts. It is our aim to build such concepts into our model.

Letter Spirit explores one kind of large-scale creative activity. The building-up of an entire stylistically-coherent gridfont requires a vast amount of "A&A" (see the introduction to this chapter). Throughout the central feedback loop of creativity, the dual opposing pressures of *letter* and *spirit* interact in interesting ways to cause unforeseen, creative by-products to become noticed, to take seed, and to grow. Inside the feedback loop, nondeterministic decision-making at the micro-level leads predictably to a great deal of top-level unpredictability.

As we have seen, many AI models of creativity lack any grounding in perception. These "blind" programs rely on their programmers' complicated and subtle human perceptual systems to separate the good from the bad throughout the creative process. It is unclear what would happen if the programs were uncoupled from the people who run them and guide them. Quite possibly they would falter and break down.

The ability to assess as well as to perceive and to understand a work-in-progress has for the most part been taken on by the human developers of many models of creativity. The resulting creative systems are therefore in truth *combinations* of program+programmer. An analogy can be drawn with the hackneyed Chinese Room intuition pump [Searle, 1980]. In the well-known and rather devastating "systems reply" to the Chinese Room argument, the entire system — all of the slips of paper with the rules of Chinese on them, taken together with the person inside who "runs" the rules — is what understands Chinese [Hofstadter and Dennett, 1981]. Neither part

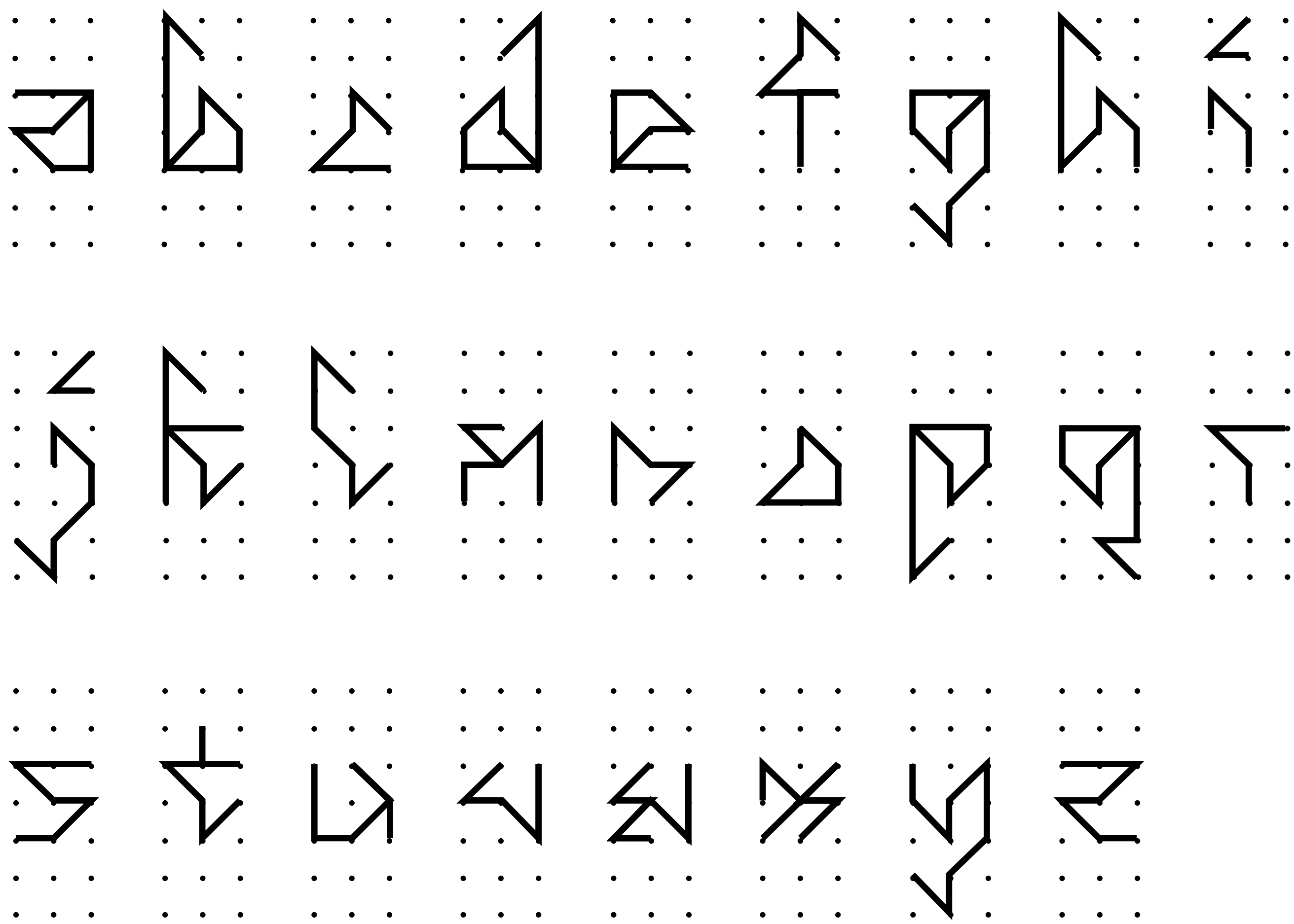
[9]From the abstract of [Chalmers et al., 1992].

alone is sufficient for understanding. The point is that the *system as a whole* turns out the be the best level for understanding what is going on. In the same way, many AI models purporting to be creative may actually be mere parts of a creative *system* that includes a human being acting as a crucial perceptual module. In order to be truly *autonomously* creative, a program would have to combine the roles of Cohen and his program Aaron, or the roles of Lenat and AM, or the roles of Chamberlain, Etter, and Racter. This is our aim in Letter Spirit.

To correctly model the creative process on a computer, a perceiver/assessor must be built into the program [Hofstadter, 1985, Chapter 10]. Relying on human help ignores a crucial aspect of creativity and breaks the creative loop. Letter Spirit will rely heavily on an internal complex model of letter perception — a model in which letters are deeply understood. Letter Spirit's creative activity will be both assessed and judged *from within* as processing takes place. In the Letter Spirit project, a complex perceptual model is being built in from the start. That way, the program will be able to assess its own products as they evolve and change throughout the A&A cycle. The ability to perceive letters in a large variety of styles, and to go beyond a simple categorization to a deeper parsing based on the conceptual level of roles, is the focus of our preliminary implementation work on Letter Spirit. Part II is about our model. Our complex, perceptually-based model of letter-understanding is thus an indispensable stepping-stone on the way towards a full implementation of Letter Spirit.

ONE SOLUTION TO THE CHALLENGE

Just after Chapter 1, we challenged the reader to try and finish designing a gridfont which we provided three seed letters for. On the next page, one of many possible solutions is shown. We have given this gridfont the name *Stony Face*. How does it compare to the gridfont you have designed? What makes the style of *Stony Face* consistent? Which letters have been most affected by the style? How was the *letter/spirit* battle resolved in each of *Stony Face*'s gridletters?



Part II

Recognition of Gridletters

CHAPTER FOUR

The Role Model: Letter Spirit's Examiner

In part one, we argued that the ability to perceive and assess creative products *throughout the design process* is a central aspect of creativity. Driven by this belief, we have spent a good deal of effort developing a sophisticated model of letter recognition. This model will serve as Letter Spirit's Examiner — the part of Letter Spirit that is responsible for determining which category a gridletter belongs to, how strongly and unambiguously so, and how well a letter's parts match the ideals specified by the roles of its categorization.

Letter Spirit is concerned with designing a complete and stylistically coherent gridfont given a handful of seed letters. It will accomplish this task by making use of analogical processing. More specifically, Letter Spirit must be able to look at one or more letters and design others such that some coherent overall style emerges. This boils down to "doing the same thing" for a new letter category that has already been done for others.

Uncovering just what this "thing" making up style is and exporting it from one letter (or set of letters) to another is a cognitive, perceptual activity of the highest order — cutting straight to the heart of esthetics, style, and creativity [Hofstadter, 1985; Mitchell, 1993; French, 1995; Hofstadter, 1995]. Strangely enough, a complex perceptual problem of the same order shows up at an even more basic level than that of style. After all, what is it that makes the 'a's of Figure 1.1 and Figure 1.8 all recognizable as "the same"? Or put another way, what is the conceptual essence of the letter 'a'? That is our driving concern here in Part II.

Thus the twin questions of Letter and Spirit both lead us to focus on a fundamental question of cognitive science and artificial intelligence: *What are the mental mechanisms underlying our ability to perceive, recognize, and categorize diverse stimuli in the flexible manner that we do?* Determining how a computer model's representations can be structured so that they are fluid, flexible, and adaptable to diverse situations instead of rigid, brittle, and unable to be adapted beyond simple mechanical tweaking is one of the key aims of this research.

Work on the Examiner is an attempt to model certain aspects of concepts and categorization. Just as research on its predecessor Copycat did [Mitchell, 1990], this work focuses on modeling concepts in an active, dynamic, associative way — modeling the kind of things that Hofstadter calls *active symbols* [Hofstadter, 1979, chapter 11]. We take a radically different tack with regard to model building than does much of the current psychological work on concepts and categories — the sort dealing with *exemplars*, *prototypes*, *graded structure* and the like [Rosch and Lloyd, 1978; Smith and Medin, 1981]. Nevertheless, we devote an entire chapter to relating our model of letter perception to those that have come before, and attempt to develop a framework within which all letter-perception models can be set.

In this chapter, we introduce a theory of letter perception based on human cognition, and called "the role hypothesis". We go on to discuss an implemented computer model of letter perception developed in accord with our hypothesis. Performance of the "role model", including both detailed analysis of single runs and statistical analysis of many runs considered together, is covered in Chapter 5. Then in Chapter 6, we discuss the relationship between the role model and other models of letter perception, and develop a framework for understanding letter-recognition approaches. Chapter 7 introduces two psychological experiments on the perception of gridletters. Finally, in Chapter 8, performance of the role model is analyzed with respect to human data and competing models of letter recognition.

4.1 The role hypothesis

Literate humans are adept at recognizing letters in many diverse forms. Large variations in style — especially apparent in display faces for publicity and advertising — do not seem to hinder our ability to properly identify characters. Idiosyncratic characters like some of those displayed in Figure 1.1 are easily recognized even when encountered for the first time.

Character recognition by people is still vastly superior to that of machines. Why is it that people are able to quickly and easily recognize letterforms while machine recognition systems fail? We believe what is required is an approach using abstract conceptual representations of letters. We argue that such high-level representations are found in the human mind and have clear effects on perceptual processing. We will describe such representations and provide empirical support for their existence.

The central tenet of the role hypothesis, lying at the core of Letter Spirit, is that letter concepts — the way we conceive of letters in our minds — are made up of sets of smaller ideas called roles. For example, the concept of lowercase 'd' can be thought of as a marriage of: 1) the idea of a straight post sitting on the baseline to the right, and 2) the idea of a 'c'-like curve (or open bowl facing right) also sitting on the baseline to the left of the post. The ways in which these two roles interact are constrained by inter-role relations called r-roles. R-roles specify, for example, where the two roles ought to touch and the fact that taken together they form a closure. Without r-roles, one might have shapes that fill the post and bowl roles very strongly yet look more like 'cl' than 'd'.

4.1.1 Parts versus roles

Palmer [1978] demonstrates the mental existence and utility of representations of structural units on a level above that of lines and/or points, yet below that of a whole figure. He claims that such structural units (similar to what we call *parts*) play a critical role in shape perception, strongly affecting categorization and similarity judgments. Palmer's structural units are attractive to perceptual systems because they have certain simple, "natural" attributes closely related to properties described by gestalt psychologists. These properties do not depend on a learned set of higher-level categories, and for that reason, Palmer's structural units are built in a wholly bottom-up manner by the perceptual system.

We readily agree with Palmer's idea of intermediate-level structural units, or parts, but go a step further and argue that such parts are formed under a mixture of gestalt-like bottom-up pressures along with top-down pressures coming from conceptual-level category representations. Under our hypothesis, roles provide top-down pressures on the parsing of letterforms into sets of parts. The emergent properties of "good" parts will tend to fall within *norms* provided by roles. This account has some similarity to Biederman's Recognition-by-Components theory [Biederman, 1987].

The parsing of a letterform into parts can be understood by analogy with the parsing of words into speech sounds. Linguists and phoneticians have attempted to exhaustively list all the many possible sounds that humans can utter in any language. They call these sounds *phones*. Each language, however, uses a relatively small selection of regions in the space of all possible speech sounds to distinguish words. These language-specific phonetic entities are called *phonemes*. The phonemes of different languages carve up the space of phones in diverse, often incompatible, ways. Thus, for example, while English speakers can readily distinguish the vowel phonemes of *beet* and *bit*, speakers of Greek or Spanish find this extremely difficult. When an utterance is perceived by a listener, its constituent phones are automatically forced into being heard as members of one or another phoneme of the listener's native language. In other words, one's native language conditions one to hear certain types of speech-sounds and not others. This is a quintessential example of top-down influence in perception. In much the same way, the parsing of letterforms into one set of visual parts as opposed to another, equally plausible set is influenced in a top-down way by the roles defined in the letter-conceptualizations or character-conceptualizations of the writing system one has grown up with.

Takano [1989] identifies four distinct and orthogonal types of information that are extracted from figures by the perceptual system. These include the identity (shape) of individual parts, orientations of individual parts, connectedness of parts to other parts, and relative orientation between part pairs. The first two types of information Takano discusses are analogous to information contained in our roles. The second two types are analogous to information contained in r-roles.

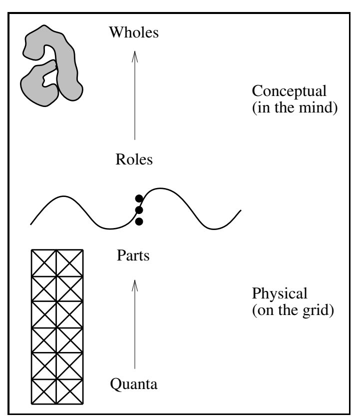


Figure 4.1: Quanta and the chunks they organize into (called *parts*) are physical entities. Roles and wholes are mental entities.

Figure 4.1 shows the relationship between quanta and parts (physical entities found on the grid) and roles and wholes (mental entities found in the mind). Parts, chunkings of actual quanta on the grid, fill mental roles. Together, one or more roles along with specific inter-role relations make up a whole.

4.1.2 Letter-conceptualizations as sets of roles

Some conceptualizations for a given letter category are better than others. A conceptualization's quality depends on how strongly the spectrum of instances that it defines are rated by viewers as members of the intended category. Those conceptualizations that define the strongest instances are more important than their rivals, and provide standard ways of thinking about a given letter. Figure 4.2 is a graphical representation of all 66 sets of roles found in the Examiner. Each lowercase roman letter category has from one to four distinct conceptualizations.

4.1.3 How roles are filled

The style of a letterform is a function of how the various roles in the letter-conceptualization are filled. Perhaps a more vivid and accurate way of saying this is that style has to do with *how norms are violated.* For instance, consider the role *crossbar*, which belongs to conceptualizations of 'e', 'f' and 't'. Possible norm violations for this role are many, but certainly include the following: "crossbar too high", "crossbar tilted upwards", "unusually short crossbar", "crossbar missing", etc. Any such violation is a stylistic hallmark that helps constitute a letterform's style. In extreme cases (involving letters at the fringes of their intended categories), stylistic aspects of a letterform have clear effects on the ease of its categorization. A letterform with poorly-filled roles will be harder to categorize than a letterform with normally-filled roles.

Figure 4.2: A graphical representation of all of the 66 role-sets (or wholes) included in the Examiner. Roles are not themselves shapes; instead, they are abstract sets of norms at the conceptual level into which parts of letterforms can be fit. These pictures are meant to show how such roles work together to define letter-conceptualizations. The black dots shown represent between-role relations (r-roles).

Figure 4.3 shows how letterforms are comprised of parts that fill a letter conceptualization's roles. During perception, such parts are formed under top-down pressure from roles and are sensitive to context. As stated in Palmer [1978, page 96], "components [of parts] enter into relationships with other components, resulting in larger structural units whose importance supersedes that of [their] constituents." We hold that most of the "importance" attributed to the emerging parts stems directly from their role-filling ability. In other words, the way in which a part fills a role directly determines its "goodness."

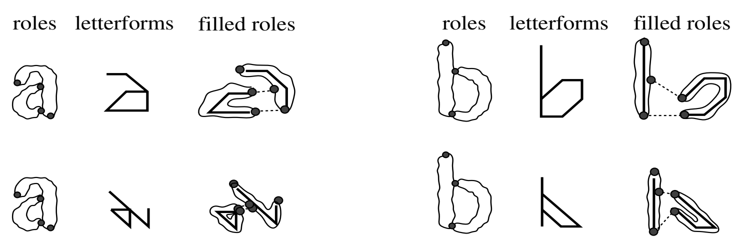


Figure 4.3: Parsing letterforms into intermediate-level parts occurs under the top-down influence of roles. The top two of these four examples show straightforward parsings (or "good" role-fillings) involving only mild violations. The bottom two letters are more complicated, and have "bad" role-fillers. The roles must be plastic enough to accommodate the somewhat nonstandard parts of the bottom two letters. In other words, roles must be able to tolerate a certain amount of norm violation on the part of their fillers.

Sometimes norms associated with roles must be violated in order to accommodate a given letterform's parts. In this sense, roles are "plastic". This plasticity is a result of allowing certain slots in the mental representation of a role to be filled by any of a range of closely related part-descriptions. Some of the norms making up the representation of a role are more malleable than others. We argue that the non-brittle nature of human letter perception is due to plasticity at the level of roles. It has been our aim to capture this plasticity in a working computer model.

4.2 The role model

Now we turn to the role model itself — a specific implementation of the role hypothesis. Over the past three years, the role model has been implemented as a working computer model. The program is written in Scheme (a dialect of Lisp) and has grown to a size of more than 25,000 lines of code.

After an overview of the role model's processing and a review of the general architectural constructs (first introduced in Chapter 1), we will get down to the nitty-gritty and discuss some code-level implementation details. We will refer to our model — Letter Spirit's Examiner — as the "role model" throughout the rest of the thesis.

4.2.1 An overview of processing in the role model

In order to understand the role model's recognition process clearly, it is useful to cast it in terms of *syntactic* and *semantic* operations. Syntactic operations are purely bottom-up chunking operations. They serve to put quanta together in a way that would be reasonable no matter what type of shape was being recognized. They are, in other words, *context-free* chunkings that presumably would arise as the result of any naturally-evolved visual system. Semantic operations, on the other hand, are *context-dependent*. They depend on the specific repertoire of categories into which the stimuli are to be filtered — a repertoire that in people has been acquired through experience. The context-supplying categories of Letter Spirit are the 26 lowercase roman letters. Semantic operations take the output of syntactic actions (which usually occur earlier in perceptual processing) and adjust and refine it. The upshot of syntactic and semantic processing is the "marriage" of fresh sensory stimuli with prior concepts.

In brief, then, operations that depend in any way on the prior set of high-level conceptual categories into which stimuli are to be eventually classified are semantic. Low-level operations that are independent of those learned categories are syntactic.

Work on the role model explores the gray area where syntax and semantics meet. The categorization process, working up from syntax to semantics, and also downwards, constitutes high-level perception (as introduced in Chapter 1).

Processing begins at the level of quanta and starts out being completely bottom-up. Quanta are syntactically chunked into parts, which are then labeled according to their low-level syntactic characteristics. Top-down semantic influence first enters the picture as the labeled parts are matched up with conceptual roles. Eventually, as the interpretation rises towards the level of letters (wholes), even more top-down influence is brought to bear. As was mentioned above, letters have more than one possible conceptualization — that is, set of interrelated roles (see Figure 4.2). Standard conceptualizations of the 26 letters (usually two to four for each letter, as shown in Figure 4.2) are provided to the program as conceptual prototypes. A crucial goal is for the whole Letter Spirit program eventually to be able to develop and test new conceptualizations during a run, but the role model on its own is not capable of this level of sophistication.

The role model operates roughly as follows, although this outline may give the impression that processing is more serial and well-ordered than it really is. In reality, processing occurs in a more parallel manner, with various aspects described below often proceeding concurrently.

- Quanta are probabilistically bonded together (by local perceptual micro-agents) with different amounts of "glue". (For example, more glue tends to be deposited at straight junctions than at angles.) The glue-depositing agents execute completely bottom-up syntactic operations.
- When enough glue has been deposited, the glued shape is metaphorically "shaken". This amounts to probabilistically breaking the glued shape into chunks of quanta at weak joints, resulting in a set of parts (usually made up of between two and four quanta). Since no concepts have been invoked up to this point, parts are

still purely syntactic entities.

- Each part is scanned by multiple instantiated codelets that probabilistically attach *syntactic labels* to the part. Labels reflect very simple properties of parts like curviness, length, width, location, and so on, and in no way involve the set of categories (either at the level of roles or wholes) into which these stimuli will eventually be channeled. Each syntactic label has a real number associated with it, standing for the strength with which the label applies to its part.
- When a part has accumulated enough syntactic labels (once again, a probabilistically determined event), it is allowed to send activation to one or more *roles*.
- The presence of a particular label on a given part serves as a cue that tends to lightly activate one or more roles with which the label is associated (*i.e.*, roles of which the given label is at least somewhat diagnostic). For instance, the labels "left-side", "straight", "skinny", and "tall" would tend to activate the "left-post" role found in 'b', 'h', 'k', and sometimes 'l'. It is important to understand that even a label such as "straight" is probabilistic, in the sense that a not-totally-straight part *might* get that label, with probability diminishing with its non-straightness. The real number associated with each label reflects this "goodness of fit".
- A part that does not strongly activate any roles will be slated for destruction, with its constituent quanta subjected to the part-forming process all over again.
- The various light activations coming from a given part's labels sum up to a *total* activation-level for each role that the part matches sufficiently. If activated highly enough, roles associate themselves with labeled parts, with the best-matching roles getting the most activation from a part. Each particular role may be filled by one part at any given time, although inter-part competition for the role's attention is ongoing and sometimes fierce.

- As roles and parts attempt to "mate", a given part may need to be slightly altered in order to be a good mate for a given role. A quantum or two may need to be stolen from one or more neighboring parts to make the part in question more attractive to a possible match. Likewise, small pieces that seem to make a part ugly in the eyes of possible role-mates may need to be given away or simply detached. (Note that the parts are merely undergoing *regrouping*; the underlying quanta defining the letterform of course remain untouched.) The resulting structures composed of quanta are now results of the combined influence of bottom-up and top-down processing. As such, these parts are no longer totally syntactic entities, and we call them *(semantically) adjusted parts.*
- Roles compete for parts throughout the letter, adjusting the parts as they go. When this adjustment-and-association phase is over, there is a fairly strong match-up between roles and semantically-adjusted parts.
- Each instantiated role has a few tags attached, stating *how well* the given part instantiates the role. This information focuses on how the part deviates from various norms associated with the role.
- Each realized role begins to alert one or more wholes (*i.e.*, full role-sets such as 'a') for which it provides evidence, in the sense of fitting a particular conceptualization of that letter. Role/whole coupling is analogous to part/role coupling, only it occurs at a higher (more semantic) level. Particular letter-conceptualizations are activated according to how strongly their component roles are realized in the actual grid letter.
- The activation level of each hypothesized whole is adjusted according to whether the whole's r-roles (*i.e.*, inter-role relations) approve of the structure discovered so far. When r-roles for a particular whole are checked, activation is taken away from any whole whose roles are not appropriately related or filled. This is a critical inhibitory aspect of the categorization process.

- Different wholes thus become activated to different extents. Each sufficiently-activated whole attempts to match itself up with the shape on the grid.
- If there is a clear leading contender among the wholes, it is deemed the winner. If there is a close race between several, the letterform is deemed ambiguous and therefore unacceptable. In a borderline case between clear-winner and close-race, a probabilistic decision is made that chooses between the two courses of action.
- In the end, the winning whole has been parsed into constituent, non-overlapping parts. These parts fill specific roles in the whole to a greater or lesser extent. The tags attached to roles relating to how well they are filled by their associated parts are also available when processing ends. For example, a 't' whose spine is too tall or is bent over at the top, or whose crossbar is too short, too high, or tilted, will have tags stating such things attached to its filled conceptualization. This information can be used in further processing, including checking the style of a letterform and designing related letterforms by analogy.

The entire labeling and role-association process happens in parallel for each of the initial parts created after shaking. In general, processing proceeds from low-level syntactic processing to high-level semantic (or conceptual-level) processing, but is not completely linear in nature. Conflicting perceptual structures compete against each other in the context of current perceptual trends. Weak structures tend to be destroyed and strong ones to be strengthened.

Figure 4.4 shows in schematic fashion how the role model might recognize a very simple 'b'. Some of the interrelations between levels of processing are abstracted away in order to make the picture more readable. It is important to remember that although the "roles" in Figure 4.4 are shown as shapes, they are actually abstract conceptual entities represented in the model as collections of norms.

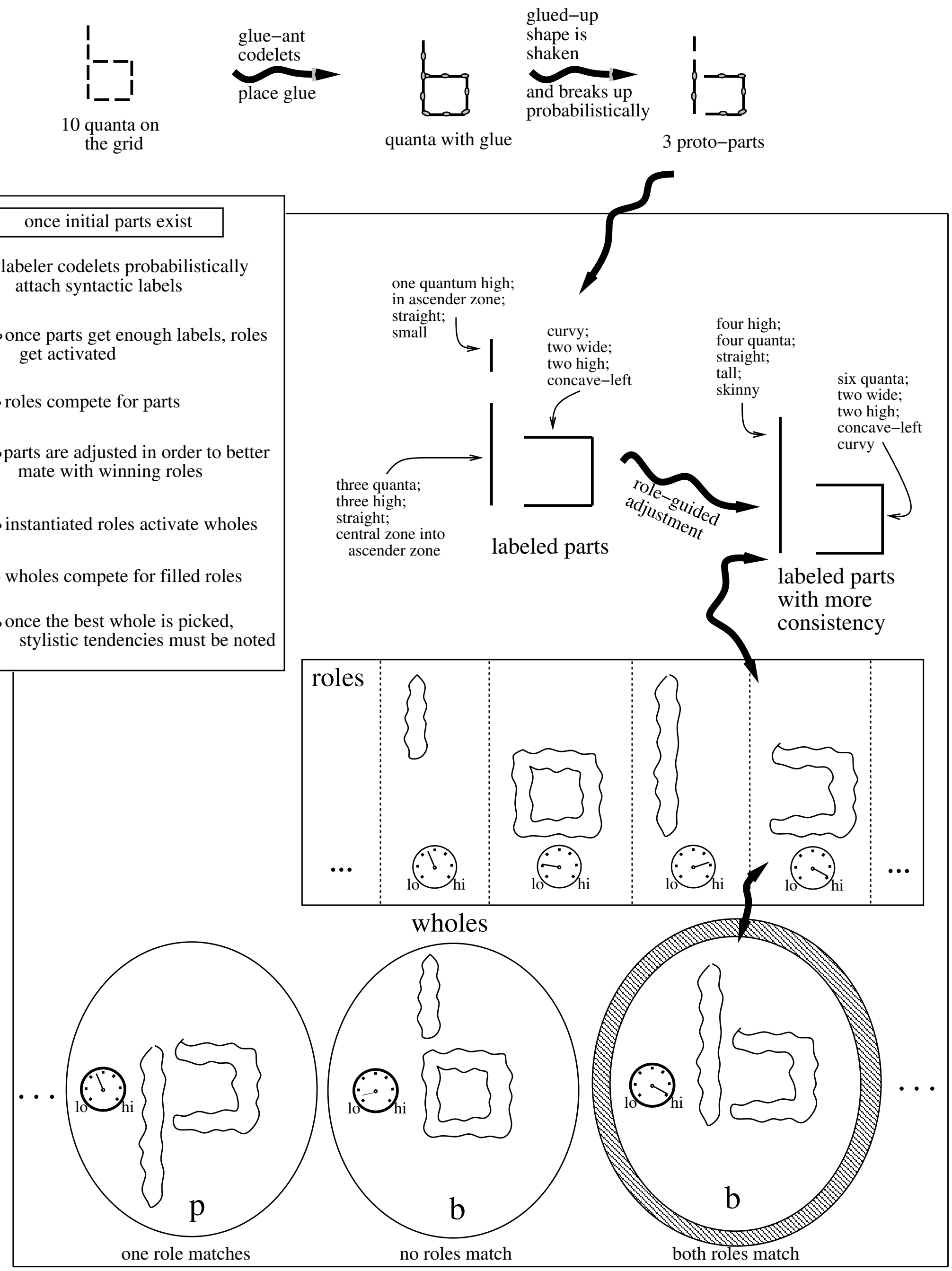


Figure 4.4: A schematic diagram showing how a 'b' is recognized by the role model. The wiggly shapes at the bottom should not be thought of as shapes, but as roles.

4.2.2 Hearsay II

Many of the ideas behind the parallel terraced scan were originally inspired by the Hearsay II speech-understanding system [Erman et al., 1980]. Hearsay II takes as its input a waveform generated by a spoken utterance. This waveform is interpreted through the cooperation of various "knowledge sources", each of which is able to perform a specific operation, such as segmenting a region of the waveform, creating phones and phonemes from segments, building-up syllable-class hypotheses from phonemes, and creating word hypotheses from syllables. The knowledge sources build data structures from the various levels of interpretation on a global blackboard, which thus serves as the locus of communication between the knowledge sources.

The various knowledge sources in Hearsay II build many levels of data structures in the process of moving from a raw waveform to a fully-parsed utterance. Each of these data structures is based on hypotheses built at the level immediately below its level (*e.g.*, segments justify phones, phones justify phonemes, phonemes justify syllables, and so on). A key feature of Hearsay II's architecture is an interaction between top-down and bottom-up processing, with structures built at lower levels providing evidence for higher-level hypotheses, and higher-level hypotheses affecting the kinds of low-level structure that get built.

The idea of simultaneous bottom-up and top-down influences interacting in the process of perception — the fundamental bedrock of the role model (as well as Copycat and Tabletop) — was inspired by Hearsay II. Reddy and Erman *et al.*'s modeling of top-down influence on speech perception maps very clearly onto Letter Spirit's role-driven model of letter-perception, as shown in Figure 4.5.

Along with all of these important similarities, there are some important differences too. They boil down to three. 1) Codelets in the role model do much smaller, local tasks than the knowledge sources of Hearsay II. For example, a single Hearsay II knowledge source segments the *entire* raw waveform all at once. By contrast, the role

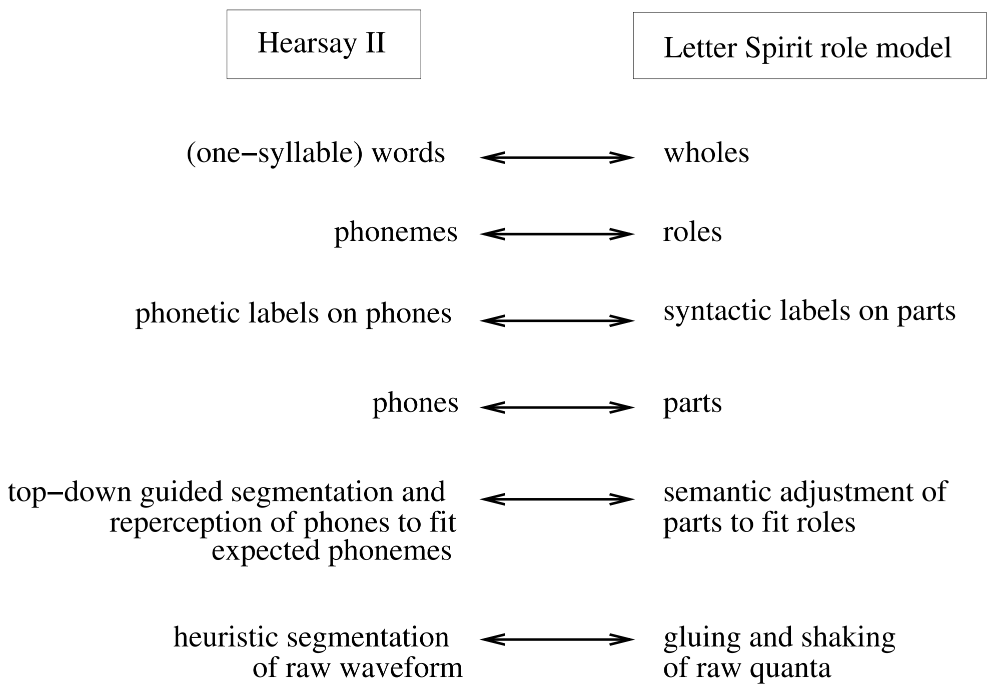


Figure 4.5: An analogy between Hearsay II and the role model.

model must build its way up to large-scale structures. 2) The parallel exploration of possible hypotheses is different. Hearsay II allows many hypotheses at each level of the program to coexist. When determining which hypothesis is best, it evaluates all of them at the same time. This strategy is potentially subject to problems of combinatorial explosion. By contrast, the role model investigates one view at a time, since only one set of physical parts can exist at once on the grid. This is actually a considerable oversimplification, since roles compete in parallel to match up with parts, reforming them semantically in the process. 3) There is also a crucial difference in the two programs' control structures. Hearsay II has a central scheduler that assigns a "priority" to each knowledge source. The scheduler picks the highest-priority knowledge source every time, in essence reasoning its way in a lock-stepped and highly rational manner. By contrast, the role model is nondeterministic — making processing in the role model more flexible than in Hearsay II.

4.2.3 The subsymbolic paradigm, emergent computation, and the parallel terraced scan

It is difficult to pigeonhole the role model as belonging to any particular well-known cognitive-science school, as it is neither fully symbolic nor fully connectionist. Instead, it takes some of the central features of both paradigms and mixes them together, and thus could be said to fall somewhere between symbolic AI and connectionism.

On the one hand, in common with connectionism, the role model has many important subsymbolic characteristics of the sort that Smolensky advocates [Smolensky, 1988]. In the subsymbolic paradigm, cognitive representations are built of *subsymbols* that in turn give rise to symbol-like structures. Subsymbols are small entities like the nodes and connections of a connectionist network. In systems of the subsymbolic vein, symbols are *statistically emergent* entities that are represented by patterns of activation over large numbers of subsymbols [Hofstadter, 1985, chapter 26]. Subsymbolic systems were invented in part to overcome the brittleness of symbolic systems, and as such, subsymbolic descriptions are in some sense "softer" and less brittle than their symbolic counterparts [Smolensky, op. cit.].

The role model's fine-grained parallelism, local actions, competition for limited computational resources, spreading activation, and emergent concepts are all faithful to the subsymbolic enterprise. Also closely related is the interaction of top-down and bottom-up processing in the model.

On the other hand, the role model's representations have something in common with more traditional symbolic methods. The emergent representations that the system develops are able to be quickly and easily *referenced* and *explicitly manipulated* since they are cut from symbolic cloth (*i.e.*, they are made up of Scheme structures, albeit with attached and dynamically varying activation values). Furthermore, these symbolic structures are built up in a workspace similar in some respects to a short-term memory — something not often found in connectionist models. Although some may argue that connectionist systems may never actually require a level of explicit symbol-processing (since networks working on distributed representations can be "built up" on top of each other, finally reaching a high level of cognitive activity), we disagree with this intuition. The notion of *reference* is a critical one in models of higher-level cognitive activities such as analogy-making [Gentner, 1983; Indurkhya, 1992]. By their very nature, symbols — even of the active, emergent variety that we model — provide a natural avenue for such reference. In other words, symbols *stand for something* and can be treated as such throughout processing [Newell and Simon, 1981]. Modeling high-level cognitive phenomena without regard to reference is a monumental (and possibly misguided) undertaking that, at least in its current connectionist guise, has yet to show a great deal of promise.[1] Although many of the anti-connectionist claims made in Fodor and Pylyshyn's famous article [Fodor and Pylyshyn, 1988] have been successfully refuted in the past few years (see especially [Chalmers, 1993]), the notions of reference and explicit structure-processing in connectionist models requires a great deal of further research.

In the role model, the high-level behavior of categorization arises as an emergent property of many small, local activities that occur in parallel. Processing is carried out by hundreds of codelets. The codelets build, rearrange, and destroy structure in a shared working area under the top-down influence of letter categories. The resulting quasi-symbolic structures are statistically emergent by-products of codelet activity.

Other projects at CRCC have focused on a similar approach to emergent, high-level perception and predate the role model in use of emergent computation and the parallel terraced scan. These include the Seek-Whence model [Meredith, 1986; 1991], Jumbo [Hofstadter, 1983], Tabletop [French, 1992; 1995], and Copycat [Mitchell, 1990; 1993].

[1] This theme recurs in Chapter 6 where connectionist research relating to the notions of reference and explicit structure manipulation is discussed in greater detail.

For a clear and comprehensive survey of the these models and the guiding principles behind them, see [Hofstadter and FARG, 1995].

Jumbo — a program to create plausible anagrams — is perhaps the most closely related to the role model since it was designed to rearrange (or "parse") a jumbled string of letters into plausible-seeming English-word-like strings using an early version of the codelet architecture later found in Copycat, Tabletop, and the role model. (Jumbo's task is quite similar in nature to the task of combining quanta in a letterform into parts that, by filling roles in a letter conceptualization, represent the physical manifestation of a letter category.) Unlike other anagram-solving programs, Jumbo has no dictionary. Its knowledge consists firstly of a table of "affinities" between letters, which lists relative strengths of consonant and vowel clusters, and secondly of a few very simple heuristics describing how these clusters themselves "glom together" to form syllable-level and word-level groups. For example, the program implicitly knows that the grouping "ough" is a much more plausible way of arranging the letters 'g', 'h', 'o', and 'u' than the grouping "ouhg". The program works by glomming together (or chunking) groups at various levels: the letter level, the cluster level, the syllable level, and ultimately the word level.

Jumbo does its work by "constructing larger units out of smaller ones, with temporary structures at various levels and permanent mental categories trying to accommodate each other" [Hofstadter, 1983]. Throughout the process of building up its answers, Jumbo engages in a large amount of back-and-forth activity — constructing, tearing down, and swapping representational structures in a "fluid" manner. Jumbo's overarching philosophy is that perception (which is how the mental build-up of anagrams is cast) is the result not of a serial and mechanical application of hard-and-fast rules, but of a parallel process in which large numbers of independent micro-actions compete with and resonate with each other as they build and destroy perceptual constructs. There is no global executive in such a system, playing the role of symphony conductor queueing in the viola section as required by the score, or the role of four-star general marshaling troops to dangerous areas on the front. Rather, independent micro-agents make local probabilistic decisions "on their own". Thus the system's philosophy is one of self-organization.

Jumbo's approach to perceptual modeling is based on two metaphors: the process by which molecules are constructed in the cytoplasm of a cell, and the parallel probabilistic search strategy used by a person in searching out a mate. Biological cells have no global executive controlling their production of molecules. Instead, molecules are formed through a series of chemically defined steps (perhaps dozens) that are carried out in parallel and essentially at random, depending upon chance encounters between enzymes and their substrates. Complex molecules arise through many waves of enzymatic activity in which products of one wave end up serving as substrates for the next, and so on. The enzymes themselves are produced according to the "needs" of a cell. Thus structures in a cell self-organize according to an interplay of chemically-defined rules and chance meetings in the cytoplasm. As in a cell, Jumbo has no central executive. Its parallel and independent building-up of structure from a limited supply of resources echoes cellular processing, with codelets playing the part of enzymes.

In Jumbo's anagram domain, there are thousands of possible ways to rearrange a given set of letters. One of the main research questions addressed by Jumbo is how a potential combinatorial explosion of possible combinations can be efficiently and successfully dealt with and defused by quickly sniffing out the more promising directions and funneling computational resources (*i.e.*, processing speed) towards them, with the most resources going to the most promising avenues. This is where the second metaphor — that of search strategies in romance — enters the picture.

When a person is initially searching for a future romance, many potential mates may be considered in a very cursory, surface-level way. Most of the people met through chance meetings, at coffeehouses, classes, concerts, and other social events, may be immediately rejected as "not the right type", but others are eye-catching and merit further consideration. Interest is sparked in this way. Time and further interest are dispensed among the few remaining potential mates according to their estimated promise as future romantic partners. More interest and time are spent getting to know those who seem to promise the most than in exploring those that seem like long shots or that turn out not to be as first imagined. Mutual evaluation occurs as potential partners get to know one another, and eventually more and more time is spent with fewer and fewer people. Ultimately, two people may commit to each other and form a couple. Of course, other chance meetings may occur as new people, initially "outside the system", are injected into it, throwing new variables into the "interest equations" as the process unfolds. In fact, even marriages, a highly stable form of couplehood, are susceptible to "flirtations" that are usually harmless, but have the potential, at least occasionally, to build up and threaten even a strong bond.

Hofstadter chose to call this kind of exploration strategy a "parallel terraced scan", "parallel" because many possibilities are entertained simultaneously (especially at the beginning), and "terraced" because those possibilities that promise the best payoff (at least as dynamically estimated throughout the course of a run) get the most resources, and also because levels of depth (*i.e.*, allocation of resources to particularly interesting possibilities) increase as processing continues. This phenomenon has also been likened to the selection process known as "rush" in the "Greek" system on American college campuses, and to picking out a book in a bookstore [Hofstadter and FARG, 1995, Chapter 2].

In the parallel terraced scan, paths are explored in proportion to their estimated promise. Such estimates are adjusted in a local manner as processing unfolds. Time and other computational constraints on systems in the real world ensure that any brute-force investigation of all possibilities is both unattainable and impractical. Likewise, deep consideration of only one or two possibilities, not even considering any others at all, may lead down a computationally expensive blind alley. The parallel terraced scan offers a solution to this dilemma (which is related to the problems that arise in the far simpler breadth-first and depth-first search strategies of AI). This strategy for parallel exploration has much in common with genetic-algorithm search methods and the solution to the k-armed bandit problem [Holland, 1975; Goldberg, 1989].

Consider a two-armed slot machine with one arm paying off at a rate of 5-to-1 and the other at a rate of 2-to-1. Enter our polyester-clad, octogenarian gamester. She does not know which arm pays off better than the other when she first starts playing the slot machine (which she calls a two-armed bandit). She does, however, have a big shopping bag filled with shiny new quarters and wishes to maximize her earnings. Obviously, if she knew which arm had the greatest payoff ratio *ahead of time*, she would pull that arm each time she fed the machine a quarter. But instead, she somehow has to use *some* of her quarters to try to figure out which arm pays off the best. In other words, she has to explore the environment. Initially she pulls either of the arms at random (*i.e.*, with no bias). But over time, as one arm is outperforming the other throughout the course of coin-feeding, she comes to notice this and adjusts her strategy to pull that arm more often. In other words, she exploits what seems to be the more favorable arm. But this being a stochastic system, it is not certain that the arm that appears to be the best *locally speaking* (say, within the last four pulls) is indeed the best *globally speaking* (overall). Our gamester's clever strategy is to pull the arm that she hypothesizes is best *proportionally more* often than the other, with the proportion being a deterministic function of the currently measured ratio of payoffs. In other words, as her information grows more and more reliable because of increasing statistics, she spends less and less time investigating the avenue that seems less promising, even though *in theory* it always remains potentially the better one. This type of strategy has been shown to be optimal for the two-armed bandit problem [Holland, 1975], and can be nicely generalized to a k-armed situation. Throw in the ability to pull arms in parallel (using multiple quarters, of course), and you have a situation in which the parallel terraced scan will provide riches.

As progress is made in a parallel terraced scan, search in the system gradually changes its mode from initially highly parallel and totally random (*i.e.*, evenly biased) to more serial and less probabilistic. This change of processing mode is due to the terracing of the scan as well as to the ability to distinguish between paths leading to structure that matches search criteria and paths leading to structure that is irrelevant. This kind of search process seems to pervade human cognition and could well be posited as a general feature of intelligence.

One of the facets of the parallel terraced scan introduced in Jumbo is a global measure of coherence or "happiness", called *temperature*. Temperature reflects how closely Jumbo thinks it is to creating a complete, happy, pseudo-word structure. The happier the program is with the structure it is building, the lower the temperature is. Temperature in turn affects the system's processing by controlling the degree of randomness. As "good" structure is built up, temperature falls from an initially very high value. Because of this, as the program moves towards a solution and structure begins to emerge, randomness in the codelet-selection process plays less of a role. This causes a speedup in the exploitation of more promising paths. As temperature falls and randomness is decreased, codelets are picked in a more deterministic manner according to their urgencies. Eventually, the system converges on a good global structure (an answer), and temperature becomes very low, allowing a solution to "freeze" into place.

The most advanced CRCC models, Copycat and Tabletop, have applied the the parallel terraced scan to the high-level cognitive activity of analogy-making. Both models combine another critical architectural feature with the parallel terraced scan — an associative network of permanent concepts that is able to provide top-down direction to the search process. Copycat, discussed in some detail in Chapter 3, makes analogies in a domain of letter strings (for example, **abc** $\Longrightarrow$ **abd**; **iijjkk** $\Longrightarrow$ **?**). Like

Copycat, Tabletop also makes analogies, but in a more down-to-earth domain — that of objects on a restaurant tabletop.

A Tabletop problem involves two people (Henry and Eliza) sitting on opposite sides of a restaurant table. The table has an array of objects on it, including cups, glasses, dishes, forks, and so on, arranged in usual ways (for instance, in slightly disrupted place settings). Henry touches an object on the table and says "Do this!" The Tabletop program, in its role as Eliza, must map its objects onto those of Henry and decide which object to touch in response to the challenge. This decision is made by thinking analogically.

Tabletop and Copycat share the same sophisticated cognitive architecture applied to two different microworlds. The Tabletop domain is slightly less abstract than is Copycat's and provides evidence for the generality of the architecture developed in the Copycat project. Differences include: Tabletop's more richly described objects; its two-dimensional, continuous world; and problems of "ownership" of objects and "territory" on the tabletop. Tabletop's implementation of the common architecture also differs in several ways from Copycat's, although in general the approach is very similar. A more detailed account of the differences between Tabletop and Copycat can be found in [French, 1992]. Regardless of the slight differences, the fact that the same general architecture has been successfully applied in two different microworlds provides evidence of the generality of the mechanisms.

The role model, using these same mechanisms, provides a strong and informative model of letter recognition and categorization (as evidenced by its flexible performance and human-like processing, discussed later in this thesis). This fact renders claims regarding the generality of the parallel-terraced-scan mechanisms even stronger. Because the models we have built work well in a variety of situations from low-level to high-level, we feel that claims about the generality and pervasiveness of the mechanisms that we have developed are empirically justified.

Though it shares many architectural features with Copycat and Tabletop, the role model differs in important ways from them. The task that it carries out involves a somewhat lower level of visual processing than the analogy problems in the other two programs. Letter recognition is by necessity closely tied to low-level vision (though we have avoided some of the problems of low-level vision by starting at the level of quanta). Both Mitchell and French point out that the mechanisms modeled in their programs should be applicable to categorization and recognition tasks. Our work on the role model proves that such claims are justified. Collectively, we have shown that the parallel terraced scan models a broad spectrum of cognitive activities, ranging from the mundane and everyday (letter recognition and categorization) to the exalted and "mystical" (feats of insight and creativity, such as when Copycat discovers the **wyz** answer). The power of the parallel terraced scan will be further tested and explored by future work on the Letter Spirit and Metacat projects.

Another important difference between the models is the level of sophistication in the modeling of concepts. The concepts in the role model's Conceptual Memory are more sophisticated than are the concepts in the Slipnets of Copycat and Tabletop. Concepts and "sub"-concepts are modeled in a nested, hierarchical way. The notion of norms and their effects on activation is explored. The concepts that the role model includes are highly interrelated and must vigorously compete to explain the same, possibly ambiguous sensory data.

The most important architectural features of the role model, however, stand squarely on common ground with Copycat and Tabletop. These features allow for the successful implementation and application of the parallel terraced scan, the modeling of the emergence of structure, and the exploration of high-level perception.

4.2.4 The three main architectural features of the role model

Codelets, the Coderack, and the parallel terraced scan

The role model makes use of the parallel terraced scan to model the process of letter perception. This is a dynamic activity involving the interplay of sensory data with the conceptual level at which categories are naturally represented. By making use of the parallel terraced scan, we can model the parallel way in which letterforms explore potential ways of being parsed during the process of recognition, with roles providing top-down guidance to the process.

The parallel terraced scan is implemented in the Examiner in much the same way that it was in Jumbo, Copycat, and Tabletop — through the use of many codelets. Intermediate- and higher-level structure emerge from the hordes of local, independent actions carried out by codelets in (simulated) parallel. Thus perceptual structure is an emergent phenomenon in our model, and the parallel terraced scan is a kind of emergent computation [Forrest, 1991].

Parallelism is simulated on a serial machine using a Coderack — a stochastic staging-area for codelets. Each codelet, when posted on the Coderack, is given an *urgency* value corresponding to its estimated importance and/or relevance to the current situation. Codelets are picked to run from the Coderack in a probabilistic manner, with high-urgency codelets having a greater chance of being picked to run than low-urgency ones.

The Visual Focus

As Copycat and Tabletop have a Workspace, and Jumbo has a "cytoplasm", the Examiner in Letter Spirit has the Visual Focus. All processing in the role model takes place in this working area — a working memory that could be described as a short-term, subcognitive memory.[2] In the Visual Focus, perceptual structure is built up and converges to a stable categorical decision during the recognition process. Codelets build and destroy structures in the Visual Focus as a letterform is categorized. The resulting perceptual structure — including a category label, a rating of how strong the letterform is as a member of the category, what its constituent parts are, and how well these parts fill their corresponding roles — is made available for further processing.

Because of the parallel nature of codelet processing, it is essential to have a common working area, like Hearsay II's blackboard, in which all codelet activity occurs. Structure in the Visual Focus acts as a sort of subcognitive "collective memory" for the codelets. That way, codelets can run, make simple changes in the Visual Focus, and die, with the results of their actions — their life's work, so to speak — being available for (and susceptible to scrutiny by) codelets that follow. Because it is a common area used by all codelets, the Visual Focus makes parallelism possible.

Parts are created, judged, destroyed, regrouped, and eventually approved in the Visual Focus. Another way of saying this is: parts *emerge* in the Visual Focus. They are created by the collective interactions of many codelets under the top-down guidance of the Conceptual Memory. As a stable categorization arises from the local computation of codelet activity, it is duly recorded in the Visual Focus.

The Conceptual Memory

One thing that Jumbo was lacking was any sort of top-down control on its processing. Top-down processing plays a very important role in both the Copycat and Tabletop projects. Top-down pressure is brought to bear in a perceptual situation — such as the building-up of perceptions of the place-setting in Tabletop or the letter-strings in

[2]See Section 1.7.

Copycat — by the concepts modeled in both models' Slipnets. Links between concepts in a Slipnet represent conceptual distances. As processing unfolds and a situation is perceived, link lengths in the Slipnet are dynamically adjusted so that concept activations and distances between concepts reflect the current perceptual situation. This makes the Slipnet sensitive to context, unlike more static semantic networks. A shortening of the distance between two concepts makes a "slippage" from one to the other more likely to occur. It is slippage that makes subtle analogies possible.

The Examiner has a model of concepts as well — the Conceptual Memory. Unlike link lengths in the Slipnet, however, link lengths in the Conceptual Memory do not dynamically adjust during the perception of a gridletter (though activation certainly ebbs and flows). As Letter Spirit develops further and other modules are added, the adjustment of conceptual distances will come to play a more central role in the model. This sort of conceptual adjustment is essential to analogy-making, as it gives rise to slippages that allow one situation to be "seen as" another. Thus during letter design, when aspects of one letterform are transferred via analogy to others, slippage will necessarily be involved.

As it stands now, the Conceptual Memory is very different from Copycat's and Tabletop's Slipnets. Letter categories in the Conceptual Memory are broken down into constituent roles. These roles are concepts in their own right, although simpler ones than wholes. In our model, then, the nested, hierarchical nature of concepts has been more explicitly modeled than in the other models, but interconceptual slippability has been temporarily omitted.

The Conceptual Memory has much structure built into it. The simplest sorts of concepts, roles, are represented as collections of norms. Each norm has a certain importance to the role, some norms being more important than others. Norms define acceptable bounds on very simple physical features associated with parts (*e.g.*, height, width, curviness, and so on). Together, sets of roles define a whole in the Conceptual

Memory. Wholes (sometimes called "role-sets" or "conceptualizations") also include r-roles, which monitor the interrelations of their constituent roles, requiring that roles be in proper relation to each other in order to form a coherent whole. As shown in Figure 4.2, each letter category is represented by a small collection of wholes in the current model ranging from one to four. Activation spreads from roles to wholes (and sometimes a bit the other way around) throughout the process of categorizing a letterform (or *seeing the letterform as* a particular whole). It is important to emphasize the dynamic nature of the Conceptual Memory. Activation values ebb and flow throughout a run, with activation coming from parts that activate roles to a greater or lesser extent, activation spreading down links, activation decaying over time, and so on.

4.3 The role model's implementation

We now move more deeply into the implementation details of the model, beginning with a discussion of the codelets used in the role model.

In the role model, just as in Copycat and Tabletop, it is essential to distinguish between *instantiated* codelets and codelet *types*. There are 16 codelet types — prespecified, rather small pieces of Scheme code (on the order of tens of lines each). During a run, these pieces of code (sometimes with arguments filled in) are instantiated many times over as specific, runnable codelets. Each codelet-instance is placed on the Coderack, where it may or may not be chosen to run, depending upon its associated urgency. Codelets compete with each other to run, inasmuch as they are picked in a probabilistic way according to their urgency values. A codelet's urgency is determined at the codelet's instantiation time according to the state of processing in the entire system, and reflects a combination of factors that take into account not only the kind of codelet that is being instantiated but also the overall stage of processing that the recognition process has reached.

The sixteen types of codelets do every stitch of the processing in the role model, from the mundane initial gluing-together of parts to the ultimate picking of the winning whole.[3] The types of jobs codelets do include: part building, part inspection, part reformulation, part labeling, role activation, activation spreading, role subsumption, and r-role checking. Because processing in the role model is broken into a series of small steps, a parallel terraced scan of possible routes to a solution can be carried out. An average run takes around 750 codelets for a typical gridletter, and up to 5000 for an especially hard one. Occasionally, no answer is reached, and codelet processing must be abandoned. Each codelet type is instantiated many times over during a run, thus reducing the processing to fine enough grains that the parallel terraced scan works quite effectively.[4]

4.3.1 The codelets

The sixteen types of codelet in the role model come in two flavors. Both are implemented as functions that are "frozen" (completely ready to run) and then inserted into a list (the Coderack). Some codelets are frozen with particular arguments. Others have no arguments frozen in. If a codelet is picked to run, it is plucked from the Coderack and "thawed" (*i.e.*, evaluated). Sometimes a picked codelet will not be able to do its job — for example, consider a codelet that is supposed to attach a label to a part that no longer exists. If that happens, the stranded codelet simply "fizzles".

The sixteen codelet types are briefly sketched below. Codelets are listed in the approximate order in which they first appear during a run. Some of the codelets (*e.g.*,

[3]Every stitch, that is, except for any guessing that may occur should role-processing fail. We do not count such guessing as a true part of the role model. See Section 4.3.12 below.

[4]In order to finely interleave large-scale processes made up of many codelets' worth of activity, the parallel terraced scan requires that such processes be broken into very small pieces. The finer the grain of these pieces, the finer the allocation and adjustment of computational resources can be.

the gestalt-checker and the role-subsumption-looker) are more complex and have less obvious motivation than others. These complex codelets are given a more thorough treatment in separate sections below.

Often, codelets post other codelets as followups during processing. The potential posting of followups has been noted in the verbal sketches below and also in Figure 4.8. The role model makes use of four "default" urgency values that are coded as parameters. Codelets are usually posted using slightly adjusted versions of these default urgency values (the direction and magnitude of the adjustments depending upon particular circumstances). The usual settings for these parameters are: low urgency, 10; medium urgency, 25; high urgency, 50; and very high urgency, 100. An experiment in which these parameters are systematically altered is reported at the end of the next chapter. If the reader desires more specific information regarding codelet implementation, the complete source code for the role model is available by **ftp** (see Appendix B).

The glue-ant codelet (no arguments)

A glue ant picks a quantum (at random) from a letterform and globs a dab of glue at a junction where the picked quantum meets a neighboring quantum. As shown below, the greater the angle between the two quanta picked, the more glue gets put down.

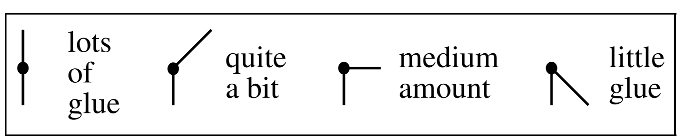


Figure 4.6: Different amounts of glue are placed between pairs of quanta having different angles as shown.

After this has been done hundreds of times, the completely glued-up shape is broken probabilistically (by the shaker codelet, see below) at joints with the smallest amounts of glue. The end result is that the shape is broken, usually at junctions with small angles, into parts that have a tendency to be chunked into "straight" lines.

The shaker codelet (no arguments)

This codelet shakes a glued-up shape — made up of several quanta glued together with varying amounts of glue — into initial parts. The number of quanta in the entire letter determine how vigorously the letterform is shaken, and hence how many parts result. Junctions where quanta meet break probabilistically depending on the amount of glue placed on them. Together, the gluing and shaking processes determine the initial (syntactic) parts that a letterform is broken into prior to labeling and role-matching.[5]

Every part created by the shaker is given a "whine value" label (initially the same value for each part). A part's whine value increases every time it is considered by a looker codelet (see below) but fails to "pass" inspection. Each part's particular whine value helps to determine whether or not that part will be picked by a pacifier codelet (also below) during processing.

The looker codelet (no arguments)

Once parts exist in the Visual Focus, this codelet picks a part at random and decides whether or not it is worth labeling. Parts must pass two tests before the looker codelet will approve them: 1) they must have either zero or two tips, and 2) they must consist of at least two quanta (this is tested probabilistically). (One-quantum parts will pass only if they are not touching any other quanta.) The at-most-two-tips test (condition 1) is a weakness in the code, sometimes causing the role model to handle "spiky" letterforms poorly. If a part fails condition 1, a breaker codelet (see below) with high urgency is posted.

[5]Experiments with initial part determination without gluing and shaking are reported in Section 5.6 of the next chapter.

Once a part has passed inspection, several labeler codelets bound specifically to the passing part are posted to the Coderack. Each "bound" labeler codelet is posted with medium urgency. If a part fails to pass, its whine value is increased and a pacifier codelet is posted with an urgency reflecting the part's whine value, meaning that whinier things engender greater urgencies, one way of implementing the philosophy "the squeaky wheel gets the oil".

The pacifier codelet (no arguments)

The pacifier codelet probabilistically picks a part in the Visual Focus, biased towards whiny ones, and attempts to figure out why the chosen part is whining so that it may do something about it. If the part that it picks passes inspection, the pacifier posts several bound labelers with medium urgency (just as if it were a looker codelet). Otherwise it attempts to glom together the whiny part and its smallest neighbor (giving probabilistic preference to small and whiny neighbors over small but happy ones). If a new part is formed, three more looker codelets are posted, also with medium urgency.

The labeler codelet (argument: a part)

The labeler codelet could be thought of as a group of several different individual codelets, one for each possible type of syntactic label that can be attached to a part. Implemented instead as one codelet, the labeler attempts to compute and attach one of several different low-level-feature labels (such as "wide", "tall", or "skinny") to its part. It picks among the label types probabilistically (see Section 4.3.2 for a discussion of labels). If the label that it chooses has already been attached to the part, the labeler fizzles. Otherwise it attaches the given label to the part. If the part has enough labels attached (determined probabilistically), a label-checker codelet (see below) of medium urgency gets posted to the Coderack.

The label-checker codelet (no arguments)

The label checker picks a random part (labeled or not) from the Visual Focus and decides whether it should post an activation-sparker (see below). If so, an activation-sparker is posted with an urgency based on the number of labels the picked part has. The *raison d'être* of the label-checker is further discussed in Section 4.3.7 below.

The activation-sparker codelet (argument: a part)

An activation-sparker codelet is posted by a label-checker that has determined that some particular part has enough labels that it should be used to try to activate roles. The activation-sparker's primary responsibility is to take a well-labeled part and attempt to activate roles with it. The activation-sparker also has a second important overall function — it is responsible for the reformulation (under top-down pressure from the roles) of any poorly formed parts.

Based on whether or not roles are activated, the codelet follows either of two possible paths. Because each of these two branches is complicated by further tests, a flowchart for the activation-sparker is shown in Figure 4.7.

If no roles are activated by the part, a further test is made to determine how many labels the part has. If the part has a large enough number of labels attached (determined probabilistically) then it is slated for breaking. In this case, a breaker codelet is posted with high urgency so that the part will be broken "soon". If the part that activates roles poorly has only a few labels, more bound labeler codelets of medium urgency are posted in lieu of a breaker.

If the part does activate roles, then depending upon *how highly* it does so, two different things may happen. If some role is very highly activated by the part ("very highly" being a probabilistic threshold), then the activation-sparker's job is considered finished, and an activation-spreader codelet with high urgency is posted. If the part

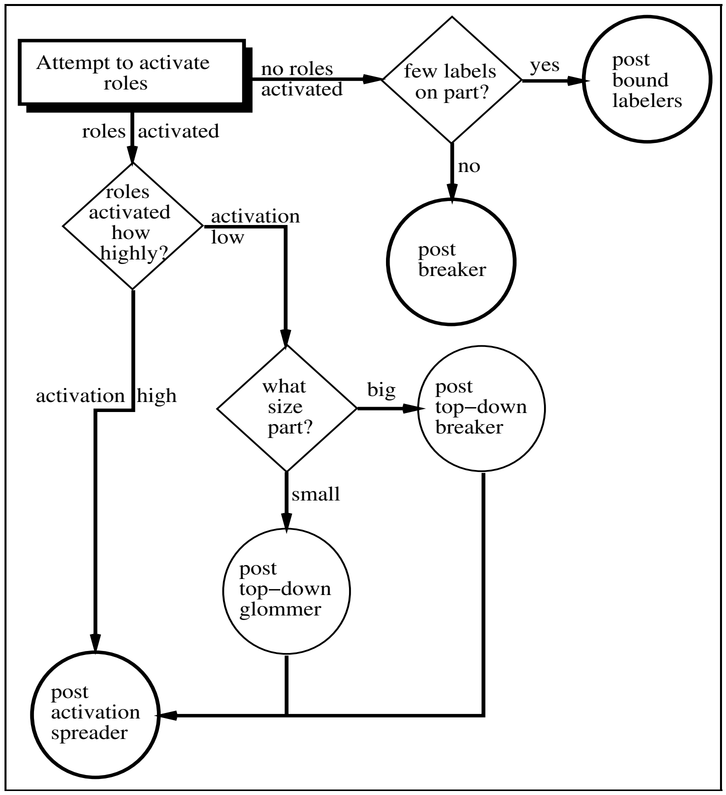


Figure 4.7: Although the activation-sparker codelet's main job is to try to activate roles with a labeled part, it also does a few other things depending upon *how well* the part activates roles. A flowchart for the codelet is displayed above. Tests other than the primary role-activation test are drawn in diamonds. Codelets that the activation-sparker may post as a result of particular tests are shown in circles.

activates roles but not very highly, it is subjected to a probabilistic test, which, depending on how highly it activates roles, may decide either to break it (if it is big) or to glom it to something else (if it is small). This test results in the posting of a top-down glommer or a top-down breaker in case the part does not activate any roles highly enough (that is, if the part is a weak role-filler across the board). This ensures that (at least probabilistically speaking) poor parts will tend to be reformed under pressure from the roles. The top-down breakers and glommers are posted with an urgency value reflecting how poorly the part activates roles. A very poor part will thus be dealt with relatively sooner than an "ok" part. After a top-down breaker or glommer has been posted, an activation-spreader codelet is also posted with high urgency (after all, low activation is still activation).

As we mentioned above, a flowchart for the activation-sparker has been provided in Figure 4.7. It is important to keep in mind that though the activation-sparker does post a variety of different followup codelets, its main job is to activate roles, given a fully-labeled part.

The activation-spreader codelet (no arguments)

This codelet simply spreads activation in the Conceptual Memory when it is chosen to run. Top-down pressure from roles is a key component in the role model's categorization process. Instead of spreading activation every set number of codelets run, activation-spreading is controlled by the activation-spreader codelet. This ensures that top-down pressures are kept under control in the system and not allowed to take over processing completely. More concretely, when a role becomes active, its effect on further processing (through activation spreading) must compete with the other processes taking place in parallel. This is because when activation spreads and wholes become active, the role model's processing is deeply affected (*i.e.*, many new codelets come into being and compete to run).[6] Spreading activation on a rigid schedule with no regard to processing context would not be beneficial. Use of a codelet to control activation spreading provides a logical alternative.

The gestalt-checker codelet (no arguments)

The gestalt-checker serves to prime the Conceptual Memory by lightly activating role-sets according to gestalt properties of the letterform being recognized. The gestalt-checker implements a kind bypassing of normal processing through parts and roles,

[6] A more detailed treatment of activation flow in the role model can be found in Section 4.3.7 below.

since activation goes directly from mere *quanta* directly to *wholes*. Eight gestalt-checkers are posted automatically once a letterform is shaken into initial parts by the shaker codelet. Gestalt-checkers are posted with low urgency, as their job is not highly critical.[7] If chosen from the Coderack, the gestalt-checker probabilistically picks one of four gestalt types (ascender, descender, closure, and no-tips) and checks to see if the letterform includes the property it has picked. If so, the gestalt-codelet lightly activates any role-sets having the property in question. It also slightly inhibits those role-sets having the "opposite" property if the property under consideration is "ascender" or "descender". The idea is to prime the Conceptual Memory in order that certain silly hypotheses can be avoided (for example, hypothesizing 'g' when "ascender" has been noticed). More on gestalt activation can be found in Section 4.3.9 below.

The breaker codelet (argument: a part)

This codelet breaks a part up into two or more mutually-exclusive subparts. It then posts several looker codelets having medium urgencies to start processing the resulting fragments.

The glom-all codelet (no arguments)

This codelet is fairly specialized and can be posted only by a successful no-tips gestalt checker. If the no-tips gestalt checker notices that a letterform has no tips, it posts this codelet, which then gloms all of the quanta together into one big part. This is useful in bringing about quick recognition of standard 'o's. The glom-all codelet, if it forms a new part, posts several medium-urgency lookers as followups.

[7]Several experiments in which gestalt checking was completely disabled are reported in Section 5.6 of the next chapter. These experiments show that priming the Conceptual Memory according to the gestalt properties of a letterform does not deeply affect performance of the role model.

The top-down glommer codelet (argument: a part)

Posted only if a well-labeled part poorly activates all roles across the board, this codelet attempts to glom the offending part to its weakest neighbor (picked probabilistically). If a new part is formed, followup looker codelets with medium urgency are posted.

The top-down breaker codelet (argument: a part)

This codelet is similar to the top-down glommer, but breaks a part up instead of trying to glom it to another part. Together, the top-down breaker and top-down glommer codelets provide a way in which roles can trigger the rearrangement and regrouping of existing parts — giving the role model substantial top-down capabilities.

The role-subsumption-looker codelet (arguments: subsuming role, subsumed roles)

Whenever roles are activated by the activation-sparker, a very quick role-subsumption test is made. Some roles can, if taken together, be subsumed under another larger role. For example, the role *left bowl* subsumes the roles *left buttress* and *basebar* since, taken together, the parts that fill the latter two roles can sometimes form a bigger part that fills the subsuming role. We call this process *role subsumption.* If a quick role-subsumption test determines that the ingredients for a possible subsuming-role exist, then a role-subsumption-looker is posted. This codelet investigates a possible subsumption more thoroughly. If the subsumption seems plausible, a new part is formed and several looker codelets are posted. These lookers have a very high urgency value, as it is important that they do their job quickly. A more thorough treatment of role subsumption, including the reasons that role subsumption was developed, can be found in Section 4.3.11 below.

The r-role–checker codelet (argument: a role-set)

This codelet is posted whenever a role-set becomes highly enough activated (through spreading activation from roles). It chugs down the activated whole's r-role list, determining to what extent each r-role criterion is met. If an r-role test fails, activation is taken away from the role-set being inspected. All of the r-role tests are explained below in Section 4.3.6. In some cases, a particular r-role test may result in some high-urgency labelers being posted. This happens when a part is missing certain labels that are required for an r-role test to be completed. If this occurs, r-role testing halts and a large amount of activation is taken from the whole in question.

R-role checking provides the normal way in which categorization ends in the role model — or nearly so. When there are several competing wholes activated at once, r-role checking serves to discount most of the hypothesized wholes (one at a time through activation subtraction), hopefully leaving only one whole with a substantial amount of activation (that is, a whole whose r-role criteria are reasonably matched). Temperature in the role model drops considerably when only one whole is active (see Section 4.3.8 below). When temperature becomes low enough (a probabilistically determined event), processing ends and an answer is returned. Thus, in most runs, a winning whole emerges immediately after an r-role test leaving only one highly active whole has been completed . (More on how processing ends in the role model can be found below.)

The special-solver codelet (argument: a role-set)

If a role-set passes its r-role tests with flying colors (determined probabilistically, of course), a high-urgency special-solver codelet is posted. This codelet brings the recognition process to a quick halt, bypassing the usual route to a solution through a longer process of activation and r-role testing involving several competing wholes.

The special-solver codelet provides the role model with a way of ending processing *unexpectedly early* in case a letterform is exceptionally and incontrovertibly strong.

The special-solver works as follows. First, it runs an r-role test for each active role-set. This serves to purge competing-wholes' activation from the Conceptual Memory. Next, the special-solver double-checks that the role-set predicted as the winner based on its superior r-role matching is indeed still the most highly activated whole. If it is, a solution has been found and the winning whole is returned as an answer. All this happens only rarely — namely, in the case of very standard letters that match r-role criteria extremely well. The more usual route to ending processing is through a lowering of temperature (as noted above).

Codelets initially posted when a run begins

At the beginning of a run, many glue-ant codelets swarm the gridletter, glomming its quanta together. Once a certain amount of gluing has taken place, a shaker codelet is run and the glommed-up shape is shaken into parts. It is at this point that the parallel terraced scan begins in earnest, all that has gone before being somewhat preliminary.

Once initial parts have been determined, two medium-urgency looker codelets are posted to the Coderack for each initial part. Also posted to the Coderack are eight low-urgency gestalt codelets. In the beginning, since little structure has been built in the Visual Focus, no information is available for determining urgency values, and so non-adjusted default values are used. Lookers are posted with an urgency value twice that of gestalt codelets.

It is up to the lookers to get the ball rolling. Once they have begun processing the initial parts, many hundreds of other codelets are posted as followups (and followups of followups, etc.) throughout the run. Figure 4.8 shows how codelets are related to one another in terms of followup links. This chart gives a general feel for flow of control in the model. Of course, followups are not guaranteed to be posted in every

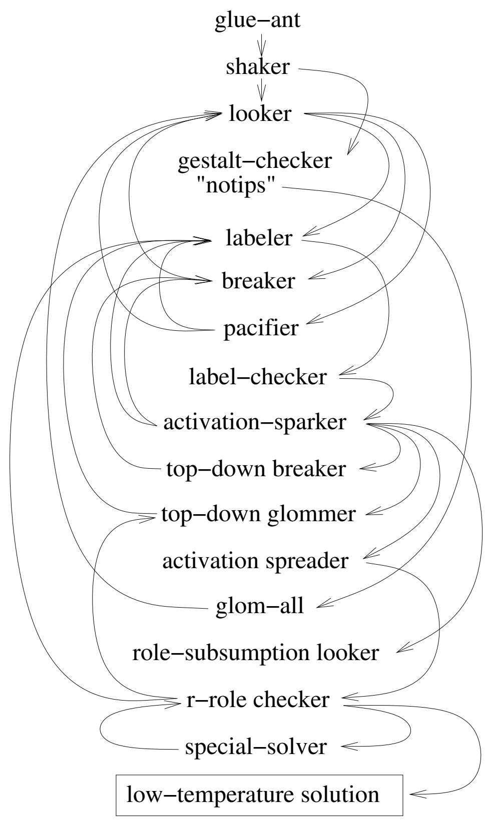


Figure 4.8: How the sixteen codelet types are interrelated in terms of flow of control in the program. Codelets can post other kinds of codelets to the Coderack as followups. Followup relations are represented by the arrows. (Of course, many possible followup situations never arise.) A categorization emerges as codelet processing proceeds.

situation, and even followups that are posted may never get picked to run.

How processing ends in the role model

When temperature in the role model drops below a threshold (probabilistically determined, but ranging around ten degrees), processing ends, and a winning whole is returned.[8] A temperature-based end is the usual way in which categorization is brought to a close. This sort of stopping occurs as competition between activation

[8]For more about computing temperature and the way temperature affects processing see Section 4.3.8.

and inhibition in the role model results in a single winning whole. Role activation and activation-spreading "compete" in some sense with r-role checking, resulting in the emergence of a fairly clear winning whole.[9] Since not just one but a number of wholes are usually activated when activation spreads, it takes several sets of r-role tests (each set carried out by a separate codelet) to determine a winning whole.

At the end of a run, a letterform has been successfully parsed into parts that fill a whole's conceptual roles. The winning conceptualization's label, as well as information regarding the number of codelets run, the temperature at the end of the run, and the final activation value of the winning whole, is returned. Information about the letterform's constituent parts, including the ways in which they both match up with and violate role-level norms, is available for further processing. Also available is information regarding r-role matching.

As was stated earlier, another way that processing may end in the role model is through the actions of a special-solver codelet. This codelet provides a way in which a letterform that is an especially strong category member may bypass the normal activation/inhibition "fight" in order to be more quickly categorized. After a successful special-solver codelet is finished, the same sorts of category and parsing information as in a normal temperature-based ending (noted above) are available for further processing.

The third way in which role-model processing may end is not as satisfying. If no whole emerges as a clear winner after several thousand codelets (currently set to 6,000), processing is discontinued. At this point a "guessing" mechanism may be invoked. This rarely happens in the role model when runs are made on fairly standard letterforms. As letterforms become stranger and more affected by style, however, guessing is invoked more often. Section 4.3.12, below, discusses guessing in

[9] This competition between activation and r-role checking is more thoroughly described in Section 4.3.7.

more detail.

4.3.2 Low-level feature labels

Each part, once created, starts accumulating various low-level feature labels, for the purpose of activating appropriate roles. Such labeling is done by labeler codelets. As was discussed above, a labeler codelet could actually be thought of as a group of different labeling codelets, one for each type of label. Although there is only one type of labeler codelet, it may attach different labels to parts depending upon a random decision made at run time.

There are seven classes of low-level feature that can give rise to labels. These classes are: height–width, curviness, weight, horizontalness, verticalness, lo-res-view ("low resolution", that is), and tips. Each of these classes in turn contains several possible labels. Figure 4.9 shows all 53 possible labels grouped under their respective classes. The classes are fairly self-explanatory (as are most of the labels). Some of the less obvious labels include: *horizontal, vertical, curvy*, the lo-res-view labels, and the tips labels. These will be briefly explained.

height–width	curviness	weight	horizontalness	verticalness	lo–res view	tips
no_height	closure	light	left	ascender	square1	points
very_short	straight	normal_weight	middle	x–zone	⋮	ttips
short	concave	heavy	right	descender	square12	rh–tip
medium	bow	huge		on_baseline		lh–tip
tall	zig–zag			on_xheight		uv–tip
very_tall	curvy			top		dv–tip
skinny				bottom		uf–tip
half_wide						df–tip
wide						ub–tip
vertical						db–tip
horizontal						

Figure 4.9: 53 low-level-feature labels make up the entire set of possible labels that can be attached to parts. They can be divided into seven classes, as shown. Most parts eventually acquire about 20 such labels. To each label is attached a real number between 0 and 1, showing how strongly it applies.

The *horizontal* and *vertical* labels (found under the height–width class) are attached with a strength corresponding to the ratio of height to width in a part. For example, a very tall part with width zero would be an ideal candidate for a strong "vertical" label.

The *curvy* label is attached according to the average angle value (computed by adding all the angles between quanta and dividing by the total number of angles) and is attached with a higher strength the greater the value is.

The lo-res-view labels are always attached with a strength of 1. They provide a low-resolution view of the gridletter in terms of elementary squares on the grid. The grid divides naturally into 12 one-quantum-by-one-quantum squares. Each of these squares is given a number, and if any quanta in the gridletter occupy the square, the label has a chance of being attached.

The tips labels are the most complex. A tip is a quantum at the "end" of a part. First off, there are four main types of tip, depending on the angle of the quantum involved: horizontal, vertical, forward-slash, or back-slash. Each of these types is further divided in terms of whether the tip points up, down, left, or right. The eight resulting classes of tip are: rh (right-horizontal), lh, dv (down-vertical), uv, df (down-forward-slash), uf, db (down-back-slash) and ub. Figure 4.10 shows all 8 classes. The tips labels are not attached with a number denoting their strength since their existence is clearly Boolean. Instead, each tip label carries with it an actual quantum number from the grid. This makes the tip information much more localized to the grid than the other labels are. Labels denoting the total number of tips (*ttips*) and the points at which tips end on the grid (*points*) also have a chance of being attached.

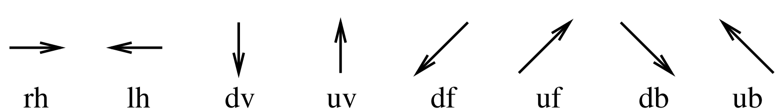


Figure 4.10: The 8 classes of tip that are possible on the grid.

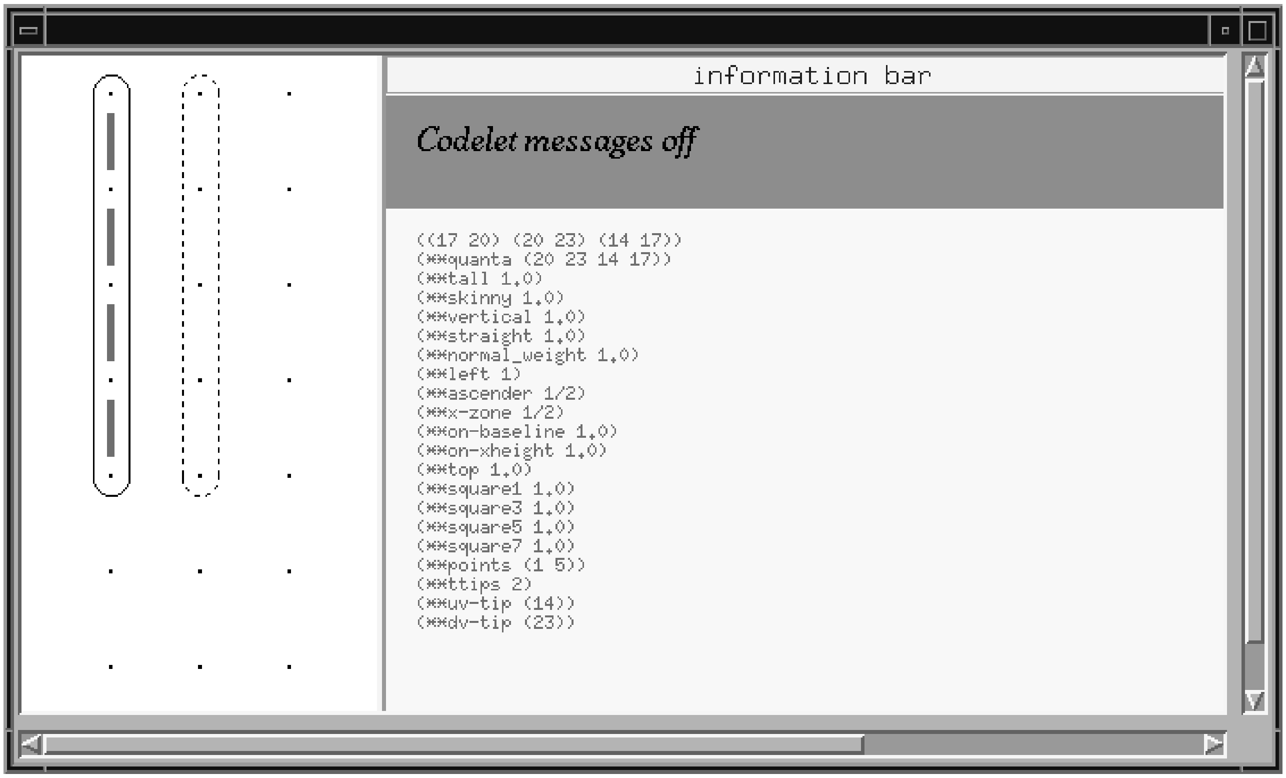


Figure 4.11: A fully-labeled part (this one a particularly strong instance of the *left post* role.)

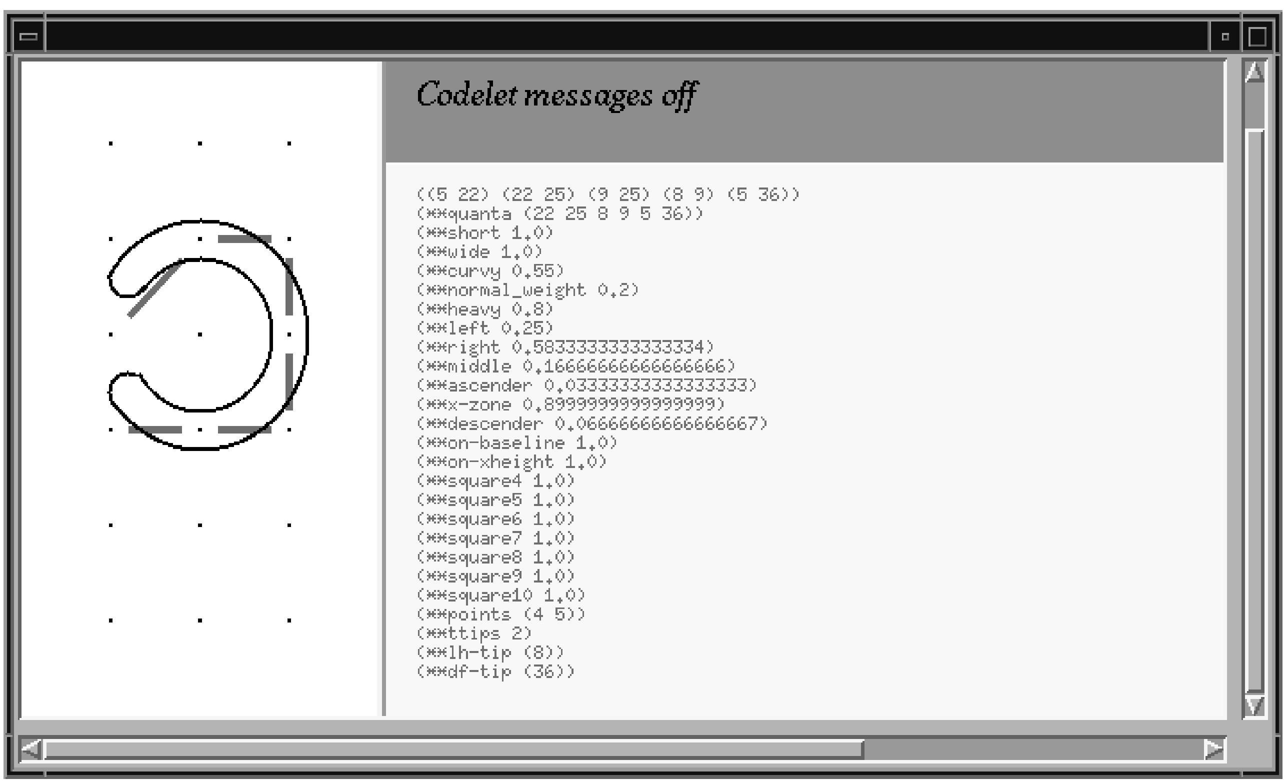


Figure 4.12: Another fully-labeled part (this one a particularly strong instance of the *left bowl* role.)

Most parts get from 15 to 20 labels attached during a typical run. A label is attached to a part by the labeler codelet with a real number denoting how well it describes the part. Numbers range from 0 to 1 (although labels with a 0 are not attached, for obvious reasons). The higher the number, the better the match.[10]

All labels and label values are very simple to compute. This is a rather crucial aspect of labels, since whenever a part is destroyed all of its labels become irrelevant. The idea is that, because labels are computationally inexpensive, throwing them away is not a terrible waste of effort, since new ones can quickly and easily be calculated.

Two different fully-labeled parts are shown in Figures 4.11 and 4.12. Remember that since labels are attached probabilistically, these very same parts could well be labeled differently on each run.

4.3.3 Roles

The role model contains 47 roles that, combined into various sets, make up its 66 wholes. Each of these roles is implemented as an object containing a list of norms defining its general characteristics (see the next section), and several slots for keeping track of activation, of part-binding, and of other administrative details. Figure 4.13 shows a graphical interpretation of each role in order to give some idea about what sorts of shapes each role's norms tend to define.

A role may of course be a member of more than one role-set. For example, the *right-post* role can be found in both the **d** role-set and the **d3** role-set. Table 4.1 shows how the 47 roles help to define the 66 role-sets. The most popular roles, in terms of how many role-sets any particular role appears in, are *circle*, *right-bowl*, and *f-slash* (forward slash).

As just stated, each role has a list of norms, and each norm in the list has an

[10]Equations for calculating label strength will not be reported. Interested readers are referred to the source code (see Appendix B for retrieval instructions).

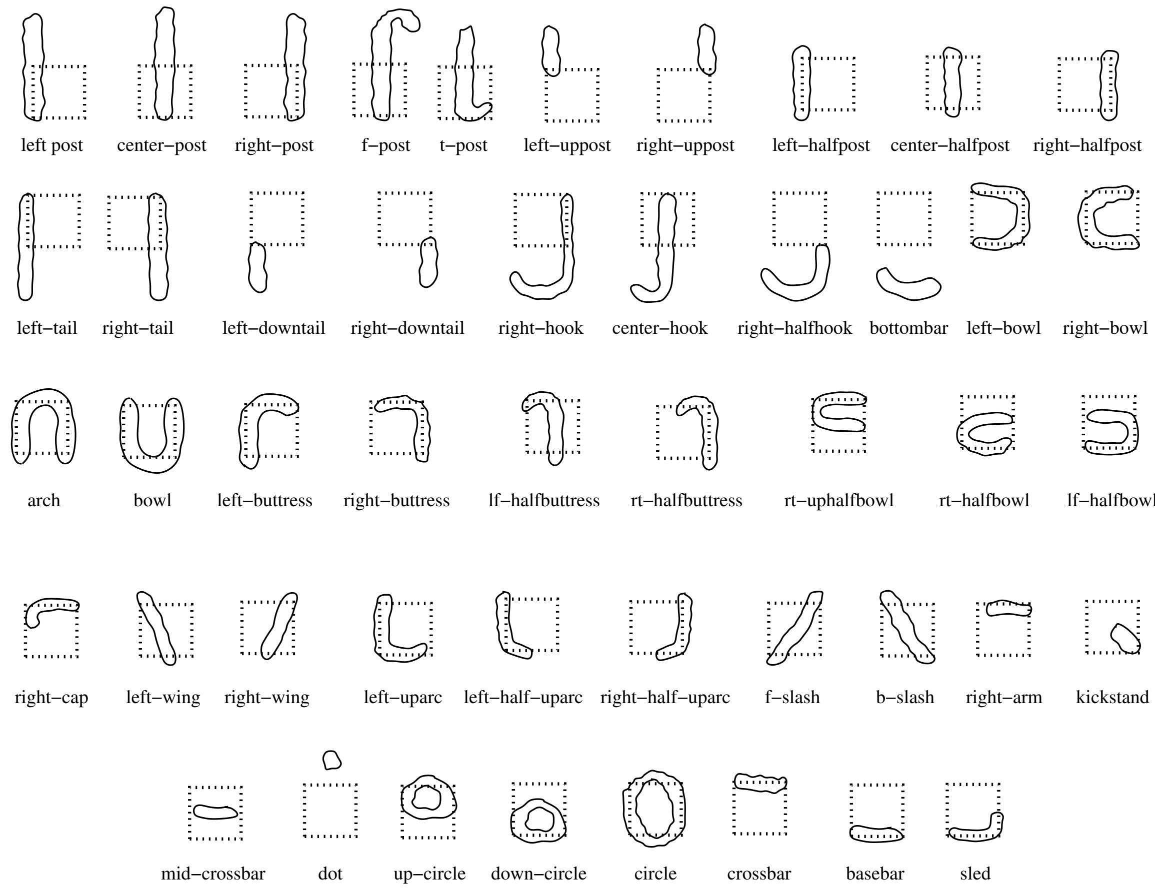


Figure 4.13: A chart showing all 47 roles in the role model. The dashed box shows where the central zone occurs relative to the role. The roles in the actual model are represented not as shapes, but as collections of norms. They are shown as globby shapes here solely for convenience.

associated value, which affectively ranks the norm in terms of its importance and relevance to the role. The norm-values were set by hand, based on analysis of system performance over thousands of runs. The exact values do not make much of a difference to system performance, as long as relative ranking and general norm-value ranges are preserved.

Figure 4.14 shows how one conceptualization of 'b' is represented in the Conceptual Memory. This particular role-set, **b**, is made up of two roles, *left-post* and *left-bowl.*

role	role-sets	role	role-sets
left-post	**b b3 h k k2 k3 k4**	right-uphalfbowl	**s4**
center-post	**l**	right-halfbowl	**a**
right-post	**d d3**	left-halfbowl	**s**
f-post	**f**	right-cap	**k4 s s3 z2 z3**
t-post	**t**	left-uparc	**e4 u y y2**
left-uppost	**b2 h2**	left-half-uparc	**w w3**
right-uppost	**d2**	right-half-uparc	**w w3**
left-halfpost	**m n r v2**	f-slash	**e3 k2 v2 x z z2 z3 z4**
center-halfpost	**i m2 w w2**	b-slash	**k3 s2 v3 x**
right-halfpost	**u v**	left-wing	**v**
left-tail	**p p3**	right-wing	**v**
right-tail	**g3 q q3 y2**	right-arm	**a2 r**
left-downtail	**p2**	kickstand	**k k2**
right-downtail	**q2**	mid-crossbar	**e k k4 s3**
right-hook	**g g4 j2 y**	crossbar	**f k3 s2 t z z4**
center-hook	**j**	basebar	**e2 s2 z z2**
right-halfhook	**g2 y3**	sled	**s3 s4 z3 z4**
left-bowl	**b p**	bottombar	**g3 y2**
right-bowl	**c d e a4 e3 g g3 q**	dot	**i j j2**
arch	**h2 m2 m3 n2**	up-circle	**e2 e4**
bowl	**u2 w2 w4 y3**	down-circle	**a2 a3**
left-buttress	**a a3 a4 h n**	circle	**b2 b3 d2 d3 g2 g4 o p2 p3 q3**
right-buttress	**r2**		
left-halfbuttress	**m**		
right-halfbuttress	**m**		

Table 4.1: How the 47 roles fit into the 66 role-sets.

In addition, r-roles specify how the two roles should interact.

Space constraints do not allow us to list the norms for each role in the Conceptual Memory (or to show all role-sets in the manner of Figure 4.14). For more information, the reader may refer to the source code.

Roles receive activation from well-labeled parts. Such parts have labels that match up with a role's norms. A role's activation value is stored in its activation slot, along with a pointer to the part that is currently associated with the role (and caused the activation). If another part comes along that "out-matches" a role's currently associated part (and thus raises the role's activation), the filler of the latter slot is

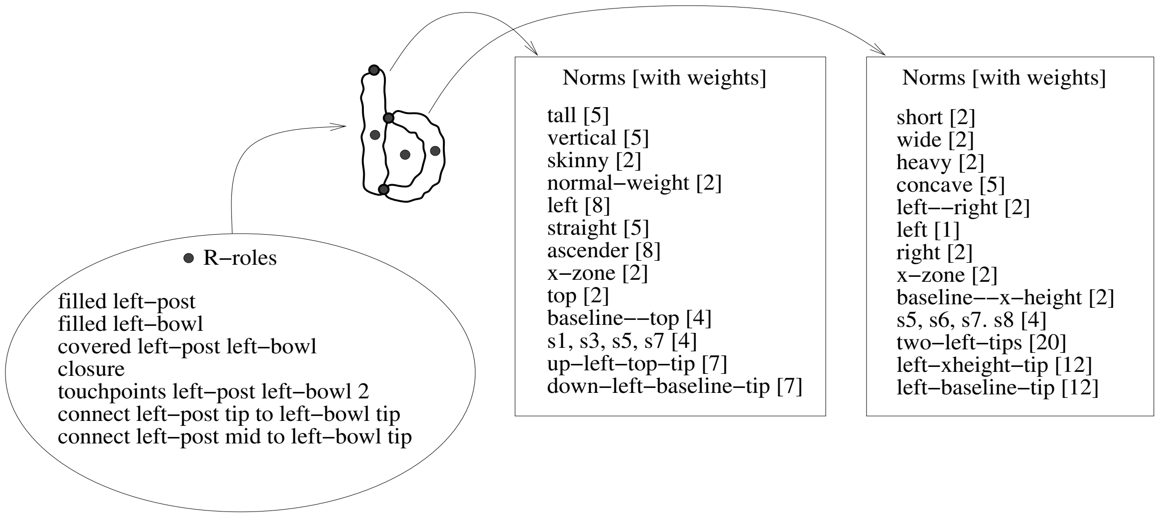


Figure 4.14: One conceptualization of 'b' is made up of two roles and seven r-roles specifying their interrelations. The norms for each role are shown above, as is a list of all seven r-roles.

changed to the new part. A role can be associated with only one part at a time. Role activation values are presented dynamically throughout a run in a window that looks like Figure 4.15. Role activation values range from a low of 0 to a high of 100.

4.3.4 The norms making up roles

The norms attached to each role serve to define how feature labels attached to parts work together to activate the given role. In this way, norms are closely related to the low-level feature labels covered in Section 4.3.2, above. A norm is given a "relevance number" that specifies how critical the norm is in the definition of a role. The *exact* numbers coded into our role model are not super-sensitive to adjustment, nor are particular values of critical importance. The order relation between the numbers, however, is. The reason for this is that some norms may be stretched further than others, and the amount of plasticity or stretchability allowed for a particular norm — or put another way, that norm's importance to the role — is defined relative to other norms in terms of relevance numbers. More important norms have higher numbers, less important norms, lower. A role's degree of plasticity is directly related to its norms and their relevance numbers. The relationship between norms and stretchability is well-founded and has been extensively researched in various psychological domains [Tversky and Hutchinson, 1986; Kahneman and Miller, 1986; Fauconnier, 1985; Norman, 1981; Tversky, 1977].

Relevance numbers help to determine the activation of a role by a particular labeled part. A part's labels match up with a role's norms in a straightforward way. Each label provides a certain amount of activation to a role depending on both the strength of the label's attachment to a part and the relevance number of the label's corresponding norm in the role in question.

tall, medium, short, very-short, no-height
vertical, horizontal
skinny, half-wide, wide
light, normal-weight, heavy
left, middle, right, left–right
straight, bow, concave, curvy, closure
top, ascender, x-zone, descender, bottom
baseline–top, baseline–t-height, baseline–x-height, baseline–midline, baseline–bottom midline–x-height, x-height–top, x-height–bottom
s1, s2, s3, s4, s5, s6, s7, s8, s9, s10, s11, s12

Table 4.2: Roles are implemented as collections of these norms.

In general, roles can be broken down into two main classes — regular norms, and norms having to do with tips. The regular norms are listed in Table 4.2. They have been divided into obvious categories based upon the particular geometric dimension that they apply to. As can be seen, norms define things like height, width, curviness, number of quanta, lo-res view, and zone of occurrence on the grid. Role-defining norms and part-characterizing labels have a clear and obvious relationship.

Tip norms, shown in Table 4.3, define desirable locations (and desirable directions) for tips that belong to roles. Since roles are meant as conceptual abstractions, their norms should certainly not be tied strictly to precise locations or directions on the grid. This is especially true of tip norms. A norm like *up-right-top-tip*, for instance, can be matched to a greater or lesser extent by various different physical tips on the grid. As is obvious from this norm's name, tips that match it well will generally point up (or close to up), and will generally be located at or near the upper right corner of the grid.

A tip set is a collection of tips that should occur together in one role. Like tip norms, tip sets are named in a logical way. For example, the tip set *right-down-tips* is a set of norms that, taken together, specify the desire that one tip should point generally down while the other should point generally right-ish. How the degree of matching between actual tips on the grid and tip sets and tip norms is calculated will not be presented in greater detail here. More information is available in the source code.

Tip norms		Tip sets
up-right-top-tip	right-xheight-tip	two-right-tips
up-right-xheight-tip	right-midline-tip	two-down-tips
up-right-baseline-tip	right-up-midline-tip	two-up-tips
up-center-top-tip	right-baseline-tip	two-left-tips
up-center-xheight-tip	right-bottom-tip	right-down-tips
up-center-midline-tip	right-center-baseline-tip	left-down-tips
up-left-top-tip	left-xheight-tip	right-up-tips
up-left-xheight-tip	left-midline-tip	left-up-tips
up-left-baseline-tip	left-baseline-tip	left-right-tips
down-right-f-tip	left-bottom-tip	notips
down-right-xheight-tip	left-center-xheight-tip	
down-right-baseline-tip	left-center-baseline-tip	
down-right-bottom-tip	left-slash-baseline-tip	
down-center-baseline-tip	right-slash-xheight-tip	
down-left-xheight-tip	left-bslash-xheight-tip	
down-left-midline-tip	right-bslash-baseline-tip	
down-left-baseline-tip	left-bslash-midline-tip	
down-left-bottom-tip		

Table 4.3: Roles also include these tip-norms and abstract tip sets.

Having norms defined at a reasonable level of abstraction (as our norms are) is important. Although the code that matches specific grid-bound shapes to these norms is very grid-specific, it is conceivable that an entirely different low-level front-end could be connected to the rest of our role-level model. This might allow a pixel-based recognizer to make use of role-level recognition as it is currently implemented. All that would need to be done is to specify perceptual pathways by which low-level parts get formed from lower-level constituents, and then labeled with our specified feature labels. Beyond the level of part formation and labeling, the rest of the role model could be applied as is.

4.3.5 Wholes (or role-sets)

The role model has 66 role-sets or wholes (shown in Figure 4.2). Each role-set consists of between one and three roles, related to each other by r-roles. The details of one particular role-set for 'b' were already shown in Figure 4.14.

Role-sets are activated from "below" by roles, which spread activation upwards along links to the level of wholes. When a whole becomes sufficiently active, an r-role–checker codelet is posted to the Coderack. Once this codelet runs, the r-roles for the associated whole are checked, and activation may be taken away from the whole for any r-role that is not satisfied. Particular r-role tests, explained in the next section, are probabilistic in nature.

Each letter category may be modeled by various distinct role-sets (as shown in Figure 4.2). (In the model as so far implemented, there are never more than four role-sets, but in principle there is no limit.) Each role-set provides an independent way of "seeing" a letterform (*i.e.*, parsing a letterform into its constituent parts). The final product resulting from a run of the role model consists of a role-set filled by particular grid-bound parts. Beyond a mere category label, a filled role-set includes information regarding how a letterform has been parsed, which role-set it has been parsed into, how particular roles are filled by the letterform's parts, to what extent the filled roles are properly related to each other, and which norms have been violated by the letterform, and how. It is precisely this information that will be used in the letter-design process.

4.3.6 R-roles

R-roles specify the relationships that should hold between roles in a role-set. Once a whole becomes sufficiently active, its r-roles must be checked, which means that for each role-set, a set of tests must be administered. Any test that is not passed results in the subtraction of some activation from the role-set. Thus r-roles act as an inhibitory mechanism in the role model. R-roles are capable of subtracting so much activation from a hypothesized whole that the whole becomes negatively activated. This ensures that particularly stupid hypotheses are immediately ruled out and strongly inhibited, and thus cannot be made over and over again throughout a run.

Table 4.4 exhibits all of the possible r-role tests that may occur. Not all of these tests must be made for each role-set. In fact, only a handful for each role-set turn out to be relevant. A particular role-set's r-role slot gives a listing of which r-role tests should be run in case that role-set becomes sufficiently active.

Some r-roles are more important than others. The most important r-role tests inhibit a failing whole to a greater extent than do less important r-role tests. The amount of inhibition associated with each test is noted in the "whack factor" column of the table. To get the actual amount of activation to be subtracted, a given whack factor is multiplied by a certain constant called the "r-role whack parameter". This parameter is set to 10 in the current system.[11] Some tests are so important that if they are failed, not only is activation subtracted from the whole, but the entire r-role checking process immediately ceases as a "failure". These r-roles are shown with a

[11] Two experiments in which this parameter was changed are reported in Section 5.6 of the next chapter.

R-role test	Whack factor	Halt?	Note
filled	8	Y	
covered	8	Y	
no-v-angle	7	Y	u versus v
two-v-angles	5	Y	v versus w
no-left-tip	5	Y	c versus s/z
no-touch	5	Y	i/j dots
m-angle	5	Y	**m3** versus **n2**
no-baseline-tip	5	Y	s versus r
no-tail	5	Y	o
total tips	4	Y	especially k and v
closure	3		
open	1		especially g versus q
touchpoints	1		
dangle	2		especially r versus n
f-hook	3		f versus t
connect	1		
touch-baseline	3		r versus n
no-touch-baseline	1		
three-down-tips	4		n versus m
three-up-tips	4		u and v versus w
v-angle	5		
cross	3		x

Table 4.4: A complete listing of r-role tests. Each item in column one names a particular test that runs using information built-up in the Visual Focus. The whack-factor column shows how much activation (times ten points — the default whack) each test is capable of taking from a whole's activation, should the test fail.

"Y" entry under the "Halt?" column in Table 4.4. Wholes that fail an r-role test in this manner cannot be considered winners in the categorization process.

The *raison d'être* behind many r-roles is to test for specific inter-category distinctions. For example, the *no-v-angle* test serves to discriminate between 'u's and 'v's. If an r-role was motivated by such a specific purpose, this has been noted in the Table under the "Note" column.

4.3.7 Activation flow in the model

As a part gets more and more labels attached to it by labeler codelets, its chance of activating roles grows. Each time a label is attached to a part, the labeler codelet checks the total number of labels and posts a label-checker if a hard-and-fast threshold is passed. When it runs, the label-checker codelet picks a part to check *at random*, and thus does not necessarily pick the part that, because of its labels, caused the label-checker codelet to be posted in the first place. This kind of random picking ensures that part-level processing (especially labeling) generally proceeds at a similar rate for all parts in the Visual Focus. If the label-checker determines that the part it has picked has enough labels (this time deciding probabilistically), an activation-sparker codelet is posted. When a part passes label-checker inspection, an activation-sparker bound to the particular passing part is posted.

It is the activation-sparker's job to attempt to activate roles on the basis of a particular labeled part. Conceptually, this stage should consist in parts sending activation directly to roles, by virtue of the fact that each norm is for the most part directly related to some particular labels. This could be accomplished by having *every* role activated to a greater or lesser extent by a labeled part. The roles best matching a part would have much higher activations than others, and some roles would be only minimally activated by a part. We avoided this obvious approach in order to make activation in the system less messy. The fact is that labels are so general that *every single role* in the memory would be activated (most likely very weakly) by *any* given part. The thought of working in this inflated economy of activation was unappealing. Our solution (explained below) is equivalent to thresholding role activation and considering only those roles that are sufficiently strongly activated by a given part. Loosening and tightening allow us to raise and lower this threshold, as will be discussed below.

The "first pass" approach to role activation that we actually make use of is simply an implementation detail. It works as follows: As the activation-sparker codelet runs, the norms of each role in the Conceptual Memory are very quickly compared to the labeled part. This quick test is two-pronged. The first prong involves comparing the physical tips of the part with each role's abstract tip-norms, and calculating a first "score". The second prong involves considering a few key features and calculating a second score. (Specifically, these features include closure, height, width, and general placement information, making up 11 of the 45 possible non-tips labels). The two raw scores are combined into a single score for each role. Based on the values computed during this first pass (which pays a good deal of attention to tips), a handful of roles are chosen to be activated by the given part. Since a part can possibly fill more than one role (but not the other way around), it is important to consider all likely candidates in parallel and not constrain consideration to only one role per part. It is at this point that roles can be "loosened" or "tightened", as we will discuss below (see Section 4.3.10). The first pass thus results in a list of roles, the number of which can be anywhere between zero and some upper bound which is determined by a parameter called the "looseness threshold". These are the roles to be activated.

Once a list of roles to activate has been determined, the norms of each role are combined in a pairwise fashion with the values of the corresponding labels on the given part. (This process was discussed above.) This determines the amount of activation each role in the list receives from the part. The part activates different roles to greater or lesser extents depending upon how well its labels match each role's norms. Activation values range from 0 to 100.

Sometimes when certain roles become activated — specifically, those roles having strong "ascender" or "descender" components — certain wholes are automatically inhibited. Roles with ascenders, for example, inhibit wholes with descenders by subtracting 50 points of activation. They also slightly inhibit (a 10-point subtraction) wholes that have no ascenders. This is one way in which activation is taken out of the system.

Once roles become activated, activation-spreading codelets are posted on the Coderack. When one of these codelets runs, activation spreads along links in the Conceptual Memory (usually from roles to wholes). Links between roles and wholes reflect, through their length, the number of roles making up a role-set. Thus "one role" wholes are slightly closer to their single role than "two role" wholes are to theirs. This allows "one role" wholes to compete on an equal footing with "two role" wholes that may be getting activation from two places at once.

Activation-spreading from roles results in some role-sets (or wholes) becoming activated. At the same time that activation is spread, activation values also decay. When an activation-spreading codelet runs, from each active role, twenty activation points are subtracted, while from each active whole, ten are subtracted. These numbers were determined experimentally and reflect the intuition that wholes should hold their activation longer than roles do, since they are in some sense deeper conceptually.[12] Decay is a second way in which activation leaks out of the system.

There is another route by which activation can enter the system — namely, through the actions of a gestalt codelet. When a gestalt codelet determines that the property it is assigned to find (*e.g.*, "ascender") exists in the letterform, it lightly activates (a 25-point jolt) all roles having that property as a norm. It also lightly inhibits (a 10-point subtraction) roles having the opposite property (if such a property exists — for example, "descender"). Gestalt checking is discussed more thoroughly in Section 4.3.9 below.

There is one other important way in which activation is adjusted in the Conceptual Memory — namely, through r-role checking (as discussed above). When a

[12]This is similar to the notion of "conceptual depth" and activation decay in Copycat [Mitchell, 1993].

role-set becomes sufficiently active, its r-roles are checked by an r-role–checking codelet. Activation points may be subtracted from the whole according to how poorly it coheres with the r-role criteria.

Together, all of the mechanisms for adding and subtracting activation in the Conceptual Memory make for a very dynamic process.

Role activation ranges from 0 to 100. Whole activation ranges from -100 to 100. Role activation reflects how well a role is filled by a given part in the Visual Focus. Whole activation reflects both how well a letterform's constituent parts fill roles and how well inter-part relations match r-role criteria.

In order to get an idea of how activation spreads in the Conceptual Memory, it may be helpful to look at a couple of pictures. Figures 4.15 and 4.16 show two snapshots of the Conceptual Memory taken at different times. Both figures, which are screen dumps of the Conceptual Memory window of the program, show a box for every possible role and whole. Darkened squares inside these boxes indicate activation, with their size reflecting the amount of activation present. On the monitor, *negatively* activated wholes are shown with *red* activation squares, and *positively* activated wholes are shown with *black* activation squares. All of the activations shown in the two figures here are positive. The screen dumps shown in Chapter 5 show positive activation as a solid black square and negative activation as a dark gray square.

In Figure 4.15, the roles *left-post* and *left-bowl* have been activated to 100%. Then activation is spread, with the result shown in Figure 4.16. Activation spreading from *left-post* activates the wholes **b, b3, h, k, k2, k3 and k4**, while *left-bowl* activates only **b**. As a result, **b** ends up with the highest activation. Also notice that the activations of *left-post* and *left-bowl* themselves have decayed.

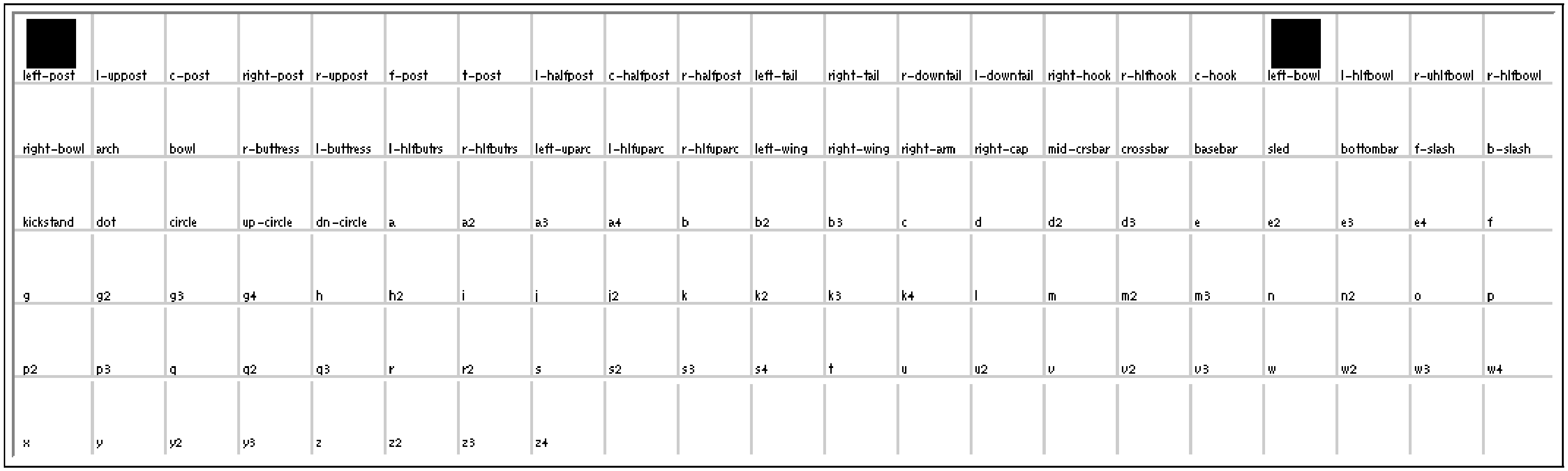


Figure 4.15: A pattern of activation (set by hand) in the roles, shown before any spreading has occurred.

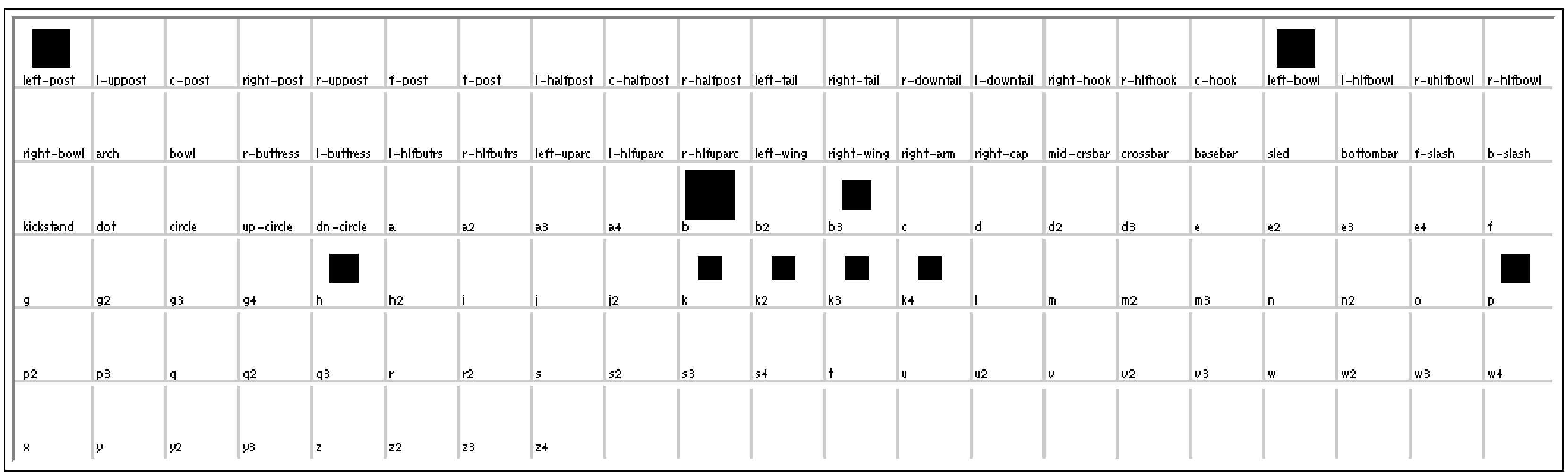


Figure 4.16: After the spreading of activation, a few wholes have become active to greater and lesser extents.

4.3.8 Temperature

Like its predecessors, the role model makes use of a global measure of the "happiness" or "goodness" of overall structure in the system. Two goodness ratings, one from the Visual Focus and the other from the Conceptual Memory, are combined together into one global measure called *temperature*. Temperature is used to control certain aspects of system processing. Since there is no structure at the beginning of a run, and no letter concepts are activated, temperature starts out high. As the system builds good parts and moves successfully towards an answer, the temperature falls. If part-level processing ends up at a dead end, so that a part must be destroyed, temperature will rise to reflect disorganization in the system. In this way, temperature dynamically reflects structural coherence in the system in a single global value. Eventually, if temperature gets low enough (determined probabilistically), the role model stops and returns an answer.

Goodness values range from a low of 0 to a high of 100 — the higher the better. Goodness in the Visual Focus is determined as an average of the goodness values of all current parts. This average is adjusted downwards if there are whiny (un-labeled) parts around. A part's goodness rating is computed with the following function:

$$goodness = \frac{activation + \frac{100n}{possibles}}{2}$$

where n is the number of labels attached to the part, *possibles* is the maximum number of labels usually attached to any part (a system parameter), and *activation* is the activation of the most highly activated role bound to the part. Thus, part goodness has to do both with the number of labels attached to a part and with how well the part matches roles. So this measure includes information regarding role-level binding, and thus reflects how well structure in the Visual Focus *coheres* with concepts in the Conceptual Memory. A goodness rating for the entire Visual Focus can be calculated as the average of all parts' goodness ratings.

Another measure reflecting goodness in the Conceptual Memory, is calculated according to activations of wholes. The formula is very simple: $100/actives$, where *actives* is the number of wholes activated above a certain threshold (currently set to 51 points). This kind of goodness is directly related to how much competition there is in the system between hypothesized wholes: the less competition, the better.

Temperature is set to 100 minus the average of the Conceptual Memory goodness (reflecting the number of active wholes) and the Visual Focus goodness (reflecting the goodness of all the parts). Thus, temperature can range from a high of 100 to a low of 0.

Temperature affects processing in the role model in two ways. Firstly, when temperature becomes low, the chance of processing coming to an end is increased. Returning an answer when temperature becomes low enough is the usual route to a solution in the role model. A probabilistic threshold ranging around ten degrees is used to determine whether temperature is low enough. Secondly, temperature controls nondeterminism in the role model. At high temperatures, when there is little evidence to go on, uniformly random decisions are made. As temperature lowers, randomness plays a smaller role in processing decisions. This effect is most predominant in the codelet-picking mechanism of the Coderack.

Temperature in the role model is not lowered according to a pre-determined, rigid schedule, as it is in simulated annealing or Boltzman machines [Kirkpatrick et al., 1983; Hinton and Sejnowski, 1986]. In Boltzman machines, temperature is used exclusively as a top-down randomness-constraining factor and is imposed on the system from the outside. In the role model, temperature also serves to affect randomness, but this time from *within the system*. Temperature is a feedback mechanism that dynamically varies according to the goodness of the perceptual structure that has been built throughout a run.

Temperature is less important in the role model than it was in Copycat. This is due to the fact that the process of perception in the role model is in some senses more constrained that it was in Copycat. Because all parts must eventually match up with roles, and roles must in turn match up with wholes, there is much more top-down constraint on processing in the role model than there was in Copycat. The upshot of all this top-down pressure is that the randomness-controlling aspect of temperature is not as critical to the performance of the role model. Temperature is still very important, however, as processing usually ends when temperature becomes very low.

4.3.9 Gestalt activation

Gestalt codelets determine whether a letterform has one of the following properties: ascender, descender, no-tips, or closure. Each gestalt codelet that is run picks at random one of these four properties to consider, and then looks at the letterform in the Visual Focus *at a global level*, paying no attention to parts or labels. For example, the "ascender" gestalt codelet simply checks for the presence of quanta in the ascender zone. If one or more quanta are there, it considers "ascender" found.

If a gestalt codelet finds what it is looking for, it lightly activates all role-sets in the Conceptual Memory sharing that property. In the case of "ascender", wholes that are activated include: **b, b2, b3, d, d2, d3, f, h, h2, i, j, k, k2, k3, k4, l**, and **t** — in other words, those role-sets *usually* having parts in the ascender zone. It also lightly inhibits role-sets having the opposite property (if such a property exists). In our "ascender" example, the role-sets **g, g2, g3, g4, j, j2, p, p2, p3, q, q2, q3**, and **y** are lightly inhibited, because they have descenders.

The purpose of this type of light activation and inhibition is to help prevent far-out hypotheses from being considered. Finding an ascender, for instance, should rule out, at least to some extent, consideration of **p** as a possible hypothesis.

The overall effect of gestalt codelet activity in the role model turns out to be minor. Experiments in which gestalt codelets were completely left out of processing are reported at the end of the next chapter, in Section 5.6.

4.3.10 Tightening and loosening roles

In the course of programming the role model, quite a few unexpected twists and turns were encountered. One such turn came when the shape shown below — a clear and obvious 'b' — was seen as a 'd'. How could this have happened?

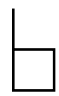


Figure 4.17: A clear and obviously strong 'b' that was once mistaken for a 'd'.

The answer is that somehow, very strange parts were formed and labeled in a bottom-up way, and they best matched the roles of the role-set **d**. These roles, as initially implemented, were so plastic that they were able to become activated by the weird parts, and then to see the parts as weak, but allowable, instances of themselves. This was certainly a very fluid perceptual act! But perhaps a bit *too* fluid, we were afraid.

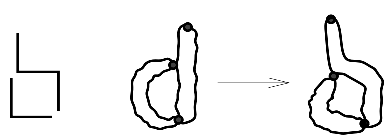


Figure 4.18: The parts that the 'b' of Figure 4.17 was parsed into. The part to the right was seen as a *right post*. The bottom part was seen as a *right bowl*. Thus this shape was seen as fitting the role-set **d**. As is shown schematically to the right of the arrow, the roles of **d** were considerably stretched in order for this to happen.

An investigation into the particulars showed that allowing "the best-matching roles" to become activated (and bound to parts), even if those activations were very low and the parts were somewhat strange, was not a necessarily a good idea. The reason was that strangely-formed parts tended to occasionally activate roles that they should have activated only under much more extreme circumstances. Instead of being broken up under top-down pressure from a more standard interpretation of roles, the parts remained intact, causing problems. The careful analysis of this processing problem led to an eventual restructuring of the activation process, bringing it into line with the "first pass" approach discussed in Section 4.3.7 above.

Adoption of a "first pass" approach to role activation allows a new variable to emerge in the system — role looseness. What this parameter does is control two thresholds: the maximum number of roles allowed to be activated by a given part at one time, and the minimum number of roles that must be activated at the same time in the first-pass stage. Controlling these two variables allows for the loosening and tightening of roles.

In initial processing, default role activation is not tampered with, and only fairly strong part⇒role matchups cause roles to become activated. (The initial processing of roles is thus conceptually tighter than it was in the case of the misperceived 'b' above.) If processing continues for a considerable time (3,000 codelets by default) and no good candidate parsing has yet emerged, roles are loosened by allowing poorer part⇒role matchups to activate roles. Since looser roles will stretch more, they will match strange parts more easily. Roles may be loosened still further at 4,500 codelets if no good parsing has yet been found. Role loosening is controlled by manipulating two parameters in the code as explained above.

Roles should not be loosened initially, since if that were done, poor parts would have little top-down constraint, leading the system to seeing 'b's as 'd's for no apparent reason, and to making other worrisome errors. A fair amount of "normal" processing — searching for a good parsing under regular role constraints — must be done before roles are loosened. Our implementation of a period of 3,000 codelets before the first loosening (determined empirically)[13], followed by a period of 1,500 more codelets before further loosening, is unfortunately very crude, though it works reasonably well. In some runs, processing seems to get absolutely nowhere until *finally* at 3,000 codelets, roles are loosened, and more far-fetched part⇒role matchups can be considered. It sometimes seems as if the processing gates have been flung open, allowing what should clearly and obviously have happened the whole time to finally take place. This makes one sometimes feel that role-loosening should set in much earlier. A much more reasonable solution to role-loosening would involve a kind of self-monitoring by the role model whereby the model could track its own processing, loosening and tightening roles appropriately in a more dynamic manner. This sort of self-monitoring process, though desirable, is very complicated in a system based on probabilistic processing by codelets. Research is currently underway at CRCC on a project involving this type of self-watching and Copycat. The Metacat project, as it has come to be called, is in early stages of conceptual design. For information about the Metacat project and the importance of self-watching, see [Hofstadter and FARG, 1995, chapter 7] and [Marshall, 1993]. Once a method for self-watching has been developed, it could be applied in the role model to set role looseness "on the fly".

4.3.11 Role subsumption

Another unexpected twist in the road to the role model came when we noticed the rather large number of possible parts that can be formed (given no top-down control) from any set of quanta. Lurking inside a very standard-looking 'b', for instance, are latent parts that would match up well with a huge number of roles (see Figure 4.19).

What happens if an "incorrect" set of parts gets formed in bottom-up processing, and these parts activate inappropriate roles? Should all possible roles be considered?

[13] Experiments with role-loosening are reported in Section 5.6 of the next chapter.

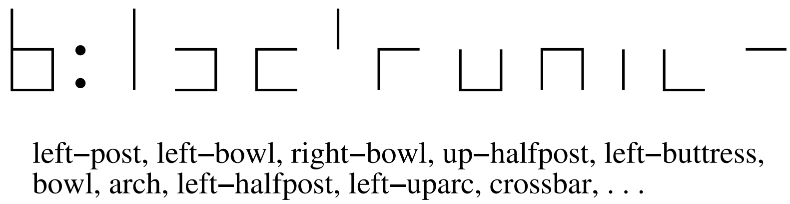


Figure 4.19: A standard-square 'b' contains a whole host of parts that can match a large number of roles *if they happen to be formed*. Some of the parts contained in this 'b' are shown to its right. Under the parts is a list of roles that might be activated by each of the parts shown.

How can this problem be alleviated?

These questions are answered in the current version of the role model by making use of *role subsumption*. The idea is very simple. Some roles can, if taken together, be subsumed under another larger role. For example, the role *left post* subsumes the roles *left-halfpost* and *left-uppost*. Likewise, the role *arch* subsumes the roles *left-halfpost* and *right-buttress*. Whenever a role is activated, a very quick check of other activated roles is made. If an entire collection of roles making up a larger, subsuming role is present in the Conceptual Memory, then a role-subsumption codelet is posted.

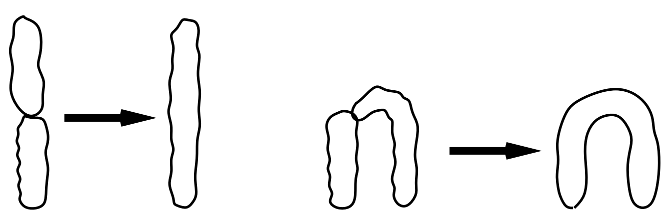


Figure 4.20: How the roles *left-halfpost* and *left-uppost* are subsumed by *left-post*; and how the roles *left-halfpost* and *right-buttress* are subsumed by *arch*.

The role-subsumption codelet checks the parts bound to a set of subsumed roles to see if they *can* and *should* be put together. The parts *can* be put together if they touch properly at endpoints and would result, if combined, in a new part having only two tips. Whether or not they *should* be put together is another matter, involving checking the current context within which the subsuming role must eventually fit. (The role of context in role subsumption is discussed further below.) If the parts bound to the subsumed roles match the combination criteria and if the proper context has been noted, then the parts are put together, forming a new part. This part, if it survives the labeling process, may eventually activate the subsuming role, though, like any other part, it is susceptible to the usual breaking procedures throughout the looking and labeling stages.

Any decision whether to combine the parts of subsumed roles is constrained by the context provided by other active roles in the Conceptual Memory. Figure 4.21 provides an example.

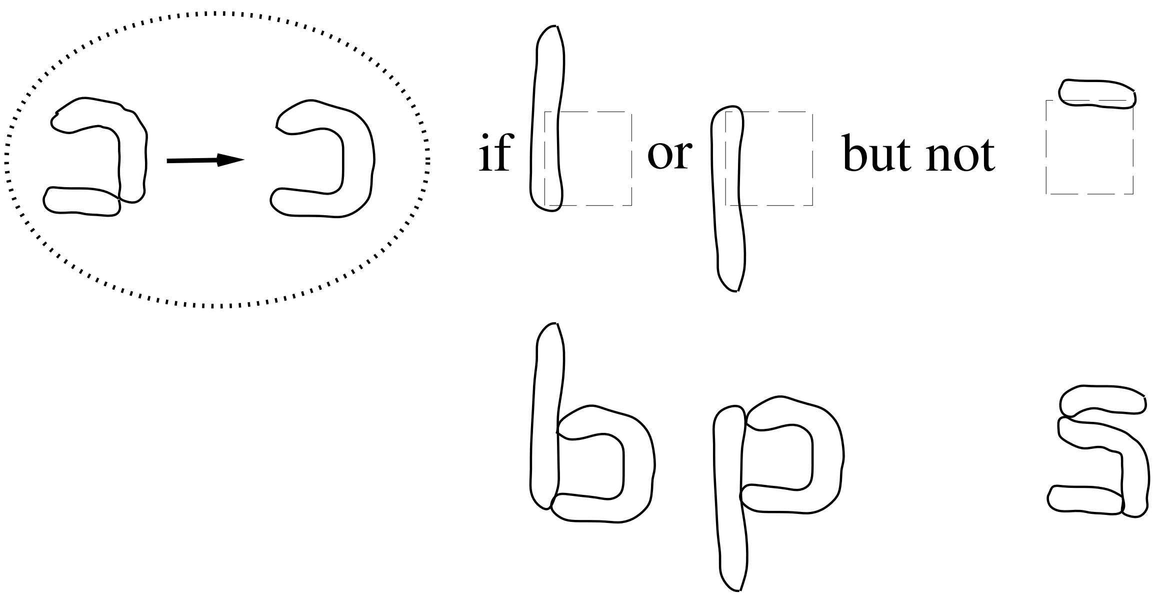


Figure 4.21: The context in which subsumed roles occur affects the role model's decision as to whether or not to combine them. The particular subsumption shown here — (*left-buttress* + *basebar* = *left-bowl*) should occur in the presence of *left-post* or *left-tail* (possibly leading to the recognition of a 'b' or a 'p'), but should not occur in case there is a *crossbar* (indicating a possible 's').

As Figure 4.21 makes clear, in some situations, even if the parts bound to a set of subsumed roles *could* be combined (that is, the parts touch properly and would combine into a new part having only two tips), they should not be.

All possible role subsumptions that we have identified are coded into the "role-subsumption list". This list contains information about which roles might be combined together as well as contextual information regarding when such combinations are beneficial (and should be pursued), and when they are detrimental (and should be avoided). An example of a run in which role subsumption occurs is included in the next chapter.

A related and complementary process called *role-breaking* can also occur in some situations, though much more rarely. This may happen when a large part bound to a single role might be better perceived as a pair of parts bound to two different roles. This usually happens in the context of other roles having been discovered. Figure 4.22 provides an example.

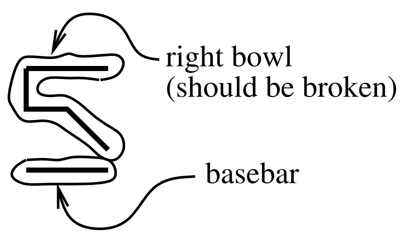


Figure 4.22: An example of role-breaking. In this case, the top of the 's' has been formed into a part that strongly activates *right-bowl*. Discovery of another part that fills *basebar* suggests that the part filling *right-bowl* should be broken up.

We do not list all of the possible role-subsumption and role-breaking situations here. Suffice it to say that we have identified 32 conditions in which sets of roles may be able to be subsumed, and 3 conditions in which roles might be able to be broken up. Once again, readers interested in such details are referred to the source code for further information.

4.3.12 Role clamping and guessing

Occasionally, the role model is unable to parse a given letterform into any role-set, even after loosening roles as much as possible and carrying out a large amount of computation trying. After some pre-specified maximum number of codelets (currently set to 6,000), the role model gives up on a letterform that has thus far proven "unrecognizable" (though in a sense, it is really "unparsable"). Once this point has been reached, there are two alternatives to abruptly ending processing.

Role clamping

The first is role clamping. If the user has specified the category of the letterform that the role model is attempting to recognize (for instance, if the Examiner has been asked to "verify" a letter or to "see" a seed letter *as* an instance of a certain category), that information may be of use to the system. Role clamping works as follows. Given a list of role-sets to clamp and a numerical parameter called *value* to use, the role model will first activate all of the constituent roles in the role-sets (with *value* activation points) and then set a flag in each role's clamp slot. Thereafter, whenever activation is spread, that role will receive a new jolt of *value* activation points. Obviously, an initial jolt of activation alone will quickly decay away. For this reason, a new jolt must accompany each activation-spreading session.

Clamped activation comes into the system from the outside and can bias processing in favor of certain categories or even particular conceptualizations (role-sets). Clamping does not amount to "telling" the system what the answer is, however. Part-building and labeling, as well as role activation and inter-role competition for parts, must still occur as usual. It is true that internal competitions will be biased in favor of clamped roles, but this does not ensure that a clamped role-set will always win.

Role clamping is like telling a person what a letterform is meant to be and then asking them whether or not they can see it as such. Obviously, this question may be answered either in the affirmative or in the negative. For example, even when told that the following shape — $\boxed{\text{W}}$ — is a 'G', we doubt seriously that anyone could see it as such. Of course, word-level top-down pressures in reading can induce arbitrarily distant shapes to be seen as instances of letters. In any case, if someone were given a weird but probably categorizable letter at the boundaries of a letter concept's space, and were told "that is a so-and-so", the act of telling them so would probably push their perceptual processing over the edge, causing them to see the shape in a new light. This is the way in which clamping can bias the role model in a proper direction.

Clamping seems to be useful in helping the system categorize some problematic letterforms. Some experiments with clamping are reported in Section 5.6 of the next chapter.

Guessing

The other alternative to a processing "dead end" once a maximum number of codelets has been reached is to try and guess the category based on non-role-based criteria. Unfortunately, since role processing has led to a dead end at this point, none of the role information can be used to make a reasonable guess.[14] After trying several guessing methods, we decided to use a template-matching approach.

A "blurred prototype" for each category is created by averaging together the binary quanta-vectors of hundreds of letterforms from a given category.[15] Given 26 blurred prototypes, it is possible to determine which prototype a given letterform is closest to by using city-block distance (*i.e.*, the sum of the absolute values of their coordinate-pair differences). By finding the prototype whose distance to the "mystery" letterform

[14] This claim has been verified through experimentation, using activations in the Conceptual Memory and parts in the Visual Focus to determine a guess. The experiments were all resounding failures.

[15] This process is explained more fully in Section 7.3.2.

is minimal according to this metric, an educated guess of the letter's category can be made.

This sort of guessing capability is rarely required, is not really part of the role model, and accordingly was not allowed during system testing. Results of a comparison between guessing and non-guessing models are discussed in Chapter 8, Section 8.2.1.

Some combination of guessing followed by role clamping might allow the system to take a "lucky guess" (which, it should be pointed out, returns only a simple category label and not a parsing) and then parlay that guess into a full-fledged parsing, using role clamping. We have not attempted such a combination, though all of the ingredients for such a recipe are present in the current role model.

4.4 Some further implementation details

The role model is written in a particular flavor of Chez Scheme [Dybvig, 1987], called SXM, developed by John Zuckerman of Motorola [Zuckerman, 1992]. "SXM" is an acronym for Scheme/X/Motif. It provides a complete set of Motif widgets that enhance Chez Scheme with a sophisticated X-window-based graphical interface.

Due mainly to the power of SXM, a graphical front-end and display system was developed for the program. Thus knowledge of Scheme is NOT required to run the role model. Some of the role model's key parameters may be adjusted interactively through the X-windows interface (see Figure 4.25). Moreover, a large number of running options and statistical displays are available (see Figures 4.23 and 4.24). The code will also run without graphics in any standard Scheme implementation. Source code for the role model is available by **ftp**. Instructions for retrieval can be found in Appendix B.

4.4.1 The role model's performance

In the next chapter we examine the performance of the role model in detail, discussing both individual runs on particular letters and collections of runs on entire datasets. Through such analysis, the degree of flexibility of the role model afforded by the mechanisms that we have discussed in this chapter becomes apparent, and is quite impressive.

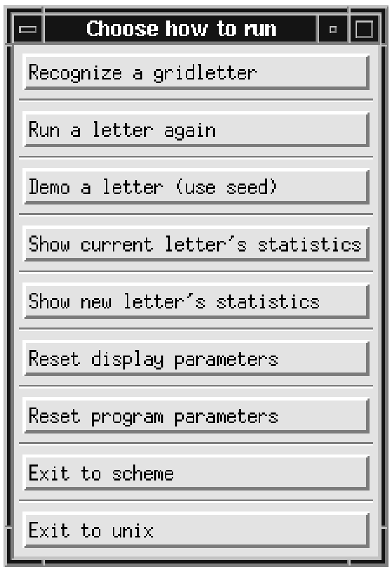


Figure 4.23: The main menu for the role model, showing several processing options.

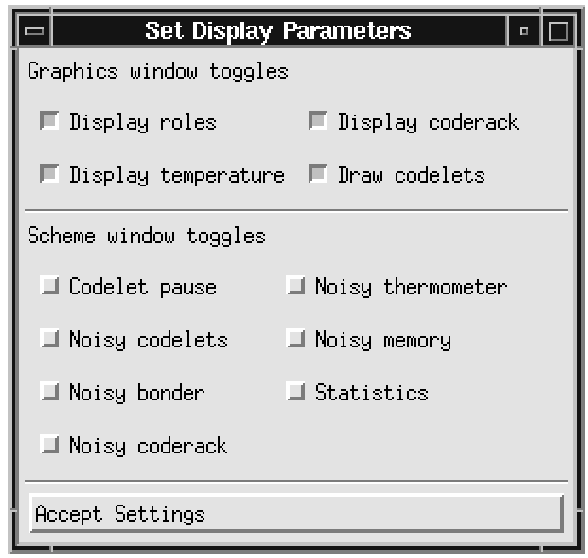


Figure 4.24: A toggle box allowing selection of display variables. There are four main display windows for the program that can be controlled from here.

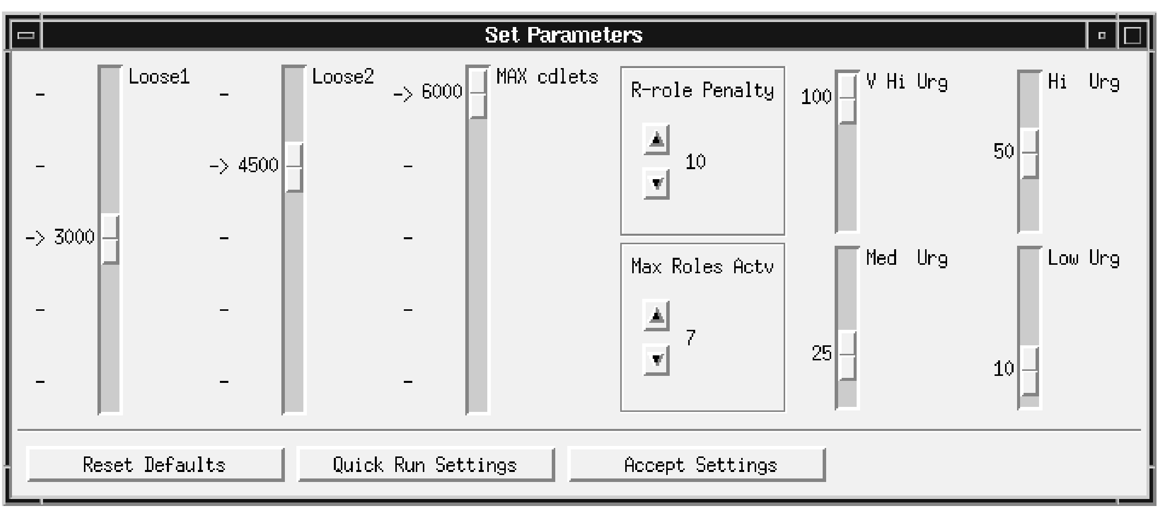


Figure 4.25: A control panel for interactively adjusting critical system parameters.

CHAPTER FIVE

Performance of the Role Model

Now that the role model's underlying philosophy and code-level implementation have been introduced, we turn to a series of example runs. Although discussing the role model's architecture is very important, the role model can probably be best understood by scrutinizing its actual recognition behavior. After giving detailed descriptions of runs on particular letters (which we trace with the aid of screen dumps), we turn to analysis of the model's performance on bigger sets of letters. Among these larger sets we include letters that have proven problematic for the role model, and we explain what makes the role model unable to recognize them.

Ultimately, the role model's behavior should be compared with human behavior on common sets of data. The next-to-last section of this chapter introduces a standard dataset that we call PSYCH. Results of psychology experiments on human perception of this dataset (given in Chapter 7) are compared with the behavior of the role model in Chapter 8.

At the end of this chapter, we include a section in which some of the role model's key parameters are manipulated. Two standard datasets are developed and used in all the tests, so that comparisons can be made among different parameter-settings, shedding light on some of the most critical system parameters.

5.1 Six simple examples

In this section we will trace a handful of runs of the role model on specific letters. Hopefully this will imbue the architectural discussion of the last chapter with some life, getting across how actual runs unfold over time.

5.1.1 Recognizing an 'i'

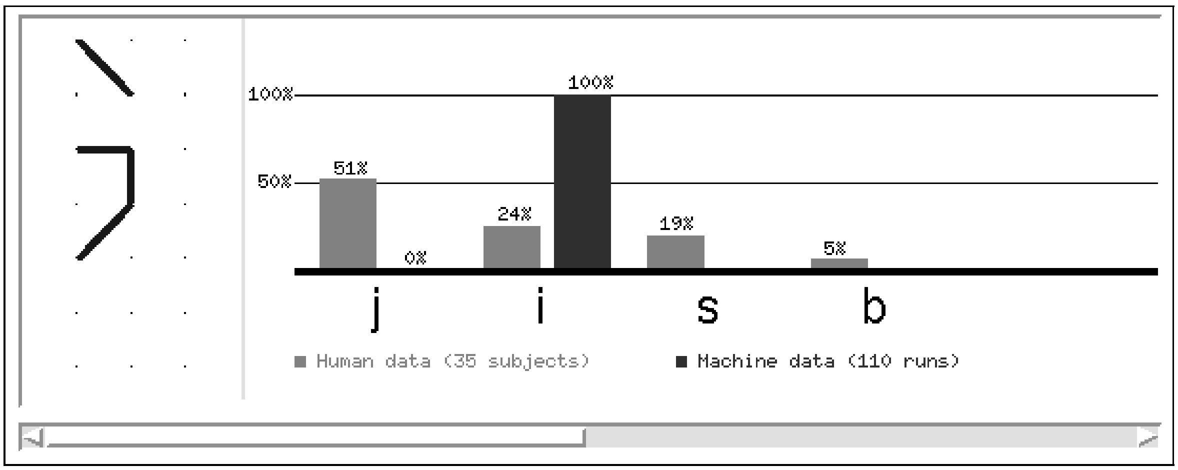


Figure 5.1: Results of 110 runs of the role model on a particular 'i' are shown along with results from the human experiments (see Chapter 7). Unlike the people in the experiment (who ignored placement constraints), the role model categorizes this 'i' correctly every time.[1]

As Figure 5.1 shows, over 110 runs on this rather simple 'i', the role model is able to recognize it correctly every time.[1] Human data are also shown in the figure (in lighter tones), but should be ignored for the time being (especially since in this case

[1]In order to count as a "correct" response, the role model must have responded with the *one* unique category in which a stimulus was placed by the experimenters (the category intended by the letter-savvy designer of the given gridletter). Note that although naïve designers often make severe design mistakes (like drawing a shape they *call* an 'o' but that in fact will generally be seen as an 'e'), *sophisticated* designers can be counted on to provide reliable categorizations for far-out, highly stylized letterforms that they create. For some of the more ambiguous tokens (including this one), categorizing turns out to be a very hard task. We use the designer's intended category in order to provide a fair comparison between humans and the role model (see Chapter 8). Use of the term "correct" below will always be subject to this disclaimer.

humans ignored placement constraints that the designer of the letterform took into account). We discuss a particular one of these 110 runs now. Because of the role model's fundamental nondeterminism, each run on this 'i' will differ from others in terms of exactly which codelets run, how many codelets end up running, what order structure is built in, which labels get attached to parts, and the activation values of roles and wholes, but despite all this low-level variability, the same overall result is reached each time. Although each run is unique, it is still very useful to trace through at least one of the runs in order to get an idea of how processing in the role model takes place.

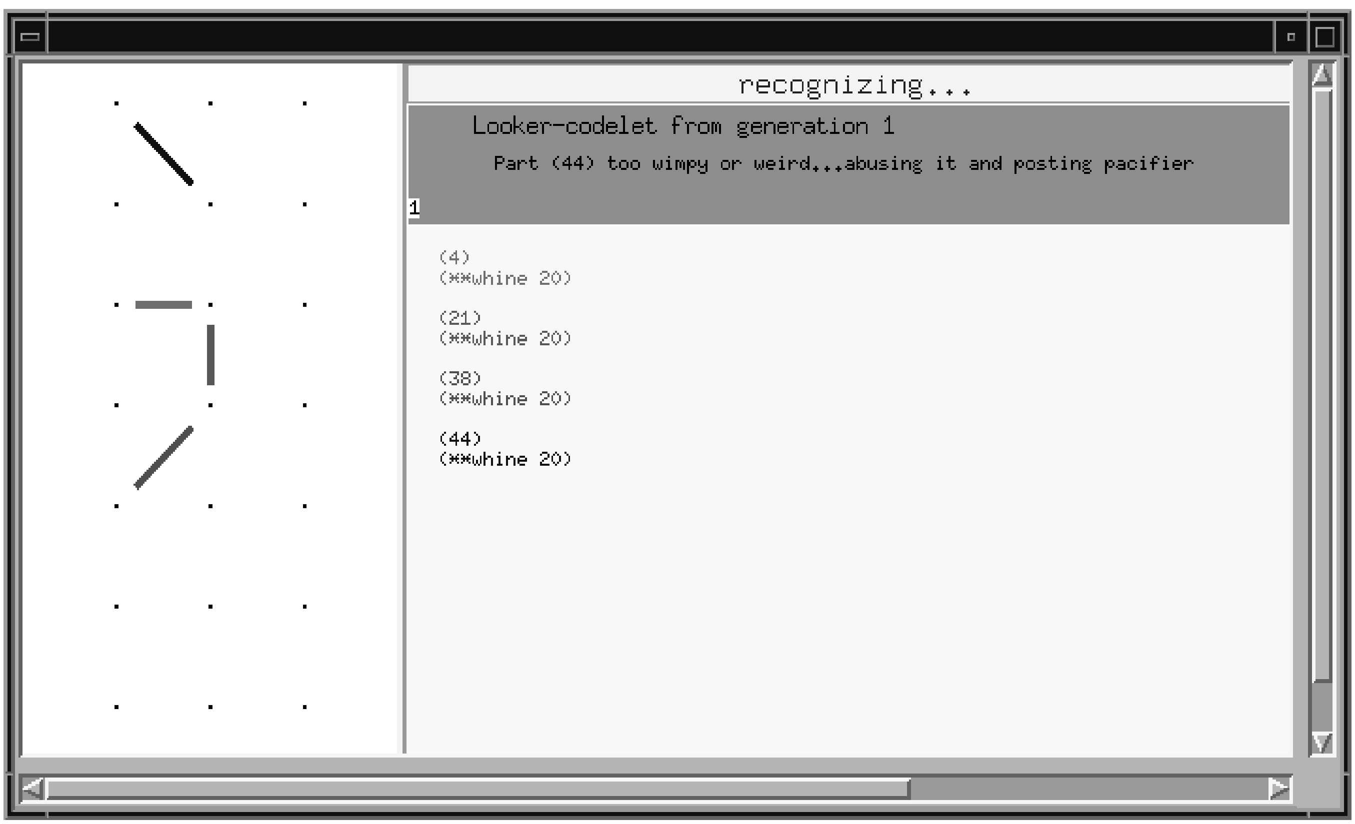


Figure 5.2: This screen dump shows the initial parts formed after gluing and shaking have occurred. Note the whine value associated with each unlabeled part.

In the beginning of this run, glue-ants swarm the shape, stochastically placing glue at angles between quanta. Then the shape is shaken into initial parts. In this particular run, four very small parts happen to emerge from the gluing/shaking process. (In a very unusual turn of events, these happen to be nothing but the four quanta that the role model was handed at the outset.) Each part is referred to in the Visual Focus by its constituent-quanta list, as can be seen in Figure 5.2. Which list belongs to which part is easily tracked on the monitor since everything is color-coded. Unfortunately, this aspect of the display is lost in a gray-scale screen dump. Each part is initially tagged with a whine value that might increase over time, should a part be considered and rejected by a looker codelet. In fact, just such an action has occurred, and part (44) — making up the dot of the 'i' — will soon have its whine value increased.

Ten codelets into the run (not counting the gluing/shaking process)[2], the shape has been glommed into two parts. Parts are shown as connected chunkings of quanta. As can be seen in Figure 5.3, the "dot" part, part (44), has begun to be labeled by labeler codelets. It is hard to make out, but the text in the dark band near the top of the display says that one labeler codelet has just unsuccessfully attempted to attach a "horizontal" label to part (44) and has fizzled.

As processing continues and more labels are attached, a gestalt codelet is picked to run. This particular gestalt codelet chooses to investigate whether the letterform has any quanta in the *ascender* zone. It does, since the dot part of the 'i' is drawn above the x-height, and so activations (and inhibitions) in the roles are adjusted to reflect this minor discovery. The ascender-finding gestalt codelet sends only a minor amount of activation to each role-set in which ascenders normally occur. It also slightly inhibits those role-sets having descenders, as ascenders and descenders are usually

[2]Gluing and shaking codelets are not counted in the role model's codelet count, for historical reasons. All codelet counts reported throughout this thesis do not include the number of gluing and shaking codelets that were run in order to create initial parts. We will mark the codelet counts in this section with a † to remind the reader of this fact.

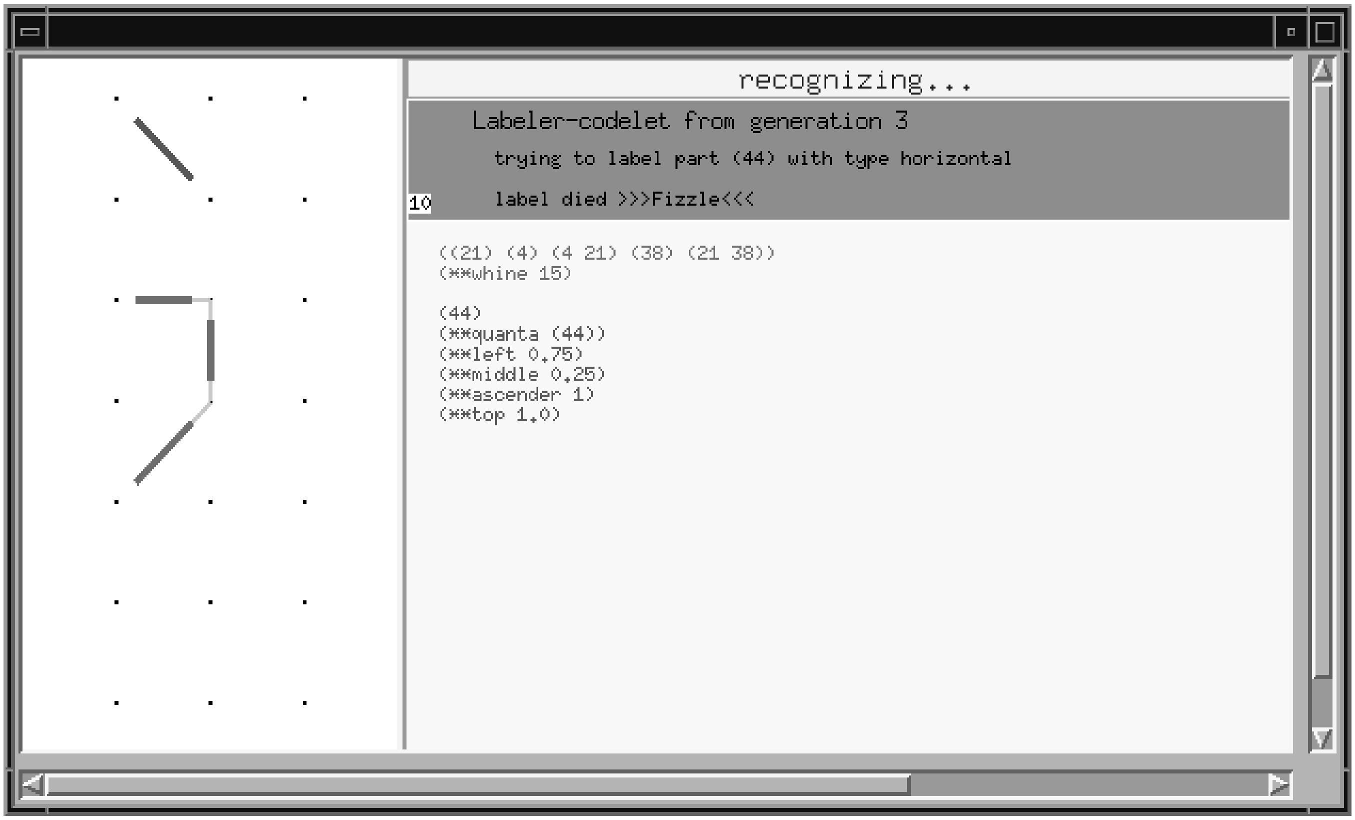


Figure 5.3: This screen dump shows the parts that have formed after ten† codelets. Note the labels attached to the "dot" part (part (44)).

mutually exclusive. See Figure 5.4, where negative activation values are shown as dark gray squares and positive activation values as solid black squares. Square size reflects activation value. (All of the very small activations in both figures are negative activations. All of the larger activations are positive.)

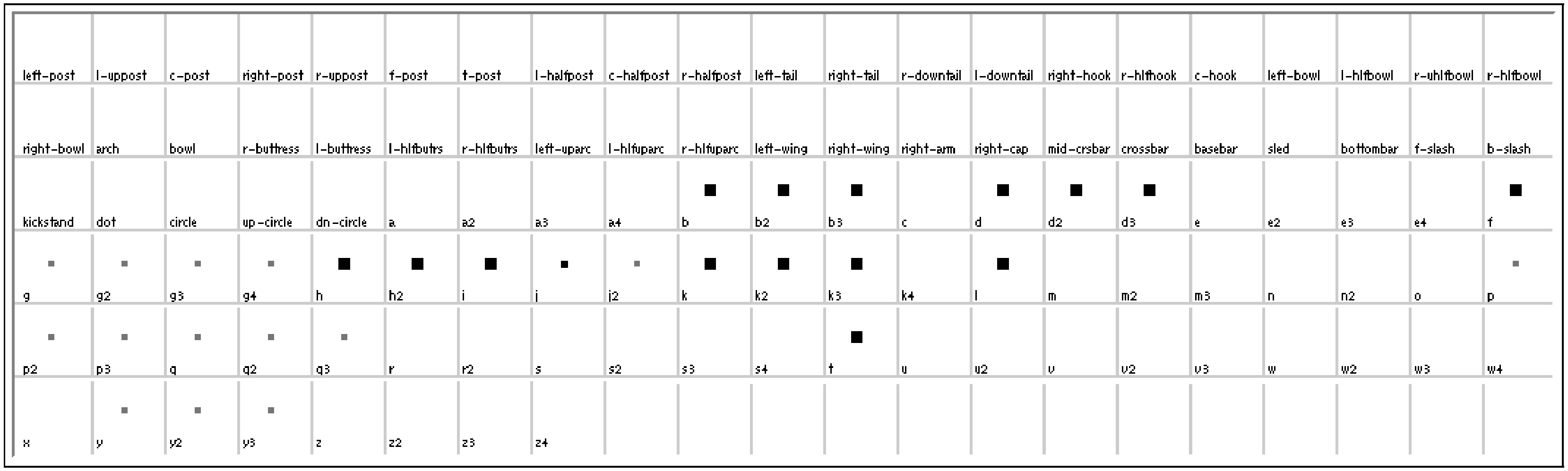


Figure 5.4: An "ascender" gestalt codelet lightly activates wholes with ascenders. Wholes with descenders are inhibited.

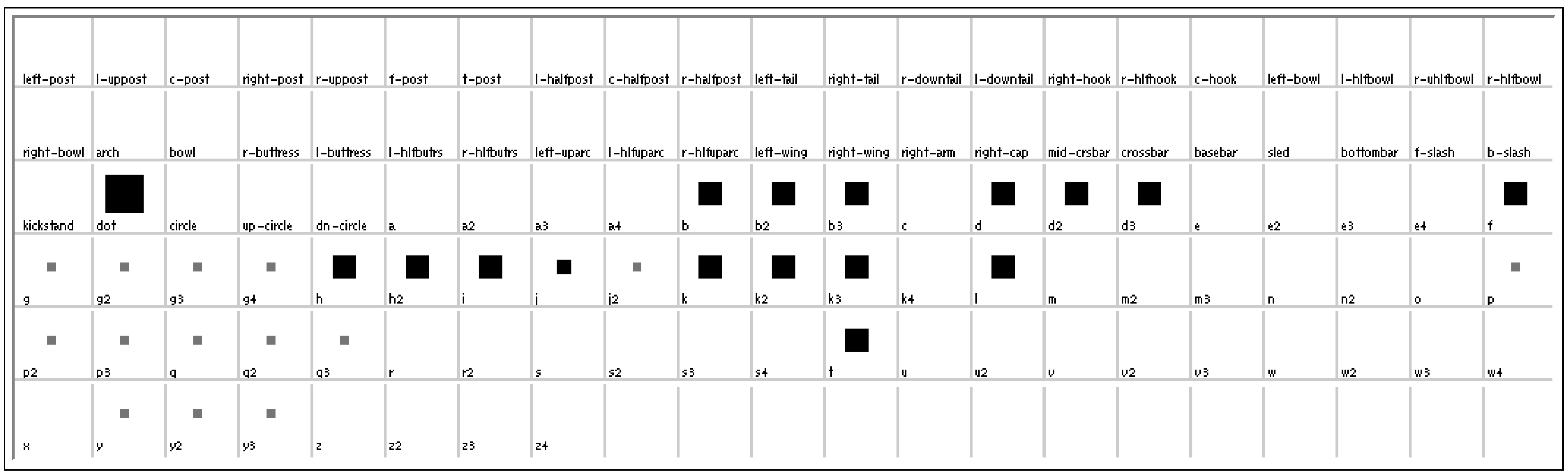


Figure 5.5: Activations of the roles after *dot* has been activated by part (44).

Around forty† codelets into the run, the "dot" part (part (44)), which has become fully laden with labels, passes inspection and is allowed to attempt to activate roles. Figures 5.5 and 5.6 show this happening. In addition to drawing activation in the role/whole window when a role becomes active, a globby shape representing the role is also drawn over the part that has activated the role. The magnitude of a role's activation is reflected in how darkly and solidly the role is drawn — with lightly dashed gray curvy lines representing slightly active roles and solid black curvy lines representing more highly active roles. The active role *dot* can be seen in Figure 5.6 as the dashed circle in the ascender zone.

Figure 5.6: The fully labeled "top" part sends activation to the *dot* role through an activation-sparker codelet. As the role *dot* becomes activated, it is drawn (as a dashed circle) over the part that it is bound to.

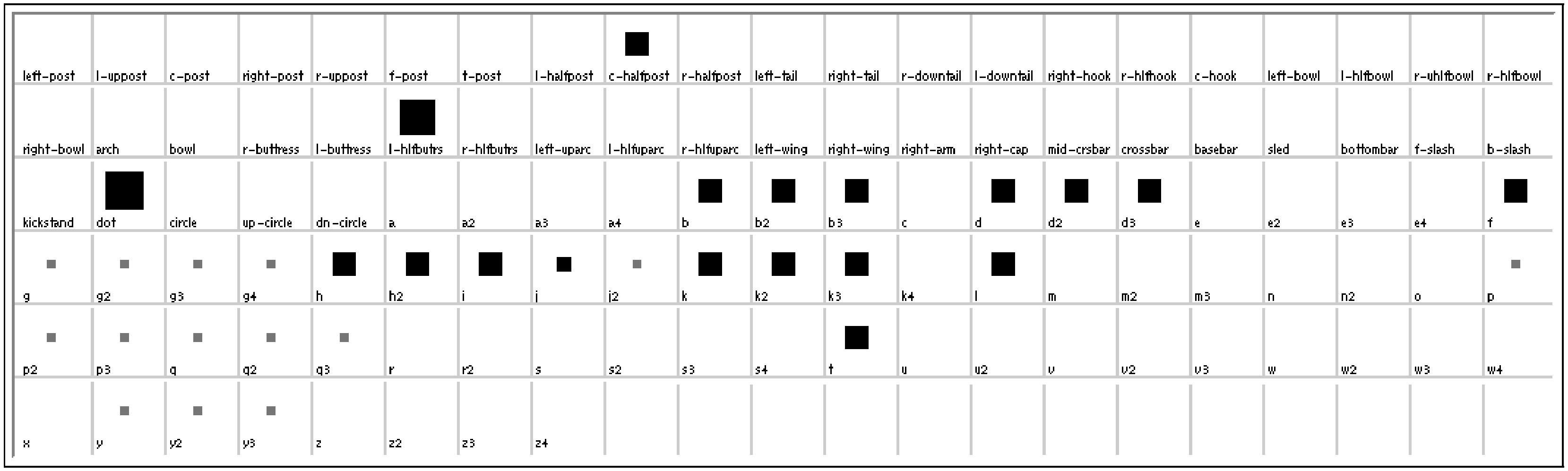


Figure 5.7: Activations of the roles before activation is spread by codelet 65†.

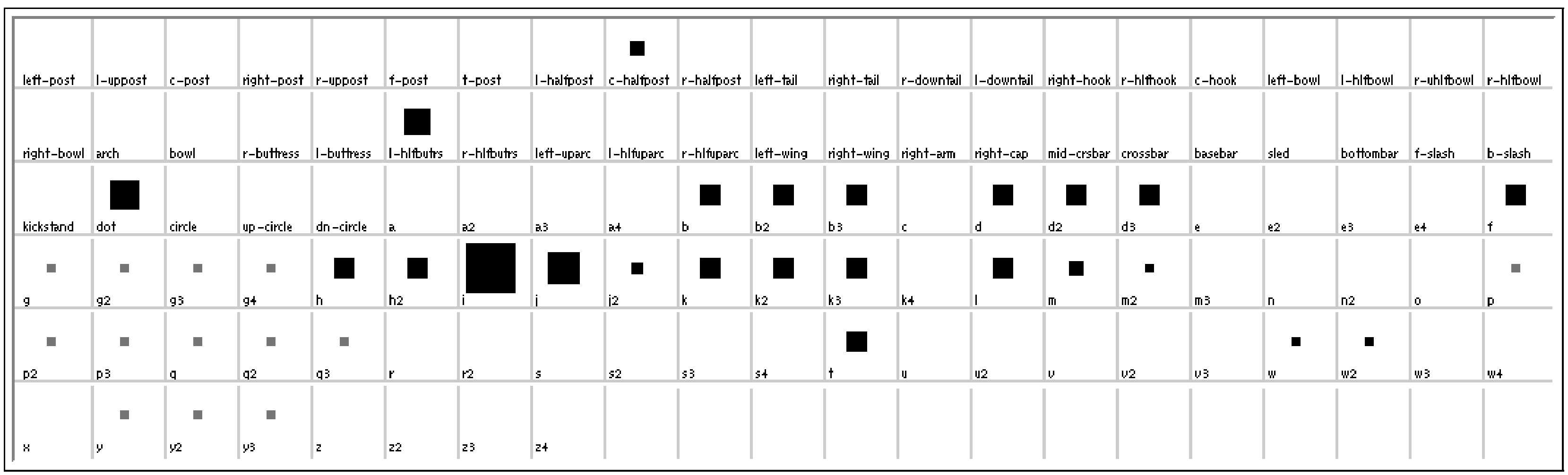


Figure 5.8: Activations of the roles after activation is spread by codelet 65†. Note that **i** and **j** are most active.

Near codelet 60†, the *center-halfpost* role becomes activated by the now fully-labeled "bottom" part. Activated at the same time is the *left-halfbuttress* role (a role from the **m1** role-set). The part in question could very well fill either of these roles. Activation is spread by the 65th† codelet to run. (The role activations before and after spreading are shown in Figures 5.7 and 5.8.) Notice that the role-sets **i** and **j** become most highly active.

Figure 5.9: The r-roles for **b** are checked by codelet 66†, and show that **b** is a bad hypothesis. The role set **b** is strongly inhibited. The three active roles are shown as curvy dashed shapes superimposed over the thicker solid lines representing parts.

Now that some role-sets have become active, r-role checking can take place. In fact, codelet 66† is an r-role checker that finds **b** a very bad hypothesis and promptly inhibits it. See Figure 5.9. (Incidentally, the whole **b** initially became slightly activated through the actions of an "ascender" gestalt codelet early in the run.) Figure 5.9 shows two more active roles than the previous figure did. These roles shown as dashed curvy shapes superimposed over the thick solid lines that represent parts. Note that the role *left-halfbuttress* is more strongly activated (and is thus darker) than its competition, *center-halfpost*. This competition has yet to be resolved.

Around 85† codelets into the run, the r-roles for **i** are checked. In addition, the r-roles of other competing wholes are checked and activation is spread again before the run ends. Figure 5.10 shows the role activations just before the r-roles of **j** (the last remaining highly-active competitor of **i**) are checked. Although the temperature is fairly low before this is done, it plummets just afterwards since the sole remaining active role-set is **i**. (Temperature always drops when there is just one active role-set.) Due to low temperature, the 89th† codelet to run can declare **i** the winner and halt processing. See Figure 5.11. A parsing of the letterform in terms of parts that nicely fill its roles is now available for further processing. The relationship between the particular labeled parts filling the roles of a role-set and the default norms of the role-set provide a great deal of stylistic information.

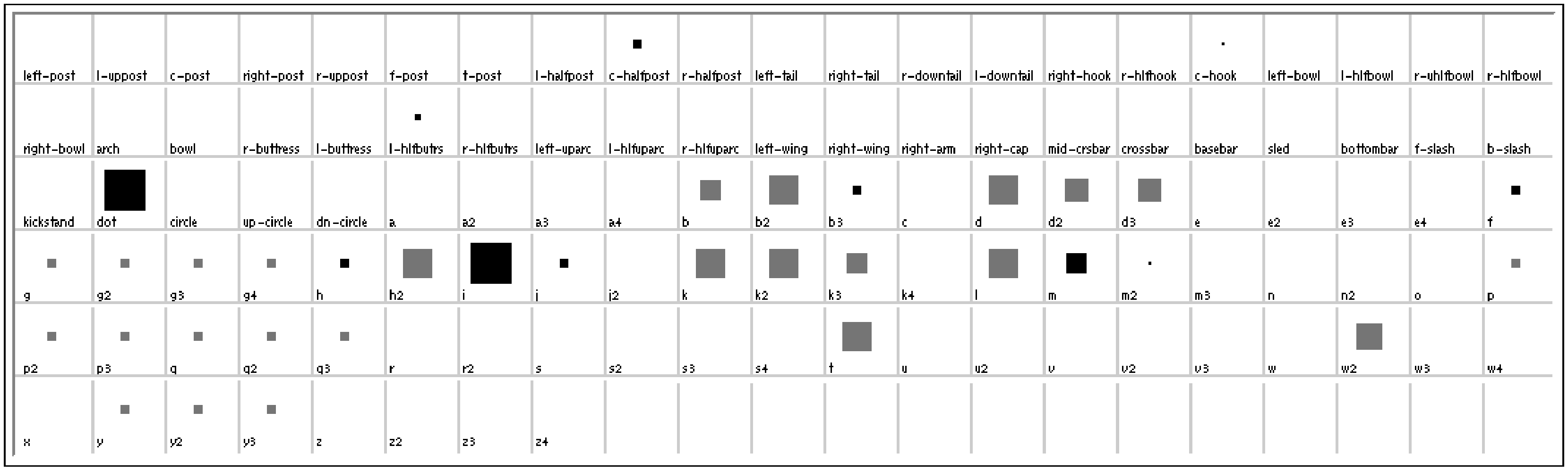


Figure 5.10: Activations of the roles just before the r-roles of **j** are checked, causing **j**'s activation to drop, temperature to fall accordingly, and **i** to be declared as the winner. Note the negative activations of some of the other wholes such as **b, h2,** and **k** (among others).

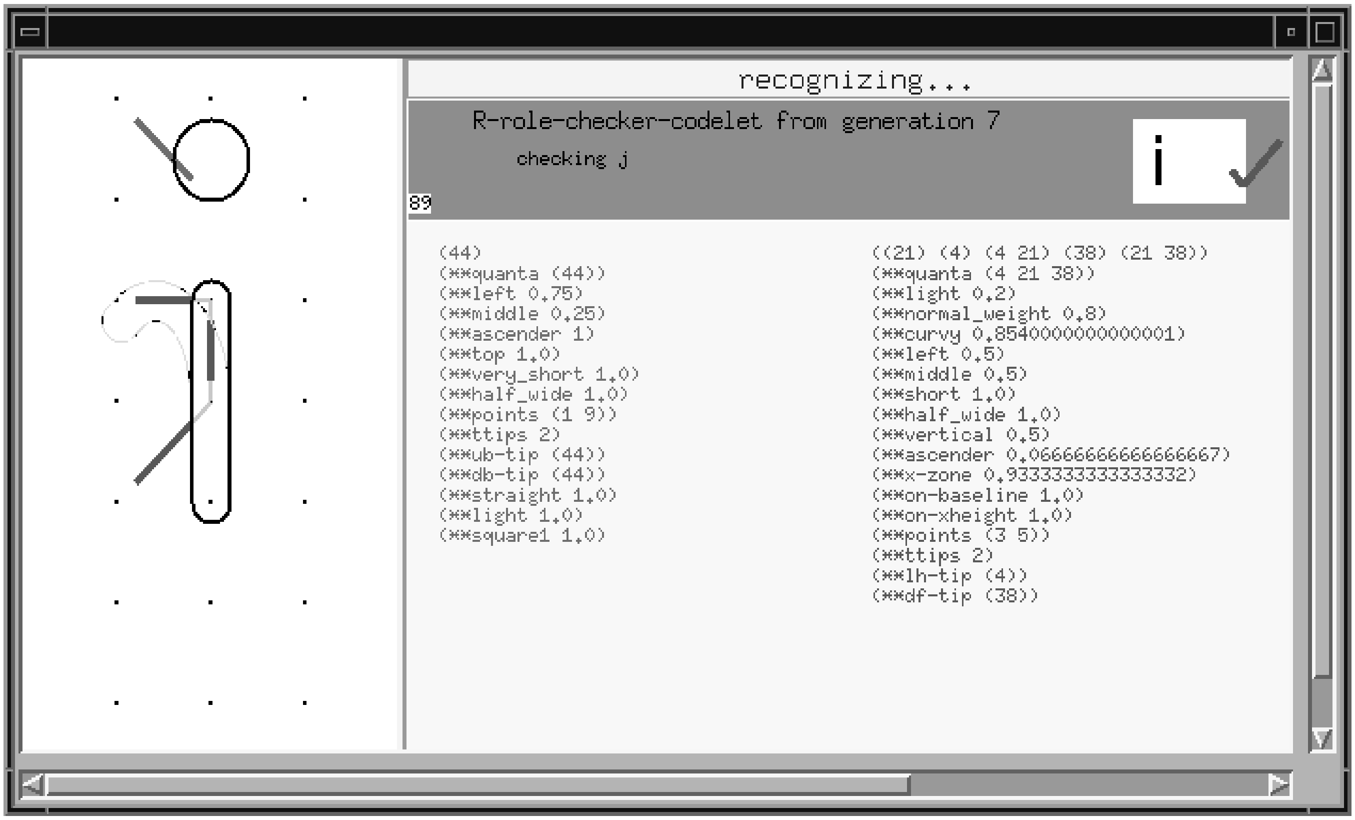


Figure 5.11: **i** is declared the winner by codelet 89† and recognition is complete.

What we have just shown are some snapshots taken during a particular run of the role model. Watching the program actually run (instead of reading about it) is much less choppy and discontinuous, and is more like watching a movie. As the run progresses, initial parts are formed, labels are attached, parts get built and destroyed, activations in the role window grow and shrink in real time, and so on. The truly dynamic nature of the program is best understood by watching it in person.

5.1.2 Recognizing a 'c'

The next letter we turn to is a very normal 'c'. Like the 'i' above, this 'c' is recognized correctly 100% of the time. However, there are two main pathways through which the role model can approach recognition of this letterform. The first, like our 'i' example, is a straightforward path in which fortuitous part formation leads the program in the right direction from the start. The second is more complicated and involves the use of *role subsumption* to recover from incorrect initial part formation (see Section 4.3.11 of the preceding chapter). We cover the fortuitous run first.

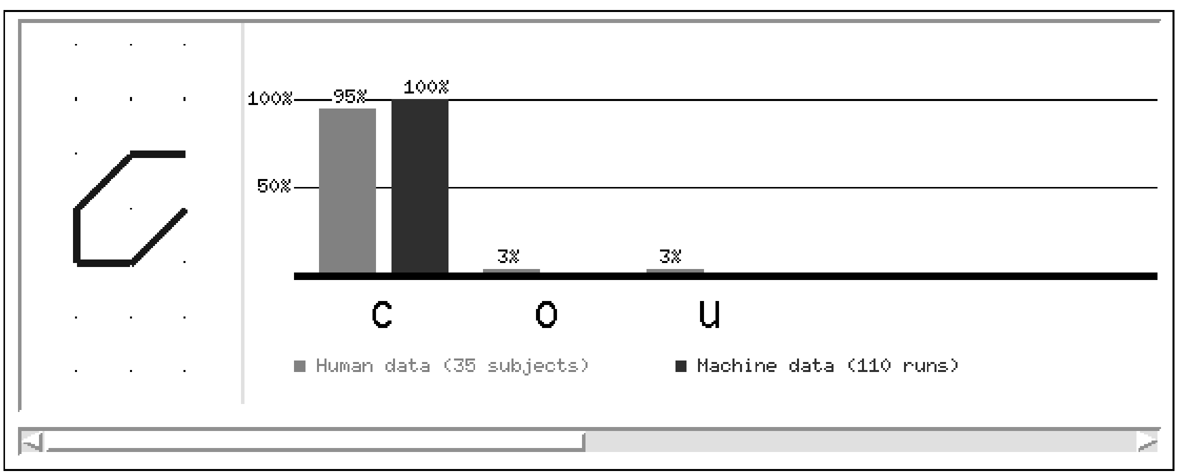


Figure 5.12: Results of 110 runs of the role model on a particular 'c' are shown along with results from the human experiments (see Chapter 7). Unlike the people in the experiment, the role model categorizes this 'c' correctly every time.

Initial gluing and shaking results in the formation of three parts. By the fifteenth post-gluing/shaking codelet, the initial three parts have been re-grouped into two, and labeling begins. A labeling codelet is trying to attach a height-related label to the upper part ((5 36)(23)(23 36)) but "fizzles" once it discovers that its job has already been accomplished by another codelet.

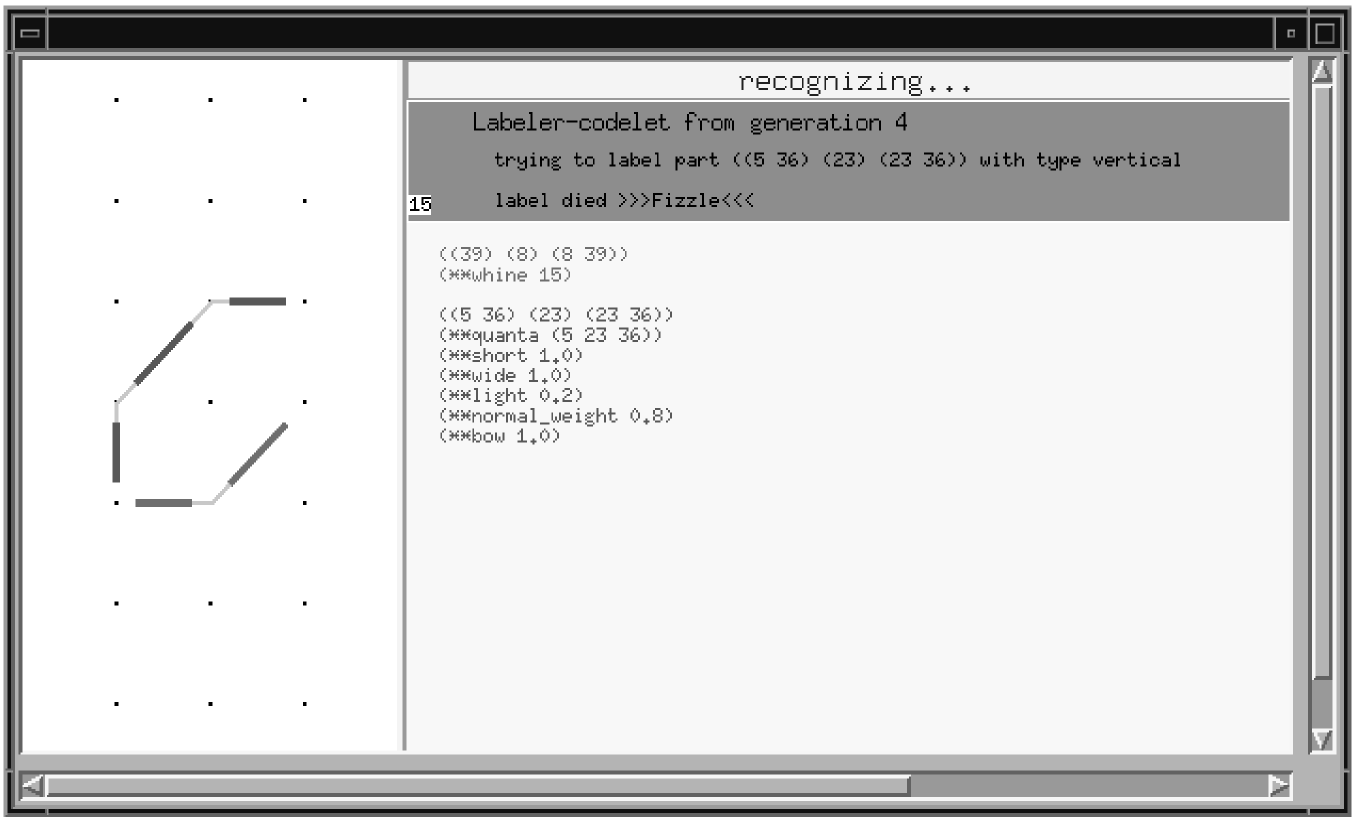


Figure 5.13: The initial two parts have been created by pacifiers and lookers. Now labeling begins in earnest.

Codelet 26† combines the two parts together into one bigger part. This act of glomming was carried out by a pacifier codelet that decided (probabilistically) to glom the unhappy lower part ((39) (8) (8 39)) to the partially-labeled upper part ((5 36)(23)(23 36)). This is a fortuitous combination, since what we are after is a right bowl (although the role model has no idea that this is the case, of course). Now that a new part has been formed, it must be labeled just as its "sub-parts" had been before they were combined together. All of the labels that were around before the two parts were combined must be thrown out (for obvious reasons). Since labels are so easy to compute, this is not a problem. In the picture, a leftover labeler codelet that wants to label one of the no-longer-around old parts fizzles.

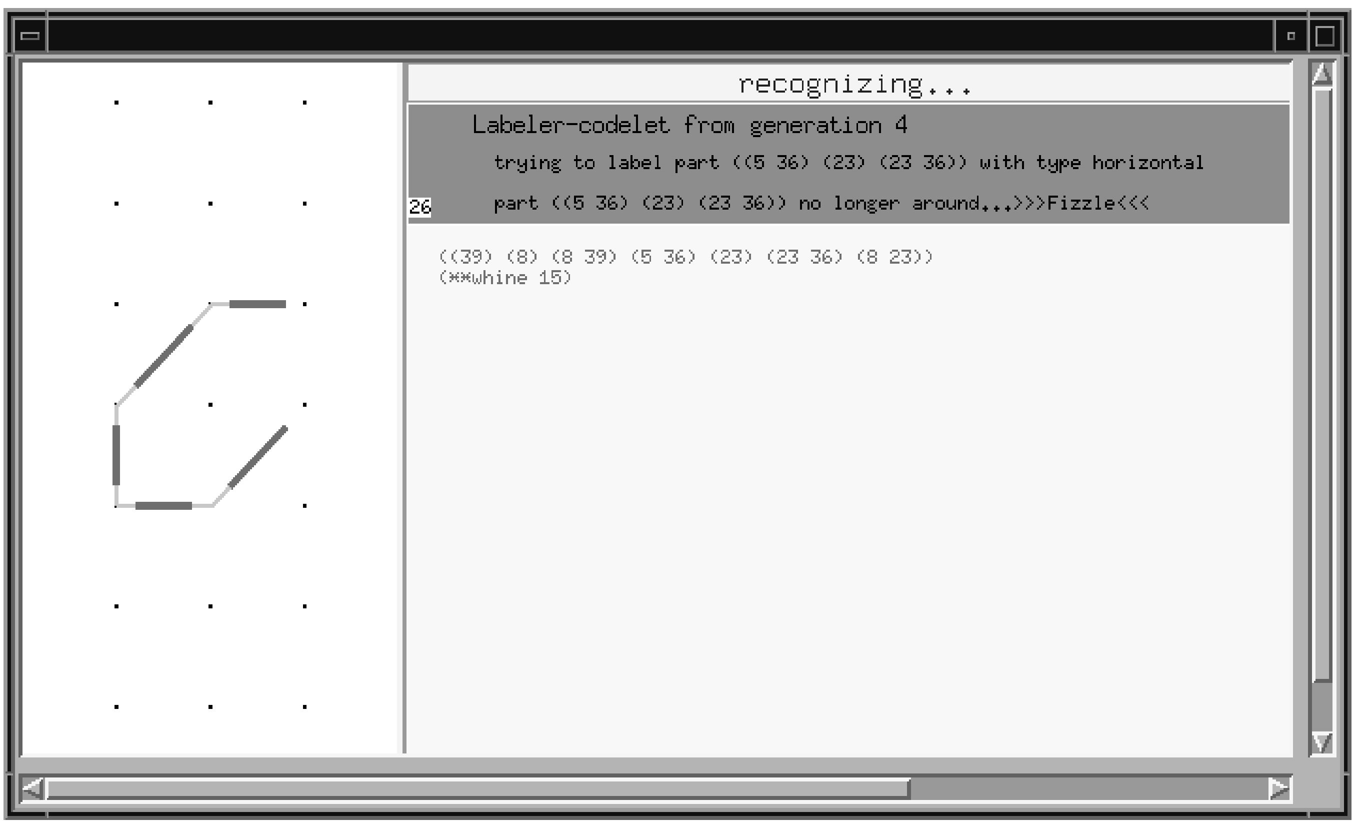


Figure 5.14: The two parts from the beginning have been combined to form a new, bigger part that must now be labeled.

By around 50[†] codelets, the new part has been labeled and is passed by a label-checker for activation-sparking. Codelet 58[†] is a sparking codelet that strongly activates the role *right bowl* in response to the labeled part. The resulting activations can be seen in Figure 5.15. The *right bowl* role is a member of 8 role sets — **a4, c, d, e, e3, g, g3, and q** — so processing is far from complete, as all of these wholes awaken and must "check things out". Figure 5.16 shows how each of these role sets becomes active once activation spreads from *right bowl.* Note that **c** is slightly more active than its counterparts. This is because the link between *right bowl* and **c** is shorter than links between *right bowl* and the other role-sets, since **c** is a one-role whole. In any case, each of the hypothesized wholes must be considered by an r-role checker before the system can discover that most of the hypothesized wholes are missing the rest of their roles and should be inhibited.

R-role checking begins, interrupted from time to time by role activation and activation spreading. A fight of sorts takes place between the r-role checkers and the activation-sparkers and activation-spreaders. That is, r-role checkers want to inhibit certain wholes (whose r-roles do not cohere with the current situation) and take activation away, but activation-sparkers and -spreaders (who pay no attention to the situation at the higher level of wholes) persist in re-activating these wholes through the activation of roles. Sometimes just as role-set activations seem to be converging on the **c**, activation gets pumped back into other competing wholes through the *right bowl* role. (Of course, the system itself is completely unaware of this emergent facet of its behavior.) The fact that *right bowl* is so common (one of the three most common roles of all, in fact), makes this battle more intense and less one-sided than usual. But eventually the battle subsides (since r-role checks always win out in the end) and **c** emerges the winner.

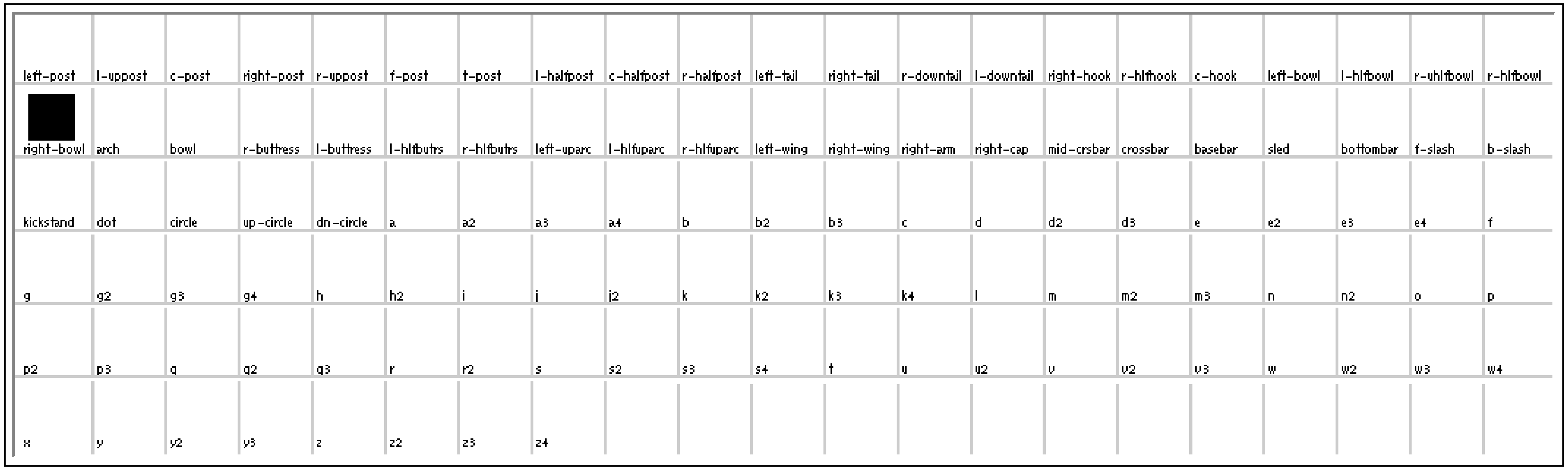


Figure 5.15: The lone role *right bowl* is strongly activated by the freshly labeled single part.

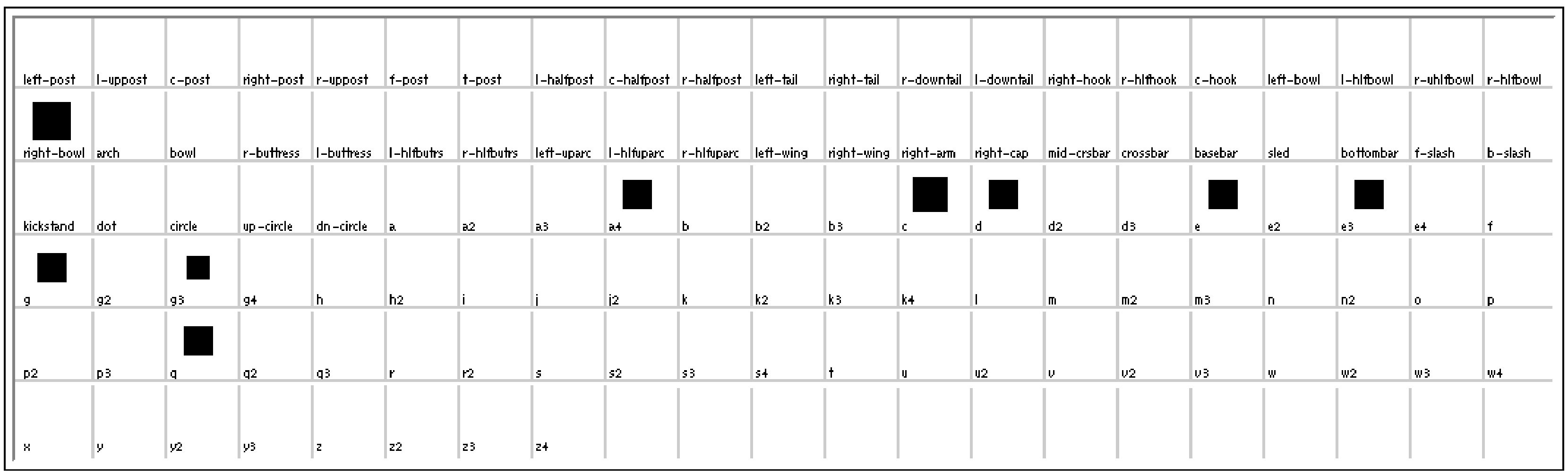


Figure 5.16: Activation-spreading causes many role-sets to become active, since *right bowl* is a very popular role.

A different route to the same answer — role subsumption

Another run on this same ‘c’ demonstrates role subsumption and a different route to the same solution. As usual, processing starts with a gluing-and-shaking phase. By 13 codelets after gluing/shaking, two parts have been formed. Labels begin to be attached to both parts, and by around 30† codelets, one of the bottom part is activating quite a few roles, but only very lightly, since it does not match any roles particularly strongly. See Figure 5.17. When the part activates roles, an activation-spreading codelet is posted to the Coderack.

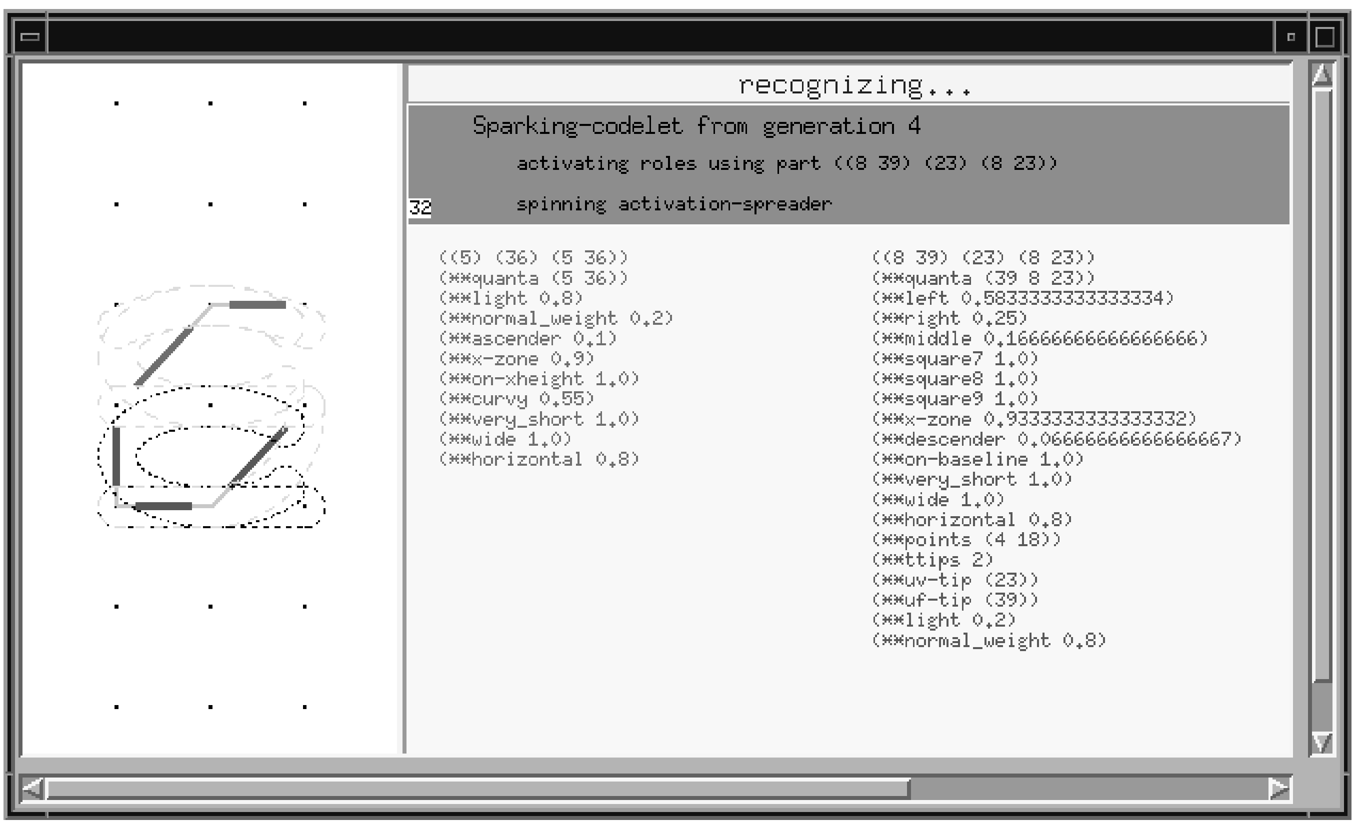


Figure 5.17: One of the two well-labeled parts (the bottom one) activates many roles lightly (resulting in quite a messy display).

Activation from this round spreads and r-roles temporarily take over processing. But by around 75† codelets, activation has pretty much decayed. The 79th† codelet is an activation-sparker bound to the top part. The top part activates three roles very strongly: *right-arm* as in **r**, *right-cap* as in **s**, and *crossbar* as in **t**. By codelet 100†, the bottom part is once again activating lots of roles lightly.

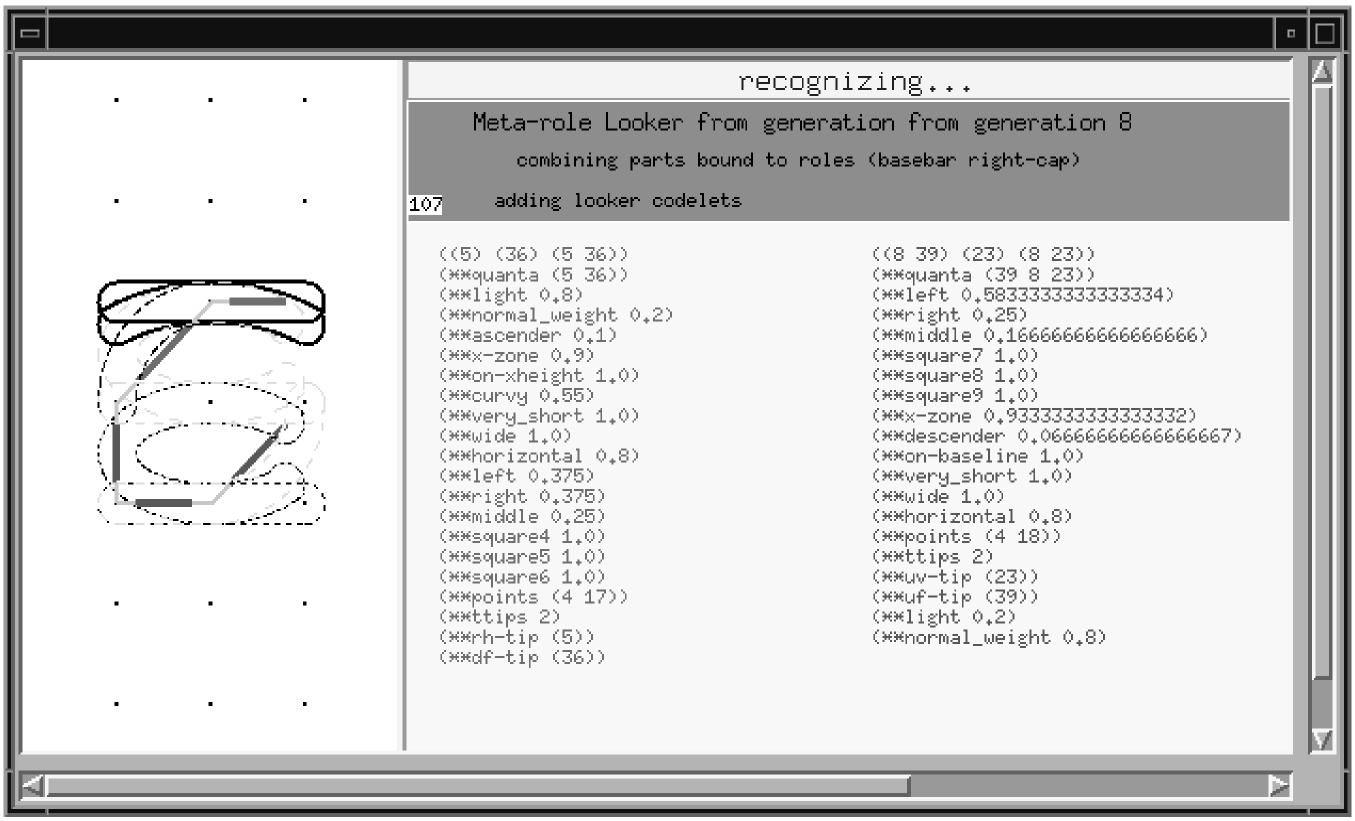


Figure 5.18: As a role-subsumption codelet (called "meta-role looker" here, for historical reasons) runs, two parts are combined together into a new part. This is a top-down process.

Because of the activation of these roles, some role-subsumption codelets have been posted to the Coderack. Such codelets are posted when a quick test shows that two roles that might possibly be combined into one are both simultaneously activated in the Conceptual Memory. In this case, the two active roles suggesting subsumption are *basebar* and *right-cap*, which can sometimes form a strong *right-bowl* if merged. The process of subsumption in this letterform (called "meta-role glomming" in the screen dump, for historical reasons) is shown in Figures 5.18 and 5.19. In some cases, the parts that fill two possibly subsumable roles do not jibe, and role subsumption does not take place.[3] During this run, however, the subsumed parts combine nicely into a new part. The combination of these two parts (each filling different subsumed roles) into a new part that *may* fill a subsuming role, should it survive the labeling process, is an example of one way in which top-down pressure from the Conceptual Memory strongly affects processing in the role model.

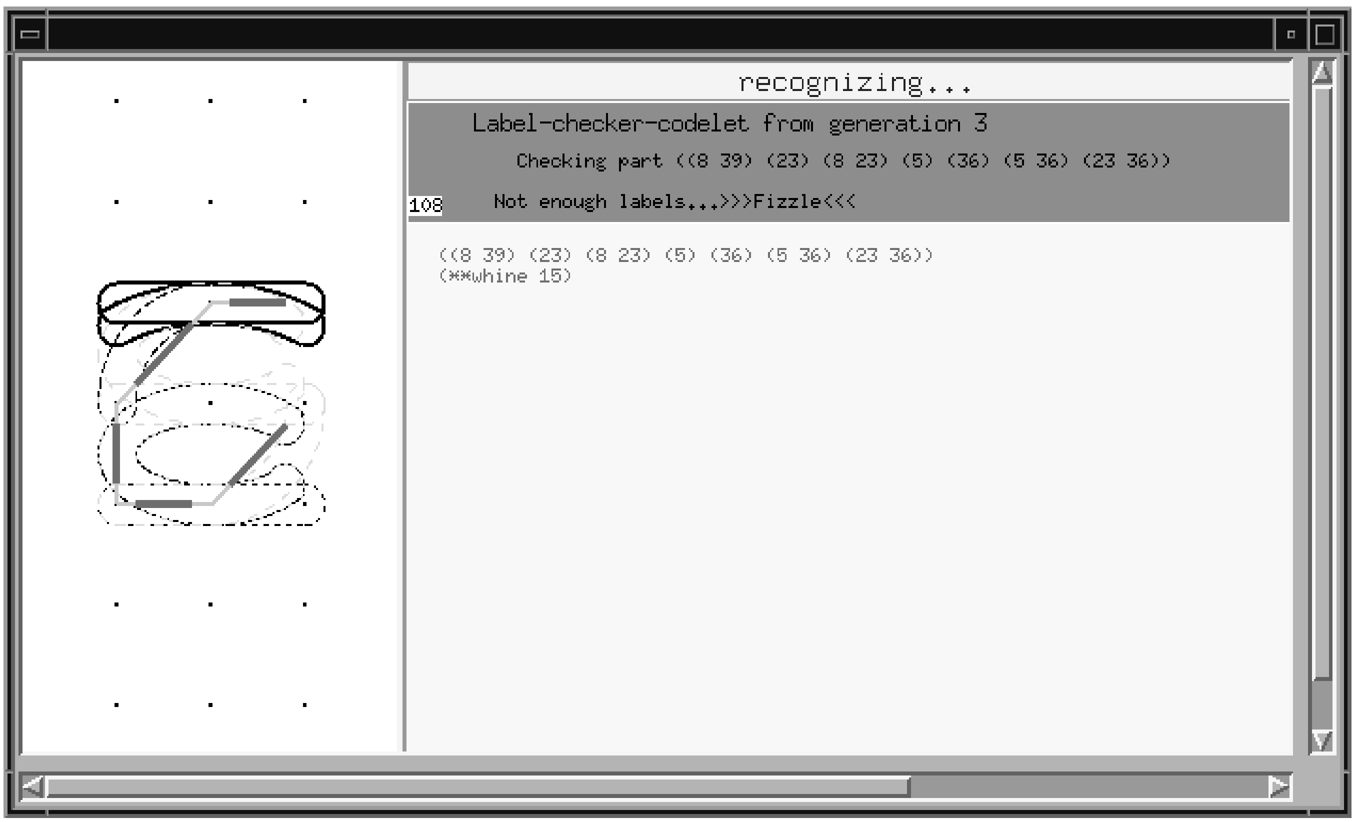


Figure 5.19: The result of role-subsumption is a new part, here consisting of all five quanta making up the letterform. Once again, the new part is subject to labeling in the usual way.

[3] For a more detailed explanation of role subsumption, see Section 4.3.11 of the previous chapter.

Now that a new part has been formed, the system turns back to labeling the part and activating roles. It takes around 100 codelets more to finish the recognition process on this run. During this period, many labels are attached to the new part, thus allowing *right-bowl* to be activated. Activation spreads to many wholes (as in the previous example), and r-role checkers run to see how well these wholes really work. Finally at codelet 198[†] (not counting gluing and shaking), **c** is declared the winner (through the usual lowering of temperature) and processing halts.

This example of role subsumption demonstrates the effects that top-down pressures from roles can exert. The role model's combination of bottom-up and top-down processing in a recognition and categorization task is rarely seen in AI models.

The two preceding examples emphasize the nondeterministic nature of processing in the role model. The fact that two such radically different paths to the recognition of one and the same 'c' exist shows the flexible nature of processing in the role model.

5.1.3 Recognizing an 'f'

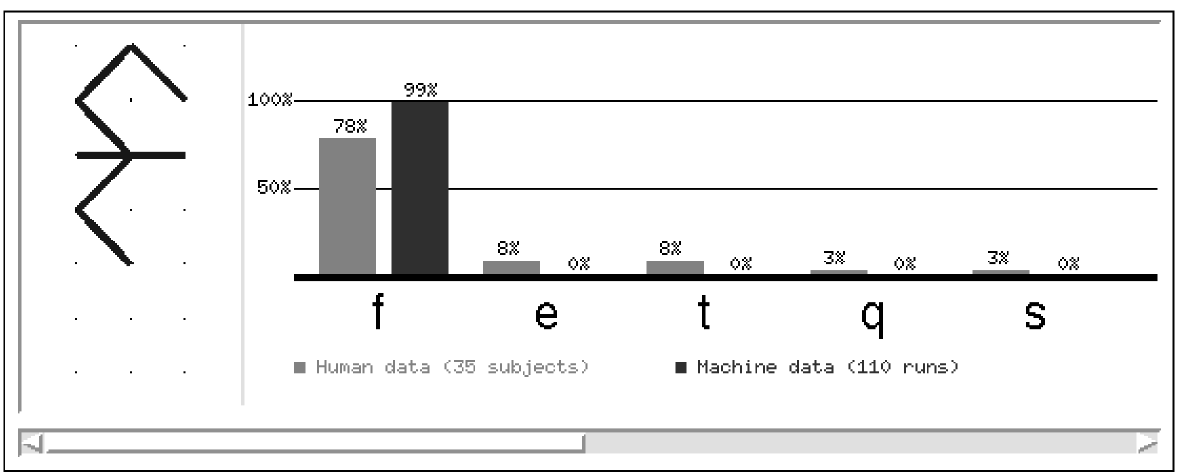


Figure 5.20: Results of 110 runs of the role model on a rather complicated 'f' are shown, along with results from the human experiments (see Chapter 7). Note that one run of the role model resulted in no answer.

Next we turn to a slightly more complicated run, this time on an 'f'. Like the 'i' and the 'c', this 'f' is classified correctly an overwhelming majority of the time by the role model (all but once, in fact). The fact that this somewhat nonstandard 'f' is so easily recognized provides a glimpse of the power of the model. The model's power becomes even more evident when its performance on several datasets is discussed below.

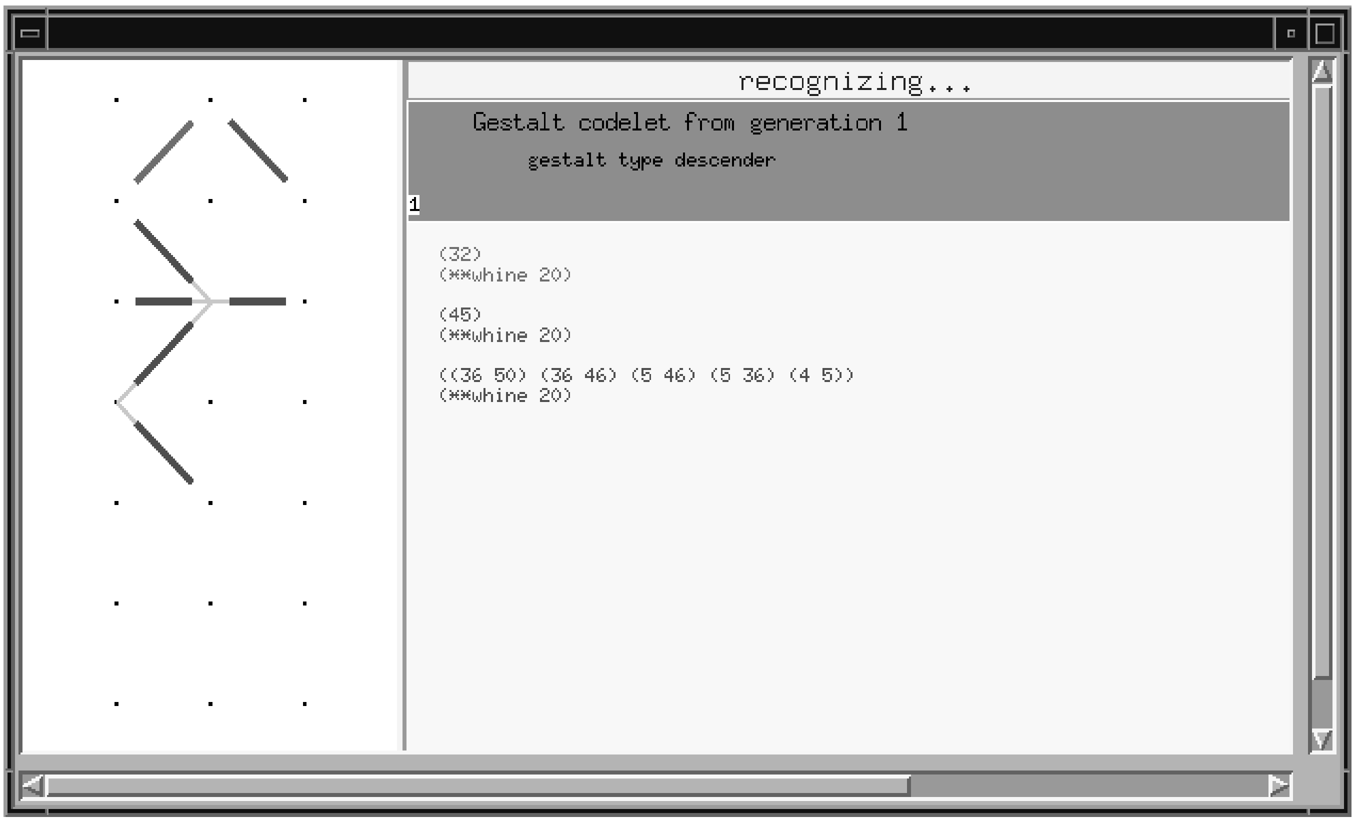


Figure 5.21: Ugly parts like the ones shown here sometimes result from the gluing and shaking process. There are three parts in this picture — two very small parts in the ascender zone, and one huge and very ugly part (with four tips) below the two small parts. Fortunately, the role model can easily recover from such inauspicious starts. (The gestalt codelet looking for "descender" here ends up fizzling.)

This run shows how the role model can recover from the formation of ugly parts at the outset. Because the gluing process is entirely bottom-up in nature and relies only on angles between quanta, shapes like this 'f' pose a problem for it. This "problem" does not really amount to much of an obstacle, though, since the part-rearranging codelets can recover quite easily (as will be shown). Notice the very poor parts initially formed by the gluing/shaking process. Meanwhile, a gestalt codelet, oblivious to any part-level concerns, merrily does its job.

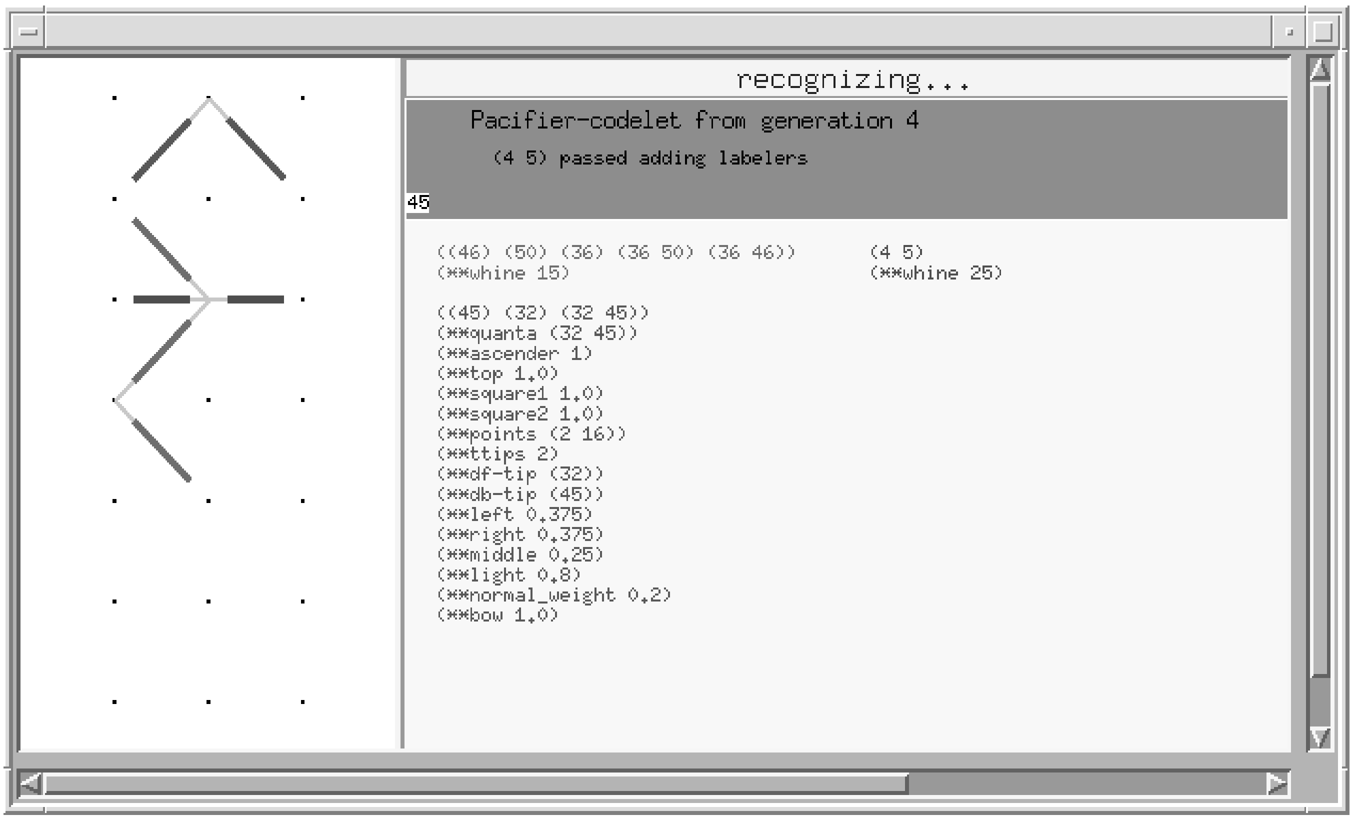


Figure 5.22: New parts formed in a bottom-up way by looker, breaker, and pacifier codelets. These three parts — one in the ascender zone, one being the crossbar, and one (overlapping the crossbar) zigzagging from the baseline into the ascender zone — are not ideal, however. (It is hard to tell from this picture, but the crossbar part has now been completely separated from the zigzaggy part that crosses it. This is very obvious when the color-coding can be seen.) In any case, the top part (which has several labels attached to it at the moment) is so bad that it eventually causes a top-down glommer to be posted.

By the 40th† codelet, the ugly part has been successfully broken up. Five codelets later, reasonable-seeming new parts have been formed. Though it is hard to tell from the picture, the largest initial part has been broken into two overlapping parts that cross each other in the middle. These two parts are much better, since they have only two tips each. The new parts get labeled in the usual way. The horizontal part strongly activates its proper role, *crossbar*, as would be expected. But the remaining two parts (one in the ascender zone, and the other stretching from the baseline into the ascender zone and crossing the crossbar) do not activate any roles very strongly. In fact, the top part activates roles so poorly that a top-down glommer codelet is posted. The results of this codelet's activity are shown in Figure 5.23. This demonstrates another way in which top-down pressure can be applied in the role model in order to "semantically adjust" parts.

Figure 5.23: New parts formed in a top-down way by a top-down glommer codelets. The parts now fit the traditional roles of an 'f'. At this point in processing, the roles of 't' are more highly activated than the roles of 'f'. As processing continues, however, the 'f' view will compete with and overpower the 't' view.

The action of top-down glomming tries to combine the very poor top part with the "weakest" of the other parts. In this case, the crossbar part is so strong that the remaining part is the obvious choice for glomming. After formation, the new part is subjected to labeling. Then role activation, activation spreading, and r-role checking occur in their usual interleaved way. Around 100 codelets later, the 'f' is recognized.

The zigzaggy nature of the part of the 'f' that fills the *f-post* role makes this 'f' is stylistically interesting. Since the role model is able to parse the 'f' in terms of its roles, it has access to deep-level stylistic information involving parts. The zigzaggy part can be subjected to further stylistic scrutiny which can in turn be supplied to other modules of Letter Spirit (those that handle design, for instance). A simple categorization whereby a letterform is called an 'f' and left at that is far less useful.

5.1.4 Seeing one 'k' two ways

The next letter on the agenda is the 'k' shown in Figure 5.24. The role model is able to correctly classify this 'k' on every run. What makes this 'k' especially interesting is that the role model in some cases parses it into the **k1** role set and other times sees it as a **k4**. These two conceptualizations of 'k' are closely related and involve forming three parts in slightly different ways. They are both shown in Figure 4.2.

Being able to see the same 'k' in two different ways requires an interesting sort of flexibility not often found in AI models. Although both conceptualizations lead to the same category label, they would clearly have different effects on the analogical design of other letterforms, depending on which of them was chosen.

The few runs that we have covered show how the role model works when recognizing particular letters. Some of the architectural features, including codelet processing, nondeterminism, conceptual flexibility, and emergent structure formation, were highlighted throughout. It is these sorts of mechanisms that endow the role model with its flexibility.

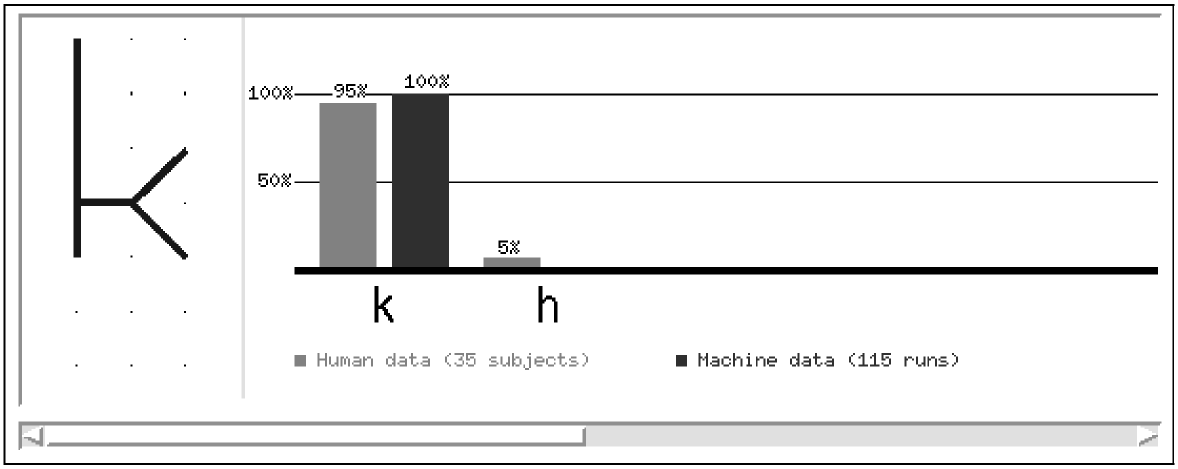


Figure 5.24: Results of 110 runs of the role model on a particular letterform are shown along with results from the human experiments (see Chapter 7). The role model is able to see this shape as a 'k' in two different ways, and never sees it as anything else.

5.2 Eighty-eight 'a's

Another way to illustrate the flexibility inherent in and emergent from the role model's architecture is to consider a series of runs on groups of letters. Doing this gives some idea of the sorts of letterforms the model is capable of recognizing and the sorts of letterforms it fails on. As mentioned in Chapter 1, there are two forces that play off each other during the design of gridletters: the *incentric* force pulls letterforms towards the category center in order to make them easily recognizable, and the *eccentric* force pushes letterforms towards the fringes of a category in order to make style more conspicuous. Figure 5.25, below, shows the extent to which the role model can handle cases where the eccentric force is very strong (as it tends to be, in gridletters).

The particular group of letters we discuss in this section is taken from an illustration in *Fluid Concepts and Creative Analogies* that appears on page 424 [Hofstadter and FARG, 1995].[4] The dataset consists of 88 lowercase gridfont 'a's. We ran the role model at least ten times on each of the 'a's in the original illustration, in order to

[4]The illustration was partially duplicated here as Figure 1.8 of Chapter 1.

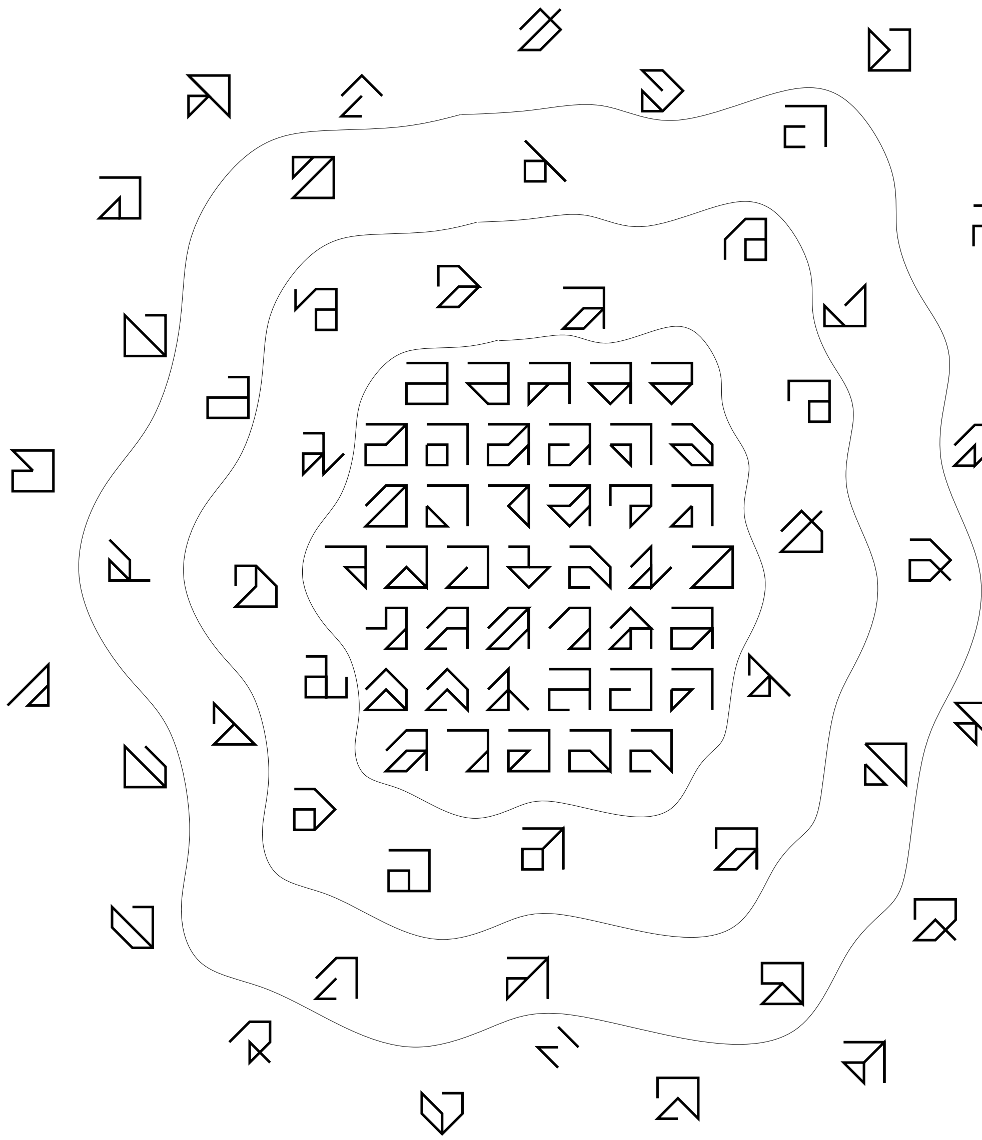


Figure 5.25: How the role model fares on 88 'a's. The role model was run 10 times on each 'a'. The 'a's in the center are always correctly classified. Moving out from the center, the first "ring" includes 'a's recognized from 90% of the time down to 50% of the time. The second ring includes 'a's recognized from 40% down to 20% of the time. The farther out in any ring letterforms are placed, the less often they are properly categorized. The 20 gridletters outside the rings were never recognized correctly.

discover which ones are easily recognized by the model and which are not.

In Figure 5.25 we have regrouped the 'a's of the original chart into a bull's-eye pattern, whose center is made up of 'a's that were correctly recognized 10 times out of 10. Concentric rings surrounding the center divide the remaining 'a's into "recognition bins", the first consisting of 'a's recognized from 5 to 9 times over 10 runs, the second being those recognized from 2 to 4 times. Finally, outside of the outermost closed curve are those 'a's that the role model never categorizes correctly. The more poorly a letter is categorized by the role model, the farther from the center of the picture it is placed. It is very interesting to note what the role model *can* do in terms of flexibly stretching its roles, and what the role model *can't* do. It seems that complicated models like Tabletop, Copycat, and the role model always have complex patterns of performance like this.

Overviews such as this provide a general feel for categorization in the role model. As can be seen from the letterforms in the center, the role model exhibits a remarkable flexibility in its categorization, making its notion of 'a'-ness very rich. Generally speaking, letterforms that are never properly recognized (those shown at the outskirts of the picture) are indeed letterforms at the "fringes of the category **a**", although there are a few exceptions that, by disappointing us, keep life lively.

The 'a's in this entire dataset are properly classified 62.7% of the time by the role model on average.[5] Errors break down as shown in Table 5.1.

The reader may wonder how the errors shown in the table compare to human errors over this dataset. The answer is, we do not know. Although we have often compared the role model's performance to that of human subjects, we have not done so in the case of this particular dataset.

[5]We did not include any guessing in this set of runs.

'a' taken for	percentage
unknown	23.6%
o	3.9%
n	2.8%
z	1.9%
r	1.8%
e	1.6%
b	0.8%
s	0.3%

Table 5.1: Error-making on the 88-'a's dataset.

5.3 What is a 'b'?

A similar dataset to the 88 'a's can be formed from 290 'b's pulled out of our extensive database of gridfonts. In order to get some idea about the expanse of the role model's internal categories, where their boundaries are, and what inter-category confusions are commonly made, we tested it on this much larger dataset. The results provide a clear window on just what the role model includes in its conception of 'b'.

In this case, we cannot show all of the tokens in a single "bull's-eye picture", as with the smaller study of 'a's. Instead we present a different chart. This new chart, shown in Figure 5.26, shows four groups of 'b's. The top group is made up of 'b's recognized correctly every time (10 of 10 runs) by the role model. There are a couple of things of note in this group. First is the range of letterforms the role model can properly classify *every time*. Role-stretching challenges represented (and properly handled) include: closure (whereby the *top* of a bowl does not properly touch a post); zone stretchings (whereby parts stray into ascender or descender zones); size stretchings (whereby letters are bigger or smaller than usual); and a fair amount of plasticity in the *left post* and *left bowl* roles. R-role plasticity is also evident in this set of tokens. It is clear from studying this top group of letters that the role model's notion of 'b'-ness is very rich indeed.

Role-level conceptual fluidity is evident in the second group as well, which consists

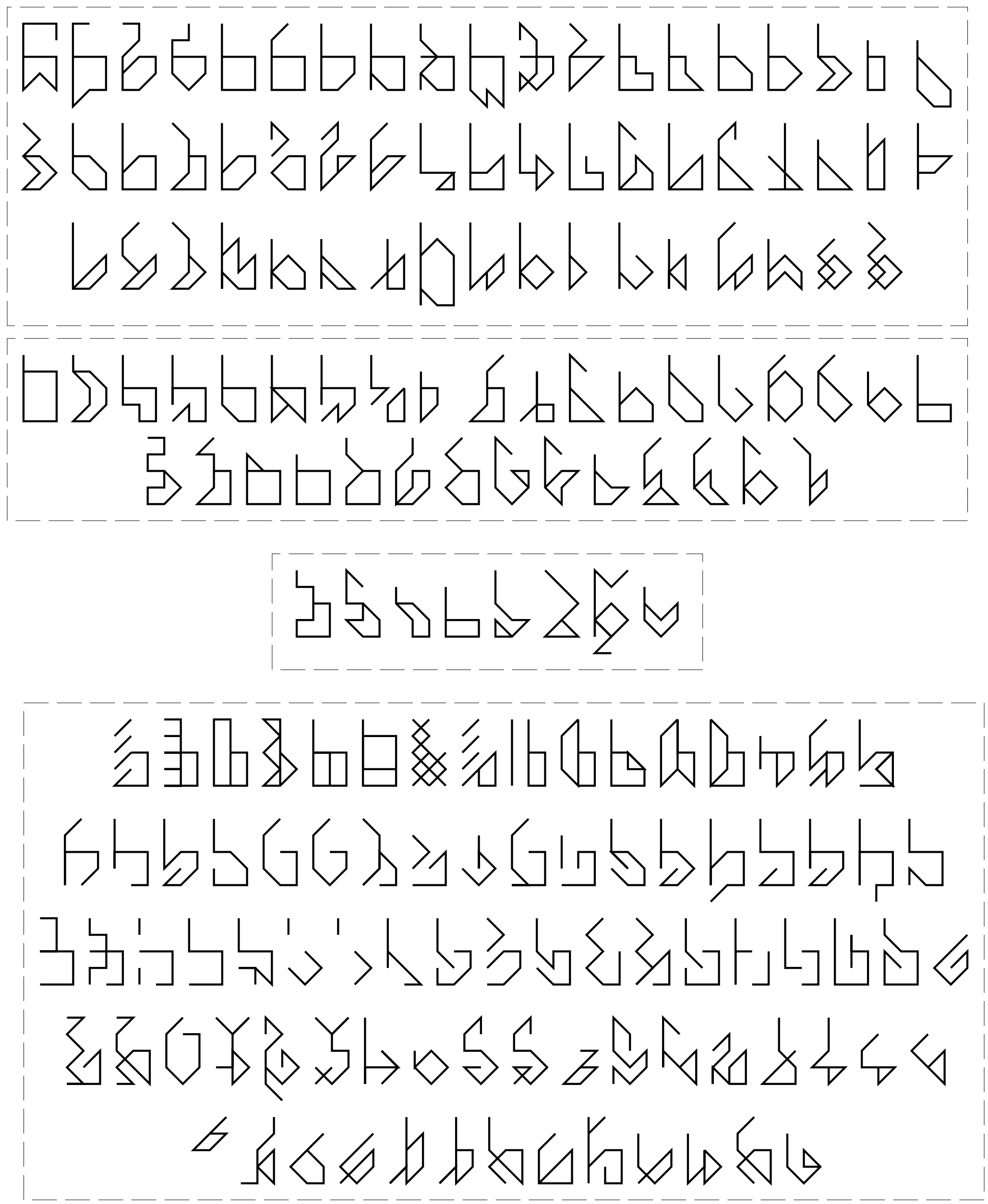


Figure 5.26: This chart shows performance of the role model on a large number of 'b's. Once again, the role model was run 10 times for each 'b'. Of the original 280 tested, 181 are shown here. The 'b's are arranged in four major groups. The first group is made up of 'b's that are always correctly recognized by the role model. The second group involves 'b's recognized from 80-90% of the time. Then the diagram skips over 20 tokens and resumes at a lower level of success. In particular, 8 tokens that were recognized 50% of the time are displayed in the third group. Then the diagram skips over some more tokens, and the bottom box contains tokens *never* correctly recognized by the role model. Between the 50% group and the never-recognized group there are 68 tokens not displayed.

of ‘b’s properly recognized between 80% and 90% of the time. In this group, role-level stretchings are slightly more extreme (*i.e.*, the norms of roles are violated to a greater extent.)

The 20 tokens that were recognized on 60-70% of runs are not displayed in the figure. However, the major trend that appears in the first two groups, whereby roles become more and more wildly filled, extends into this group as well.

The next group displayed in our Figure is made up of eight tokens that were recognized 50% of the time. These tokens are characterized by multiple stretchings and involve norm violations in both of their constituent roles at the same time.

Once again, a group of tokens — those properly recognized between 10% and 40% of the time — are not displayed. There were 68 such tokens.

The final group shown consists of tokens *never* correctly recognized by the role model. This group has been further subdivided into 5 lines:

1. Tokens shown on line 1 present “thickness” challenges. That is, the parts filling the *left post* role involve thick patterns of quanta (sometimes drawn as closed shapes and other times as regular patterns of lines). Although people do not seem to be challenged by thick parts of this sort, the role model cannot handle them at all. This is a weakness of the model that stems from its programmed-in tendency to see letterforms only in terms of thin lines rather than “filled-in” closed shapes (whether such shapes are filled in by white space or by other regular patterns). Enhancing the model to handle thick shapes of the sort seen here would probably require a number of quite radical extensions to the architecture, and has not been investigated. This level of vision is in a sense too retinal and too close to three-dimensional perception to be in the true scope of the Letter Spirit project. In fact, it was deliberately excluded from the outset, and gridfonts designed using these visual features were never intended as input to the program by the original designers.

2. Line 2 in the "never-recognized" group is made up of tokens with closure problems. The role model is probably too insistent on perfect closure, as can be seen here. People, but not the program, easily fill in shapes of this sort with "invisible" or "hallucinated" quanta, so that interrupted line segments are seen as jumping across gaps. Thus many of these letterforms would not present recognition problems to people, though they stymie the model. Handling "invisible quanta" in the role model should be possible. A mechanism whereby two short collinear segments could, under considerable top-down pressure, "hallucinate" such a quantum by some sort of mutual activation is entirely conceivable.

   One interesting subclass of tokens shown on line 2 is made up of "open circuit" tokens, in which a part (usually just a single quantum) has been swung open like a gate, opening the usual closed circuit of a letterform. People can easily swing such gates open and shut in their mind's eye. Such mental "gate-swinging" activity is far beyond the capability of the current role model.

3. Line 3 is composed of tokens having closure problems more difficult than those in line 2. These problems are compounded by complicated role-fillers that involve a fair amount of role-level norm-violation in their own right. The fact that the role model cannot recognize these tokens is not terribly disturbing.

4. Line 4 tokens are just plain weird. The fact that the role model cannot see these shapes as 'b's is probably *good*, at least in some cases.

5. On the other hand, some of the tokens shown in line 5 should probably have been properly categorized at least some of the time. There is, needless to say, only so much plasticity in the role model, and the model is certainly nowhere near as flexible as is the human visual/cognitive system. The role model's mechanical substrate shows through when it is exposed to tokens like these.

It is interesting to consider both the tokens that are properly recognized in this set and those that are not. Taken all together, these groups give an intuitive feel for what the role model is capable of, how flexible it is, and where the bounds of its flexibility are.

Similar studies could be made for each of the 26 categories that the role model knows about. Unfortunately, space constraints do not permit the presentation of such an undertaking here. (However, see Appendix A Figures 9.4, 9.4, and 9.4, for pictures showing how the role model performs on eighteen particular gridfonts.) In any event, although studies of this sort make for interesting pictures and are certainly worthy of mention, they give only a rough idea of *why* the role model makes the mistakes that it does, and have nothing at all to say about *what sorts* of conceptual mistakes the role model makes (*i.e.*, the categories into which it incorrectly classifies "missed" letterforms). There are two ways to get at information like this. One is to consider in detail specific tokens that the model cannot recognize. The other is to statistically analyze thousands of runs. We provide data from both of these approaches in the next two sections.

5.4 Letters the role model cannot recognize well

Although the role model is capable of recognizing a very large variety of gridletters from all 26 categories of the lowercase roman alphabet, it is obviously far from a perfect model. One way to understand its weaknesses as a model is to consider some of the particular letters that it cannot categorize as well as it should be able to. In this section, we discuss fourteen particular examples.

The fourteen poorly-handled letters are shown in Figure 5.27. The three numbers displayed to the right of each letter show the percentage of correct responses given by the role model on the displayed token recognized under three different conditions.

The first number was computed during a set of regular runs; the second, during runs with roles clamped to 25; and the third with roles clamped to 50. Role clamping, during which activation is continually fed into the roles of certain wholes throughout a run, improves categorization in half of the cases. As was discussed in the previous chapter, role clamping provides a way to bias the system towards a certain category, but does not force its hand completely.

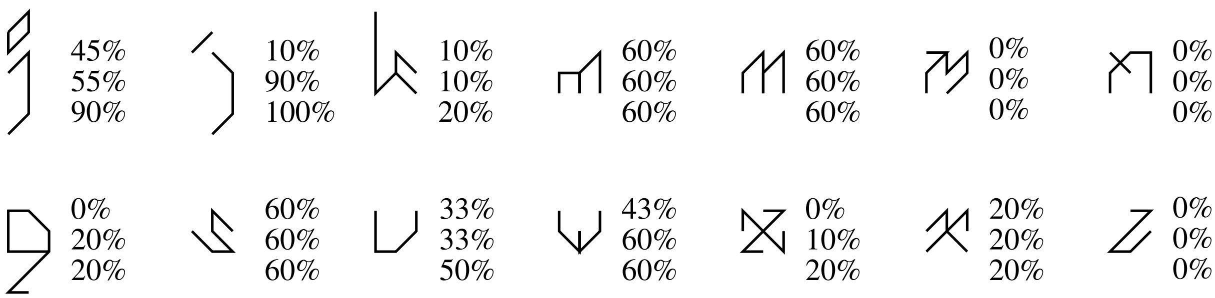


Figure 5.27: Fourteen tokens that the role model handles poorly. Next to each of these tokens are three numbers — all percentages. The first is % correct during regular runs; the second, % correct with roles clamped to 25; and, the third, % correct with roles clamped to 50. Role clamping appears to be helpful in some situations, and not very useful in others.

Each of the 14 tokens is handled poorly due to a variety of different factors. The first 'j' has a filled-in dot that the role model cannot see as a little filled-in "object", the way people do effortlessly. The second 'j' falls almost exactly between the two role-sets for **j**. In the ensuing fight, neither **j** nor **j2** wins. This is caused mostly by positional problems and a very low dot. "Fratricidal" fighting between role-sets representing the same category is a very rare phenomenon in the role model, but as this shows, it can happen. The next three tokens are simply hard for the model to parse into appropriate parts. In all of these three letters, the desired parts are too small for the role model to handle. The role model has a strong preference for parts having more than one quantum. This sometimes causes problems in letterforms with very small parts. Moving on to the right, the third 'm' is seen as an **n** every time. Actually, it is true that the letterform fills the roles of an **n** quite nicely (and can easily be seen as such by people), so this is not too serious a problem. The following 'n' is unrecognizable due to a minor weakness in the role model: there is no role-set for 'n' that allows for a serif on the upper left corner. A new role-set would have to be added to the system in order to handle this token, since the "lone quantum" in the upper-left corner cannot be glommed to the remaining part. As it is currently implemented, the gluing phase cannot glue together quanta that cut each other diagonally. This weakness rarely has serious consequences, but occasionally does matter.

We now move to the second line. The 'q' has too strange a tail (one that swings way over to the left), causing it to be seen more often as a **g**. The 's' has a whole host of strange, and, above all, *small* parts. As was mentioned above, the role model has a strong preference for parts having more than one quantum. In this particular 's', this problem is exacerbated by the orientation of the letterform itself, which leans over to the right substantially. The next shape is seen as a **v** more often than as a **u**, which is not a very serious problem, as it is indeed ambiguous. The 'w' once again illustrates problems with small parts. Because the rest of the letterform matches **v** so strongly, the small center post is sometimes overwhelmed. Small parts are occasionally overwhelmed by pressures from other letter categories, causing this sort of problem. The next two 'x's are made up of weird parts — requiring too much stretching for the role model to handle. The first 'x' is especially interesting since each of its four tips has the "wrong" orientation, but they are all systematically wrong in the *same* way. Getting the program to recognize the systematicity inherent in the wrongness of the tips would be a tricky problem indeed — an analogy problem in its own right, but at the level of low-level vision. Finally, the 'z' has a great deal of competition from **c**, since the one-quantum part at the top, which should play the role of *crossbar*, gets glommed instead to the *slash* part. Once again this weakness stems from a propensity in the model away from small one-quantum parts.

Together, these tokens illustrate some of the weaknesses of the role model. Nevertheless, these few weaknesses are not so serious as to cause the model a great deal of trouble in most situations.

5.5 A standard dataset and a confusion matrix

One of the driving concerns of this work is to develop a model exhibiting the kind of fluid categorization that is so natural for people. In order to determine the success or failure of our approach along these lines, we have compared our model with human data gathered in a set of psychological experiments. The experiments themselves are reported in Chapter 7. Psychological validation of the role model in terms of these data (and comparisons with the performance of other simple models) make up the contents of Chapter 8.

A comparison of this sort necessitates the development of a common dataset. We have formed such a dataset and introduce it here for the first time. Our entire dataset is made up of 544 letters, which can be divided into several non-exclusive subsets. Each subset will be given an appropriate (and hopefully mnemonic) name which will be used throughout the next few chapters. The full data set, which we call PSYCH, may be meaningfully separated into several non-exclusive subsets:

> NORMALS: This is a set of 388 tokens, representing each of the 26 lowercase roman letter categories from 9 to 24 times. All tokens are relatively strong category members, ranging in style from very standard to only somewhat stylized. The complete dataset is shown in Appendix A as Figure 9.2. Another way of looking at the NORMALS dataset is as a collection of letters taken from a variety of fairly standard but diverse typefaces. Thus the set as a whole has a number of diverse letters as its actual members, some of which are slightly more eccentric than their compatriots.

FONTS: This set consists of six entire gridfonts (shown in Appendix A as Figure 9.3), each a coherent set of 26 tokens meant to share the same stylistic identity (or *spirit*). The styles of these fonts range from somewhat stylized to highly stylized and "eccentric". There are 156 tokens in FONTS.

TRAIN: A subset of the NORMALS made up of 130 tokens, which represent each of the 26 roman lowercase letter categories exactly 5 times each. These were intended to be very central examples of each letter. Simple computer models of letter perception that were created as rivals to the role model were trained using this group. The first five tokens in each letter category shown in Appendix A as Figure 9.2 make up TRAIN.

NORMALS-minus-TRAIN (or, NORMALS−TRAIN): These 258 tokens represent each letter category from 4 to 19 times. The dataset consists of the tokens NORMALS with the tokens of TRAIN removed, and was created for testing the programs trained using TRAIN.

PSYCH-minus-TRAIN (or, PSYCH−TRAIN): The PSYCH data set minus the TRAIN data set (or, put another way, the FONTS dataset plus the NORMALS−TRAIN dataset.) This subset is used in the comparisons of Chapter 8.

Our dataset differs drastically from those usually found in psychological letter-perception work, in that it includes tokens from *more than one typeface*.[6] Each category is represented by at least 15 distinct letterforms that can be naturally divided into two classes of tokens — standard and eccentric. Because of the range of tokens within each category, and because of the fact the we have included stylistically-charged

[6] Citations supporting this claim can be found in the following three chapters.

tokens in our dataset as a whole, we feel that our stimuli capture far more of the natural variability among letters than do the datasets of most previous approaches. This leads to a much deeper and more thorough analysis of error-making than has been reported in past work on letter perception.

Table 5.2 shows a stimulus-by-response confusion matrix created by testing the role model on the PSYCH dataset. This category-level matrix was made by collapsing the 544 specific token-level confusion vectors ("This particular 'q' was taken for a 'g' 6 times") into 26 category rows, then summing ("Overall, 'q' was taken for 'g' 53 times"), and finally, normalizing ("Overall 'q' was taken for 'g' 8.5% of the time"). It represents data gathered from just over 50,000 runs of the role model. (Because of the stochastic nature of the role model, it is important to run the model multiple times on each token in order to properly evaluate its behavior.) Each token's row (before collapse) was calculated as the total performance over a large number of runs. More specifically, each token in the NORMALS dataset was run at least 100 times, for a total of 42,831 runs, and each token in the FONTS dataset was run at least 35 times, for a total of 7,485 runs.

The size and boldness of each entry in Table 5.2 reflects its magnitude. Entries greater than 80% are shown in boldface. Entries between 1.1% and 79.9% are standard size. Entries less than 1.1% are shown in scriptsize. Finally, entries of magnitude zero are shown as dashes. Note that the numbers across a row do not usually sum to a full 1.00 because some runs of the role model ended in no classification at all (all guessing aside).

The confusion matrix shows the percentage of the time that tokens belonging to a particular category were assigned to each of the 26 response categories. Correct-response percentages can be read down the diagonal.

response

stimulus	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
a	.725	—	—	—	.023	—	—	—	—	—	—	—	—	.050	.055	.002	—	—	—	.003	.002	—	—	—	—	.002
b	.009	**.866**	—	—	.003	—	—	.033	—	—	—	.019	—	.006	—	—	—	—	—	.002	—	—	—	—	—	—
c	.002	—	**.820**	—	—	—	—	—	—	—	—	—	—	—	—	—	—	.004	.066	—	—	—	.002	—	—	—
d	—	.002	—	**.870**	—	—	—	.011	—	—	—	.050	—	.002	.003	—	.001	—	—	—	—	—	—	—	—	—
e	—	—	.038	—	**.804**	—	.001	—	—	—	—	—	—	.016	.044	—	.022	—	.004	—	—	—	—	—	—	.004
f	.002	—	—	—	—	.795	—	.007	—	—	—	—	—	—	—	—	—	.002	—	.100	—	—	—	.001	—	—
g	—	—	—	—	.001	—	**.801**	—	—	—	—	—	—	.010	.002	.008	.041	—	—	—	.014	—	—	—	.039	—
h	—	.036	—	—	—	—	—	**.800**	—	—	—	.008	.040	—	—	—	—	—	—	.019	—	.002	—	—	—	—
i	—	—	—	—	—	—	—	—	**.939**	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
j	—	—	—	—	—	—	—	—	.045	**.876**	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—
k	—	.009	—	—	—	—	—	.045	—	—	.667	—	—	—	—	—	—	—	—	—	—	—	—	.049	—	.003
l	—	.002	—	—	—	—	—	—	—	—	—	**.896**	—	—	—	—	—	—	.018	—	—	—	—	—	—	—
m	—	—	—	—	—	—	—	.004	—	—	—	—	.790	.054	—	—	—	—	—	.003	—	—	—	—	—	—
n	—	—	—	—	—	—	—	—	—	—	—	.002	.001	**.801**	—	—	—	—	.001	.005	—	—	—	—	—	—
o	.015	—	—	—	—	—	—	—	—	—	—	—	—	—	**.862**	—	.001	.001	—	—	—	—	—	—	—	—
p	.007	—	—	—	.001	—	.012	—	—	—	—	—	—	.002	.013	**.868**	.002	—	—	.001	.004	—	—	—	.028	—
q	.002	—	—	.001	.001	—	.085	—	—	—	—	—	—	.002	.026	.001	.781	—	.002	—	.006	—	—	—	.021	—
r	—	—	.020	—	—	—	—	—	—	—	—	—	—	.056	—	—	—	**.850**	—	—	—	.019	—	—	.001	—
s	—	—	.001	—	.001	—	—	—	—	—	—	—	—	.003	—	—	—	.074	**.800**	—	—	—	—	—	—	—
t	—	—	—	—	—	.011	—	—	—	—	—	—	—	—	—	—	—	—	.002	**.955**	—	—	—	—	—	—
u	—	—	—	—	—	—	—	—	—	—	—	—	—	.001	—	—	—	—	—	—	.796	.132	—	—	—	—
v	—	—	.003	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	.002	**.962**	—	—	—	—
w	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	.005	.066	**.890**	—	—	—
x	—	—	—	—	—	—	—	—	—	—	—	—	.006	.001	—	—	—	—	—	—	—	—	—	.758	—	.002
y	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	—	.002	.007	.004	—	**.901**	—
z	.005	—	.057	—	.006	—	—	—	—	—	—	—	—	.055	—	—	.001	—	.025	—	—	—	—	—	—	.551

Table 5.2: Confusion matrix of the role model's performance on PSYCH.

The unidirectional error rates shown in the table can be averaged together to provide a bidirectional category error rate. As an example, consider 'a's and 'b's. 'a' tokens were properly categorized 72.5% of the time, and 'b's 86.6% of the time. 'a's were never taken for 'b's, while 'b's were taken for 'a's 0.9% of the time. The resulting bidirectional category error a↔b is thus 0.45%.

It is informative to compare the confusion matrix shown here with the human confusion matrix reported in Table 7.2. This will be done, using a variety of standard statistical techniques, in Chapter 8.

5.6 Systematic variation of some key parameters

In order to investigate the importance of various parameters in the role model, we introduce two datasets. The first, called *easy test*, is made up of 26 very normal and easy-to-recognize gridletters, one from each category. The second, called *hard test*, is made up of 26 harder-to-recognize gridletters, one from each category. The idea was to compare the performance of the role model in its original guise to that of various degraded or otherwise manipulated versions.

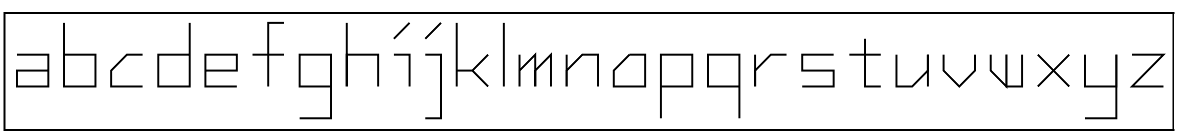


Figure 5.28: The *easy test* dataset, made of 26 letterforms, one from each category. All are very easy to recognize, given default parameter settings in the role model.

Figures 5.28 and 5.29 show the two datasets. Although neither of them is a stylistically coherent gridfont, these two collections of 26 letters are made up of either very-easily-recognized letterforms (*easy test*) or hard-to-recognize letterforms (*hard test*). By noting performance over the two datasets, we can establish a metric with which altered models can be compared. It is important to test both the recognition of easy letters and that of hard letters, since system performance degrades differently in either of the two conditions.

The default role model (as used in the experiments above and in those presented in Chapter 8) is able to recognize all of the *easy test* tokens 99.3% of the time over more than 100 runs on each token. The average number of codelets per run over all of the runs is 366. The average temperature at the end of the runs is 4.3. These numbers will serve as a baseline for comparison throughout the rest of this section.

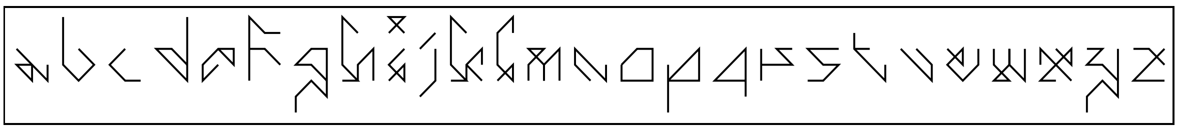


Figure 5.29: The *hard test* dataset, made of 26 letterforms, one from each category. All can be recognized by the role model at least some percentage of the time with default parameter values. This collection of gridletters could be considered an "anti-gridfont".

The default role model is able to recognize all of the *hard test* tokens 84.3% of the time over more than 100 runs on each token. The average number of codelets per run over all of the runs is 1,574. The average temperature at the end of the runs is 6.2. These numbers will serve as a second baseline for comparison throughout the rest of this section.

5.6.1 Eight versions of the role model

Eight new manipulated versions of the role model were formed by changing the values of key parameters. They are:

- *No gestalt codelet:* The gestalt codelet was removed from the role model in order to determine the effect of Conceptual Memory priming through gestalt activation and inhibition.

- *Maximum roles activated:* When roles are activated by the activation-sparker, there is usually a maximum number of roles that are allowed to be activated simultaneously by one part. This parameter changes as roles are loosened, starting at 7 roles, moving to 15 roles, and finally ending at 20 roles. For the "max roles" test, this value was permanently set to 47 (the total number of roles currently in the model), meaning that a part was allowed to activate anywhere from 0 to 47 roles at one time.

- *Completely loose roles:* For this version of the model, roles were loosened completely at the outset of each run. The object was to determine what effect completely loose roles have on recognition.

- *Equal urgencies:* All default urgency values were set to the same value — 10.

- *Urgency reversal:* Default urgency values were reversed. The "low urgency" parameter was set to 100, "medium urgency" to 50, "high urgency" to 25, and "very high urgency" to 10.

- *R-role whack low:* The default r-role penalty parameter was changed from 10 to 2. This essentially weakened the role of inhibition in the model.

- *R-role whack high:* The default r-role penalty parameter was tripled from 10 to 30, strengthening inhibition in the role model.

- *No gluing/shaking:* The standard gluing and shaking process usually used to determine initial parts was scrapped. Instead, each quantum in a letterform was made into an initial part at the outset of a run. Part glomming, part breaking, and part labeling were allowed to take place as before, once single-quantum initial parts were formed.

Tables 5.3 and 5.4 show the key performance values — accuracy, number of codelets per run, and final temperature — of all eight manipulated models and the default role model. In terms of accuracy, performance on the *easy test* degrades the most

	Easy test results								
	Default	No gestalt	Max roles	Loose rls	Equal urg.	Urg. rev.	Low r-role	High r-role	No glue
Acc.	99.4	98.2	99.5	91.0	99.6	99.3	87.2	99.8	95.3
Cdlts	366	362	444	554	369	511	924	554	780
Temp	4.3	4.3	4.1	4.4	4.5	12.1	22.3	5.7	3.5

Table 5.3: Performance of several variants of the role model on *easy test*.

	Hard test results								
	Default	No gestalt	Max roles	Loose rls	Equal urg.	Urg. rev.	Low r-role	High r-role	No glue
Acc.	84.3	80.7	65.6	71.7	81.7	84.0	36.8	39.8	73.0
Cdlts	1574	1462	979	1164	1454	1932	2132+	1307+	2274
Temp	6.2	5.6	4.6	5.4	6.2	6.3	—	—	7.4

Table 5.4: Performance of several variants of the role model on *hard test*.

in the "low r-role" condition, suggesting that the r-role whack parameter, controlling inhibition, is the most critical parameter in the role model. Set too low, this parameter dampens the critical effect of inhibition in the model. Of all model variants tested on *easy test*, the "low r-role" model takes the longest to complete a run on average. In addition, average final temperature is very high. This trend is apparent in the *hard test* runs as well, where once again the "low r-role" model suffers the worse degradation in performance. Interestingly, the "high r-role" model, in which the default inhibition level is tripled to 30, performs fairly well on *easy test* and very poorly on *hard test*. This is most likely because the role-filling parts of the letterforms of *easy test* match r-role criteria quite well, and thus are never severely punished; while in *hard test*, roles have messier relationships to each other, resulting in an unacceptably large amount of r-role punishment if the whack parameter is increased. It is clear that an intermediate amount of inhibition has been correctly chosen as the role model's default. It is also clear that inhibition is a central and important feature of the role model.

After the "low r-role" condition, the "loose roles" condition shows the second worse degradation. We were already aware of the fact that roles should not be loosened prematurely, for the reasons presented in Section 4.3.10 of the previous chapter. Our experiment here confirms our earlier reasoning. *Hard test* performance in the "loose roles" condition is also degraded substantially. Role loosening must happen gradually, so that normal role matching can be attempted before too much radical role stretching is allowed.

The "noglue" model also degrades a fair amount on both the *easy test* and the *hard test*, showing that the gluing and shaking process that we have implemented in order to get a set of stochastically-determined initial parts is a reasonable one. Apparently, the gluing and shaking process starts the role model off in a reasonable direction for most runs. It is especially noteworthy that the most severe degradation in the "noglue" model is apparent in the number of codelets that it takes to complete a run. The role model is able to (mostly) recover from having single-quantum initial parts on the *easy test* dataset. Thus its accuracy rate drops only a bit, though the codelet count is more than twice its normal value. For the *hard test*, as the codelet count increases, accuracy also drops, suggesting that recovering from single-quantum parts is much harder given strange letterforms.

All of the remaining conditions show only slight degradation on the *easy test*, with accuracy values remaining very near those of the default role model, and codelet counts being only slightly larger. The "max roles" model shows a slightly increased codelet count, but performance is generally similar to the default model. The "equal urgency" model's performance is almost exactly the same as the default model's, and the "urgency reversed" model's performance is hampered only by an increased codelet count. This was to be expected, as changes in urgency affect which codelet gets picked to run when. The urgency-changing tests are unfortunately less informative than they should be, since formulæ for internally determining some critical urgency values — that is, those not reliant on the default urgency parameters — remained unchanged. The "no gestalt" model's performance is also quite similar to performance of the default role model, with the accuracy rate falling due to much worse performance on the letter 'o'. This is a direct result of 'o's reliance on the notips gestalt codelet for quick and accurate recognition. Other than the clear effect on 'o', gestalt priming plays only a minor part in the role model.

Some of these *easy test* results carry over into the *hard test* case as well. Once again, the urgency-changing experiments were less informative than they should have been. Similar effects on performance were noted — namely, recognition taking much longer in the "urgency reverse" condition than in the default condition. The "no gestalt" model showed slightly more degradation on the *hard test* than it did on the *easy test*, but, predictably, the main change came in the recognition of 'o'.

The "maximum roles activated" model shows a much greater drop in accuracy on the *hard test* than it did on the *easy test*. This has a direct relation to how well the parts of letterforms in the two tests match roles. The less standard parts of the *hard-test* letters tend to activate a greater number of roles than do the very standard parts of the *easy-test* letters. Thus a change in the maximum number of roles allowed to be activated by one part strongly affects performance on the *hard test*.

All of these experiments serve to shed some light on key system parameters in the role model. It is interesting that in every case in which a parameter was changed, performance in the model degraded along some dimensions. The importance of some parameters — notably, the r-role whack parameter — is much greater than the importance of others — the default urgency values, for example. Some design decisions made during the course of programming the model — for example, the implementation of the gluing-and-shaking procedure, or the implementation of the role-loosening schedule — seem to be justified by our experiments. Performing experiments of this sort in order to determine the effects that design decisions (made during the course of programming) have had in the final model is both entertaining and informative.

In general, these experiments with degraded versions of the role model result in less striking conclusions than similar experiments carried out on the Copycat model [Mitchell, 1993, Chapter 8]. This is most likely a reflection of the level of the task being undertaken. In the lower-level task of letter recognition, there is far less competition between "plausible" or "reasonable" solutions for a given run than can be found in Copycat's letter-string domain.[7] In other words, the basins of attraction in the letter recognition task are larger. If one imagines the running model as a ball rolling on a hilly surface, tweaking parameters can be likened to lightly "hitting" the ball as it rolls around. In the case of the role model, recovery from such a hit is often possible since the basins of attraction (representing a particular answer) are so large. In the Copycat experiments, such hits often are enough to cause the ball to veer off into a new basin of attraction.

5.7 Summary

In this chapter we have shown the role model's performance on a large number of diverse runs. Analysis at all levels, ranging from the details of which-codelet-runs-when in a particular run to overall trends and accuracy values, has proven to be valuable and revealing.

In the remainder of Part II of the thesis, we have two goals. The first is to relate the role model to other models of letter recognition in the literature. The second is to support the role hypothesis and its implementation, the role model, with empirical data from human studies. The PSYCH dataset introduced above plays a key role in the attainment of our second goal, and will need to be referred back to as our story unfolds.

[7] See the discussion of Section 3.2.9 for some example Copycat problems with many possible answers.

CHAPTER SIX

Competing Models of Letter Recognition

Not surprisingly, a large body of work on letter perception has been generated over the last three decades. What is surprising, though, is the apparent division of that work into two camps with different approaches. These camps can be roughly characterized as the "engineering" approach and the "psychological" approach. On the one hand, much work has been done by the engineering camp under the rubric of Optical Character Recognition (OCR). The engineers are interested in actual working programs (hopefully easy to implement, efficient, and fast) to use in real-world applications. In general, they do not care how people recognize letters. On the other hand, the cognitive scientists of the psychological camp would like to explain human letter recognition. They build models to try to explain the data they gather through human experimentation. They *do* care how people recognize letters, but their models are often not actually implemented as running programs.

Unfortunately, there has been little cross-fertilization between the two camps. OCR research tends to stress low-level processing and prototype or template matching (although a few more sophisticated model-based approaches stand out in the crowd), while cognitive psychology generally stresses the importance of high-level letter categories with hierarchically-structured representations. Our letter-recognition work can be seen as an attempt to advance research in both camps simultaneously, as we are building a working computer model that is informed by a theory of human cognition. Since humans are by far the most sophisticated letter-recognition systems in the world, it behooves us to understand how they work and to co-opt human solutions for use in our computer models.

This chapter is divided into two large sections. The first, called "External Competition", deals with competing models of letter recognition developed by researchers outside of CRCC. The second, "In-house Competition", introduces two simple models we have developed ourselves as rivals of the role model.

We begin with the introduction of an overall framework covering all models of letter recognition (both psychological and engineering-style). We then present a bird's-eye view of the opposing camps, discussing where in the framework a few particular models can be fit. Next, we swoop down to consider two sophisticated models in detail — one from each camp: Thomas Sanocki's psychological model, and Hinton and Williams' mathematical OCR program. Last, we will unveil two simple models that we have developed ourselves (in parallel with the role model). These models are meant to illustrate fundamental flaws in two common AI approaches to recognition. One important advantage of having developed these last two models is that we can test them, alongside the role model and human subjects, on the same data sets.

6.1 A framework for approaches to letter perception

We find it useful to divide the many systems developed for letter recognition into two distinct groups or approaches. In first approximation, these two general approaches might be labeled "feature-based" and "model-based". However, we will not use these labels because they are too simplistic and thus misleading. Although these approaches roughly correspond to lines of research coming out of the "engineering" and "psychological" camps respectively, the story is slightly more complicated. The complication stems from the fact that a few renegade OCR models make use of perceptual hierarchies, and a few early psychological approaches use simple feature lists. Because these models violate the usual standards of their respective camps, any distinction along these lines is slightly less clear than it should be. To remedy this confusing state of affairs, we introduce a new framework for thinking about general approaches.

Both Figure 6.1 (on the next page) and Table 6.1 (below) show different aspects of the two groups that, for lack of better names, we call the "Flat Approach" and the "Structured Approach". This division is based on five critical dimensions: representation, process, reference, relation, and result. Each of the five dimensions is actually a continuum with most models falling somewhere in between the opposing ends (see the bottom of Figure 6.1). In Table 6.1, the extreme opposing ends of each dimension are listed under the appropriate approach.

Dimension	**Flat Approach**	**Structured Approach**
Representation	Category models based solely on collections of features or templates	Hierarchically structured internal category models (part→whole hierarchies)
Relation	Lists of features having little or no relation to one another	Complex structural relations between parts
Process	Non-hierarchical processing of features (usually using statistical distance)	Explicit processing of parts (parts compete to explain low-level data)
Reference	Simple categorization with no intermediate-level parsing	The ability to refer to intermediate-level parts after they have been formed from sensory input data, and to use them in further processing (*e.g.*, extraction of style information)
Result	Category label	Category label plus a parsing in terms of parts

Table 6.1: The five (continuous) dimensions of the structured/flat framework.

At the bottom of Figure 6.1, we list a few names (of specific models or authors) under each approach. Since the structured/flat distinction is actually a continuum,

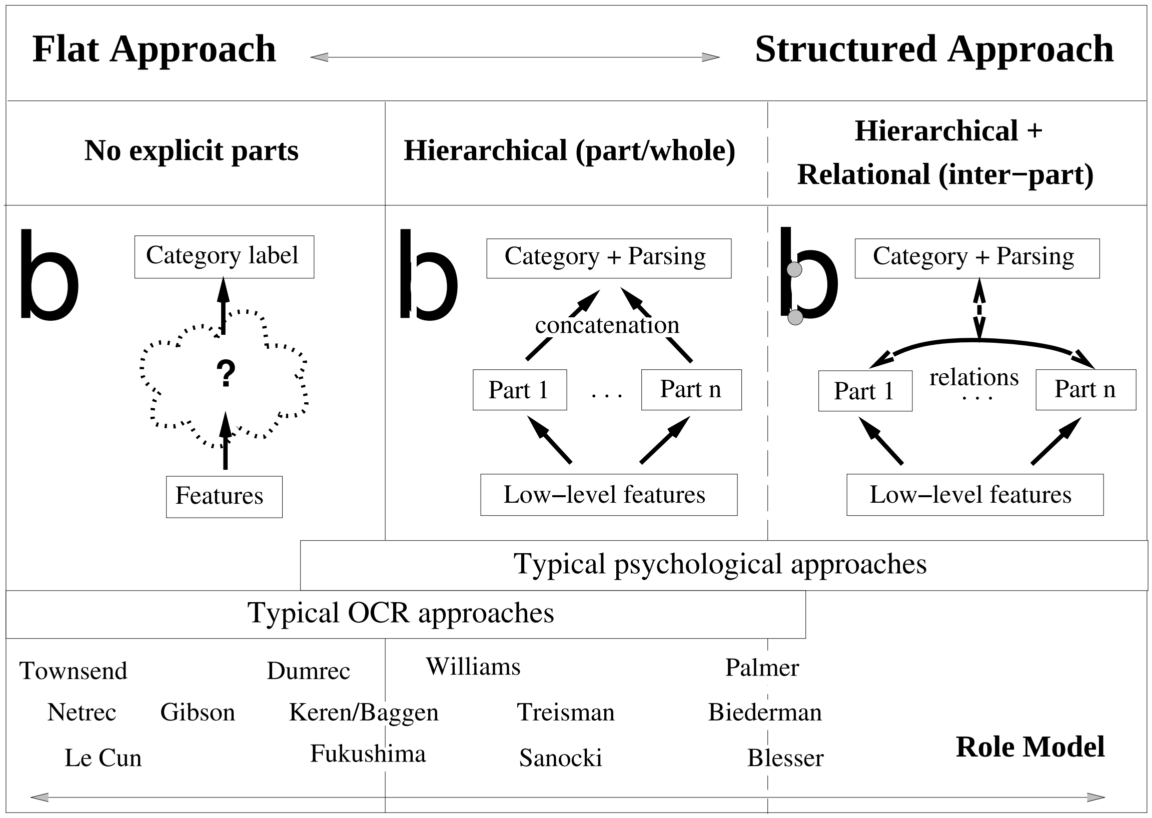


Figure 6.1: We find it useful to divide letter-recognition models into two distinct approaches, as shown above. The "Flat Approach" is characterized by systems with no explicit part processing and, thus, no hierarchical or relational structure. The "Structured Approach" includes systems having explicit category models, usually involving hierarchical and/or relational structures made up of parts. Structured systems do more than simply categorize — they also parse input data into parts that may be used in post-categorization processing. Systems that make use of explicit part processing involve the use of two kinds of "structure". Namely, hierarchical systems have a built-in part/whole structure. So too do relational systems, but parts in relational systems are further constrained according to their relations with other parts. At the bottom of the chart, pointers to some common systems are listed under either approach.

we have tried to place the names along the continuum as a function of the amount of structure or flatness that each involves. Thus some fall along the "borders". These names serve as pointers to letter-perception models that will be introduced throughout this chapter. Referring back to these charts may provide a good way of understanding how specific models relate to each other in the big picture.

Structured approaches have what we call "internal category models", which get used during processing. The best kinds of category models feature constituent parts that (at least in the most sophisticated models) have specific constraints on their inter-part relations. Throughout the process of categorization, low-level input data are filtered and chunked into intermediate-level parts. In the end, the fruits of this "parsing" — what we have been calling "parts" — are available for further processing. Categorization using the structured approach thus results in much more than a simple category label at the output end.

The Structured approach can be subdivided into two classes according to the *kind* of representations involved in a system's internal category model. These two classes correspond to groupings along the *relation* dimension. Both of these classes — labeled "hierarchical" and "relational" — are shown in Figure 6.1. Hierarchical structure is exemplified by part$\rightarrow$whole hierarchies. Obviously, every categorization model making use of explicit parts will necessarily include this kind of structure (as the parts taken together make up a whole). The second and arguably more interesting kind of structure is relational structure. This kind of structure supplies constraints and guidelines on inter-part relations. For example, a post and a bowl must be related in certain ways to make a 'd' (**c+l=d** given the right relations, but given wrong relations they may make **cl**). More specifically, they should touch in two specific places, form a closure, etc. In the role model, this type of structure is coded into the r-roles. Note that we do not count simple concatenation (or, more accurately, "co-presence") as a kind of relational structure. This allows us to distinguish between classes of models having simple hierarchical structure and those having both hierarchical *and* relational structure.

As is indicated in the figure, our role model falls under the structured approach to letter recognition and has both hierarchical and relational structure. We believe that parsing a letterform into its constituent physical parts and aligning them with pre-existent abstract roles is not just the key to the perception of letterforms in a large variety of styles, but also the key to letter design. The line of reasoning behind the latter claim can be briefly sketched as follows: 1) in order to be able to properly model design it is vital to include perception in the model; 2) perception involves parsing sensory data into hierarchical perceptual structures aligned with hierarchical conceptual structures (like wholes made of roles); 3) we believe that analogy drives the design process; and 4) constituent part/role matchups are what between-letter analogies are based upon.

In our perceptual model (see Chapter 4), the lowest-level data (quanta) become organized into parts under the top-down influence of roles. To keep this all straight it is important to remember that conceptual-level roles are filled by actual shapes that constitute a letterform's parts. Some authors refer to what we call "parts" as "higher-order" or "high-level" parts. This is because our parts are themselves formed out of lower-level sensory data items, like pixels, line segments, or, in our case, quanta.

In contrast to all this, the "flat" approach is characterized by its reliance on feature lists and/or templates. Some early models tended to use these lists as binary checklists, but those models make up only the extreme end of the flat approach. In our view, flat approaches also include mathematically sophisticated models that apply statistical techniques (in Figure 6.1 such statistical techniques are represented by the question-mark in the cloud) to squeeze a categorization out of a collection of features. Typically, these intermediate processes are based on calculations of distances between feature vectors in Euclidean vector space. Sometimes they allow features to interact (*e.g.*, input features combining in the hidden layer of a three-layer connectionist network), though not as collections making up parts. These intermediate processes may or may not be learned by reinforcement from the environment. The key to our structured/flat distinction is that an internal category model involving construction and processing of *explicit parts* is nowhere to be found in what we call the flat approach.

As we shall see, psychological evidence tends to favor a structured approach to letter recognition. (This is especially true of the isolated-letter, style-conscious flavor of letter recognition we are interested in.) Why, then, are flat approaches so common? Indeed, why are they fairly successful (as is shown by the retail OCR programs that use them)? The answer is simple. Or better yet, the answer is simplicity.

Flat approaches are easy to implement, computationally cheap, and fast. They are not, however, anywhere close to a complete solution to the problem of letter recognition. Their collective Achilles' heel lies in their inability to handle fonts that they have never been exposed to before. Although some flat systems do exhibit a small amount of generalization over sets of fonts (including the ability to handle some fonts that have never before been seen by the system), the stylistic distance between the novel fonts and the known fonts is actually very small.[1] So in the end, the tradeoff seems to be between fast-and-easy flat approaches that work on a limited number of (usually regular) fonts, and computationally-intensive, hierarchically- and relationally-structured approaches that work on many fonts (including ones never seen before) having letters that venture towards the boundaries of their intended categories. Obviously, there is room in the world for both types of approach. As soon as the act of *designing* letters enters the picture, however, and further post-categorization processing is required, much more weight gets added to the structured-approach side of the scale. The need

[1]Most OCR systems stick with recognizing standard bookfaces and are never exposed to the more interesting display faces. See Appendix A, Table 9.1 for a chart showing how Omnipage (a state-of-the-art commercial OCR program) fared on our PSYCH dataset.

to refer to a letter's parts *after* the process of categorization is finished provides the weight.

The reasonably clear waters of our structured/flat distinction are unfortunately muddied by systems that learn (such as backpropagation networks). This is because a learning system that starts with low-level features and is allowed to learn to categorize *in any way it chooses* via environmental feedback may well spontaneously develop implicit part-processing techniques on the way to a solution. Since backpropagation networks with a hidden layer make extensive use of distributed representations [Smolensky, 1988], it may be hard to discern whether, hidden away in such representations, some processing of implicit distributed parts is happening. Let us (for the moment) assume that such implicit distributed parts have been developed by some system through learning. It is conceivable that without explicit referential pointers to these distributed parts, further analogical processing (as in design of new letters) using the entire distributed representation might be carried out, but this would appear to be a very complex, tricky scenario.

We have two things to say about these theoretical possibilities. Firstly, although machine-learning techniques like backpropagation might in theory allow a learning system to develop distributed representations that include parts, it is unlikely in practice that such types of representations would develop on their own, because it is easier for a system to carve the space up without resorting to a hierarchical, part-influenced structuring of the feature space. A "blind" backpropagation algorithm would probably require some fairly extensive guiding — by way of architectural constraints (*e.g.*, the right sort of modular architecture) and/or specialized training subtasks — in order to develop intermediate-level parts in its hidden layer (see Section 6.2.1 for more on cramming hierarchical and/or relational structure into connectionist networks). Secondly, connectionist systems that do higher-level cognitive tasks involving explicit reference to structured distributed representations and manipulation of these representations (such as analogy or planning) are few and far between. (Recall that "reference" is one of the five dimensions in the structured/flat framework.) "Recycling" a distributed representation for use in ways not originally intended during training is a tricky thing. Work exploring this gray area between symbolic and subsymbolic processing, and concentrating on re-using distributed representations in further processing includes: research on distributed proto-plans for robot control [Meeden, 1994], investigation into the possibility of holistic syntactic transformations on distributed representations [Chalmers, 1990], and in-depth study of detectors, decoders, and transformations using recursive auto-associative memories (RAAMs) [Blank et al., 1992]. So, though some preliminary work in this direction has been done, there is a vast amount of work still to be done.

In summary, our division of models into two groups — "structured approach" models and "flat approach" models — seems to be a useful one, providing a common framework within which the many diverse models of letter recognition can be compared and contrasted. A further classification of structured approaches can be made along the lines of relational structure between parts. We argue that hierarchically- and relationally-structured approaches provide the best way of modeling and understanding letter recognition.

6.2 External competition

Now that the structured/flat distinction is in place, we turn our attention to some existing models of letter recognition that have been developed outside of our research group. Discussion of these models is organized with reference to the engineering and psychological camps. First we will present a broad overview of the two camps. Then we will consider two models in detail — one taken from each camp.

6.2.1 Optical Character Recognition (OCR)

Letter recognition by machines can be separated into various branches depending on the particular subtype of recognition being done [Mantas, 1986]. The most relevant approach, in terms of rivals to our role model, is known as Optical Character Recognition (OCR). Another major branch of machine recognition of letters is known as *on-line recognition* [Tappert et al., 1990]. In on-line recognition, timing information regarding *when* parts of letterforms were drawn (and thus how a letterform was constructed) is used during the recognition process. There are also some specialized branches of OCR, including cursive-script and printed-handwriting recognition. In these two specializations, partitioning word-level or document-level data into separate letters (called the segmentation problem) is a hard pre-processing problem. We will limit our discussion to standard OCR approaches — those applied to the task of recognizing various mechanically printed typefaces — because they more closely resemble what we are trying to get our system to do.

OCR involves parsing documents set in a variety of fonts into machine-standard form (*i.e.*, ASCII). Although there are papers in the OCR literature with impressive titles like "On the Recognition of Printed Characters of Any Font and Size"[2], and OCR machines and software are available commercially, a survey of the literature shows that the challenge of OCR — specifically, successfully scanning any document set in any typeface — has not been satisfactorily met. New methods are still being proposed and explored. A system that can recognize letters in many typefaces that it has never seen (or been trained on) has yet to be developed. In his book *The Age of Intelligent Machines*, OCR pioneer Raymond Kurzweil states, "While machines exist that can accurately recognize many type styles in common usage, no machine can successfully deal with the level of abstraction required by ... ornamental forms." [Kurzweil, 1990,

[2]Kahan, Pavlidis, and Baird [1987] actually tested their model on just six fairly standard fonts.

page 241].

[Suen et al., 1992; Mantas, 1987; Mantas, 1986; Mori et al., 1984; Gaillat and Berthod, 1979] all provide overviews of methodologies in character recognition from the engineering standpoint. These studies, which focus on OCR, show that for the most part, engineering approaches rely on statistical and mathematical methods (*e.g.,* nearest neighbor, Bayes rule, hierarchical clustering) applied to low-level local features of letterforms. In terms of our framework, this means that most OCR programs favor the flat approach. Of course, most engineering solutions are not meant to provide cognitively plausible models of letter recognition, but it is clear that the limitations of such models are a direct consequence of their non-human approach to recognition. We think there is much the OCR field can learn from cognitive psychology.

Gaillat and Berthod [1979] present a taxonomy of classical OCR systems that includes the following broad classification of methods:

- point-to-point comparison
- global transformations (*e.g.*, Fourier transforms or the method of moments)
- local features and curvatures
- structural decomposition into parts

This list still seems adequate to classify most systems that have been developed in the fifteen years since their survey was made. The results of our model-based studies reported in Chapter 8 (and elsewhere in this dissertation) help to make clear, and to explain the reasons for, the limitations built into the first three of these approaches, and recommend heavily the fourth [McGraw et al., 1994a; McGraw et al., 1994b].

Connectionist approaches

Connectionist models, for which claims of cognitive plausibility are often made, have also been applied to the task of letter recognition. These models have been no more successful than other OCR approaches, and suffer from many of the same faults. The interactive-activation reading model of McClelland and Rumelhart [1981] makes use of a geometric-feature-based recognizer. Its letter-recognition capability is limited to uppercase letters of one sole typeface. Indeed, each letter "concept" learned in the model is simply a specific, fixed shape. Other flat-approach connectionist models, also based on integration of low-level geometric features, exist in the literature [Guyon et al., 1989; Lee and Oldham, 1990; Le Cun et al., 1990; Fukushima, 1992], but like their symbolic OCR counterparts, these models have not been shown to be capable of recognizing ornamental or eccentric or fanciful letters such as those that characterize the gridfont domain.

Among the most sophisticated approaches using neural networks are those of Le Cun *et al.* [1990], Fukushima [1989,1992], and Hinton, Williams, and Revow [1992]. The first two of these are surprisingly dissimilar in approach to the third — with the latter work (described in detail in Section 6.2.4) taking a weakly-structured, category-model-based approach while the Le Cun and Fukushima projects are based on completely flat processing of local features. This shows that connectionist models can in principle run the gamut from flat to structured.

The Le Cun *et al.* model is a middle-of-the-road flat approach. In their network, a normalized 16×16 pixel image is fed through four hidden layers [H1..H4] and classified at the output layer. The H1 and H3 layers act as feature extractors, while each of H2 and H4 computes a localized average of the feature map from the layer immediately below. H1 computes the convolution of four different, trainable masks/filters with the image. H3 does a similar convolution, this time of twelve masks with the output of H2. H3 learns to combine the features of H2 into higher-order (but not part-like) features that are in some sense task-appropriate. Note, however that H3 does not make use of natural letter parts. H4 is fully connected to the output layer. There are 98,442 connections in the complete model.

Fukushima's "Neocognitron" model is a very similar flat approach [Fukushima, 1989; 1992]. Like the Le Cun *et al.* model, the Neocognitron is also multi-layered. In the Neocognitron, layers in the network alternate between low-level feature-extractor planes of neurons (called "S-cells" by Fukushima) and positional-invariance planes (made up of "C-cells"). The network is completely feedforward, with the lowest-level input being a pixel representation of a letter and the output layer being a simple categorization. S/C cell layers cover ever-larger physical areas of the input pattern as they stack on top of each other towards the output layer. In this way, local features (at what *we* would call the level of quanta) are extracted in lower stages of processing and are gradually integrated into more general features. It is important to distinguish these "abstract", integrated features of the upper levels of processing in the Neocognitron from what we call letter parts. Nowhere in the Neocognitron are letter parts classified or used *as such*. Instead, the Neocognitron uses collections of features, or feature-bundles, that are very much tied to the lowest levels and have nothing whatsoever to do with the parts that people divide letters into naturally under the influence of conceptual roles.

Fukushima has also investigated models with top-down connectivity (involving feedback of activation from higher levels back down to lower levels) [Fukushima, 1992]. Although the top-down idea is very intriguing and may be able to be applied to the task of part extraction and recognition *within one roman letter*, Fukushima applies it only to the problem of the extraction of individual letters from a collection of input data such as a cursive word. A task much more closely related to part extraction — that of the extraction of radicals from Chinese characters — is also briefly discussed. However, the radicals that Fukushima is able to extract are much more closely related to complete roman letters than to letter parts. For example, two radicals of the kind Fukushima investigates, when *placed together*, much as roman letters are placed together in an English word, make up a complete Chinese character.

Le Cun *et al.*'s work and Fukushima's Neocognitron, because they are based on strictly local features, are susceptible to a rather critical problem — a jumbled image containing the right features but in the wrong spatial relationships might well be classified incorrectly as a very strong category member when it should obviously not be. This criticism, which is discussed in [Lee, 1991], applies to *all* flat-approach models using local features (and not just connectionist flat-approaches, at that). Local features alone, without any hierarchical structure to group them, are simply not sufficient to determine category membership. Lee showed that at least one common neural-network approach has the unfortunate ability of producing very confident incorrect classifications of images that look nothing at all like characters.[3] Fukushima acknowledges this problem in his article and discusses the care that must be taken in the tuning of S-cell→C-cell connectivity in order to avoid it [Fukushima, 1992, page 224].

An obvious way around this problem is to force some structure into the system by moving from a flat approach to a structured approach. However, creating hierarchical and relational structure from scratch (that is, learning it) and being able to refer subsequently to the structure for use in, say, analogical mapping is an incredibly hard problem for connectionism, and has yet to be solved. The core of the problem is this. Putting structure into a connectionist model is hard, but clearly possible (see the discussion of RAAM below for one example). Being able to *manipulate the structure in its distributed encoding* is an entirely different matter, and a much harder one, to boot. Work at CRCC has shown how deeply interrelated high-level perception and analogical thinking are [Mitchell, 1993; Hofstadter and FARG, 1995]. If connectionism is to tackle analogy, its strengths as a learning model and auto-associator must be

[3] Similar arguments about local features were made in the famous Minsky and Papert book [Minsky and Papert, 1969], *Perceptrons*, which had the unfortunate side effect of de-funding neural-network research for a decade and a half.

buttressed with the ability to create and manipulate hierarchical perceptual structure. Some research in connectionism seems to be moving in this direction. A brief survey of such work is in order.

First comes RAAM. Pollack's recursive auto-associative memory (RAAM) is able to encode arbitrary tree-like structure into a distributed representation [Pollack, 1988; 1989]. It works like a conventional stack in computer science, in that it is possible to push things into memory and pop them off in reverse order later. The problem is that the structure is completely specified in the training set. This means that the RAAM simply amounts to an encoder/decoder. Some preliminary research extending the relevance of RAAM by focusing on holistic processing of RAAM-encoded representations (that is, being able to process "distributed parts", and thus, in a weak sense, being able to "refer" to separate parts) is presented in [Chalmers, 1990] and [Blank et al., 1992] (briefly discussed above in Section 6.1).

Hinton's family-trees research [1990], explores the representation and processing of part–whole hierarchies in connectionist networks. In Hinton's networks, all internal representations are learned by the system based on a set of training propositions. A bank of units for each role — agent, patient, and relation — is completely specified in the network architecture. The network is trained on a variety of agent–relation–patient triples representing two different, but closely related, family trees. The network is able to do three things: 1) to successfully create internal representations that make it possible to express implicit regularities in the domain (*i.e.*, it creates sensible internal representations for the various people — determining sex and age aspects, for example, from information like "father of" — as well as learning the relationships between the people); 2) to generalize correctly to the few examples on which it was not explicitly trained (4 of 104 instances of relationships, to be precise); and 3) to make use of the isomorphisms between the two family trees in its encodings (*e.g.*, having a unit that, after training, ends up encoding which generation a person belongs to). The main conclusion of the work is that it is essential to use the *same parts* of a connectionist network (*i.e.*, the same nodes and links) at *different times* to represent different parts of a part–whole hierarchy. In the end, a system very similar to a RAAM is proposed. Once again, though, it is unclear how the representations that are encoded through learning can be used in further processing (*e.g.*, analogical mapping).

Smolensky's work with tensor-product representations provides another approach to connectionist structural encoding [Smolensky, 1990]. The dot product of two vectors, one representing a role and the other its filler, can be used as an input encoding. Using structurally-related dot products can help a network organize its hidden layer appropriately. Blank's work in progress on the analogical mapping of simple scenes makes use of tensor products to combine symbolic and subsymbolic representations in order to represent *what* and *where* information in such a way that his network can use this distinction in its processing [Blank, 1995; 1994]. In preliminary studies, Blank's network seems to be able to perform a kind of simple perceptual grouping in its hidden layer. It can then map the distributed representation of the perceptually-structured source onto the target in order to make an analogy.

Regier's thesis work on simple scene description (*e.g.,* describing a scene as "the square *is over* the triangle") is another important start towards getting roles into a network [Regier 1991; 1992]. The fundamental weakness is that the two objects being related are represented on separate banks of units in the input layer before processing even begins. Object location (and thus separation) is thus supplied to the network at the outset. The network must come up merely with a descriptive phrase relating the objects. In other words, the perceptual parsing is largely done before the network even starts.

If a connectionist version of our role model is to be created (and we sincerely hope that one will be, since it will provide a connectionist underpinning for our ideas and in this way further justify them), the problem of structural reference within connectionist models will need to be solved first. We argue that in the meantime, hybrid models that resort to some symbolic processing (such as our own) provide a more reasonable starting point.

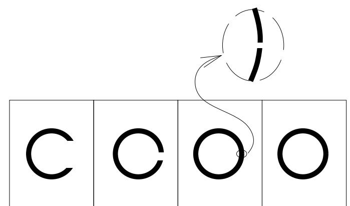
Intermediate-level parts in OCR

Let us quickly review some of the claims we have made so far. Little work on machine-based recognition systems has considered intermediate-level parts (sometimes called "high-level features" in the literature), let alone collections or groups of such parts [Mori et al., 1984; Gaillat and Berthod, 1979]. Restated in terms of our framework: OCR models generally do not take a structured approach. This is probably due to two reasons: 1) OCR research has, for the most part, had little to do with cognitive psychology; and 2) flat approaches are fast and cheap.

One important exception to this trend is the work done some two decades ago by Barry Blesser's research group[4] [Blesser et al., 1973]. Blesser's approach to machine letter recognition was, like ours, strongly based on the psychology of human letter perception [Naus and Shillman, 1976; Shillman et al., 1976]. The idea was to describe letters not in terms of their physical attributes, but in terms of more general descriptions of their underlying representations. Blesser argues that these representations are made up of functional rather than physical invariants.

Figure 6.2 shows three levels of the attribute "closure": functional, perceptual, and physical. *Physical* closure can be determined by a topological test. *Perceptual* closure is determined experimentally by asking human subjects "Is it closed?". *Functional* closure takes contextual effects into account and is determined by asking humans a similar question, but this time at the level of letters: "Is it an **O** or a **C**?" Blesser's approach is to search for functional attributes by experimenting with ambiguous letters.

[4]Blesser's group included Shillman, Cox, Naus, Kuklinski, Ventura, and Eden.



Functional Closure	–	+	+	+
Perceptual Closure	–	–	+	+
Physical Closure	–	–	–	+

Figure 6.2: Three levels of the attribute "closure": functional, perceptual, and physical. Taken from [Blesser et al., 1973, page 212]. Blesser argues that the functional level is the correct level to apply to letter recognition.

One published set of experiments investigating the attribute "leg" (which separates **Y** from **V** or **H** from **U**) is reported in [Shillman et al., 1976]. Once a collection of several of these attributes has been identified, then they can be applied to an automatic recognition system.

Throughout the work by Blesser's group, the role of stylistic context, even *within a character itself*, is highlighted. It is argued that graphical context rules specify the relationship between the physical attributes of a character and the functional attributes that specify its identity. This interrelation of the physical level and the conceptual level is very much in the spirit of our distinction between parts and roles. In [Cox et al., 1971], it is suggested that once graphical context rules have been discovered for one letter in a given typeface, they can serve to positively influence further perception (or possibly even design) of other letters in the same typeface.

Rich also briefly mentions an idea for a part-based model of letter recognition in her introductory AI text [Rich, 1983]. The model she sketches is similar in spirit to the implemented work of Sanocki (see Section 6.2.3 below).

Lam and Suen [1988] are interested in the natural decomposition of characters into sets of strokes (especially as pertains to Chinese). In some respects, strokes are similar to what we call *parts*. The method that Lam and Suen use to classify characters works generally as follows: first, preprocess images to obtain a polygonal approximation to the contour; next, decompose this polygonal representation into curves and line segments making up strokes; finally, use a relaxation algorithm to determine the quality of the match between this representation of the test image and each of a set of stored templates. This approach has some important resemblances to our work, but it neither addresses the notion of conceptual decomposition into roles (as roles are not always the same as strokes) nor provides a very elegant solution to the template-matching problem (as a brute-force algorithm is applied).

OCR and the flat approach

The flat approach to pattern recognition results in a simple categorization — an appropriate category label for a single input object. This approach is based on the intuition that each category constitutes a large basin of attraction, making classification of an input pattern reasonably easy. Simple, regular patterns (such as the very standard characters found in fonts like Times or Helvetica) do tend to fall nicely into the basins. But as patterns become messier (and characters begin to be seriously compromised by the style of their font), they start landing in between-basin zones of the space. Deciding which basin to roll into (if any) becomes a messy and much harder task.

Furthermore, flat, feature-based pattern recognizers do not generally possess or construct complex internal models of the object being classified, and can therefore say very little about the way such an object is constructed, what its parts are, and what makes it stylistically interesting. In contrast, the structured approach to recognition (including our role model, Blesser's model, Lam and Suen's model, Sanocki's model, and Williams' model) necessarily involves a deeper "understanding" of the categorized object. The object is in some sense *parsed* into its constituent parts. Information about how parts are formed may be useful for future processing. This is especially relevant to the task of design, as well as to performing simple additional tasks, such as determining object orientation or extracting elements of style based on the results of a recognition task.

In any case, structural, model-based OCR approaches — though clearly superior to flat, feature-based approaches as cognitive models — still have important lessons to learn from cognitive psychology. We are convinced that letter recognition by machines will remain inferior to that by people until and unless the nature of the human recognition system is taken much more carefully into consideration.

6.2.2 A psychological perspective

Feature lists, templates, and mathematical models

There are almost as many different psychological theories of letter perception as there are OCR models. Some of the least plausible are based on purely mathematical methods, and have much in common with the flat OCR approaches discussed above [Townsend, 1971; Gilmore et al., 1979]. Keren and Baggen [1981] show why low-level geometric approaches like Townsend's are not psychologically substantive. Although models based on Luce's Choice or multidimensional scaling can often do the curve-fitting required in order to fit at least some aspects of human perceptual data, they tell us little about the psychological processing underlying human letter perception.

Template or prototype models, wherein a description of a letter is made at a holistic or gestalt level, are (as was pointed out long ago by Neisser [Neisser, 1967]) untenable for all but unrealistically circumscribed alphabets. Nevertheless, this type of model is still commonly found in commercial OCR programs. Template models can be classified as flat approaches since they involve no explicit part processing. In the next chapter, we will discuss the behavior of two template models in detail and elucidate their weaknesses.

Keren and Baggen propose a feature-analytic model based on a feature list very similar to Gibson's [1969; 1971] (although their features are arguably slightly higher-level in nature). Though their model works quite well on the data set they developed, they readily admit that it *cannot handle more than one typeface*. This points out a major problem associated with the use of feature lists: they will apply only exceptionally to more than one set of stimuli. They are certainly not general enough to handle multiple typefaces, especially when the styles under consideration go out towards the fringes of letter-categories, as is often the case in display faces for advertising (or in gridfonts, for that matter). Making a general feature list that could be used in the recognition of letterforms of multiple typefaces would require an implausibly large number of features that would grow each time a new typeface was to be added.

Features	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
Extender																										
Direction	-	U	-	U	-	U	D	U	-	D	U	U	-	-	-	D	D	-	-	U	-	-	-	-	D	-
Side	-	L	-	R	-	C	R	L	-	C	L	C	-	-	-	L	R	-	-	C	-	-	-	-	R	-
Hook	-	-	-	-	-	+	+	-	-	+	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+	-
Central Portion																										
Enclosed	+	+	-	+	+	-	+	-	-	-	-	-	-	-	+	+	+	-	-	-	-	-	-	-	-	-
Round	+	+	+	+	+	+	+	+	-	+	-	-	+	+	+	+	+	+	+	+	+	-	-	-	+	-
Side Gap	L	-	R	-	R	R	-	-	-	-	R	-	-	-	-	-	-	R	L	R	-	-	-	L	-	L
Top/Bottom Gap	-	-	-	-	-	B	-	B	-	-	T	-	B	B	-	-	-	B	-	-	T	T	T	T	T	-
Doubling	-	-	-	-	-	-	-	-	-	-	-	-	+	-	-	-	-	-	-	-	-	-	+	-	-	-

Table 6.2: A feature list for use only in identifying letters from Helvetica. Taken from [Haber and Haber, 1981].

An example of a feature list is in order. [Haber and Haber, 1981] includes an interesting section on feature lists and feature-list creation that clearly points out some of their most obvious weaknesses. In their discussion, the shape-based feature list shown in Table 6.2 is introduced. This feature list can be easily coded into a binary decision tree. It lists a distinctive set of features for Helvetica, and provides an example of a typical feature list. The features are divided into two sets: those dealing with the central portion of a letter, and those dealing with ascenders and descenders. Briefly, the features can be explained as follows:

- Extender direction: Does the letter have ascenders (U) or descenders (D)?
- Extender side: Which side is the extender on [left (L), right (R), or center (C)]?
- Hook: Is there a curved hook at the end of an extender?
- Enclosure: Is any of the central part of a letter completely enclosed?
- Round: Is any of the central part curved?
- Side gap: Is there an opening in the central portion? Which way does it face [left (L) or right (R)]?
- Top-bottom gap: Is there an opening facing up or down [top (T) or bottom (B)]?
- Doubling: For a pair of letters is one the double of another (*e.g.*, consider 'm' and 'n')?

The example feature list from [Haber and Haber, 1981] was developed for use only in identifying which of the 26 Helvetica categories a given Helvetica letterform belongs to. It is not meant as a general feature list for use in the recognition of letters from any other styles, nor could it be easily adapted for use as such. Its features are custom-tailored for Helvetica and are not built of interchangeable building blocks of the sort that might possibly be arranged into more general features. Simply put, the feature list shown here is completely non-general and inflexible — a criticism that we believe applies across the board to all feature lists of this sort.

All working feature-models developed to date use features that are too limited. It is highly unlikely that developing a set of general features for letter recognition, much less design, will ever be possible [Hofstadter, 1985]. We plead the case that complex stimuli like letters are more easily and more naturally modeled by richer representational systems (using the structural approach) than by feature lists (using the flat approach).

Treisman and Gelade [1980] argue that "local elements or parts of shapes function as separable features which must be integrated by focused attention." They go on to say that this notion of separable features applies to letters as well (considering letter parts as features). Their argument hinges on the belief that strong evidence of separable features in any domain is provided by the phenomenon of "illusory conjunctions", where separate features are incorrectly integrated in order to form an illusion. Thus if subjects are very briefly exposed to two pictures in different areas of their visual fields — for example, a blue square and a red triangle — they often report having seen a red square.

One of the illusory-conjunction experiments reported in [Treisman and Gelade, 1980] involved asking subjects to search for a target 'R' in either a background of 'P's and 'Q's, or a background of 'P's and 'B's. The idea is that if, during perception, the letters are decomposed into parts and then re-integrated, illusory conjunctions will be found in the first condition and not in the second. This will cause search times in the first condition to have a linear time increase with display size, but no such increase in the second condition. Figure 6.3 shows how the parts of the 'P' and 'Q' can be combined to make an 'R', while those of 'P' and 'B' cannot. Experimental results show the predicted linear increase.

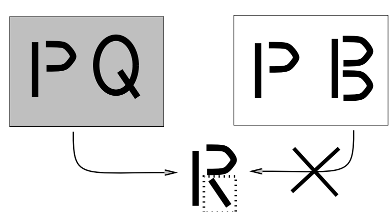


Figure 6.3: Treisman's letter-search task shows evidence of illusory conjunctions in the first condition ("find the 'R' in 'P's and 'Q's") [darkened] and not the second ("find the 'R' in 'P's and 'B's"). This is because, as shown here, 'P' and 'Q' have parts that, taken together, completely specify all of the parts of an 'R', while 'P' and 'B' do not. This result provides strong evidence for part processing in human perception of letterforms.

These results suggest that feature-integration theory applies to letters as well as to more complex shapes, and that letters are indeed broken into higher-order parts and reintegrated. We agree with Treisman and Gelade that letterforms are made up of parts (the parts that correspond to and fill conceptual roles), but we go further and argue that the parts themselves are higher-level structures made of sets of simple features and are not single, complex "features", as Treisman and Gelade state.

Furthermore, we believe that parts are constrained in their relationships to one another by relational structure (the relation dimension of the structured/flat framework). Treisman and Gelade's parts are in some sense free-floating before integration. After integration, their features (what we would call "parts") are simply concatenated together. This is too simplistic a view.

Problems with feature lists can be summed up as follows. Feature lists:

1. lack flexibility, since features are not built of building blocks, and are by definition non-decomposable (as emphasized during the critique of Haber and Haber's feature list above);

2. cannot represent a writing system with a potentially infinite number of features, as letterforms, especially advertising faces, seem to require (see the discussion of Keren and Baggen above);

3. are not efficient, since implausibly large numbers of features are often required even for one typeface (see the Keren and Baggen discussion above);

4. cannot express the intuitively clear relationships between different features (as argued in the discussion of Treisman and Gelade above).

Considered in this light, the benefits of higher-level representations certainly outweigh the complications they may add to processing in a model.

Higher-level relational structure

Hock *et al.* [1988] show that similarity and categorization judgments are based on parts and relational configurations in a pattern, rather than on elementary constituents of a pattern (*e.g.,* specific quanta or dots). Their patterns consisted of 8 dots randomly placed in a two-dimensional space. Subjects judged the similarity of dot patterns made by two transformations. In one type of transformation, a single dot was moved to a new area of the space. In the other type of transformation, three neighboring dots were moved as a unit to a new area. Patterns made from one another by the second transformation were rated by people as more similar to each other than patterns related by the first, indicating that similarity is a function of higher-order perceptual units rather than of elementary dot locations.

This result, together with those of Palmer [1977; 1978] and Takano [1989][5], suggests that intermediate-level perceptual units (parts) play a key role in categorization. Although Palmer's shapes were somewhat letter-like in appearance (and were made on a grid similar to ours), they were not letters. A natural, although controversial, question follows: Are letters decomposed in a similar fashion into intermediate-level perceptual parts, or are they somehow "unitized" into a single gestalt? This question is answered in favor of intermediate-level parts both by our experiments and by those of Treisman and Gelade [1980], discussed above, and by the work of Sanocki [1986], whose approach is detailed in the next section.

Schyns and Murphy (1994), discussing visual perception in general, also argue for the utility of segmentation into intermediate-level complex parts. For them, a necessary aspect of such parts is their *functional* nature. Their work shows that parts are not perceptual primitives, as Treisman would argue (*e.g.,* complex line segments, edges, etc.); rather, they are functional units that are typically defined in terms of

[5] Discussed in Chapter 4 with reference to the role hypothesis.

several primitive features and that have been learned to be important in categorizing and representing the object in question. As far as letter perception is concerned, the functional account is supported by the work of Blesser (discussed above) [Blesser et al., 1973]. Our theory of roles fits these results perfectly, and argues that functional parts ("roles", in our terminology) are useful also in letter perception.

Other work involving representation of objects by parts includes [Tversky and Hemenway, 1984] and [Biederman, 1987]. Biederman's recognition-by-components (RBC) hypothesis is especially relevant as it makes much of the relations among components. Our roles are somewhat related to Biederman's "components" (although they are not three-dimensional). Just like components (also called "geons"), roles provide top-down constraints resulting in the "representational regularization" or conceptual packaging of the infinite number of possible role-filling parts into fairly simple representational objects. Thus heavily-stylized letters can be recognized if they can be parsed into parts that fill appropriate roles. We agree that although objects come in highly complex and irregular forms, the conceptual units by which they are recognized are simpler and more regular than huge lists of all possible parts.

Our work suggests that the intermediate-level pieces into which letters are segmented are essential to the categorization process. Roles provide top-down influence on the segmentation of letterforms into such parts. Our experiments[6] show that human letter-recognition data, especially when the stimuli include nonstandard letterforms, can be clearly and easily explained in terms of roles, while other theories are far less powerful in terms of explanation and prediction.

[6]See Chapters 7 and 8 as well as [McGraw et al., 1994a; McGraw et al., 1994b].

6.2.3 Sanocki's letter-recognition model

In his doctoral thesis, *Visual Knowledge Underlying Letter Perception: A Font-Specific Approach*, Thomas Sanocki introduces a structured model of human letter perception based on what he calls *attributes* [Sanocki, 1986]. Sanocki's attributes share a common philosophical foundation with our notion of roles. His model was developed "with the goal of learning about the internal representations of visual knowledge that underlie letter perception" — one way of stating our goal as well.

Sanocki's intuition was that internal representations of letters are necessarily more complex than the simple feature lists championed by many psychologists (see, for example, [Gibson, 1971], [Keren and Baggen, 1981], [McClelland and Rumelhart, 1981], and [Townsend and Ashby, 1982]). In most of these feature-based models, letters are represented as sets or lists of features. When one of a letter's features becomes activated, the letter itself becomes slightly activated. Groups of activated features together provide enough activation so that one letter category "wins" by becoming the most highly activated.[7]

This activation process should sound familiar as it is precisely the process used in fully-trained feedforward backpropagation networks. In simple networks, the connections between features on the input layer and appropriate letters on the output layer are learned as the network is trained over time. (In fact, McClelland and Rumelhart's model does exactly this, as does our Netrec model, introduced below.) There are two types of features that researchers have experimented with: local parts or segments (such as those used by McClelland and Rumelhart) and more abstract global properties (like "open to the left" or "has a closure" [Gibson, 1969]). Sanocki felt that simple feature models of this sort do not tell the whole story, mostly because coming

[7] This process is very similar in nature to the bottom-up way in which simple groups of syntactic features of parts in our model activate roles, except that it is applied at the level of entire letter categories instead.

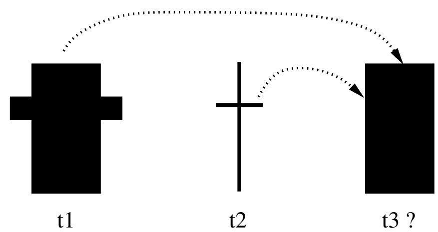


Figure 6.4: The parts of **t1** and **t2** — their crossbars and posts — are not completely independent, as some feature-based models would suggest. Instead, their style obviously matters. Consider the shape labeled **t3**, which is made up of the crossbar of **t2** and the post of **t1**. Is it a good 't'?

up with a set of features that explains how people recognize letters in *many different styles* seems to be impossible. As we argued above, most feature models (*e.g.*, Keren and Baggen's model) cannot easily explain how letters are recognized across fonts. In the worst case, a new set of features must be created for each font, which is absurd.

Sanocki points out that another fundamental problem with some feature-based models is the underlying assumption that features are *independent* — having no effect on each other. Consider the case in which the features in question correspond to letter parts. An obvious problem rears its head. How could it be that the crossbar of a 't' would have no effect whatsoever on the nature of its post? Figure 6.4 caricatures the problem. If parts were indeed completely independent, then **t3**, which is made up of the crossbar of **t2** and the post of **t1**, should be a good 't'. It clearly is not. Something must be wrong with the assumption of feature independence. Sanocki argues that the relationships between features (parts, in this case) should also include some aspects of style. We wholeheartedly agree.

In a series of experiments, Sanocki showed that people's letter representations must contain font-invariant information in order to be able to recognize letters across a variety of fonts [Sanocki 1987; 1991]. The ability to recognize letters of such widely varying shapes and sizes stems directly from the underlying representational system. Gibson was among the first to point out that in order to recognize so many diverse

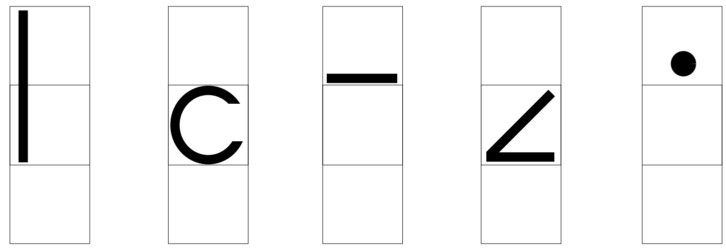


Figure 6.5: Letter categories in Sanocki's model are made up of these five attributes. These attributes were allowed to vary along the dimensions of size, shape, and thickness. Very roughly speaking, Sanocki's attributes have some tenuous similarity to roles in our model.

letterforms so well, the perceptual system must make use of abstract relational features that are invariant across fonts [Gibson, 1969]. Sanocki's model builds on this idea, emphasizing the importance of structural hierarchy in the relationships among parts. In this sense, Sanocki's structural model has its roots in both Palmer's perceptual-hierarchy theories and Neisser's work on schemata [Palmer, 1977; Neisser, 1967]. Our role model shares a similar spirit.

Sanocki's main result, which he calls the "font regularity effect", boils down to showing that a perceptual model becomes tuned to regularities within a font. This implies that changing a font in mid-stride — that is, once the model has become attuned to a particular font — will affect further processing, slowing it down until the system tunes itself to handle stylistic aspects of the new font. This property of the model is empirically shown to be true of human subjects.[8] Blesser's theory regarding the effects of stylistic constraints on recognition is supported by Sanocki's empirical results [Cox et al., 1971][9].

Obviously our role hypothesis has much in common with Sanocki's theory, in

[8]Although this result is in some respects tangential to our model, we can nevertheless encompass its results within our model by allowing style-relevant activations in the conceptual memory to affect future processing. As this task is not related to our intended use of the model, we have done little research along these lines.

[9]Note that [Cox et al., 1971] is an article from the Blesser group.

that we make much of perceptual hierarchies and the relationships between roles; however, there are important differences. The biggest difference lies in the weakness of Sanocki's "attributes", which, together in groups, define a letter category's representation. Sanocki's model uses only five attributes to form all 26 font-invariant representations of the lowercase letters. The attributes are shown in Figure 6.5. These attributes are defined at too low a level and are too tied to specific fixed shapes. They have only a tenuous similarity to the more natural decomposition of letters into roles. In contrast, our model posits 47 roles (at a higher level than Sanocki's attributes), which are in turn represented as collections of weighted norms (see Figure 4.2). Letter categories in Sanocki's model are represented as flat collections (*i.e.*, simple concatenations) of attributes. Figure 6.6 shows the representational structure of all 26 categories in Sanocki's model. A picture of each letter in terms of its attributes is shown to make the formal (Lisp-like) representation clearer.

Representations like Sanocki's were once commonly found in many AI models but have fallen out of favor as AI has become more "sophisticated". Using Lisp-like symbolic grammars defined in terms of atomic primitives (which is what Sanocki has done) leads to the kind of brittleness now commonly associated with many forms of symbolic AI. (One cogent discussion of the inherent brittleness of classical symbolic systems of this sort can be found in [Holland, 1986].) In addition, such representations are in some sense isolated crystalline structures that are usually only weakly isomorphic to the real-world domain being explored. One way to avoid these problems of brittleness while retaining some of the important features of hierarchical representations is to use stochastic processing and a model of active concepts [Hofstadter, 1979; Hofstadter and FARG, 1995] as we have done in our model .

Although Sanocki's model does recognize letters from 12 different fonts and does provide important results concerning the processing of diverse styles, every one of the

a	STROKE(right, base) + LOOP (base, 0, 90) + a–type[LOOP]
b	STROKE(left, top, base) + LOOP(base, 0, 270)
c	LOOP(base, 180, 90)
d	STROKE(right, top, base) + LOOP(center, 180, 90)
e	LOOP(base, 135, 180) + BAR(base)
f	STROKE(middle, top, base) + BAR(top) *
g	STROKE(right, base, sub) + LOOP(base, 0, 0) + LOOP(sub, 180, 270)
h	STROKE(left, top, base) + LOOP(base, 0, 180)
i	STROKE(middel, base) + DOT
j	STROKE(middle, base, sub) + LOOP(sub, 225, 270) + DOT
k	STROKE(left, top, base) + k–type[ANGLE]
l	STROKE(left, top, base)
m	STROKE(left, base) + m–symmetrical[LOOP, LOOP]
n	STROKE(left, base) + LOOP(base, 0, 180)
o	LOOP(base, 0, 0)
p	STROKE(left, base, sub) + LOOP(base, 0, 180)
q	STROKE(right, base, sub) + LOOP(base, 180, 90)
r	STROKE(left, sub) + LOOP(base, 0, 90)
s	s–symmetrical[LOOP,LOOP]
t	STROKE(middle, top, base) + BAR(top) + LOOP(base, 225, 180)
u	STROKE(right, base) + LOOP(base, 0, 180)
v	ANGLE(225)
w	w–symmetrical[ANGLE,ANGLE]
x	x–symmetrical[ANGLE, ANGLE, ANGLE]
y	ANGLE(180) + LOOP(sub, 180, 225)
z	z–symmetrical[ANGLE, ANGLE]

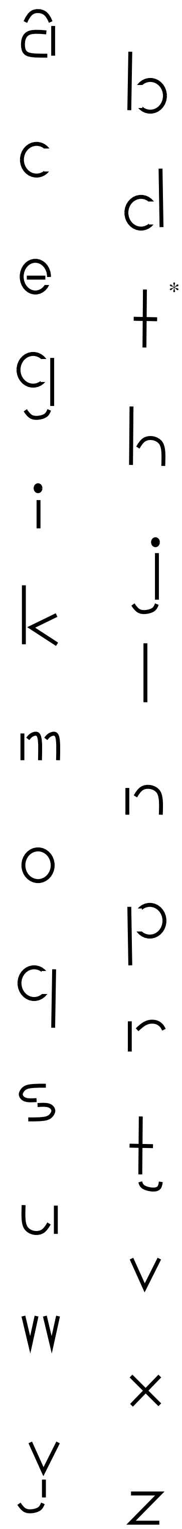


Figure 6.6: The 26 letter categories of Sanocki's model are made up of five attributes. Each letter category has only one representation. Graphical representations of each category are illustrated to the right. The formal representation each letter category is taken directly from Sanocki. (* = [sic.]).

fonts in question could be pegged as fairly normal.[10] Stylistic influences are clearly visible in Sanocki's letterforms, but they are not so strong as to greatly interfere with the categorization task. Sanocki completely avoids questions of deep style (see Chapter 1) by making only one attribute set (a role-set in our terminology) available for each letter category. In contrast, because its roles are defined at a higher, more natural level, and because of its multiple role-sets for each letter, our model recognizes a much greater range of letterforms — letterforms that stray farther from the security of normalcy and deeper into the jungle of style.

Near the end of Chapter 2 of his thesis, Sanocki discusses what form a letter-recognition processor might theoretically take with regard to serial and parallel processing. He divides processing into four levels: feature detection, attribute activation, letter category activation, and a decision stage. He argues that the first three levels are each separately done in parallel (mostly by pointing to [Treisman and Gelade, 1980; Treisman, 1986] and [Shiffrin and Schneider, 1977], with whom we basically agree). But the decision stage, he goes on, is more serial in nature (and the font regularity effect occurs just before the decision process).

Our notion of an emergent perceptual process that zeroes in on a letter category coheres with this story, while providing a slightly less clean-cut division of levels. Instead of dividing neatly into four cascaded levels of processing, our model to some extent mixes the processing together, blurring the boundaries between processing stages while preserving some aspects of seriality at the global level. While some agreement with Sanocki's four levels might be teased out of our system's processing, this is simply a side effect of the temporal nature of our parallel emergent processing. A global sort of seriality falls out of our system because: parts must be created and at least partially labeled (in a parallel fashion) before roles become activated; roles

[10] The fonts that Sanocki used were created in his lab especially for his experiments. They are all closely related to the Apple][psych font (a simple, sans-serif, low-resolution font).

must become activated (also in parallel) in order for wholes to become active; and the decision process (slightly more serial in nature) can occur only after at least one whole has been activated. So there is necessarily some order to the processing, but there are no sharp lines between processing stages. In fact, "later" stages may have some noticeable effect on the processing of "earlier" stages. We find this notion of overlapping stages and the mixture of bottom-up and top-down processing much more natural, even if more complex.

All in all, we agree with Sanocki that attributes (or roles, in our case) must be stylistically consistent with one another. This applies to letter-level representations (wholes) as well. Our main disagreement concerns the level of our roles, and the way we represent roles as collections of even simpler features. Sanocki admits that he glosses over the notion of attribute recognition (and/or formation) from sensory data, and chooses instead to concentrate on the higher levels. Since we start our recognizer with raw configurations of quanta and have it build up the parts, our model explains not only higher-level processing, but an earlier stage of processing as well. In this light, our model could be construed as something of a hybrid of feature-based and hierarchical models — with feature-based roles working together in a hierarchical fashion to define wholes. In any event, Sanocki's central claim — that letter representations must contain information regarding the *stylistic relationships between attributes* — is philosophically consistent with our own view.

6.2.4 Williams and Hinton's deformable-models system

Christopher Williams' doctoral thesis, *Combining Deformable Models and Neural Networks for Handprinted Digit Recognition*, while also a structured approach (but only weakly so), provides a system completely antithetical to Sanocki's. Instead of basing his system on psychological-plausibility arguments and human experiments, Williams attacks the recognition problem with a mathematically sophisticated, model-based approach.

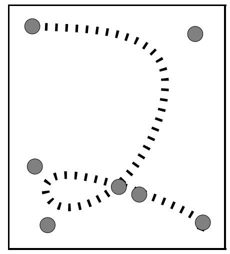


Figure 6.7: The generic cubic spline model for a '2' in Williams' program. Note the absence of any intermediate-level parts in the model. Unfortunately, Williams provided examples of spline models for only two of the numbers in his system: '2' and '3'. This was, in our view, a very serious omission.

In a nutshell, Williams' program recognizes isolated handprinted digits '0' through '9' using trainable deformable models. Each digit is represented in the program as a single cubic B-spline. Figure 6.7 shows the cubic spline used to model a '2'. The basic shape of each digit is defined by the "home" positions of the control points. The shapes have an elasticity that results from a Gaussian distribution over displacements of the control points away from their home positions. The matching of an input image with a spline model results in a measure of match quality. The quality metric is based on the likelihood that the image data could be generated by Gaussian "ink generators" placed along the length of the spline. Imagine, for example, Gaussian ink-squirters sitting at each of the dashes in the '2' spline shown above. The measure of match of the '2' spline model to an actual digit has to do with the probability that the ink-squirters would cover the digit once the spline had been "deformed" enough to allow the best possible ink-squirting attempt. Elastic matching (based on the Expectation Maximization algorithm) is used to adjust each of the ten spline models in such a way that an energy function (a function of both the deformation energy — that is, the cost of moving the control-points around — and the data likelihood — that is, the probability that the ink-squirters lying along the spline would cover the shape) is minimized. The spline with the lowest energy cost wins. A neural network is used to help provide reasonable starting points for the search through deformation space.

A brute-force comparison of all ten (possibly deformed) splines to the digit being recognized is not very expensive (relatively speaking, that is). However, if such a brute-force method of comparison were expanded to the 26 lowercase letterforms, or the 47 role-sets, or, worse yet, the full ASCII character set, the computational cost would go up considerably.[11] Imagine the trouble a Williams-like system would have trying to read Chinese characters (of which there are between 10^4 and 10^5 tokens)! We believe that *our* model would scale up gracefully due to its built-in parallelism.

The spline models are hand-coded by Williams and are not learned by the system. Such coding-in — which is essentially what we have done with the roles in our model — is a slow and difficult process (as we can attest). On top of that, any model-matching process will by its very nature be computationally intensive. The painstaking process of model design and the computational intensiveness of matching may well explain why feature-based flat systems are so pervasive; after all, they are easy to implement and they are fast. However, when it comes to recognizing letters with more than a small degree of variability (such as handwritten digits or gridfont letters) the feature-based flat models quickly break down.

Williams basically ignored the computational overhead problems and developed a model-based system without regard to the computational intensity of the matching process. He readily admits this weakness of his system in a straightforward way. We acknowledge this problem with model-matching approaches as well. However, we do our best to circumvent this problem in our model by relying on the parallel terraced scan. The parallel terraced scan, as well as other related search techniques (like genetic-algorithm search), helps tackle problems of computational intensity head-on. These sophisticated techniques rely on self-organization and dynamic adaptation, and

[11] One way to avoid this problem would be to dedicate a recognition "chip" to each spline model and do all the comparisons in parallel. William's does not discuss such parallelization.

are inspired by biological systems [Holland, 1975; Forrest, 1991].

A more fundamental problem with Williams' work relates to the structural flatness of Williams' deformable splines. Consider, for example, the fact that his model for '4' consists of a single spline. In our system, a '4' would have two or three parts determined by its conceptual roles. Although we feel that Williams' endorsement of model-based approaches is fundamentally correct, we hesitate to support the idea of such simple "one-part" models (especially if they are represented as very shape-wedded splines). Instead we refer to the psychological literature on letter perception (and the data we have gathered) to back up our role hypothesis. Basing recognition on models as opposed to feature lists is a good first step. Making these models psychologically plausible is the next.

6.3 In-house competition

Models like Sanocki's and Williams' provide views contrasting with our own on the problem of letter recognition, as do many other systems. It would be very enlightening were it possible to compare the behavior of all such systems on one or more sets of standardized data.[12] Unfortunately, most models are developed for special purposes (like our gridletter recognizer) and cannot be easily modified to allow testing with other datasets.

One partial solution to this dilemma is to develop a range of simple competing models to test on a set of standard data relevant to the problem at hand. Simple models can act as probes into the behavior of more complex ones, helping explain how they do their processing and why alternative approaches may not work. They also provide good metrics for system performance.

[12]Williams, in fact, made use of one such large dataset of handwritten Zip codes available from the National Institute of Standards and Technology (NIST).

The next two sections are concerned with two very simple computer models of letter perception. One of the main reasons these models are so simple is that they both take a completely flat approach to recognition. These models have been developed for two reasons. First, they are intended to serve as benchmarks against which the strengths and weaknesses of Letter Spirit's role-based recognizer can be compared. There is little doubt that inter-architecture comparison and analysis of this sort will provide useful data for cognitive modelers. Second, as will be seen in Chapters 7 and 8, the rival models serve to shed some light on the human data collected (and vice versa). We suggest that comparing the performance of the simple models (which we can selectively degrade) with the performance of human subjects should be considered an interesting data-analysis technique in its own right. It is especially useful when data are as complicated and processing is as varied as is the case in letter perception. By selectively degrading the models we can determine what factors influence people in letter-categorization tasks. The information gleaned from this process can then be used in the development of more sophisticated cognitive models.

6.3.1 Dumrec

As its name is intended to suggest, Dumrec is a brute-force symbolic-AI gridletter recognition program. Like the role model, it is written in Scheme (a dialect of Lisp). Dumrec is an exemplar model, meaning that it compares a mystery letter with a large number of stored "prototype" letters, counting agreements and disagreements at various levels of structural complexity, computing a score for each category as a weighted sum of such agreements and disagreements, and then awarding victory to the category with the highest score. Dumrec's database of stored letters can be changed to include any number of different "prototype letters".

Dumrec is initialized by reading in a data file containing a large number of prototype letters (several for each of the 26 letter categories) and computing a property list for each letter in the list. After this, Dumrec is given any number of mystery letters, one at a time. When a mystery letter is presented, its property list is computed in exactly the same way as for each of the prototype letters. The mystery letter's property list is then compared, using a weighted metric, to the property lists of all of the prototype letters, and the winner is picked.

Dumrec uses two types of comparison. The first is a direct approach in which the quanta of the mystery letter are directly compared to the quanta of each prototype letter with no abstraction whatsoever. Dumrec has six comparisons of this type, involving what we call Level-0 data: a rating based on the Hamming distance between the quanta list of the mystery letter and that of the prototype letter; an "on"-bit rate, which is the number of quanta shared between the mystery letter and each of Dumrec's prototype letters; and four simple scores based on the number of horizontal, vertical, forward-slash or back-slash quanta in a letter. Points are awarded to each prototype letter based on how well its properties match those of the mystery letter.

All other points awarded to prototype letters are based on comparisons of the pre-computed feature list of each prototype letter to the mystery letter's feature list. These features can be separated into three levels, based on the amount of abstraction involved in their computation.

Level-1 data involve only a small amount of abstraction, as level-1 properties are concerned with sets of multiple quanta. Level-1 data include position and number information for a variety of simple angles and shapes. These data are computed by sliding simple shapes over the grid and noting matches. For example, a line segment composed of two horizontal quanta is slid down the grid. Each of the seven possible heights at which such a line can occur on the grid gets a value of either 1 or 0 depending on whether the segment as a whole is present. The total number of such matches is also computed. The vector pertaining to horizontal-segment occurrence in a mystery letter can then be compared to the corresponding vectors already computed for the prototype letters, and a score assigned. Shapes that were considered in this manner include: $n \times m$ rectangles (where n is either 1 or 2 and m ranges from 1 to 4); all possible triangles made of sides having either 1 or 2 quanta; vertical line segments with lengths ranging from 2 to 4 quanta; horizontal lines of length 2; forward and backward slashes of length 2; and all possible angles (*i.e.*, meetings of two quanta) of 45, 90, or 135 degrees. Also contributing to a prototype letter's level-1 score are comparisons between the total number of certain types of such shapes, including: simple closures, boxes, rectangles, triangles, lines, and angles. Such a brute-force approach to searching for shapes is easily implemented, given the relatively small number of quanta on the grid. Level-1 data are all quanta-based and locally calculable, but are slightly more abstract than level-0 data.

Level-2 data consist of slightly more abstract properties — namely, "mass", "gravity", and sectorization data. The grid has six sectors (the three zones shown in Figure 1.6, divided into left and right halves), each of which may or may not include quanta from the letterform. Sectorization data consist simply of the number of quanta in each grid sector. A letter's "gravity" is the location of the sector with the most quanta. A letter's "mass-vector" is slightly more complicated. It consists of a vector with its origin in the lower left corner of the grid that extends into the grid according to the position and amount of quanta making up the gridletter. This vector changes according to how much "stuff" there is in a letter, and where that "stuff" is located. Comparing these properties between gridletters requires some knowledge of the structure of the grid and the relation between sectors. For this reason, level-2 data are slightly more abstract than level-1 data.

Level-3 data constitute the most abstract data in the Dumrec program. Properties at this level include "point" and "tip" information. "Points" are the 21 intersection points of quanta on the grid. All the points where a gridletter terminates are recorded for the point property. For instance, most versions of the letter 'c' on the grid have two points associated with them: the locations where the ends of the (conceptual) curve are located. "Tips" are related to points, but also include information about the direction of termination and its quantum type (vertical, horizontal, forward-slash, or back-slash). Each tip is a quantum with a direction of termination and a connectedness to the rest of the letter. For example, the vertical ending of the part filling the *post* role on the left side of a very standard letter 'b' would be a vertical tip, pointed upwards, and connected to the rest of the letter at its bottom. Most instances of the letter 'k' have 4 tips, two vertical ones having opposite directions (either end of the part filling the *post* role), and two kinds of diagonal ones on its right side. The number, locations, and types of tips in the mystery letter are compared with those of the prototype letters. These calculations involve a small amount of knowledge about the relationships between types of tips. Also note that they cannot be computed completely locally, and instead require knowledge of all of the endpoints in the entire shape. Thus, level-3 data are the most abstract properties found in Dumrec (even if they are not very abstract), and require the most computation. Obviously, though, even the most abstract of the Dumrec levels is still far lower, conceptually speaking, than the level of roles.

Once the property list has been calculated for the mystery letter, Dumrec calculates a score for each prototype letter based on the match of its property list with that of the mystery letter, with different weight-factors attached to different types of comparisons. Comparisons of properties range from the trivial (for level-0 data) to rather complicated (for level-3 data), but are all relatively simple computations. After raw scores are calculated, any of several different selection metrics (explained below) can be applied to yield the final answer.

Obviously, the weight-factors play a crucial role in Dumrec's performance and are very important. The weights used for our experiments were set by hand over a very large number of runs (during which TRAIN was used as the prototype set).

That set of weights was then used in each of the different Dumrec models reported below. Because Dumrec's weights interact with each other in nonlinear and hard-to-predict ways, they cannot be tuned by trivial parameter-tuning algorithms. In fact, an attempt to tune the weights using a genetic algorithm (GA) [Goldberg, 1989] provided different, but not objectively as successful, weights than those we tuned by hand. As has been discovered over and over again in machine learning, an intelligent user is often the best way to tune complex models.

The three faces of Dumrec

Chapter 8 presents the results of three different Dumrec models. The models discussed there differ according to the properties utilized in computing a letter's score (*i.e.,* level of the data allowed in processing). Table 6.3 shows all of the models arranged by type and introduces the names we will use to refer to them below.

Altogether we studied three versions of Dumrec, two degraded and one non-degraded: Dumrec-S, a model using only the bit-level properties of Level 0; Dumrec-NT, a model using Level-0 through Level-2 data but no Level-3 information (*i.e.,* no tips or points); and Dumrec, the full model.

Properties included in the model		
Level-0 only *simple*	Levels 0-2 *no tips*	All properties $\approx$ *abstract*
Dumrec-S	Dumrec-NT	Dumrec

Table 6.3: The three flavors of Dumrec. "Properties included" corresponds in some sense to degrading the model.

Dumrec was originally intended as a winner-take-all (WTA) model in which the scores would yield a simple rank-ordering of the prototypes. The selection rules we tested reflect this philosophy simply and easily. Under the WTA-*token* selection scheme, the prototype token (from TRAIN) with the highest score (meaning that its properties are in some sense the most closely matched to the mystery letter's) wins, and the mystery letter's *category* is presumed to be the same as that of the winning *token*. We also extensively tested a WTA-*category* selection scheme (computed by averaging together the scores of the five prototype tokens in each of the 26 possible categories, and then picking the category with the highest average score), but, curiously, performance using this scheme was essentially the same as that of models using the WTA-token scheme.

For some of the results reported below, the WTA-token selection scheme was modified in order to make Dumrec's output continuous-valued (and no longer winner-take-all), so that its performance could better be compared with that of humans. To make the program less dependent on winner-take-all selection, we used an exponentially decreasing scoring function to assign a "probability score" to each letter category based on its prototype tokens' relative rankings in terms of score. At most, 6 categories were awarded points: the highest-ranking prototype's letter category was awarded 0.5 points, the next highest 0.25, then 0.125, and so on, decreasing by half until the number of points awarded became negligible (after 6 applications). Note that the sum of all these scores is nearly 1. The exponentially-decreasing function was especially appropriate in this case for two reasons: firstly, it provided a natural way to convert the rank-ordering into a continuous-valued scoring scheme (while preserving the winner-take-all ranking); and secondly, because TRAIN happened to have 5 prototype letters for each letter category, it lent itself to a scoring function that gave the most weight to only the top few ranks. The resulting "probability scores" for the top categorizations of each mystery letter allow for the construction of a fuller, more realistic error matrix for the Dumrec models.

Although use of the exponentially-decreasing scoring rule certainly affected correlations of the model with human data, the overall effect (in terms of general system performance) of changing selection rules turned out to be quite minor.

As would be expected, Dumrec is reasonably good at gridfont letter recognition, most of the time either guessing the correct letter or picking a "reasonable" wrong one. Results of Dumrec's performance will be discussed below (both in Section 6.3.3 and, more extensively, in Chapter 8).

6.3.2 Netrec

We have also completed several experiments in gridfont letter recognition using very simple two- and three-layer feedforward connectionist networks. These "Netrec" networks were trained (using backpropagation) on a variety of gridletters and then tested on letters they had not yet seen. Our results show that connectionist networks trained on normal letters can perform fairly well on recognition tasks if the test letters are also very normal [McGraw and Drasin, 1993; McGraw et al., 1994a; McGraw et al., 1994b]. The ability to generalize from input patterns and to create complex association maps between input patterns and their categories is of course the key to gridletter recognition — in fact, to recognition of anything at all. However, due to the statistical, quanta-level nature of their associations, networks trained on normal letters have a very hard time categorizing eccentric letters.

Our networks use a localist representation for the input of a gridletter. The vector provided to the network has exactly the same form as the one provided to Dumrec, but Dumrec has much more knowledge programmed into it from the start. Unlike Netrec, for example, Dumrec knows about tips, closures, simple features, and so on. The input layer consists of 56 nodes, one for each quantum on the grid. If a quantum is "on" (*i.e.,* it has a line drawn in its position), the quantum's associated node gets high activation. If it is "off", its node gets little or no activation. This representation is sparse (*i.e.*, strictly localist), and provides no spatial-relationship information to the network. That is, the fact that quantum 0's grid-location happens to be very near quantum 14's on the grid (or that quantum 13's grid-location happens to be very far

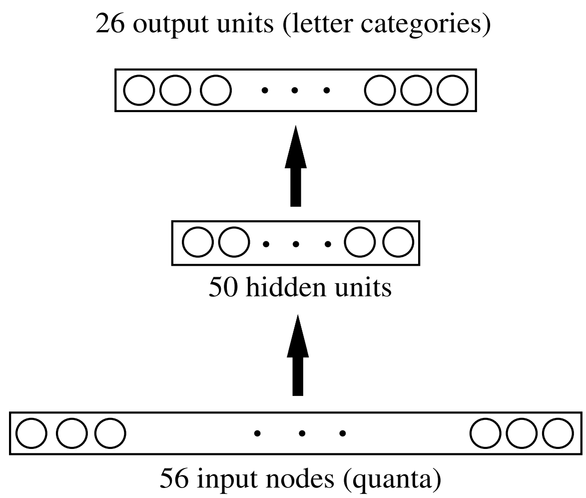


Figure 6.8: Netrec's architecture. The solid arrows denote full connectivity.

away from quantum 14's), is not reflected in the numerical ordering imposed on the input layer. Instead, the network is forced to glean this information from its training set during the learning phase.

The output layer is simply a vector of 26 nodes, one for each letter of the alphabet. We have experimented with hidden layers consisting of anywhere from 10 to 70 nodes.[13] The network is trained to associate patterns on its input layer with patterns on its output layer, using standard backpropagation of error to adjust connection weights.

We have also trained and tested networks using a richer "feature-based" connectionist approach more similar in nature to those found in [Fukushima, 1989], [Guyon et al., 1989], and [Regier, 1991]. Our approach was to train a number of small "subnets" to detect certain features, and then to combine these subnets together into a letter-recognizer. Surprisingly, recognition results for such feature networks were not significantly better than or even different from those of regular three-layer nets. For this reason, we have chosen not to report them here. See [McGraw and Drasin, 1993]

[13]Some experimentation has also been done with perceptron nets having no hidden layer.

for more information.

One of the open problems in connectionism involves deciding how to pick both a learning rate for backpropagation and the number of hidden units to use. We know of no theoretical results that help guide the choice of these two crucial parameter values. There are some fairly obvious rules of thumb that can be consulted for guidance, but they do not always apply. We opted to determine values for these two parameters experimentally by testing all pairwise combinations of networks with 14 different learning rates (.001, .01, .05, .1, .15, .2, .25, .3, .35, .4, .45, .5, .8, and .99) and 10 different numbers of hidden units (10, 20, 30, 40, 50, 55, 58, 60, 65, and 70) on two different learning tasks. Based on results of experimentation, we decided to use a learning rate of 0.4 and 50 hidden units for the comparison tests reported below. The backpropagation momentum rate was set to 0.00001.

	Learning rate													
HU	.001	.01	.05	.1	.15	.2	.25	.3	.35	.4	.45	.5	.8	.99
10	2.28	1.91	2.00	1.27	0.61	0.39	0.30	0.26	0.22	0.20	0.18			
20	2.17	1.94	1.57	0.47	0.27	0.21	0.17	0.15	0.14	0.12	0.11	0.09	0.00	0.06
30	13.4	1.96	1.28	0.36	0.23	0.18	0.15	0.13	0.12	0.11	0.10	0.08	0.06	0.05
40	max	10.8	1.5	0.34	0.21	0.16	0.14	0.12	0.11	0.10	0.09	0.08	0.06	0.05
50			16.1	1.03	0.29	0.19	0.15	0.13	0.15	0.10	0.09			
55	16.5	14.2	2.15	0.44	0.26	0.20	0.17	0.15	0.13	0.12	0.09			
58	max	max	max	23.1	18.6	14.5	0.53	0.21	0.15	0.13	0.11			
60		max	max	24.1	20.4	16.7	0.46	0.29	0.17	0.14	0.11			
65		16.0	14.7	9.32	2.54	0.34	0.20	0.16	0.14	0.13	0.12			
70		max	max	max	max	max	max	max	max	max	max			

Table 6.4: Average error rates of various networks trained and tested on *Standard Square*. HU stands for hidden units. In general, an error rate greater than 2 signifies very bad performance (non-convergence) and an error rate less than 0.2 signifies almost perfect convergence. Note that a very low error rate may also signify overlearning.

In the course of empirically determining the learning rate and hidden-unit parameters, we came across an anomalous result that is worth briefly reporting here. Table 6.4 shows a convergence chart for three-layer networks trained on a particular gridfont

(*Standard Square*) for 50,000 cycles. We actually tested more networks than those shown on the chart, but none of them performed better than the ones shown. The numbers on the chart represent average error rate over all output nodes for a test run. The lower this number, the better the convergence. A graphical interpretation of part of the chart is shown in Figure 6.9. We were surprised to find that none of the networks with over 70 hidden units converged, no matter what the learning rate was. We are not sure why this is, and would like to explore this further in the future.

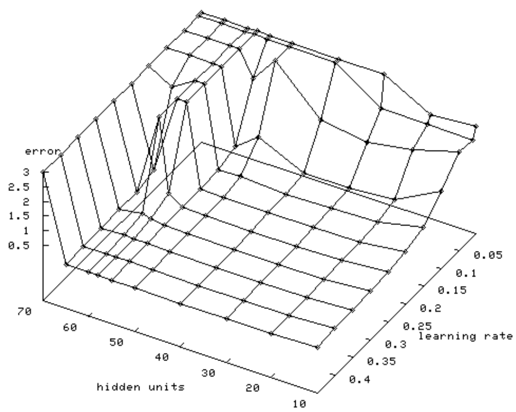


Figure 6.9: Netrec convergence on the gridfont *Standard Square*.

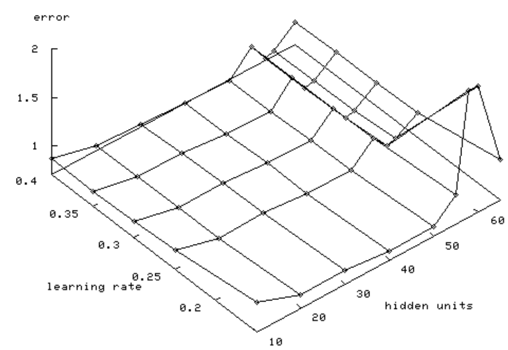


Figure 6.10: Netrec convergence on the dataset NORMALS.

A similar convergence pattern (having the same inexplicable many-hidden-unit non-convergence property) is shown in Figure 6.10. These data are from a three-layer network run on NORMALS. It would be interesting to investigate the causes of non-convergence and determine to what extent they can be accounted for by theoretical considerations.

Representational and training issues in Netrec

After we had published some initial results, it was brought to our attention that Netrec's performance might be drastically improved if two things were done: 1) improve its somewhat weak localist representation; and 2) train it on more than five examples of each letter [Doug Blank and Mike Gasser, personal communication]. Our interest in these suggestions led us to experiment with the second (and by far the easier) of these possibilities to see how far Netrec's simple architecture could be pushed, given a richer training set.

It was quite simple to revamp the training set. We took a set of 455 gridfonts and removed the entire PSYCH−TRAIN test set. Fonts in the new training set range from plain-Jane to completely strange and provide around 200 distinct and highly variegated tokens from each category. The new dataset includes the original TRAIN set as a subset. After duplicates have been removed, the training set contains 5481 unique tokens, with an average of 211 tokens for each of the 26 categories. By contrast, the original dataset has only 130 tokens, with just 5 examples of each category. After backpropagation training using the same parameter settings that we used in the Netrec model, we tested the trained network on the PSYCH−TRAIN dataset.

Performance of the Netrec model trained with the huge new dataset skyrockets, jumping 19.4% in accuracy from 59.8% for the original model to 79.2% for the new model, which we will call Netrec+. This vast improvement should come as no surprise, however, since the new training set includes many tokens *very much like* the tokens that the network is eventually tested on (see Figure 6.11). It is well-known that three-layer backpropagation networks like Netrec+ are able to learn statistical mappings from their input units to their output units and to generalize to never-before-seen examples — that is, as long as the never-before-seen examples are sufficiently close, statistically speaking, to the training patterns [McClelland and Rumelhart, 1986]. In terms of our task, this means that if a network of this sort is trained on a variety of weird gridletters, it will surely be able to recognize gridletters that are *weird in the same kind of way* as its training letters.

To make this concrete, consider the letter 'b' and Netrec+. In our huge dataset, there are 249 particular 'b's that the network is trained on. These include a fair

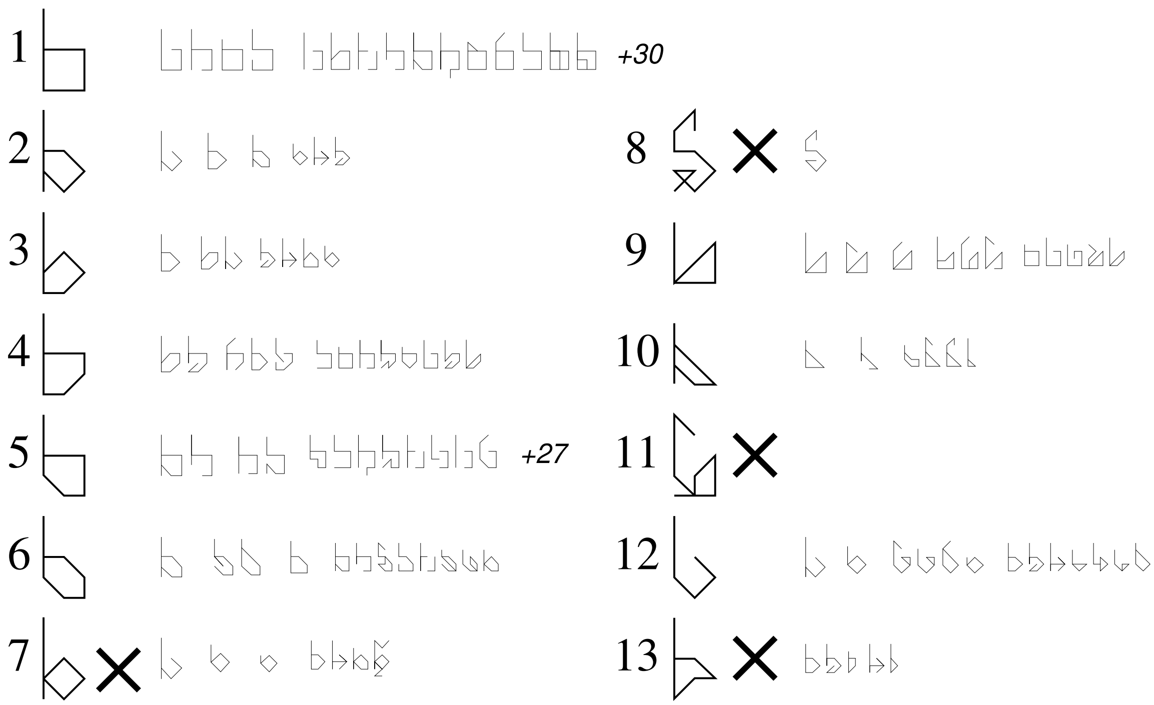


Figure 6.11: The 13 'b's that Netrec+ was tested on (4 of which it missed) are shown here. The 4 missed tokens are marked with an **X**. Next to each of the 13 test letters, the most similar letters from the training set (where similarity is measured in terms of quanta-level overlap) are shown. Each training letter is displayed with a size inversely proportional to its Hamming distance away from the test letter. Less similar training letters are smaller. Only letters within a Hamming distance range of 1 to 4 quanta are included in this chart. *All* of the closest training letters are shown for each test letter (except in the case of test letters 1 and 5, as they had too many similar training letters). As is apparent from this chart, Netrec+ is unable to recognize letterforms that are too far away (in terms of quanta overlap) from letterforms that it was trained on.

number of "regular" 'b's (especially those making up the TRAIN subset) as well as a large number of quirky, strange, grid-affected 'b's. After being trained on each of these 'b's around 200 times[14], Netrec+ was tested on the 13 'b's of PSYCH−TRAIN. Netrec+ was able to successfully categorize 9 of these 13 'b's. Figure 6.11 shows all 13 of the 'b's, and marks the four that were incorrectly categorized with a dark '**X**'. All 9 of the 13 properly categorized examples are, on a quantum level, nearly identical to some of the training 'b's. The training 'b's most similar to the testing 'b's — in terms of quantum-level overlap — are shown to the right of the 13 test 'b's in the figure. So Netrec+'s performance, in terms of 'b's, anyway, is just what is to be expected. It is able to learn associative connections between sets of quanta and categories *in a low-level way.* Since its task (which is simply to activate one of the category nodes highly given a 56-node input pattern) requires no explicit role-based processing, the network has no reason to develop internal representations in terms of roles. It can solve its task using lower-level statistical associations, so it does.

The upshot of all this is that although the performance of Netrec+ by far exceeds that of Netrec in terms of accuracy, there is strong evidence that the kind of post-learning processing done by the Netrec+ model (with the better training set) is very much the same as Netrec's — statistical in nature, focusing on low-level associations between quanta and categories. Our simple experiment just goes to show the importance of picking a rich training set when training a model like this. Netrec and Netrec+ use exactly the same algorithm to attack their respective tasks. The only difference is the training set. Backpropagation networks (especially given our simple architecture) are fantastic mimics. *What* they are trained to mimic, however, makes all the difference in the world.

[14]Netrec+ was trained for 1,000,000 cycles, after which it was determined (empirically) that further training, though still leading to slight convergence over the entire training set, was no longer affecting performance on the test set.

Our suspicions are further borne out through analysis of the errors that the Netrec+ model makes, compared to those that humans make, on the PSYCH dataset.[15] Correlations of the Netrec+ model with human data are superior to those of the Dumrec models (and the original Netrec too, of course) but do not equal those of the role model. Another analysis, this time of bidirectional category errors, leads to the same conclusion, with the Netrec+ model outperforming both Dumrec and the old Netrec in terms of its prediction of human errors, but trailing significantly behind the role model. In short, the kinds of errors that Netrec+ makes are consistent with low-level processing involving just quanta and no higher-level structures.

It is not surprising that even with an improved training set, processing in the Netrec+ model is completely constrained by the statistical methods exploited by backpropagation in order to associate quanta with categories. This kind of model, though it can certainly learn to categorize very well indeed, works at too low a level. We argue that the processing should occur at the level of *parts* (as defined conceptually by roles), and not at the level of quanta or at the level of small groups of associated quanta *not constrained by roles*. What is missing in Netrec+ is this kind of higher-level, explicit part processing. Without more realistic training subtasks, possibly coupled with a more advanced architecture — two things that could force a network to pay attention to the level of roles — no network of Netrec+'s type has any reason to develop implicit higher-level processing by itself. Nor should such a model be expected to develop role-like structure, short of a miracle.

Our Netrec models are very simple and do not make use of the most sophisticated techniques that connectionism has to offer. Perhaps by changing Netrec's architecture and/or manipulating the way it is trained, a new type of Netrec could learn to categorize letterforms with regard to their constituent parts (and in this way serve

[15] The human experiment is reported in Chapter 7. Netrec+ is analyzed in terms of the human data, along with all of our other models, in Chapter 8.

as another implementation of the role hypothesis). A connectionist approach based on part-level processing is certainly conceivable, but such an approach would push the limits of what can be done today in connectionism.[16] Further research into these issues is certainly warranted.

The role model has nothing to say about where roles come from or how they are learned. Of course, neither does Netrec, as it shows no sign of role-level processing. But in any case, a model that developed the right kind of role-level processing *on its own* would have a distinct advantage. Coding in knowledge by hand is a tedious and time-consuming process that would be better avoided. Care must be taken to address the sorts of issued raised by the role model, however (such as emergent structure-building, a mix of top-down and bottom-up pressures, etc.). It is clear that a naïve learning approach, like Netrec's, does not do so.

One critical point that applies equally to Dumrec, Netrec, and Netrec+ has to do with the end result of the recognition process. The role model yields a parsing of a letterform in terms of its constituent, role-filling parts. It provides the sort of information that can be used in further analogical processing during the design of other letters. None of the other models do this.

Consider the following hypothetical situation. What if a "perfect" model of categorization existed, but it did not provide any information about what a letterform's parts were? That is, suppose it correctly categorized each letter it was given, but did not parse it into roles. Could such a model conceivably play the Examiner role in an implementation of Letter Spirit? Without some way of explicitly being able to refer to letter parts, further processing is severely hampered and may well be impossible. The role model provides an answer to this problem by avoiding it from the outset. The proper strategy is to explicitly fit a letterform into roles during the process of

[16]This point is related to our previous discussion of *reference* and *part/whole processing* in connectionism. See Section 6.2.1.

categorization.

6.3.3 A brief look at some general results

Stimulus	Dumrec models			Netrec	Netrec+
	S	NT	Full		
PSYCH−TRAIN	55.9	62.7	69.6	59.8	79.2
NORMALS−TRAIN	64.5	74.9	83.4	71.0	89.7
FONTS	41.7	42.3	46.8	41.0	63.0
Bowtie (26)	38.5 (10)	53.8 (14)	53.8 (14)	38.5 (10)	64.4 (17)
Double Backslash (26)	19.2 (5)	15.4 (4)	19.2 (5)	19.2 (5)	47.2 (12)
Flournoy Ranch (26)	34.6 (9)	42.3 (11)	46.2 (12)	34.6 (9)	59.9 (16)
Hint Four (26)	80.8 (21)	65.4 (17)	76.9 (20)	73.1 (19)	84.5 (22)
Close (26)	34.6 (9)	26.9 (7)	26.9 (7)	26.9 (7)	61.3 (16)
Sabretooth (26)	42.3 (11)	50.0 (13)	57.7 (15)	53.8 (14)	64.2 (17)

Table 6.5: Recognition performance of the Dumrec and Netrec models on several data subsets. The main numbers recorded here are correctness percentages computed at the token level. Also in parentheses is the total number of letter categories properly recognized for each font.

Table 6.5 shows accuracy values for our simple models on several data subsets. These accuracy values are computed by dividing the total number of tokens correctly classified (pooling all categories together) by the total number of tokens. Note that the large data set reported here is PSYCH−TRAIN, not PSYCH. This is because the programs were trained on TRAIN, and testing the models on the training letters is of no value at all. In particular, Dumrec immediately recognizes its prototype tokens and categorizes them correctly every time. Netrec's performance on the training set is also very good, although not perfect. Since we are interested primarily in the generalization performance of the models (which sheds more light on the representations the models develop and the type of processing they use), we chose not to count performance on TRAIN, and to test the models only on letters they had never seen before. The numbers reported above (as well as those reported in Chapter 8) for the Netrec models are an average of the performance of twenty different networks, each fully trained on their respective training sets.[17]

The best overall performance (in terms of accuracy) is that of the Netrec+ model. Next best is the performance of the complete Dumrec model. The Dumrec model outperforms the degraded Dumrec models for the NORMALS and two of the fonts. The simple connectionist networks (Netrec) fall somewhere between the Dumrec-S and Dumrec-NT, with nothing exceptional about their behavior.

The behavior of Dumrec and Netrec is better understood in the context of human data. After introducing the human experiments in Chapter 7, we will compare and contrast the performance of Dumrec, Netrec, humans, and the role model in Chapter 8.

[17]Each of the twenty networks was initialized with a different set of random weights before training.

CHAPTER SEVEN

Gridletter Perception by Humans

In the previous chapter, we introduced a distinction between "flat" and "structured" approaches to object recognition. We noted that many important psychological theories of perception favor a hierarchically- and relationally-structured approach, in which intermediate-level structures are built up from visual stimuli [Schyns and Murphy, 1995; Hock et al., 1988; Biederman, 1987; Palmer, 1978; Palmer, 1977]. Furthermore, some psychological research suggests that a structured approach can be successfully applied to letter recognition [Treisman and Gelade, 1980; Sanocki, 1986]. Our role model shares this conviction.

In this chapter we present the results of two simple human experiments in letter perception. These experiments strongly support the idea of a structured approach to letter recognition. As will be shown, the results of our experiments are all consistent with the role hypothesis. More detailed discussion and analysis of the human data introduced here makes up the contents of the next chapter, in which human and machine performance are compared.

Many cognitive scientists have noted the importance of studying error-making in order to shed light on cognitive processing [Norman, 1988; Dell, 1986; Cutler, 1982; Fromkin, 1980]. For this reason, in addition to considering *accuracy* rates in this chapter (and the next), we pay special attention to error-making, trying to understand why certain errors are made by people, and what this tells us about their cognitive processes. Our emphasis on error-making carries over into Chapter 8, where performances of our models are compared and contrasted using the human data presented here.

7.1 Predictions of human performance

The experiment reported below is concerned with the recognition of gridletters. The purpose of the experiment is twofold: to gather data for comparison with cognitive models of letter perception, and to shed light on the human perception of lowercase roman letters that are stylistically eccentric. It is important to keep in mind the difference between the task we are asking people to do and other tasks that are related to letter recognition, like reading or unlimited-time categorization. Our task focuses on the quick recognition of single, isolated letterforms, without regard to word-level or other contextual effects.

Letters come in myriad styles, ranging from fairly bland (like the Times typeface in which this thesis is set) to highly stylized (like the 'a's of Figure 1.1). We are interested in the ways in which people perceive all kinds of letters — including ones that are eccentric. In designing our data set, we created two large groups on either end of the continuum of style. Each of the two groups — one with prototypical, non-eccentric, vanilla-style letters, and the other with nonstandard, eccentric, style-loaded letters — has many diverse letters as actual members. In the "normal" group there are some letters that are more daring than their compatriots, and in the "eccentric" group there are some letters that flirt with normality. The validity of our subjective division of these letterforms into the two groups is confirmed by the objective data that we gathered in the course of the experiment.

We now proceed to discuss the results of our experiment in terms of the role hypothesis. Although our results are compatible with the role hypothesis, they are also compatible with other structured approaches to letter perception (see Chapter 6). Stronger empirical evidence for the role hypothesis is presented in Chapter 8, where the performances of the role model, Dumrec, and Netrec are compared with that of people.

Our central predictions all relate to the hypothesis that letter concepts are comprised of simpler conceptual *roles* (see Chapter 4). The abstract, conceptual nature of roles is what allows people to easily recognize letters that populate the fringes of a letter category. According to our theory of roles, human letter perception should exhibit the following characteristics:

1. Reaction time should be longer and recognition rate lower for complicated stimuli, in which roles are either ambiguously or poorly filled, than for relatively normal letters, with fairly straightforward role-fillers. In other words, letters with role-fillers on the edges of acceptability should both take longer to recognize and be recognized less successfully, on average, than letters that have normal role-fillers. For example, the top two letterforms of Figure 4.3 should be more quickly and successfully recognized than the lower two. That complicated letters are harder to recognize is not a surprising result. What is important for our purposes is that our experimental observations are compatible with the role theory.

2. Low-level proximity (*i.e.,* degree of overlap of the quanta of a test letterform and the quanta of prototypical templates) alone should not account for errors in human letter recognition. Instead, errors should be explainable in terms

of role plasticity, parts that fill roles, and role-level similarity between letter-conceptualizations. Errors made by human subjects should be most prevalent between letter categories that have similar sets of roles — roles that are defined by similar sets of norms. The roles shown in Figure 4.2 suggest why 'n's and 'r's, or 'q's and '*g*'s, should be easily confused by people, whereas 'n's and 'm's, or even 'e's and 's's (which are very often *syntactically* similar in the sense of having nearly identical sets of quanta, but are not closely related at the role level) should not.

3. The timing of particular errors should reflect the underlying role structure of letters. For instance, errors made in classifying letterforms with parts that fit reasonably well into one or more sets of conceptual roles should be made more quickly than errors made in classifying letterforms with parts that are too eccentric to fit any role-set well. The more ambiguous the parts and the more tenuously they fill roles, the more time-consuming and less accurate the classification should be.

7.2 The experiment

Method

Stimuli. A black grid was displayed against a white background on a Macintosh screen. The 21 vertices were drawn with a unit distance of 1.31 cm, making the entire grid 7.86×2.62 cm. A solid square with sides of 0.13 cm was drawn at each vertex. Every legal quantum in the grid was drawn in with a line thickness of 0.04 cm. Thus the grid provided noise for each gridfont letter (henceforth, *token*) displayed. A token was displayed by drawing its active quanta with a line thickness of 0.13 cm. One example from the set of stimuli is shown in Figure 7.1. Subjects were seated, but allowed to maintain whatever distance from the monitor they found best.

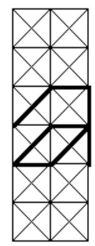


Figure 7.1: A sample stimulus (captured directly from a Macintosh screen). The presence of the grid provides some degree of noise. This gridletter is an 'a'.

Procedure. A dataset of gridfont characters was shown to each subject. The tokens were presented individually, with the screen blank for two seconds between each response and the next stimulus onset.

Subjects were told that they would see a series of stimuli, each of which was intended to represent a lowercase roman letter. Responses were to be rapid keyboard keypresses that identified the stimulus as one of the 26 lowercase roman letters; identification of a stimulus as any other character in any other alphabet was disallowed. The importance of correct identification was stressed over that of speeded response, although subjects were aware that their response times were being recorded. It should be noted that keyboards are labeled with uppercase letters, and so do not provide novel prototypes of lowercase letters.

Some of the datasets introduced in Chapter 4 were used in these experiments. They are listed again in Table 7.1 as a convenience. Dumrec, Netrec, and the role model were all tested on the same datasets. Human and machine performance will be compared and contrasted in Chapter 8.

For 21 of the subjects, the dataset was NORMALS plus four of the gridfonts from FONTS (104 more tokens, for a total of 492): *Bowtie*, *Double Backslash*, *Flournoy Ranch*, and *Hint Four*. Two additional gridfonts (*Close* and *Sabretooth*) were added for the final 14 subjects only (for a total of 544 tokens). The order of presentation was uniquely randomized for each subject, to alleviate possible context effects. Subjects were not made aware that any letters were drawn from stylistically consistent gridfonts (in FONTS).

NORMALS	388 tokens, all relatively strong letters ranging from very standard to somewhat stylized
FONTS	six complete gridfonts (156 tokens) ranging from somewhat stylized to completely eccentric: *Bowtie*, *Double Backslash*, *Flournoy Ranch*, *Hint Four*, *Close*, *Sabretooth*
PSYCH	NORMALS plus FONTS (544 tokens)

Table 7.1: Datasets used in the experiment. The datasets are displayed in Appendix A as Figures 9.2 and 9.3.

Subjects. 35 subjects were drawn from the Indiana University community and included undergraduates, graduate students, and staff. None were involved in work related to gridfonts, or to fonts of any kind. All subjects were compensated for their participation and were debriefed after the experiment.

7.3 General results

Our experiment involved the presentation of 17,973 trials divided over the 35 subjects. 27 of the trials were thrown out — one due to an impossibly short response time (which occurred due to a keyboard entry error) and 26 because of illegal responses (*e.g.,* typing '9' instead of a valid letter). Subjects gave between 63.7% and 95.6% correct responses, with a mean of 84.0% correct.[1] Average response times ranged between 0.84 seconds and 3.12 seconds, with a mean of 1.84 seconds. In individual trials, response times varied between 0.47 seconds and 56.4 seconds.

Table 7.2 shows a stimulus-by-response confusion matrix for the dataset PSYCH

[1]As we pointed out in Chapter 5, in order to count as a "correct" response, a subject must have responded with that category in which a stimulus was placed by the experimenters (the category intended by the designer of the given gridletter). For some of the more ambiguous tokens (see especially Figure 7.6), categorizing turns out to be a very hard task. We use the designer's intended category in order to provide a fair comparison between humans and the recognition programs discussed in Chapter 8. In the case of naïve designers, it is possible that the intended category is recognized by very few people; however, when designers are experienced, such design gaffes do not arise often, and it is reasonable to take the designer's intention as synonymous with "correct response". Use of the term "correct" below will always be subject to this disclaimer.

generated by averaging over the 35 subjects and collapsing tokens into their respective categories. The table shows the percentage of the time that tokens belonging to a particular category were categorized in each of the 26 response categories. Correct-response percentages can be read down the diagonal. The unidirectional error rates shown in the table can be averaged together to provide a bidirectional category error rate. As an example, consider 'a's and 'b's. 'a' tokens were properly categorized 82.9% of the time, and 'b's 91.4% of the time. 'a's were taken for 'b's 0.9% of the time while 'b's were taken for 'a's 0.2% of the time. The resulting bidirectional category error 'a'↔'b' is thus 0.55%.

Our confusion matrix differs considerably from others like it in the psychology literature since it includes data about a variety of lowercase alphabetic styles instead of just one. All studies of lowercase letter recognition that we are aware of rely on data from just one font, usually a very plain and standard font at that [Jacobs et al., 1989; Geyer, 1977].[2] Because the style-permeated nature of some of our tokens made them hard to recognize, we did not have to resort to short stimulus-display times or other tricks for degrading performance (such as distance reading [Bouma, 1971]) to get a well-filled confusion matrix. The stylistic complications of some of our tokens, aided and abetted by perceptual noise provided by the grid, were sufficient to cause the generation of a non-sparse matrix.

Even though our dataset is made up of arguably idiosyncratic gridletters, our stimuli are more realistic than many past data sets, in the sense that they capture more of the natural variability found among letterforms. This leads to a more thorough treatment of errors than has been evident in past work. As confusion matrices have been shown to shed light on letter similarity [Gilmore et al., 1979], it is informative to note the most common perceptual errors and to try to understand why they occur.

[2][Boles and Clifford, 1989] presents a confusion matrix including both upper- and lowercase tokens, while [Townsend, 1971] presents an often-cited uppercase confusion matrix.

stimulus	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	u	v	w	x	y	z
	response																									
a	**.829**	.009	—	.028	.017	—	.010	—	—	.004	—	—	—	—	.021	—	.011	.003	.001	.003	.009	.010	.017	.013	—	.015
b	.002	**.914**	—	.005	.004	—	.002	.005	—	.002	.005	.008	—	—	—	.003	—	—	.043	.002	.002	.005	—	.002	—	.002
c	.014	—	**.863**	.002	.015	—	.023	—	.004	—	—	.010	—	—	.008	—	—	.002	.002	—	.012	.021	.017	.003	—	.004
d	.035	.010	—	**.890**	—	—	.012	—	—	.028	.001	—	—	—	—	—	.003	—	.009	—	.003	.003	.001	.001	—	.003
e	.006	—	.012	.001	**.905**	—	.009	—	—	—	—	—	—	—	.002	.032	.022	—	.001	—	—	—	—	—	.001	.008
f	.002	—	—	.002	.031	.717	.005	.040	.015	.018	.009	.003	.020	—	—	.006	.008	.009	.006	.104	—	.002	.003	—	.002	—
g	.006	—	—	—	—	.003	**.891**	.004	—	.008	—	—	—	—	—	.004	.058	—	.022	—	—	—	—	—	.004	—
h	—	.025	.004	—	.005	.002	.007	**.890**	—	—	.007	—	.002	.025	—	—	—	—	.014	.002	.002	.002	—	.002	.012	—
i	—	.007	.004	—	.005	.017	.012	—	.755	.097	.001	.003	.003	.001	.005	.004	.004	.005	.046	.015	.001	.004	.004	.004	.003	.001
j	.001	.008	—	.001	.002	.001	.007	.001	.052	**.857**	.001	.002	—	—	—	—	.002	—	.057	.002	.001	.001	—	.001	.003	—
k	—	.011	.005	—	.005	.009	.006	.080	—	—	.799	.008	.003	.003	—	—	—	.002	.006	.002	.003	.006	.002	.051	.002	—
l	—	.028	.036	—	.002	.015	.003	.003	.175	.019	.003	.592	—	—	—	—	.003	.007	.030	.029	.025	.003	—	.002	—	.024
m	—	—	—	—	.008	.008	—	.001	—	.001	.001	.001	**.917**	.017	—	.001	—	—	.003	.003	.001	—	.003	.019	.017	—
n	.027	.003	.002	.007	.003	.004	.003	.022	.008	.003	—	—	.042	**.804**	.005	—	—	.012	.002	.011	—	—	.004	.021	.010	.008
o	.024	.014	.003	.055	.007	—	—	—	.003	.004	.001	—	—	.004	**.837**	.001	.011	.003	.003	—	.009	.020	—	.001	—	—
p	—	—	—	—	.013	—	.016	—	—	—	—	.001	—	—	.001	**.939**	.018	.006	—	—	—	—	—	—	.004	.002
q	.004	.002	.001	.001	.002	—	.184	—	.001	.002	—	—	—	.002	.002	.012	.779	—	.005	—	—	—	.001	—	.001	.002
r	.011	.004	.066	—	.012	.003	.001	.008	.017	.003	.003	—	—	.069	.004	.025	.001	.676	.013	.015	—	.004	—	.029	.027	.009
s	.004	—	—	—	.003	—	.004	—	.001	.003	—	.003	—	.013	.003	—	.001	.005	**.924**	—	.004	.011	.001	.004	.005	.012
t	.003	.011	.024	—	.005	.013	.009	—	.008	.002	.006	.006	—	.002	.002	—	.003	.019	—	**.854**	.003	—	.002	.018	.009	.003
u	.008	.003	.004	.001	.003	.001	.003	.001	—	.007	.009	.001	—	.001	.003	—	—	.004	.001	.003	.812	.100	.018	.003	.014	—
v	—	.007	.017	.001	.017	—	.001	—	—	.002	—	—	—	—	.009	—	—	—	.002	.001	.206	.658	.076	.001	—	—
w	—	—	—	—	.012	—	—	—	—	—	.011	.001	.001	—	—	—	.002	—	.002	—	.004	.011	**.931**	.019	.005	—
x	.001	—	—	—	—	.003	—	.004	—	.002	.006	—	.010	.002	—	—	—	.003	.009	.019	—	—	—	**.928**	.004	.010
y	.001	—	—	—	.004	—	.005	—	—	.002	—	—	—	—	—	—	.002	—	.004	.001	.010	.001	.005	.001	**.960**	.004
z	.003	—	.026	.006	.022	—	.007	.004	.003	—	—	—	—	—	.002	—	.008	.007	.013	.015	.008	.007	.003	.036	.037	.790

Table 7.2: Confusion matrix of human performance on PSYCH.

Stimulus set	accuracy	response time	# of quanta
PSYCH	84.0	1.84	6.80
NORMALS	90.1	1.46	6.97
FONTS	68.7	2.80	6.36
Bowtie	52.1	3.79	7.65
Double Backslash	66.2	2.88	6.00
Flournoy Ranch	57.4	3.44	7.54
Hint Four	85.3	1.95	4.92
Close	74.8	2.12	5.31
Sabretooth	76.7	2.61	6.73

Table 7.3: Human recognition performance averaged over 35 subjects. Accuracy ratings reported here were computed at the token level.

7.3.1 Prediction 1

Table 7.3 shows average percent-correct and response-time values for some of the data subsets. These values were computed using token-level computations, in which percentages are simply the number of tokens correctly categorized divided by the total number of tokens. Category-level analysis is not included in these numbers. For all groups, with the exception of the gridfonts *Sabretooth* and *Close*, accuracy and speed have corresponding ranks. In general, correct responses came more quickly than errors, but the letters that were identified the most quickly were not necessarily those identified correctly most often. Generally, as response time increases, accuracy decreases ($r = -0.74$, $p < 0.01$ for a correlation over all tokens averaged over all subjects). Both of these measures can be understood in terms of a third variable corresponding to an intuitive notion of *difficulty*, with response time rising and accuracy falling as difficulty rises.

Difficulty can be explained in terms of the ease with which a letterform can be parsed into parts that fill roles. Complicated letterforms take longer to parse into roles, causing response time to rise. Likewise, letterforms with parts that match role-sets poorly or ambiguously cause more errors than letterforms with parts that easily fit within the norms provided by role-sets. Figure 4.3 provides an example of "good" role-fillers vs. "bad" role-fillers. The role hypothesis is thus consistent with the relationship between response time and accuracy, as stated in Prediction 1.

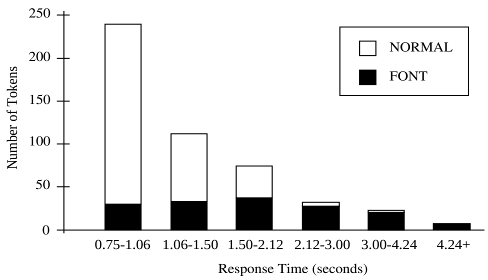


Figure 7.2: Each column shows the total number of tokens that occurred within certain response time ranges. Tokens are separated into FONT and NORMAL groups that (stacked together) sum to the column total.

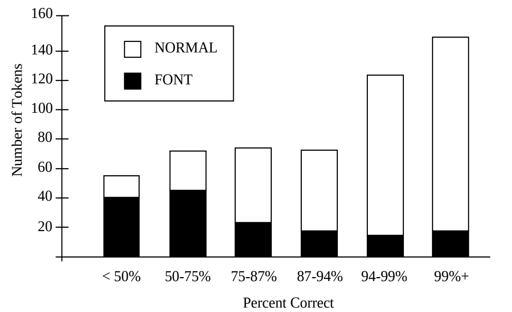


Figure 7.3: The number of tokens, separated into NORMALS and FONTS groups (as in the previous figure), that occurred within certain accuracy ranges.

Of all the gridfonts, the one with the hardest letters to recognize was the complicated *Bowtie*, in which the highly eccentric style pushes the gridletters to the fringes of category space. The easiest font was the relatively simple *Hint Four*. Easier than the full fonts were the letters belonging to the class of NORMALS, which, of course, we picked for their normalcy (see Figure 9.2). There is a striking difference in response time (and accuracy) between FONTS and NORMALS. Figure 7.2 shows a histogram of the number of tokens in the dataset that fell into certain response-time bins. It shows that most of the tokens in NORMALS are recognized quickly, while most of the FONT tokens take longer. Figure 7.3 shows a similar histogram with accuracy along the x-axis. Clearly, most NORMAL tokens are recognized more accurately than the FONT tokens are. These two effects also hold within the FONTS category itself, with some fonts having a lower recognition rate while at the same time taking longer to recognize. The NORMALS generally have standard parts that easily fill roles, whereas FONTS have nonstandard role-fillers. The fact that NORMALS are more easily recognized than FONTS lends further support to Prediction 1.

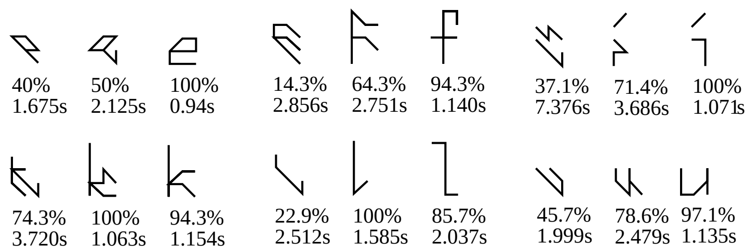


Figure 7.4: Standard vs. nonstandard role-fillers as exhibited in *Double Backslash* (left letterform of each triplet), *Sabretooth* (middle), and tokens from NORMALS (right). There are six sets of triplets, one for each of the letters 'e', 'f', 'i', 'k', 'l', and 'u'. Starting at the left of each triplet, where roles are poorly filled, roles are filled in a more standard fashion towards the right. Role-related difficulty is reflected in the percent-correct and response-time data.

The number of quanta in a gridletter does not alone determine its difficulty, as is apparent in Table 7.3. As a result, response time does not correlate strongly with the number of quanta ($r = 0.097, p > 0.1$); nor does the number of quanta explain a great deal of the accuracy variance ($r = 0.18, p < 0.01$).[3] The NORMALS and FONTS data of Table 7.3 make this especially obvious — note that both *Close* and *Hint Four*, which have far fewer quanta on average than the NORMALS, are clearly harder to recognize. The role hypothesis can account for this result as follows: some gridletters with a higher-than-average quantum count still have well-filled roles, while others having an average number of quanta, or possibly even a lower-than-average quantum count, have troublesome role-fillers. This effect is also apparent within the FONTS group, with *Sabretooth* having fairly strong but heavy role-fillers (making it relatively easy to recognize) and *Double Backslash* having roles filled in less standard ways (making it harder to recognize). Of course, most NORMALS tokens, having very normal role fillers, are even easier to recognize than the *Sabretooth* tokens.

[3]Again, these correlations are made over all tokens.

Figure 7.4 shows some specific tokens (as well as accuracy and response-time data) from *Sabretooth*, *Double Backslash*, and NORMALS to illustrate this point.

7.3.2 Prediction 2

Table 7.4 displays the eleven category pairs that are closest, as determined by overlap of quanta, in the PSYCH dataset. In essence, the list shows the letter-pairs that, on a purely syntactic, quantum-level-only basis, one might expect to observe confusion between. These similarities are exactly of the kind exploited by (flat) template models often found in commercial OCR programs (see Chapter 6). The list was created as follows. The *blurred prototypes* of the PSYCH dataset, graphically represented in Figure 7.5, were first created by averaging together the binary-valued length-56 vectors of all of the letters in a given category, thus providing a real-number value between 0 and 1 for each of the 56 quanta. This results in a single length-56 vector for each of the 26 letter categories. Each such vector acts as a "blurred prototype" template for its letter category. Given any two such vectors, the city-block distance between them (*i.e.,* the sum of the absolute values of their 56 coordinate-pair differences) gives a low-level syntactic measure of category similarity.

Despite the apparent naïveté of the blurred-prototype model, the approach works to some degree. This is shown by the performance of a recognition program that assigned a token to the category that its syntax matched best, in the city-block-distance sense discussed above. This simple model (essentially a variation on the Dumrec-S program we described in Chapter 6) recognized the tokens in our experiment at a reasonable rate (55.9% recognition on PSYCH−TRAIN). Similar models, more flexible than this approach only in the sense of allowing 2-D translation of images or pixels, form the basis of some OCR models in the literature [Mantas, 1986; Mori et al., 1984; Gaillat and Berthod, 1979].

It is interesting to contrast the category-pairs determined by quanta overlap in

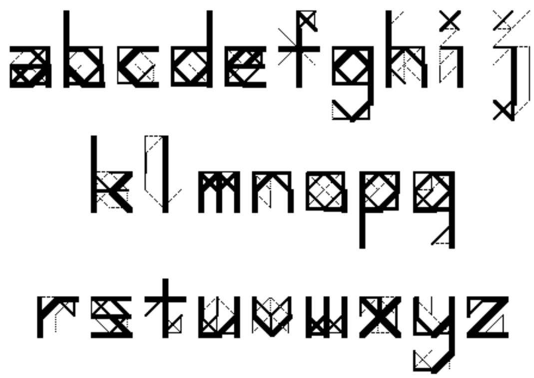


Figure 7.5: The blurred prototypes of the PSYCH dataset. The visual strength of a quantum indicates its relative frequency.

Template	
Error	Error Rate
m↔n	8.80
n↔r	8.35
u↔w	7.15
c↔o	5.75
b↔h	4.90
g↔q	4.80
e↔s	4.60
e↔o	4.55
n↔o	4.30
h↔n	4.15
a↔o	4.15

Table 7.4: The eleven closest categories as calculated by the city-block distances between the blurred prototypes of PSYCH. The bidirectional error rate is taken from the associated prediction matrix (see Section 8.4.1 of Chapter 8). These pairs give some idea of the quantum-level relationship between categories of PSYCH.

Humans	
Error	Error Rate
u↔v	15.30
g↔q	12.10
i↔l	8.90
i↔j	7.45
f↔t	5.85
v↔w	4.35
h↔k	4.35
n↔r	4.05
c↔r	3.40
j↔s	3.00

Table 7.5: The top ten category errors made by humans. The bidirectional error percentages are computed from the confusion matrix shown in Table 1.

Table 7.4 with the most common human category errors, shown in Table 7.5. Although a few human errors seem to be due to simple overlap of letter shapes, most are not. The only human errors in the top ten list that can be explained or predicted in terms of quantum-level proximity are 'n'↔'r' and 'q'↔'g'. Note, however, that both of these letter-pairs share many common roles as well. For instance, 'q' and 'g' both have a bowl or a circle in the central zone, and both have some type of descender on the right. Similarly, 'r' and 'n' both have a stem on the left that goes from the baseline to the x-height, as well as some type of role that attaches to the stem near its top on the left and arches over to the right. It is this role-level similarity of 'r' and 'n' that can cause ambiguity for people. Depending on how close the filler of the latter role gets to the baseline, the amount of 'n'-ness of the letterform will vary. Such structural similarities are suggested by Figure 4.2. In fact, we argue that it is the role similarities that give rise to the shape proximities in these cases, and that confusion in these cases is a result of higher-level role similarities (as stated in Prediction 2).

The cases where quantum-level proximity differs from human errors are more interesting. We consider first the entries in Table 7.4 that represent very rare human errors, and second, the entries in Table 7.5 that represent very distant letter-pairs on a quantum-level basis. Finally, we look at some common token-level errors (as opposed to category-wide averages), and see how they fit the role hypothesis.

First, consider the pairs in Table 7.4 that do not correlate with high-frequency errors in human perception (in Table 7.5). The following pairs are very close in blurred-prototype space but do not occur in the top ten human errors (the numbers in parentheses are taken from Table 7.2): 'm'↔'n' (2.95%), 'u'↔'w' (1.10%), 'c'↔'o' (0.55%), 'b'↔'h' (1.50%), 'e'↔'s' (0.20%), 'e'↔'o' (0.45%), 'n'↔'o' (0.45%), 'h'↔'n' (2.35%), and 'a'↔'o' (2.25%). People do not make these errors often even though the letters are very similar at the quantum level. This result supports Prediction 2.

Our results corroborate the work of Medin et al. (1993) and Goldstone et al. (1991), who show that literal physical properties are often less important in similarity judgments than are higher-level relations. This particular result is also compatible with all accounts of categorization that consider emergent features (*e.g.,* closure) [Treisman and Paterson, 1984]. However, since the role hypothesis is concerned with structured representations made up of emergent features — as each role is comprised of norms governing groups of features, and r-roles provide information about the interaction between roles — it goes a step beyond standard emergent-feature models such as Treisman and Paterson's.

We hypothesize that for any pair of syntactically similar letters, there are salient differences in their role-sets that cause people not to confuse them. For example, the difference between 'b' and 'h' is simply a matter of closure at the bottom, but this seemingly small difference plays a very significant role in people's categorization processing, precisely because they know of the existence (and the conceptual proximity) of the other category. That is, the closeness of the role-sets makes people pay special attention to such closure or its absence when they are categorizing 'b's and 'h's. Similar stories can be told about 'c' and 'o', about 'm' and 'n', and about 'e' and 's'. These pairs of letters are rarely confused by people because the *conceptual* differences are too great.

Now we turn to the obverse side of the coin, and consider the most common human errors not found in the list of the most syntactically-similar pairs. The errors include: 'u'↔'v', 'i'↔'l', 'i'↔'j', 'f'↔'t', 'v'↔'w', 'c'↔'r', and 'j'↔'s'. Consider the default roles for 'v' and 'u'. As the "point" of a 'v' (at the bottom) becomes rounded, the letter becomes more 'u'-like. Similar role-level analyses hold for other letters: a 'j' can become more 'i'-like as its descender gets straighter and shorter; an 'f' can become a 't' if the stem is bent differently; an 'i' can become more 'l'-like as the space between its dot and its stem gets smaller; a 'v' becomes a 'w' if its point gets bent upwards instead of down. Because the grid is so coarse, sometimes highly-eccentric, style-driven letters

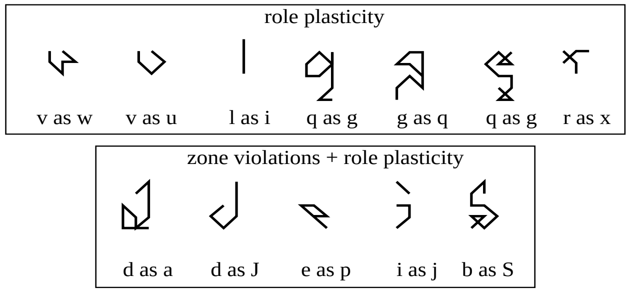


Figure 7.6: Examples of some commonly miscategorized tokens. These can be divided into two classes, as shown. Note how reasonable these errors are.

may well push a letter into territory where the kinds of role confusions noted above occur. Similar stories can be told for the remaining common errors not found in the syntactic clustering. Because roles are plastic, human letter perception is flexible and extremely reliable. The plasticity comes, however, at the price of occasional error-making. By closely examining the errors, we can come to understand the relationships between letter-concepts.

We argue that *all* of the most common human category errors shown in Table 7.5 can be understood in terms of role plasticity. Precisely because of the plasticity of roles, ambiguous or strange parts can be accommodated and seen as instances of roles. Low-level syntactic proximity is clearly not as powerful as an explanatory method (as anticipated in Prediction 2).

That errors are easily explained by the role hypothesis becomes even more obvious when specific errors, rather than category-level error rates, are considered. At the token level, all common human errors can be seen to be role-level errors. We look at this effect next.

Figure 7.6 shows a number of commonly misperceived tokens from PSYCH. The first thing to note is that all of the errors are fairly reasonable — in fact, it is not clear that some of them are "errors" at all.[4] People make an occasional dumb mistake,

[4] See footnote 1.

but most category misidentifications can be easily explained in terms of conceptual proximity. Generally, there are two types of problems causing miscategorization. The first type involves perceptual mistakes where nonstandard parts that serve as role-fillers cause miscategorization. In the example shown, the roles of a 'w' are filled quite nicely by what was supposed to be a 'v' — with the kink on the right side being seen as forming the bottom hump of a 'w'. Such errors are commonly caused by the coarseness of the grid, which often forces letterforms to the fringes of category space.

In some cases, the perceiver allows very poor parts to be seen as role-fillers in order to make sense of weird shapes, resulting in miscategorization. This usually occurs with difficult or otherwise nonstandard letterforms and is probably magnified under time pressure. In the case of 'l' seen as 'i', changing the default height of 'l' to fit the unusually short token is thrown out in favor of filling the dot role of 'i' by the top part of the higher-than-usual token (or perhaps perceiving the token as an uppercase 'I').

The second class of error includes ones in which subjects either made reference to categories outside the stated limits of the experiment (*e.g.,* capital letters) or violated the placement constraints of the grid. Both are types of error that the computer models discussed in the next chapter cannot make. An example of this is making a classification based on a resemblance to an uppercase letter (as in 'd' seen as 'J', above) or overriding the conventions of the grid itself (as in 'e' seen as a very tiny 'p' with no quanta in the "official" descender zone). We consider the "official" ascender and descender zones as the only proper place for ascenders and descenders, but not all of our subjects were like-minded.

7.3.3 Prediction 3

Every token elicited a certain number of responses from the group of subjects. While some tokens were always identified correctly (thus placed in only one category), others were occasionally misclassified but with only one wrong answer given instead of the correct answer (for a total of two categories). Still other tokens were categorized in several different ways. As would be expected, there is a strong correlation between the number of categories in which each token was found and its average reaction time ($r = 0.85, p < 0.01$). Fast response times are seen when there are few plausible category bins. As the number of plausible answers increases, response time becomes longer.

The number of categories implies a continuum that we might call "n-guity". A token that is always identified correctly has parts with a high "goodness-of-fit" to the roles of only one category. A token whose responses are divided between two categories could be considered ambiguous. As a token becomes stranger, the appeal of one or two categories will be lost in the crowd of poor part–role fillings, and the number of responses will grow quite large — as high as fourteen in our experiment.

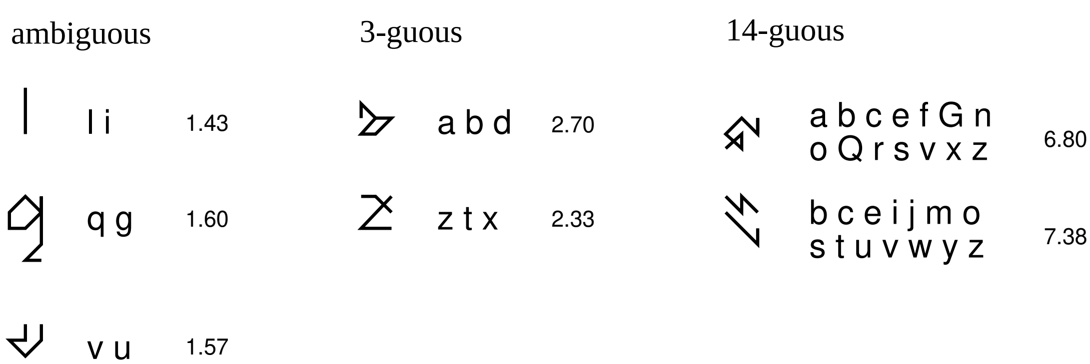


Figure 7.7: Ambiguous and strange letterforms (14-guous being the strangest of all). Beside each gridletter are the categories people guessed and the average response time.

Ambiguity and strangeness both cohere nicely with the role hypothesis, as was anticipated in Prediction 3. When a token is ambiguous, its parts can fill two or more sets of roles, all in a fairly reasonable fashion. This is illustrated by the first two columns of gridletters in Figure 7.7. A choice must be made between the possibilities. This choice takes longer as more competing categories are considered. When a token is strange, its parts do not really fit any sets of roles well (although clearly they will fit some letter-conceptualizations better than others). The problem is that too many sets of roles can be fit into poorly, so that role-filling is not much help, and finally, after an extended period of time, an informed guess of some sort is made. The two tokens in the "14-guous" column of Figure 7.7 are very strange tokens indeed, each having elicited fourteen possible categorizations.

The ambiguous/strange distinction is also evident in the fonts *Sabretooth* and *Close*. *Sabretooth* is a somewhat stranger font than *Close*. In other words, *Sabretooth*'s gridletters have parts that do not fill roles in a normal fashion, so the font as a whole has a higher average response time than *Close* does, even though the accuracy rates of the two fonts are very similar.

7.3.4 Double tokens and the question of determinism

While most models of letter recognition are deterministic and take into consideration only the stimulus and the ability of the subject, our experiments show that human letter recognition is not deterministic, in the sense that presenting the same stimulus twice to a subject does not necessarily evoke the same response.[5] Of the 544 tokens, there were eight pairs of tokens that happened to be identical.[6] These tokens therefore wound up being presented twice to each subject, although the trials were of course separated randomly. The reaction times of the first exposures and the second exposures correlated surprisingly poorly ($r = 0.235$, $p > 0.01$). Not only the reaction times but also the category responses differed over the two exposures. The average agreement

[5]This result has some relation to the research of Ann Dowker on nondeterminism in arithmetical estimation [Dowker, 1995] as well as to observations about nondeterminism in the game of Numble made by Daniel Defays [Hofstadter and FARG, 1995, Chapter 3].

[6]This came about unintentionally (though eight times), as a result of some given letterform being both a member of one or more of the fonts and at the same time a member of the NORMALS dataset. Obviously, we did not remove "duplicates" before the experiment was run.

in responses between two trials from the same subject on the same (double) token was 90.0% — virtually the same as the overall accuracy of categorization (89.4%) on the entire set of double tokens. That is, the chance of a subject correctly identifying any one of these tokens is equal to the chance that both categorizations of that token will be the same. There does appear to be a facilitating effect in repeating stimuli, as the average response time for second exposures (1.24s) is significantly lower than the average response time of first exposures (1.50s). (Of course, this facilitating effect may well be simply due to general practice on the gridletter-recognition task rather than to specifically reacting more quickly to an already-seen shape.)

While this finding does not indicate that a successful letter-recognition system *must* be stochastic, it does show that humans achieve very good performance while being heavily influenced by factors beyond those that are commonly assumed to completely determine the outcome of a trial. It may well be that human flexibility is an outcome of the nondeterministic nature of our cognitive processing — a theme recurring in much of the research at CRCC (see [Mitchell, 1993], [Hofstadter and FARG, 1995], and Section 3.2.9 of Chapter 3.)

7.4 A follow-up experiment

An additional experiment investigating the possible top-down effects that roles have on the parsing of letterforms into parts was also carried out. Ninety-two subjects were divided into two conditions — "shape" and "letter" (forty-six subjects to each condition). Letter-condition subjects received stimuli that were always gridletters. Shape-condition subjects received the same stimuli, except rotated through 90 degrees and mirror-image reflected. Each group was asked to choose between two alternative parsings of a stimulus, in a forced-choice test.

Of the two possible parsings, one was deliberately created with parts that filled

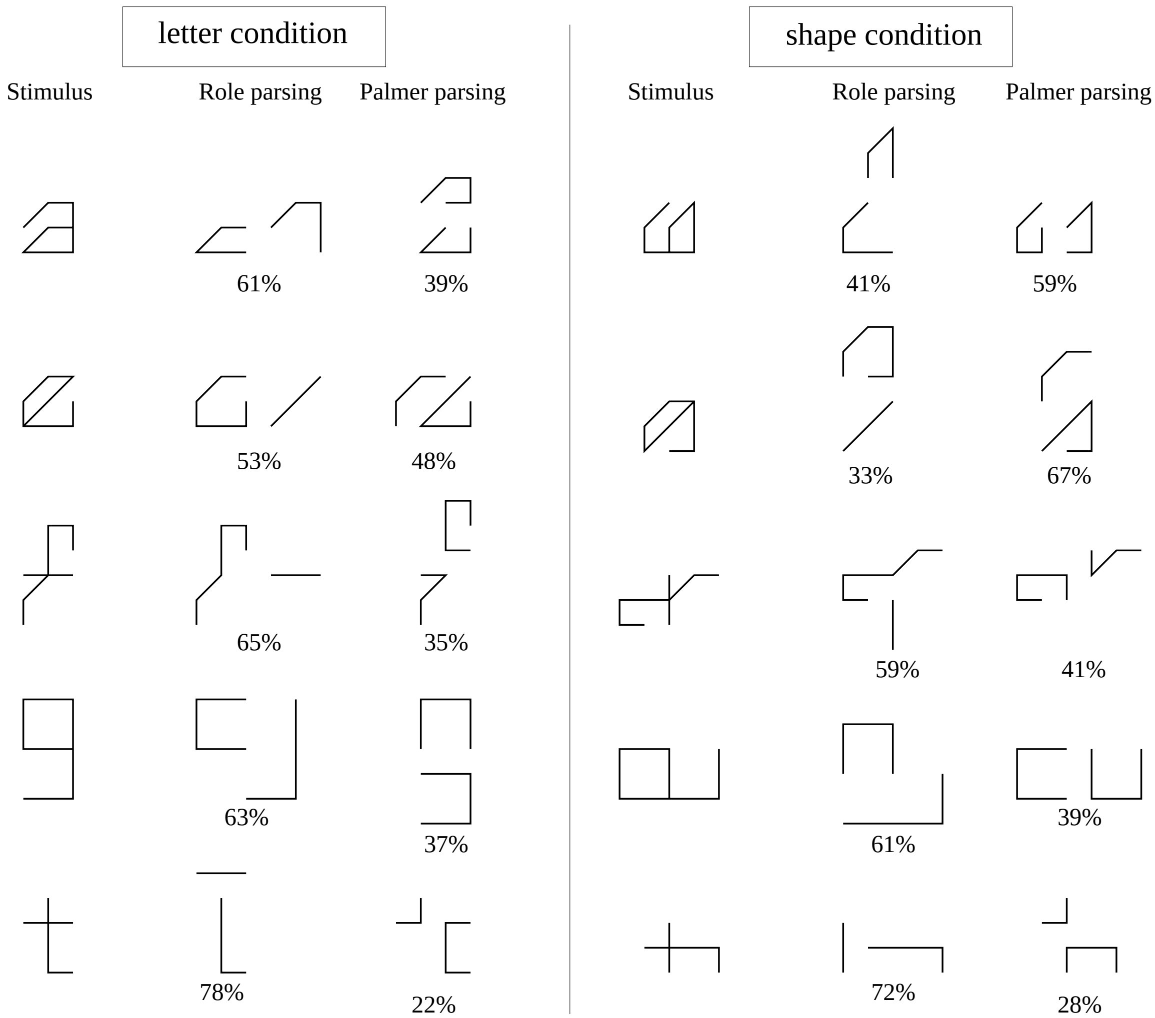


Figure 7.8: The five stimulus pairs from the part–role parsing experiment (letter condition on the left, shape condition on the right). The numbers shown below the stimuli are the percentage of subjects who chose the corresponding parsing as best. Note that the "same" parsings (rotated through 90 degrees and mirror reflected in the shape condition) result in different preferences, depending on the condition. Letter stimuli are shown on the same line as their related transformation-derived shape stimuli. The data show a clear and significant top-down effect from roles, as the role-based parsings are preferred significantly more often in the letter condition than they are in the shape condition (see the text).

the letter category's roles extremely well, while the other was deliberately created not to match roles at all. However, the poor role-filling parsings were made in such a way that the "goodness ratings" of their parts were much higher (yielding an average of 0.181) than the goodness ratings of the parts from the role-parsings (yielding an average of 0.094). Goodness ratings for parts were computed using Palmer's part-rating model (previously mentioned in Chapter 4) with the regression weights reported in [Palmer, 1977].[7] A goodness rating in Palmer's model is computed as a combination of properties of proximity, connectedness, orientation, length, and continuity — all purely gestalt features with no top-down influence from any learned set of target categories (*e.g.*, roman letters, Chinese characters, etc.).

Generally, results show that role-parsings are preferred over Palmer-parsings in the letter condition (64% to 36%). Role-parsings are also preferred in the shape condition, but by a much smaller margin (53% to 47%).

Figure 7.8 shows all five stimuli used in the second experiment. An ANOVA on the data shows that there is a significant difference in subject preference between the letter-condition parsing choice and the shape-condition parsing choice ($F(1,90) = 5.786$, $mse = 0.272$, $p < .05$ [$p = .0182$]). In other words, there is a significantly stronger top-down effect from roles in the letter condition than there is in the shape condition — meaning that role-parsings are chosen significantly more often in the letter condition than they are in the shape condition. This result runs counter to Palmer's claim that goodness ratings of parts should be the same in both conditions [Palmer, 1977], and (less importantly for our purposes) that the Palmer parsing should always be preferred. Note that Palmer's model holds that a part's goodness-rating does not change even if a part is rotated and/or reflected. Contrary to this view, our data

[7] One important caveat: Palmer claims that the regression weights need to be recomputed for each particular set of stimuli. We did not do this. Instead we used the reported weights. We feel that the plasticity of the weights seriously weakens Palmer's model, since reweighting the six high-level properties will almost always result in a fit to experimental data.

show that if the stimulus is a letter then it will be more strongly subject to top-down role effects than if it is not.

7.5 Comparing people and machines

Results of the first experiment reported above are consistent with the role hypothesis of letter recognition. Stronger evidence of the top-down effect of roles on letterform perception was presented in experiment two. In the next chapter we will compare the role model (and other models of letter recognition) with the human data gathered in the experiments. The analysis presented in the next chapter provides further evidence that human letter perception makes use of roles.

CHAPTER EIGHT

Human and Machine Performance

In this chapter we will compare the performance of our three models — the role model, Dumrec, and Netrec — to human performance on the same datasets. Table 8.1 reviews the makeup of the datasets we use. The PSYCH dataset, first introduced in Chapter 4, can be reasonably divided into two subsets, NORMALS and FONTS.

Letters within PSYCH range from fairly bland to highly stylized. We are interested in the ways in which our models perceive all kinds of letters — including ones that are eccentric. It is important to recall that even within the NORMALS subset there is a considerable range with respect to the style of tokens. That is, there are some tokens in NORMALS that are clearly more style-filled and eccentric than others. Because of its internal range of styles, the NORMALS dataset differs substantially from datasets usually used to probe recognition in computer models. Regardless of their collective name, some tokens in NORMALS tend to stray from the usual standard associated with vanilla bookfaces. This "stylistic range" is enormously extended and amplified by use of the FONTS dataset, whose tokens flirt with the edges of letter categories. Note that the same sort of story that applies to NORMALS with regard to their internal range also applies to FONTS, only this time in

NORMALS	388 tokens, all relatively strong letters ranging from very standard to somewhat stylized
FONTS	six complete gridfonts (156 tokens) ranging from somewhat stylized to completely eccentric: *Bowtie*, *Double Backslash*, *Flournoy Ranch*, *Hint Four*, *Close*, *Sabretooth*
TRAIN	130 very standard tokens from NORMALS, 5 from each category, used to train the Dumrec and Netrec models
NORMALS−TRAIN	NORMALS minus TRAIN (258 tokens)
PSYCH	NORMALS plus FONTS (544 tokens)
PSYCH−TRAIN	PSYCH minus TRAIN (414 tokens)

Table 8.1: Datasets used in the comparisons between people and programs. The NORMALS and FONTS datasets are displayed in Appendix A as Figures 9.2 and 9.3.

reverse. Some tokens in FONTS stray towards "normalcy". For these reasons, the PSYCH dataset considered as a whole provides a genuine panorama of tokens ranging from entirely standard and prototypical to completely nonstandard and eccentric, with all intermediate levels represented as well.

Some of the tokens in FONTS are so completely swamped by their stylistic aspects as to become very hard to "correctly classify" by humans (see Chapter 7). For this reason we must restate the caveat regarding "correctness" that we have mentioned earlier. In order to count as a "correct" response, a human subject or computer model must respond with that category in which a stimulus was placed by the experimenters (the category intended by the designer of the given gridletter). We use the designer's intended category in order to provide a fair comparison between humans and the three recognition programs discussed below.

This rigid and narrow notion of "correctness" is "fuzzified" by our consideration of the types of errors made by both humans and machines. We subscribe to the general cognitive-science stance that error-making sheds much light on internal aspects of cognitive processing. Thus we place a great deal of importance on the correlations between confusion matrices of models and of people. In what follows, we will discuss both category-level confusions and specific token errors made by each particular model, relating the errors to those made by humans.

8.1 A quick review of the models

A quick review of the three models introduced in previous chapters is in order. The *role model* is our main computer model of letter recognition and is the focus of our implementation of Letter Spirit to date. It is closely based on the role hypothesis presented in Chapters 1 and 4, which states that letter conceptualizations (or wholes) are best understood in terms of their constituent roles. The roles in a letter conceptualization are related to each other by a set of r-roles. Thus the role model takes a hierarchically- *and* relationally-structured approach to letter recognition. During the recognition process, explained in detail in Chapter 4, physical parts of letters are perceived (*i.e.*, coalesce) in a self-organizing way under the top-down influence of roles. Letter parts fill roles either well or poorly. The way in which roles are filled and the way in which such filled roles relate to each other are the principal determinants of categorization. The role model is based on Hofstadter's parallel terraced scan, in which perceptual possibilities are scouted out in parallel, but at different rates, by swarms of independent micro-agents that individually make stochastic microdecisions, but that at a collective level exhibit a robust, emergent intelligence [Hofstadter and FARG, 1995].

Because of the role model's nondeterministic behavior, it is important to process each token many times. For this reason, the numbers reported in this chapter are compiled from over 50,000 runs. For each token in FONTS, the model was run at least 35 times, for a total of 7,485 runs. For each token in NORMALS, the model was run at least 100 times, for a total of 42,831 runs.

Dumrec, discussed in Chapter 6, is a simple deterministic exemplar-based model of letter perception. Dumrec builds a property list for each of its training letters.

Dumrec's three flavors — the simple model, the no-tips model, and the full model — involve different amounts of abstraction in the creation of these property lists. A letter to be categorized is compared, based on its own property list, with each of the prototype property lists, using a weighted-feature approach. The letter is considered to belong to the category of the prototype that it matches most closely.

Netrec and *Netrec+* are simple three-layer backpropagation networks. Like Dumrec, these models require a training set. The network is trained hundreds of times on each of its training letters, gradually adjusting its weights in order to properly carve its input space into categories. Once trained, Netrec classifies a new token by feeding it through its connections. The result is a category classification on the output layer. Simple Netrec is trained using the 130 distinct tokens of TRAIN. Netrec+ is essentially the same three-layer network as simple Netrec, but trained on 5,481 distinct tokens (which collectively do a much better job at saturating the gridfont space). Together, Netrec and Netrec+ show how the final behavior of an architecturally-simple learning model trained using backpropagation is completely determined by its training set. Given a training set that finely samples the entire space probed by a testing set, the behavior of such a model is guaranteed to be quite good. With a sparser training set, behavior is severely degraded.

Netrec and Netrec+ are discussed in detail in Chapter 6. The numbers reported below for both Netrec models are an average of the performance of twenty different networks, each fully trained on the training set.[1]

8.1.1 Computer models and recognition

Implementation of the computer models was motivated in part by the potential for comparison with human subjects. Of course, there are important inherent differences

[1] Each of the twenty networks was initialized with a different set of random weights before training.

between the human and machine conditions, so that comparisons are meaningful only if these differences are taken into consideration. For instance, the Dumrec and Netrec recognition programs employ simple feedforward processing, so time of performance is always constant, and hence any information regarding timing is meaningless. Another important difference is that in the case of human perception, the pixel-level stimuli must first pass through the low-level portions of the visual system in order to arrive at the medium-level quantum information that we provided directly to our programs. Various parameters of stimulus presentation irrelevant to high-level cognition could affect human error rates. We attempted to tune the amount of noise in the stimuli (by regulating the level of visibility of the surrounding grid) so that human and computer error rates could be reasonably compared. The differences in relative proportions of errors made should, we reasoned, be due primarily to fundamental differences in representations at *higher* levels, though learning systems like Netrec complicate this story.[2]

8.2 Accuracy

Table 8.2 shows accuracy values for all models and humans on several data subsets. These values were computed by averaging the correct-response percentages of each letter category within each dataset (*i.e.*, averaging the numbers on the diagonal of each dataset's confusion matrix). They differ from the accuracy numbers reported in Chapters 6 and 7, since those numbers were computed by counting the number of tokens correctly categorized regardless of category and dividing by the total number of tokens. Although the two sets of numbers are clearly related, those reported here provide for a more accurate comparison.

[2]For a discussion of *how* learning models complicate the story, see especially Section 6.1 of Chapter 6.

Stimulus	Humans	Role Model	Dumrec models			Netrec	Netrec+
			S	NT	Full		
PSYCH−TRAIN	80.1	76.6	42.4	52.7	60.4	59.8	79.2
NORMALS−TRAIN	85.0	93.8	49.0	63.0	71.8	71.0	89.7
FONTS	65.4	51.4	33.8	38.5	43.7	41.0	63.0
Bowtie	51.5	39.3	32.8	43.3	45.6	38.5	64.4
Double Backslash	67.7	41.9	14.1	15.9	19.8	19.2	42.2
Flournoy Ranch	58.0	38.3	30.5	39.5	43.3	34.6	59.9
Hint Four	83.2	66.4	60.8	55.5	65.1	73.1	84.5
Close	73.8	65.5	23.8	28.5	35.8	26.9	61.3
Sabretooth	77.6	56.0	35.9	48.1	52.5	53.8	64.2

Table 8.2: The people vs. the programs. Recognition performance of the models and people on several data subsets. All of the numbers recorded here are correctness percentages computed as averages of confusion-matrix diagonals.

Note that the large dataset reported here is PSYCH−TRAIN, not PSYCH. This is because two of our models, Dumrec and Netrec, were trained on TRAIN, and testing the models on the training letters is of little value.[3] In particular, Dumrec immediately recognizes its prototype tokens and categorizes them correctly every time. Netrec's performance on the training set is also very good, although not perfect. Since we are interested primarily in the *generalization* performance of the two simple models (which sheds more light on the representations they use), we chose not to count performance on TRAIN. As in the human case, we tested the models only on letters they had never seen before.

Humans outperform all models on the PSYCH−TRAIN dataset considered as a whole, with Netrec+ and the role model both showing similar accuracy values. (The differences between the accuracy percentage of humans and that of the role model amounts to a 14-token difference in correct categorization over 414 tokens.) The slight difference in overall performance is due to the humans' superior performance

[3]Netrec+ was trained using the 130 tokens of TRAIN plus an additional 5,351 tokens taken from a large variety of gridfonts.

on the FONTS dataset (which also accounts for Netrec+'s advantage over the role model). However, the role model beats both the humans and Netrec+ on the NORMALS dataset. Although all of the models *and* humans experience a fair amount of degradation in performance from NORMALS to FONTS, the role model suffers the greatest drop.

The role model consistently outperforms its simpler rivals, with the exception of Netrec+, when considered at the level of FONTS and NORMALS. However, its performance on individual fonts is only significantly better than Dumrec and simple Netrec for two fonts: *Double Backslash* and *Close*. In terms of accuracy, its performance on the remaining four fonts is not significantly different from that of Dumrec or Netrec. Netrec+'s performance on FONTS is superior to the role model's, with the 11.6% difference stemming from Netrec+'s better performance on 18 individual tokens. As was discussed in detail in Chapter 6, Netrec+'s ability to categorize strange gridletters so well is a direct result of its having been trained on very similar strange gridletters. This fact also accounts for its slightly degraded performance on NORMALS.

Performance of the degraded versions of Dumrec follows our expectations, with the simpler versions' performance generally worse than the more abstract models. Degradation can be seen in all datasets, the only surprise being Dumrec-S's performance on *Hint Four*. The simple Netrec networks fall somewhere between the Dumrec-S and Dumrec models, with nothing exceptional about their behavior (besides an excellent performance on the font *Hint Four*).

Figure 8.2 shows how the models and humans fare on each separate letter category. The graphs summarize performance on the dataset PSYCH−TRAIN. This gives some idea about which categories are found most difficult by humans and by each of the programs.

The fact that font-level accuracy rates for all models diverge in distinct ways from

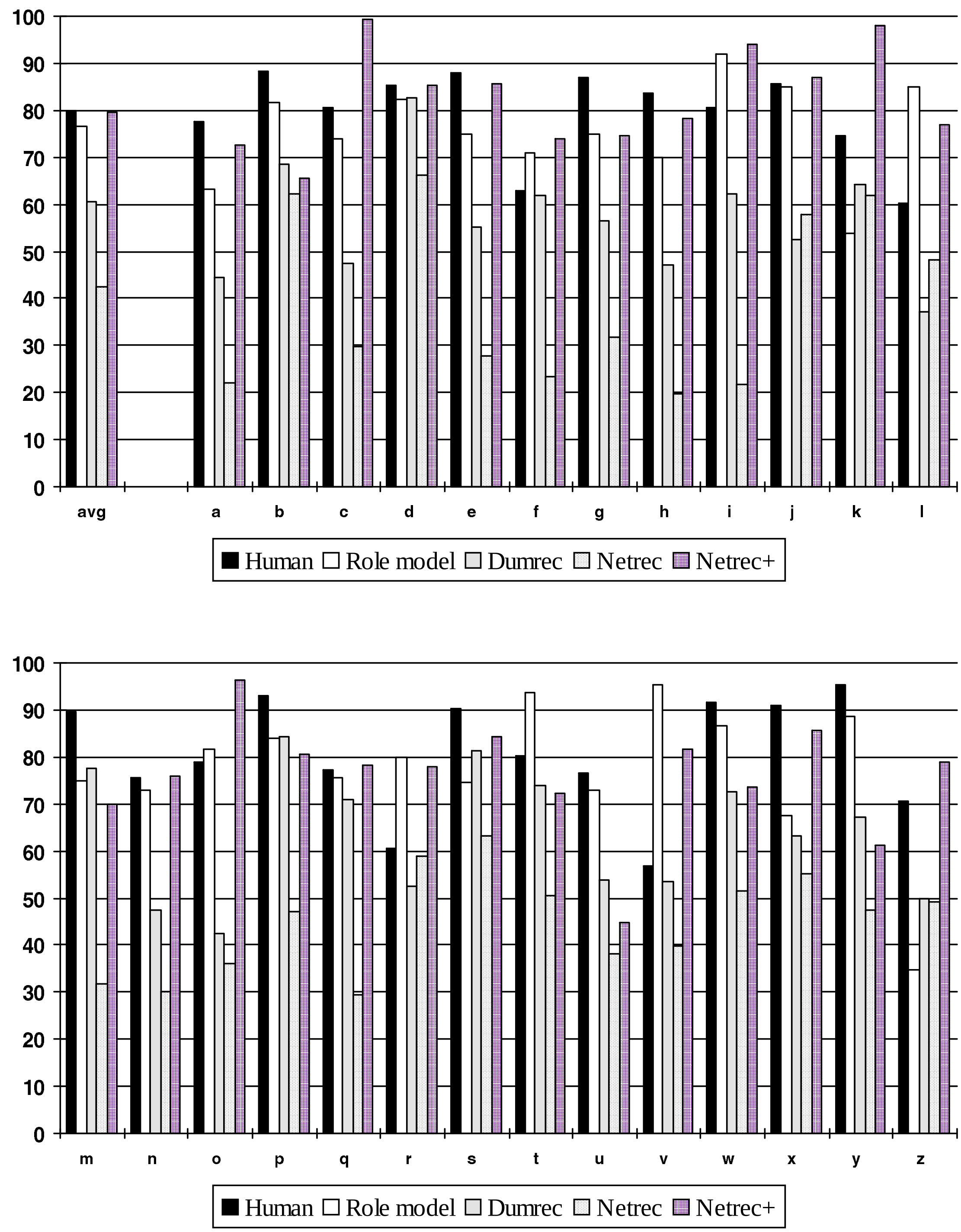


Figure 8.1: These two graphs show how humans and all of the models compare in each letter category. The accuracy rates recorded are for the PSYCH−TRAIN dataset.

the human data leads us to suspect that a different sort of error-making is happening for each font. Correlations of the human confusion matrices for each font against all of the other fonts show that this is (at least weakly) the case. Correlations range from a low of 0.71 between *Flournoy Ranch* and *Bowtie* to a high of 0.87 between *Close* and *Hint Four*.[4] These correlations mean that human performances on the different fonts share from 50% to 75% of their variance (the r^2 values).

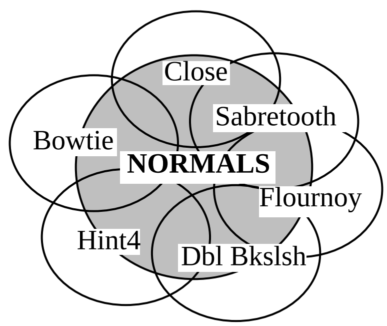


Figure 8.2: A schematic view of how the error sets of individual fonts may overlap with the error set of NORMALS. The overlap between the individual fonts with other fonts does not necessarily match actual data.

Considered individually, the fonts seem to result in different kinds of errors. How, then, do the NORMAL and FONT errors compare? Is it valid to create a confusion matrix over the entire PSYCH dataset? To investigate this, we correlated the human category-level confusion matrix on FONTS with a similar matrix for NORMALS. The correlation (over 676 points) was 0.97. This suggests that together, the fonts as a group result in a *very* similar set of errors to the NORMALS. Figure 8.2 shows a schematic diagram of how error sets for each of the fonts and the NORMALS overlap. This overlap was further confirmed by correlating each individual font matrix with the

[4]All p values reported in this chapter are less than 0.0001, unless otherwise stated.

NORMALS matrix. Each font correlated more highly with the NORMALS than it did with each of the other individual fonts. This time, correlations ranged from a low of 0.77 (for *Bowtie*) to a high of 0.94 (for *Hint Four*). Thus we can conclude that a PSYCH-level confusion matrix, as reported in Table 7.2, provides reasonable data about letter category confusion *in general*.

The fact that errors appear to be very similar in both the FONTS and NORMALS conditions bodes well for the role hypothesis. The role hypothesis predicts that all category errors are based on poor role-fillings and/or poor inter-role relations. Thus there should be no great difference between errors for NORMALS and FONTS, since the same roles apply in both conditions.

Considered in terms of accuracy alone, the role model does not seem to outperform Dumrec in a very dramatic way (especially with regard to the FONTS), and is slightly worse than Netrec+. The problem is not in the role model, however, but in the measure used thus far to compare its performance to the other models. Simply put, accuracy values provide a view of performance at too low a resolution. They leave out all sorts of information, including: what inter-category confusion errors the models make, which exact tokens are missed, and how all these pieces of information relate to the human data.[5] Later in this chapter we will delve more deeply into our data to answer some of these questions, but first we turn briefly to the notion of "guessing" in the role model, and its effect on the experiments reported in this chapter.

8.2.1 Investigating guessing in the role model

As we brought up in Chapter 4, the role model is sometimes unable to categorize letterforms that it is presented with. Usually this happens when letters are complicated

[5]Comparisons in terms of accuracy also gloss over the fact that the role model is *parsing* a letterform into constituent parts that can be used in further processing, but that is another story altogether.

(like some of those in the FONTS dataset), although occasionally the role model will become "stuck" on an easy letter and not be able to parse it. Just to provide some idea of how often this happens, here are some numbers. In 42,829 runs on tokens in NORMALS, the role model hit a processing wall 663 times (or 1.5% of the time). On the more complicated FONTS dataset, 7,485 runs resulted in 1,168 non-parsings (or 15.6% of runs).

The question of what to do when a processing wall of this sort is hit is a complicated one. The simplest possible solution is to guess a category based on some non–role-related criteria. One version of this sort of guessing, based on "blurred prototypes", was introduced in Chapter 4. Although this sort of guess will result in a categorization (wrong though it may be), no parsing will be available for further processing.[6] As a very simple experiment, we implemented a simple guessing function based on the blurred prototypes of the PSYCH dataset.[7] Thus we have a template-matching guesser that computes the city-block distance from a token to each of the category templates and picks the closest one.

We kept track of any guessing that the role model resorted to. The question is how to handle these data when one is correlating the role model with human data and with other models of recognition. We explore three possible solutions. Table 8.3 shows the results. The first solution is to drop the guess idea altogether and simply use a confusion matrix for the role model having rows that do not necessarily sum to 1. This is the [noguess] condition. (Note that this is mathematically equivalent to adding an extra "guess" category to all of the other models' correlation matrices. This new column would be filled by zeros in all matrices other than that of the role model.) The second solution is to record the guess in the guessed category as if it had

[6] Clamping roles according to a guess of this sort and trying again is one possible solution to this problem. See Chapter 4.

[7] See Chapter 7 for a discussion of blurred prototypes and categorization.

PSYCH [14,144 points]				
Humans	1.0000			
Role Model [noguess]	0.9136	1.0000		
Role Model [guess]	0.9023	0.9591	1.0000	
Role Model [split]	0.9150	0.9985	0.9605	1.0000
	Humans	RM-noguess	RM-guess	RM-split
NORMALS [10,088 points]				
Humans	1.0000			
Role Model [noguess]	0.9621	1.0000		
Role Model [guess]	0.9644	0.9932	1.0000	
Role Model [split]	0.9623	0.9997	0.9935	1.0000
	Humans	RM-noguess	RM-guess	RM-split
FONTS [4,056 points]				
Humans	1.0000			
Role Model [noguess]	0.7274	1.0000		
Role Model [guess]	0.7014	0.8513	1.0000	
Role Model [split]	0.7309	0.9952	0.8554	1.0000
	Humans	RM-noguess	RM-guess	RM-split

Table 8.3: Correlations of the token-level confusion matrices of three versions of the role model and humans (including all error-making information) on three datasets. The three approaches to guessing in the role model lead to very similar correlations. Ultimately, we decided to go with the [noguess] condition for comparison purposes.

been generated normally by the role model. This is the [guess] condition. The third alternative is to split the entire guess value (the number that would have been entered in the "extra guess column" above) over the twenty-six categories evenly — the [split] condition.

As is apparent in the correlation table, none of these alternatives seems to make a very large difference. Even in the FONTS case, where the largest number of guesses must be made, guessing does not affect correlations that greatly. For example, the correlation between the [split] condition and the [noguess] condition is a very high 0.9952. For this reason, we decided to go with the first alternative, essentially throwing out all guesses. The accuracy values reported in Section 8.2 reflect this decision. All correlations discussed below are for the [noguess] version of the role model as well.

8.3 Considering errors

8.3.1 Correlations of confusion matrices

Table 8.4 shows the correlation matrices for all models and humans on PSYCH−TRAIN, NORMALS−TRAIN, and FONTS, correlated over token-level confusion matrices.[8] Category-level correlations, in which all token data are collapsed into their respective letter categories, are higher in all cases, but are less informative as they have many fewer data points. Thus we choose to report the token-level correlations.

Since error-making tends to highlight the type of processing that a cognitive system is doing, it is important to take errors into account during correlation (hence a correlation over *entire* matrices, instead of just over diagonals, which represent only correct categorizations). This is one of the reasons we developed the continuous-valued version of Dumrec (see Chapter 6). Use of the "probability scoring" results in a far richer error matrix for the Dumrec models.

The difference in correlation between models and humans on NORMALS vs. FONTS is striking. All of the models correlate much more strongly with people on the simple NORMALS−TRAIN dataset than on FONTS (although the role model's correlation with FONTS surpasses Dumrec-S's correlation on NORMALS−TRAIN). The role model has the highest correlations with humans of any model over all datasets (0.95 on NORMALS, 0.73 on FONTS, and 0.89 overall). Other models correlate with humans in the same rank order no matter what dataset is considered. The order is: Netrec+ with the highest correlation, then Dumrec, Dumrec-NT, Netrec, and

[8] Token-level confusion matrices differ from category-level confusion matrices in that they have one row for each token in the dataset. In contrast, category-level matrices have only 26 rows into which the token data are averaged, no matter how many tokens there are.

PSYCH–TRAIN [10,764 points]							
Humans	1.000						
Dumrec	.7791	1.000					
Dumrec-NT	.7029	.8500	1.000				
Dumrec-S	.6079	.7036	.7830	1.000			
Netrec	.6595	.6800	.7066	.7103	1.000		
Netrec+	.8401	.7316	.6901	.6042	.6521	1.000	
Role Model	.8872	.7544	.6766	.5748	.6442	.8144	1.000
	Humans	Dumrec	Dumrec-NT	Dumrec-S	Netrec	Netrec+	Role Model
NORMALS–TRAIN [6,708 points]							
Humans	1.000						
Dumrec	.8689	1.000					
Dumrec-NT	.7940	.9069	1.000				
Dumrec-S	.6682	.7684	.8312	1.000			
Netrec	.7349	.7653	.7774	.7588	1.000		
Netrec+	.9117	.8427	.7899	.6773	.7480	1.000	
Role Model	.9511	.8551	.7853	.6521	.7321	.9144	1.000
	Humans	Dumrec	Dumrec-NT	Dumrec-S	Netrec	Netrec+	Role Model
FONTS [4,056 points]							
Humans	1.000						
Dumrec	.5786	1.000					
Dumrec-NT	.5132	.7374	1.000				
Dumrec-S	.4887	.5803	.6953	1.000			
Netrec	.4717	.4822	.5565	.6161	1.000		
Netrec+	.6845	.5081	.4954	.4667	.4334	1.000	
Role Model	.7274	.5184	.4397	.4153	.4082	.5859	1.000
	Humans	Dumrec	Dumrec-NT	Dumrec-S	Netrec	Netrec+	Role Model

Table 8.4: Correlations of the token-level confusion matrices of each model (including all error-making information) on three datasets. The role model is clearly superior.

finally Dumrec-S.

The Netrec+ correlations are higher than those of the Dumrec and simple Netrec models, always falling between the full Dumrec model and the role model. The extreme difference in correlations between the simple Netrec model and the Netrec+ model serves to emphasize the critical importance of the nature of the training set selected for a statistical learning model such as the very simple three-layer network we have devised. It is interesting to note the extent to which a model such as Netrec+ can explain human data. Considered in terms of error-making, Netrec+'s performance is surprisingly similar to that of humans, even though processing is different. This is most likely due to the fact that higher-level categories such as roles are strongly tied to the lower level of quanta. That is, roles affect parts, parts are formed of quanta, and thus even errors at the level of roles have a clear relationship to quanta-level similarity. This makes teasing quanta-level processing apart from role-level processing a hard problem — one reason why it is important to include the weaker Netrec model in our discussion.

Netrec and Netrec+ are trained using the very same algorithm. This algorithm forms associations between quanta and categories by backpropagation of error. Given a very rich training set, better associations form. But the *kind* of processing going on in Netrec+ is *exactly the same* as the kind of processing done by Netrec.[9]

The differences in correlation values between Netrec+ and the role model over each dataset are all significant according to Fischer's r-to-Z transformation ($p < .001$). The fact that these correlations are made over very large datasets (involving thousands of points each) means that even small differences in r-values — though they may seem to be insignificant on the face of it — are important.

[9] Incidentally, this is not an argument against all connectionist models *per se* — just against simple three-layer backpropagation networks. More sophisticated connectionist models that process in accord with the role hypothesis are certainly conceivable, though they would push the limits of current connectionist systems.

Of the three main models, the role model's r-values degrade the least between NORMALS and FONTS (0.2237). This suggests that the role model captures human error-making in a very general way. Still, the correlations do degrade. Once again we have shown that models that do reasonably well on conservative characters will not necessarily work well at all on eccentric characters. This is especially apparent in the Dumrec model, which exhibits the biggest degradation (0.2903). There is an important lesson to be learned here. Most models in the literature to date have been tested on only one or two unadventurous typefaces, and commercially available OCR programs work only on the most normal of typefaces. If a computer program is to be taken seriously as a cognitive model of human letter perception, it must also account for recognition data on more spirited typefaces. There is a brittleness often found in standard artificial-intelligence programs that causes them to rapidly break down when exposed to input that is just slightly richer than usual [Holland, 1986]. Although our NORMALS dataset is itself very rich and covers much stylistic ground (as it features tokens with many different styles), the FONTS dataset provides a truly critical test for each of the models.

Also apparent in the FONTS correlation table is the fact that Netrec+ and the role model handle FONTS very differently from each other. The correlation between the two models is only .5859. Netrec+ apparently makes very different errors from those made by the role model — errors that do not correlate well with human errors.

The three Dumrec models have correlation values that reflect their levels of abstraction. As more abstract data are allowed in processing, the models improve. The ranking of the Dumrec models remains in place even on the FONTS dataset. In any case, it is clear that the role model is superior to Dumrec in all respects.

Also of note is the fact that the correlations of the simple Netrec networks with human data place those networks clearly ahead of Dumrec-S. This implies that such networks are doing something more than simple quanta-level processing, but that they are able to find and exploit some of the more abstract features used in Dumrec (such as tips) only if given a rich enough training set. Both of these conclusions are further confirmed by the analysis given in Section 8.4.

8.3.2 Principal-component analysis

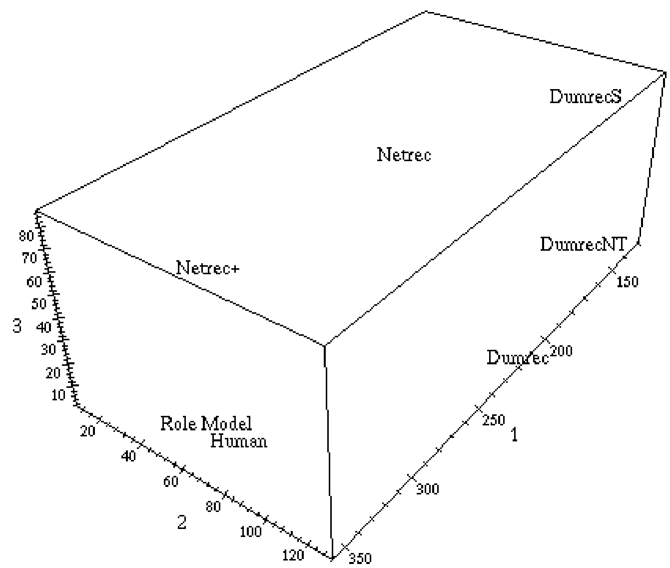


Figure 8.3: All models compared on the first three principal components of the 676-dimensional space made by turning the 26-by-26 category confusion matrix from PSYCH−TRAIN into a vector for each model.

Principal-component analysis is another tool that allows us to compare the performances of programs and of people. By considering the performance of each program (and the composite human performance) to be a point in the 676-dimensional space defined by the 676 entries of a generic 26×26 error matrix, we can locate the principal dimensions that separate the programs from one another and from people. Points for the programs and for people are plotted according to their values along the first three principal components in Figure 8.3. The most significant dimension seems to measure something similar to overall percent correct. The second dimension distinguishes between the Netrec group, the Dumrec group, and people together with the role model, suggesting perhaps fundamental differences and likenesses. The third dimension serves to further distinguish Netrec+ from the role model and humans. Higher dimensions were not easily interpretable. As can be seen, the role model outperforms all other models on this scale as well. Interestingly, although not surprisingly, as Dumrec becomes more abstract, it moves towards people along component 1.

8.3.3 Error-making in the role model

Since the role model is not trained on the TRAIN subset like Dumrec and Netrec are, we can correlate its performance with human performance on the entire PSYCH dataset. Over the 14,144 points of PSYCH, the role model's correlation is 0.9136. Over the entire NORMALS (10,088 points) subset, the correlation is 0.9621. These correlations are slightly higher than those shown in Table 8.4 mostly because the tokens of TRAIN are easily recognized by both humans and the role model. One other correlation value is of interest. Calculated at the category level instead of the token level, the role model's r-value on the PSYCH dataset is an extremely high 0.9821. This suggests that the role model's overall performance considered at the category level is extremely human-like.

In order to investigate to what extent error asymmetry (*e.g.*, mistaking 'a's for 'b's versus mistaking 'b's for 'a's) affected correlations, we transposed the human matrix across its diagonal and correlated the resulting matrix against the role model. The human matrix correlated with its own flipped version with an r-value of 0.9932, suggesting that error asymmetry plays a very minor role in the human data, accounting for only 1.4% of the variance at most. Furthermore, the role model's correlations with *both* human matrices are essentially equivalent, which suggests that the role model's high correlations with human data do not stem from its ability to model asymmetrical confusions. For this reason, we have chosen to concentrate on bidirectional category errors in our discussion of category-level error-making.

Error	Role Model	Humans	Human Rank
u$\leftrightarrow$v	6.70	15.30	1
g$\leftrightarrow$q	6.30	12.10	2
f$\leftrightarrow$t	5.59	5.85	5
r$\leftrightarrow$s	3.71	0.90	-
a$\leftrightarrow$o	3.50	2.25	-
b$\leftrightarrow$h	3.50	1.50	-
c$\leftrightarrow$s	3.35	0.10	-
v$\leftrightarrow$w	3.30	4.35	6
c$\leftrightarrow$z	2.85	1.50	-
n$\leftrightarrow$r	2.80	4.05	8
Human error sum		47.9	

Table 8.5: The top ten category errors made by the role model. The bidirectional error percentages are computed from the confusion matrices shown in Table 5.2 (role model) and Table 7.2 (humans). Human error rates (also percentages) are listed for comparison purposes. Also reported is the rank of any error found among the top ten human errors. Considering Tables 8.5–8.8 together, the human error sum provides for a simple comparison between Dumrec, Netrec, and the role model.

Table 8.5 lists the top ten bidirectional category errors (*i.e.,* the percentage of the time that tokens in either category are misclassified into the other) made by the role model. Similar Tables for Dumrec and the Netrecs are provided below. By every measure, error-making by the role model is the most human-like.

The role model's top ten errors include five of the top ten human errors ('u'$\leftrightarrow$'v', 'g'$\leftrightarrow$'q', 'f'$\leftrightarrow$'t', 'v'$\leftrightarrow$'w', and 'n'$\leftrightarrow$'r'), with the top two errors matching in exact order. By contrast, Dumrec and Netrec+ match four of the top ten human errors apiece, and Netrec matches only three (see Tables 8.6 , 8.7, and 8.8 below). Another simple comparison involves considering the sum of the human error rates corresponding to each model's top-ten errors. The model whose errors correspond more strongly to those of humans will have a higher sum. As can be seen by comparing Tables 8.5–8.8, this is the case for the role model.

Of the remaining top ten errors made by the role model, two are very poor ('r'↔'s' and 'c'↔'s') and do not have strong human counterparts (although humans do sometimes make these errors). Together with 'c'↔'z', these errors seem to be due to weaknesses in the role model by which inter-role similarity is incorrectly specified in terms of norms. This is not a major problem (as is apparent from the very high correlations between the role model and the human data). The 'a'↔'o' and 'b'↔'h' errors are quanta-level in nature and are the kind of error predicted by the blurred-prototype template model discussed in Chapter 7.

Considered in terms of category errors, the role model once again beats out Netrec+, Dumrec, and Netrec in terms of explanatory power. These results further substantiate the correlations of Table 8.4. All together, our results suggest that roles play a critical role (sorry) in human letter perception and provide a strong predictive ability in terms of error-making.

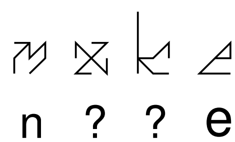


Figure 8.4: A complete listing of errors made by the role model that were never made by people. The role model's most frequent categorization is shown below each of the tokens. (Though the 'e' is most frequently categorized correctly, it is also mis-categorized as an 'r' too often. See the text.)

Figure 8.4 shows the complete set of tokens that the role model sometimes misclassifies but that people always classify correctly. There are only four such errors. The role model makes the 'm'→'n' error 100% of the time. This happens because the letterform is parsed into two shapes broken at the three-quanta intersection point, one of which activates *left halfpost*, the other activating *right buttress*, causing the model to see the shape as an 'n'. Although people in our experiment never classified this letterform as an 'n', they were probably subliminally pushed toward the answer 'm' because of the letterform's resemblance to capital 'M'. All told, this particular role-model error is a quite reasonable one. The 'x' error occurs because the figure can be parsed into coherent roles only 2.5% of the time. The remainder of the time the letterform is not classified. Considered in terms of the forward-slash/backward-slash conception of 'x', every one of the four tips is in an incorrect place (requiring a great deal of stretching right off the bat); moreover, the two parts, even when formed, are overly jagged and angled. The role model is not able to bend its roles in a flexible enough manner to handle this particular shape. The 'k' is actually correctly identified 26% of the time, but on 40% of runs no categorization is reached, on 25% of runs it is incorrectly categorized as an 'h', and 14% of the time it is seen as a 'b'. The 'k's "droopy" arms cause fairly severe problems (especially since one of them lies along the baseline). The 'e' is properly classified 40% of the time. However, it is also misclassified as an 'r' 39% of the time, and not classified at all on 21% of runs. None of these errors is too severe, but together, they do point out some weaknesses in the role model.[10]

8.3.4 Error-making by Dumrec

Table 8.6 shows the ten most common bidirectional category errors made by Dumrec. Dumrec's top errors have more in common with human errors than do simple Netrec's (shown in Table 8.7), but less in common with human errors than the role model's and Netrec+'s. This ranking holds across the human error sums for each model as well.

Four of Dumrec's top errors ('f'↔'t', 'g'↔'q', 'h'↔'k', and 'v'↔'w') can be found among the top ten human errors (see Table 7.5). Two of the remaining errors ('n'↔'m' and 'u'↔'w') are predicted by syntactic proximity as defined by the blurred-prototype

[10]These 4 errors, along with some others, are all further discussed in Section sc-hard-letters.

Error	Dumrec	Humans	Rank
m$\leftrightarrow$n	21.90	2.95	-
u$\leftrightarrow$w	19.05	1.10	-
v$\leftrightarrow$w	17.40	4.35	6
e$\leftrightarrow$z	15.05	1.50	-
a$\leftrightarrow$z	14.65	0.90	-
f$\leftrightarrow$t	13.10	5.85	5
h$\leftrightarrow$k	12.70	4.35	7
c$\leftrightarrow$e	12.25	1.35	-
g$\leftrightarrow$q	11.60	12.1	2
g$\leftrightarrow$y	11.45	0.45	-
Human error sum		34.9	

Table 8.6: The top ten bidirectional category errors (by percentage) made by Dumrec. Human error rates (also percentages) are listed for comparison purposes, as are the human error rankings (if within the top ten). Considering Tables 8.5–8.8 together, the human error sum provides for a simple comparison between Dumrec, Netrec, and the role model.

model (see Table 7.4). The 'c'$\leftrightarrow$'e' and 'e'$\leftrightarrow$'z' errors, however, are not. The errors 'a'$\leftrightarrow$'z' and 'g'$\leftrightarrow$'y' have very low human counterparts, making them especially important as negative evidence against Dumrec's mechanisms. As expected, Dumrec's errors seem to be based on low-level processing to a far larger extent than the role model's. This is of course a result of its reliance on prototype matching and its weak capacity of abstraction.

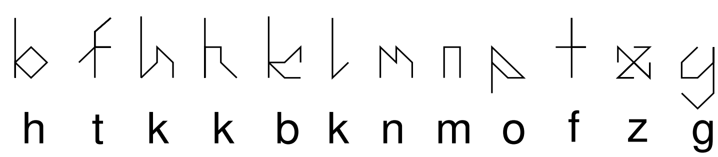


Figure 8.5: A complete listing of Dumrec errors that were never made by people. Dumrec's incorrect categorization is shown below each of the tokens.

Figure 8.5 shows all of the tokens that Dumrec misclassified but that people always classified correctly. There are 12 such errors, three times as many as the role model makes. Although three of the "token errors" are predicted by the naïve blurred-prototype model (Table 7.4), these errors seem generally more reasonable than the simple-Netrec errors (discussed in the next section), but less reasonable than the role model's errors. It is interesting to note that three of these errors are also made by Netrec+, and two (different ones) by the role model. It is certainly the case that Dumrec makes *some* dumb errors based on quantum-level processing, but it makes fewer of them than simple Netrec does. The five worst errors are 'k'→'b', 'l'→'k', 'n'→'m', 'p'→'o', and 'x'→'z'. The first two errors demonstrate weaknesses in the Dumrec code: the 'k'→'b' error occurs because Dumrec's closure-rating system is not sophisticated enough (in this case, the 'b' that won barely beat out a 'k'), and the 'l'→'k' error happens because all of the 'l' prototypes happen to have their ascender in the *middle* of the grid, whereas the ascender of this particular 'l' was placed, without thinking, against the *left edge* of the grid. Since Dumrec is very sensitive to placement, it saw this letter as being very distant from all 'l's, and much closer to letters like 'b', 'k', and 'h'. The role model has no such bias. The remaining three of the five worst errors are caused by the weakness of the "tips" calculations: the skinny 'n' matches 'm' prototypes' tips better than it does 'n' prototypes; the 'x's tips are all too tricky for Dumrec (since they are not in or near a standard location); and the short stem on the 'p' results in a tip in an unfamiliar location, which provides no cues as to the shape's 'p'-ness. In general, the abstract properties that Dumrec makes use of seem to help. However, their level of abstraction stops short of what is needed. The brittleness of the Dumrec program is clearly brought out by these errors.

8.3.5 Error-making by Netrec and Netrec+

Table 8.7 shows the top ten bidirectional category errors made by Netrec. All but three of the errors ('f'↔'t', 'g'↔'q', and 'v'↔'w') are rarely made by humans. The 'c'↔'o', 'e'↔'s', 'o'↔'u', and 'a'↔'s' errors all occur very rarely as human errors and are thus especially serious. Four of the top ten errors ('c'↔'o', 'e'↔'s', 'u'↔'w', and 'g'↔'q') are predicted by the closest blurred-prototype categories shown in Table 7.4.

Error	Netrec	Humans	Rank
c↔o	13.65	0.55	-
e↔s	11.55	0.20	-
s↔z	10.60	1.25	-
v↔w	10.20	4.35	6
f↔t	9.00	5.85	5
u↔w	8.60	1.10	-
o↔u	8.60	0.60	-
a↔s	8.45	0.25	-
g↔q	8.40	12.1	2
b↔l	8.35	1.80	-
Human error sum		28.05	

Table 8.7: The top ten bidirectional category errors (by percentage) made by Netrec. Human error rates and rankings are listed for comparison purposes. Considering Tables 8.5–8.8 together, the human error sum provides for a simple comparison between Dumrec, Netrec and the Role Model.

Error	Netrec+	Humans	Rank
g↔q	23.66	12.1	2
u↔v	15.18	15.3	1
v↔w	12.74	4.35	6
c↔s	11.49	0.10	-
b↔k	8.99	0.80	-
m↔n	8.82	2.95	-
a↔o	7.76	2.25	-
o↔s	6.65	0.30	-
f↔t	6.39	5.85	5
a↔z	6.27	0.90	-
Human error sum		44.9	

Table 8.8: The top ten bidirectional category errors (by percentage) made by Netrec+. Human error rates (also percentages) are listed for comparison purposes as are the human error rankings (if within the top ten).

This implies that to some extent, the simple nets rely more heavily on low-level quanta-level features than do Dumrec and the role model. The 's'↔'z' error fits the syntactic-feature theory as well — although it shows that Netrec is doing more than simple overlap of quanta — since gridfont 's's and 'z's share similar features at the x-height and the baseline. The b↔l error probably occurs because both 'b' and 'l' extend into the ascender zone and some of the 'l's have diagonal serifs in the central zone.

Netrec+'s most common errors are shown in Table 8.8. They are very different from the simple-Netrec errors, the two models having only three errors in common. Four of Netrec+'s top ten errors are included in the top ten human errors ('g'↔'q', 'u'↔'v', 'v'↔'w', and 'f'↔'t'). However, this good behavior is offset by four especially damaging errors that humans only rarely make: 'c'↔'s', 'b'↔'k', 'o'↔'s', and 'a'↔'z'. Three of the errors — 'g'↔'q', 'm'↔'n', and 'a'↔'o' — are predicted by the blurred-prototype template model of Chapter 7. The two remaining errors ('b'↔'k' and 'o'↔'s') are harder to explain, but seem to be based on quanta-level similarity and poor handling of closure. In general, Netrec+ blends very-predictive, human-like errors with poorly-predictive, non-human-like errors in a strange way.

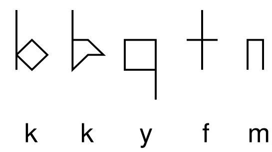


Figure 8.6: A complete listing of Netrec+ errors that were never made by people. Netrec+'s main categorization is shown below each of the tokens.

Figure 8.6 shows all of the tokens that Netrec+ misclassifies but that people never miss. There are five such errors — very few in comparison to simple-Netrec and Dumrec. Three of these errors are made by Dumrec as well, and are the same three made by the simple-Netrec model. It appears that most of Netrec+'s token errors are a consequence of low-level feature mistakes. Closure, for example, though very salient to people, seems to be less so to Netrec+. This allows Netrec+ to make the b↔k errors and the q↔y error. The remaining errors show a peculiar type of brittleness. not often associated with connectionist networks. The n↔m error and the 't' error, for instance, are hard to explain without recourse to quanta-level comparison. These are the sorts of errors that a role-based system or a human would never make.

Together, the bidirectional category errors and the individual token errors suggest that Netrec+, though able to generalize impressively well over its training set, develops no internal representations at the crucial intermediate level of roles. Instead, it develops a sophisticated statistical mapping between sets of specific quanta and categories. This is not surprising in light of theoretical knowledge about backpropagation, and is just what should be expected from a network with such a simple architecture and training task.

Figure 8.7 shows all of the tokens that simple-Netrec misclassified but that people

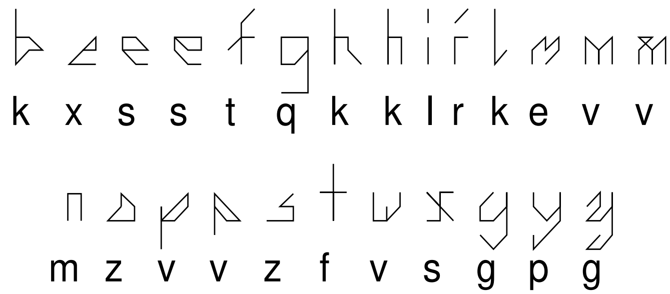


Figure 8.7: A complete list of Netrec errors that were never made by people. Netrec's incorrect categorization is shown below each of the tokens.

always got right. There are 25 such errors, roughly twice as many as Dumrec's and five times more than Netrec+'s. It is clear from studying the figure that most of the errors are caused by low-level feature mistakes. See especially 'e'→'s', 'g'→'q', 'n'→'m', and 't'→'f', which are predicted by the closeness of letter categories in quanta-overlap space as defined by blurred prototypes (see Table 7.4). These errors reinforce the theory that the network is paying close attention to low-level geometric features near the level of quanta instead of to higher-level parts. Some of the other errors are clearly based on collections of particular simple features or sets of quanta. Consider, for instance, the 'i'→'l' error, the 'i'→'r' error, and the 'b'→'k' error. In general, errors made by the network seem far less reasonable than people's errors (see especially the 'm'→'e' error).

The Netrec and Netrec+ models raise important questions about the link between abstraction, averaging, and mimicking. It is clear by considering the Dumrec data that more abstraction generally improves system performance. The role model confirms this view, as its level of abstraction is even higher (since it is based on the role hypothesis) and its performance is far superior. Netrec+'s vastly improved performance over both simple-Netrec and Dumrec suggests that its representations must be more abstract than theirs are. The question is *why* its representations might be more abstract, and, if they are, exactly *how* they are.

By considering the simple-Netrec model, we can cast some light on the situation. It is clear that simple-Netrec is doing mostly quanta-level processing. It may be generalizing, but its generalization amounts to a kind of statistical averaging over its training set. It is able to mimic the kind of input that it is trained on quite well, and its post-training categorizations strongly reflect this. But Netrec+'s behavior is so different from simple-Netrec's that if one did not know, one might be inclined to assume that its processing was not related. However, this would be a mistake.

Consider a lookup table. We can create a model using a lookup table whose accuracy values and errors are exactly the same as people's. But would this model tell us anything at all about the kind of processing done in human letter recognition? Obviously not.

Netrec+'s behavior is in some respects similar to such a lookup table's. Netrec+ is trained on tokens that are *very* similar to the tokens that it is tested on,[11] and so the fact that it does very well on our task is not surprising. But this does not mean that Netrec+ has developed internal category models that are different *in any deep sense* from simple-Netrec's. It simply has a better training set. In the end, then, we must conclude that there is no reason to suspect that any role-level processing is going on inside of Netrec+.

This may be a damning criticism of Netrec+, but Netrec+ has one thing going for it. Strangely, its advantage over the role model is directly related to its downfall. It boils down to *learning*. The role model provides no explanation for where roles come from. Neither does Netrec+, in its current guise, as it exhibits no role-level processing. But a related learning model that both conformed to the role hypothesis *and* explained where roles come from in terms of learning would be one large step beyond our current role model.

[11] See the Netrec+ section of Chapter 6.

8.3.6 Degraded Dumrec

The general trend of the correlation results in Table 8.4 shows that the full Dumrec model correlates better with human data than its degraded counterparts do. The slightly more abstract features included in Dumrec clearly improve performance. This is also evident in Figure 8.3, which shows the models progressing towards human performance as abstraction is added. Correlations between the Dumrec models and the role model also suggest that Dumrec's trend toward abstraction in processing is moving in the right direction. Improvement in performance is also evident for the FONTS data set.

The basic lesson is that although abstract features of the kind added to Dumrec unquestionably help, this type of feature-based, prototype model is still not powerful enough to handle eccentric letterforms. The large difference in processing between humans and Dumrec was probably magnified by the fact that the model was tuned using quite standard letterforms. Dumrec operates at too low a level — that is, it is too *shape-based* — to be able to deal well with category outliers. Its abstraction tools are not powerful enough.

8.4 Theoretical error-prediction matrices

The comparisons with human data drawn above suggest that the role model provides a quite good, perhaps excellent, model of human letter recognition. They also highlight the limitations of the Dumrec and simple-Netrec computer models we have implemented. In particular, Dumrec and simple-Netrec behave in an especially brittle manner when exposed to the exotic letterforms of FONTS, and often make poor category judgments.

The relationship between the role model and Netrec+ is, as we have seen, complicated by the fact that Netrec+ is a learning model that has been trained over an especially advantageous training set, whereas the role model is not. Nevertheless, the role model's predictive powers as far as error-making goes make it significantly more realistic as a psychological model.

While the role model, Dumrec, and Netrec are three specific implemented computer models of recognition, we can also make comparisons with theoretical error-prediction matrices. Such theoretical matrices, although they do not themselves simulate the recognition process but are instead based solely on notions of intercategory distances (*i.e.*, specific *tokens* are never involved at all), provide an additional way to characterize the behavior of recognition systems — either human or computer — in terms of their representation of categories. We investigate two types of theoretical matrix, using category representations based on category-level templates and local features, respectively.

The two systems of representation that we evaluate correspond to two of the most commonly implemented approaches to letter perception. A variety of models make use of either template-matching or low-level feature-matching. (See Chapter 6.) Commercially available OCR systems are especially wedded to these two approaches.

We suppose that any letter category can be thought of as a vector representing a point in some high-dimensional similarity space. The distance between letter categories in this space is directly determined by a theory of representation of categories. For example, consider the quanta-level blurred prototypes (introduced in Chapter 7), in which each category is represented as a 56-dimensional vector with real-valued entries. This representation is based on a theory of *low-level* proximity of categories (based on the idea of quanta overlap, which is a primitive kind of template-matching). There are a variety of different theories that can be used to define other types of representations. Below we discuss three different theories of intercategory proximity, giving rise to two sets of predictions regarding the likelihood of intercategory confusion.

8.4.1 Generating error-prediction matrices

Given a theoretical method for defining categories, we can generate theoretical predictions of error rates by first computing the distances between all possible pairs of letter categories (resulting in a 26×26 distance matrix), and then transforming this matrix into a 26×26 error-prediction matrix using a power function, as described below. By comparing (via correlation) the resulting error-prediction matrices with those of an actual system (human or machine), we can find out how closely processing in the actual recognition system coheres with the category theory used to build the theoretical matrix.

More technically, our error matrices were generated as follows. Each theoretical approach provides a method of representation of categories (all are described in detail below). Using these representations, we compute a real-valued distance between every pair of letter categories. Every entry, d_{ij}, in the resulting distance matrix is then transformed using a function well known to psychologists: $a_i e^{-kd_{ij}}$, where each a_i is simply a normalizing factor that sets the total of the ith row to unity, and k is adjusted to set the overall percent correct (average of entries along the main diagonal) to a particular value. This exponential formula converts each unit increment in intercategory *distance* into a reduction of that intercategory *error* by a constant factor. Greater distances mean rarer errors (hence the negative sign in the exponent). The value of k controls the dispersion of values away from the main diagonal. Large values of k will leave the diagonal entries at 1 (since the distance between a category and itself is always zero), but will minimize all predicted error-rates. By contrast, very small values of k (near 0) bring all entries close to equality. For each theory's matrix, we tuned k so that the overall percent correct of the matrix (the average of entries along the main diagonal) equals the overall percent correct of the human data.[12] This

[12]For the purposes of the correlations below (shown in Table 8.10), we set the total percent correct

factor therefore adjusts the overall percent correct of the matrix to be in accord with that which resulted from a particular dataset and particular stimulus-presentation conditions. It does not, however, change the order relation among errors.

8.4.2 Theories of intercategory proximity

We consider three different theories of intercategory proximity, which use representations based on: template-matching, local features, and global features. The template-matching approach simply amounts to noting the presence or absence of quanta in a letterform, and corresponds to the level-0 data used in Dumrec. The local-features approach results in a vector recording each of the intersections and ends of letter segments (such as T-junctions and tips) tagged with the features' location. In the global-features approach, properties are derived from the entire gestalt of a letterform, such as "nonclosed letter standing on a broad base" [Keren and Baggen, 1981]. The theories that we investigate are in direct competition to our role hypothesis.

Template-matching

Inter-letter distances between the 56-dimensional blurred-prototype vectors (described in Section 7.3.2 of Chapter 7) were calculated using the city-block method.

Local features

OCR systems have employed detection of local features as a basis for recognition [Guyon et al., 1989; Gaillat and Berthod, 1979]. A list of all possible local features on the grid (such as quanta-intersection points) was created and "blurred feature prototypes" were formed using the resulting 1,696-dimensional space (in the same

(*i.e.,* accuracy) of each theoretical matrix to the human value. Experimentation showed that correlation r-values were quite consistent over a large range of target accuracy values. The data we report reflect tuning to the human accuracy value, as the human data are of primary interest.

manner as with the blurred-prototype approach discussed above). Once again, distances between letter-category vectors were calculated using the city-block method. The top ten bidirectional category errors predicted by the local-features matrix are shown in Table 8.9.

Local Features	
Error	Error Rate
m↔n	16.10
u↔w	7.90
c↔e	6.20
b↔h	5.80
o↔u	4.25
v↔w	4.00
u↔v	3.85
e↔o	3.85
n↔o	3.60
n↔r	3.40

Table 8.9: The top ten bidirectional category errors from the local-features prediction matrix.

Global features

Psychologists have proposed several feature lists as bases for human perception of various figures, notably uppercase letters or numerals [Gibson, 1969; Gibson, 1971; Keren and Baggen, 1981; Haber and Haber, 1981; Treisman and Gelade, 1980]. These features are usually global and boolean in nature, so that in this scheme a letter category amounts to a vector of 1's and 0's, each of which corresponds to the presence or absence of one of the global features. We sought to create theoretical error matrices for at least one such feature list, but found that these lists were too rigid to apply beyond their original single-font data sets. In addition, the feature lists of [Gibson, 1971] and [Keren and Baggen, 1981] were intended solely to discriminate between uppercase letters, and are irrelevant to the discussion of lowercase letters. That is, for both feature sets there are several pairs of lowercase letters that possess exactly the same set of features. No theoretical error-matrix of reasonable quality could be formed on the basis of either of these feature lists (so we did not create one). Although the originators of these lists admit as much, this shows plainly that reasonably short high-level feature lists cannot be constructed that serve to explain recognition of larger sets of stimuli. Lacking generality, they cannot serve as a basis for a theory of categorization.

8.4.3 Results from the theoretical error-prediction matrices

Table 8.10 shows the Pearson r-values comparing all of our models, and humans, with the two theoretical matrices. These correlations take into account only off-diagonal confusions.[13] On the one hand, it is clear from this chart that Dumrec, simple-Netrec, and, to a lesser extent, Netrec+, all rely heavily upon template-matching and local features in arriving at their decisions, with almost half of the variance of the simple computer models coming from these low-level factors in some cases. On the other hand, humans and the role model hardly use this information at all (see Table 8.10). Much of the variance in human data and role-model data that can be explained by template-matching or local features is likely due to the correspondence between those simpler approaches and the role hypothesis. An argument to that effect can be concisely stated as follows: roles affect the kinds of parts that a letter can have by providing norms for such parts; specific parts of a letterform are formed out of quanta; thus roles provide top-down constraints on the low-level properties of letters, and their effects on the lowest level can be easily seen.

Both the role model and the human data have low correlations with the two theoretical matrices, and move from low on template-matching to even lower on local

[13] We omitted the diagonal so that the correlations would not be overwhelmed by the strength of the correct category vector.

	Theoretical error matrices	
	Template-Matching	Local Features
Template	1.00	
Features	0.81	1.00
Netrec	0.40	0.30
Netrec+	0.44	0.39
Dumrec-S	0.59	0.54
Dumrec-NT	0.66	0.61
Dumrec	0.52	0.59
Role Model	0.37	0.32
Humans	0.33	0.29

Table 8.10: Category-level correlations between theoretical prediction matrices, computer models, and people. This table sheds light on the *kind* of processing done by the models and by people. The numbers shown are off-diagonal correlations.

features. Netrec+ has lower correlations with the theoretical matrices than do Dumrec and simple-Netrec, but higher than does the role model. The simple-Netrec model shows an obvious tendency towards template-matching. The Dumrec-S and Dumrec-NT models share this tendency of increasing correlation towards template-matching, but to a lesser extent. The full Dumrec model, however, shows an opposite tendency from all the other models — it has a stronger correlation with the local-features matrix. Thus, the kind of processing used in each of the simple computer models (Dumrec and simple-Netrec, and to a lesser extent Netrec+), as well as the level of abstraction of processing in each of the models, are rendered clearer by consideration of theoretical error-rate prediction matrices.

It is worth noting that the most outrageous Dumrec, Netrec, and Netrec+ errors — exhibited in Figures 8.5, 8.6, and 8.7 — are those in which the token has a strong resemblance to the incorrectly guessed category if viewed on the level of quanta or that of local features, but in which the discrepancy at the level of roles is huge, and so such errors are only rarely made by humans.

8.5 Discussion

8.5.1 What is missing in the Dumrec and Netrec models?

The main weakness of the Dumrec and Netrec models is a lack of flexible higher-order representations in the recognition process. There is obviously more to human letter recognition than the noticing of simple geometric features (as in Dumrec and Netrec). Although the enhanced Dumrec models make use of more abstract properties, they still rely too heavily on low-level geometric features (as is apparent from Table 8.10). Netrec+ pushes the limits of a low-level feature (or "flat") model to its maximum potential, reinforcing the observation that a statistical learning model is capable of mimicking its training set very well. People are nonetheless much better at recognition precisely because their representations of letterforms are more abstract. Of course, this abstraction comes, at least in part, from exposure to an enormous variety of inputs (as with Netrec+).

As we have shown, people's representations seem to jibe with a description at the level of roles. There are certainly limits to the amounts of stretching that people's representations can accept, but some ability to adjust flexibly under pressure is required [Jackendoff, 1995; Hofstadter, 1985; Blesser et al., 1973]. The two simple families of models reported here, Dumrec and Netrec, contrast with the human case and with the more complex role model. It is clear that more sophisticated models are required. We suggest our role model as one possible implementation of the role hypothesis.

8.5.2 Why the role model is good

Based on all of the analyses given above, it is clear that the role model is far superior to both the Dumrec and simple-Netrec models. It is also seen to be superior to Netrec+ when error-making is taken into account. The role model's overall behavior correlates very strongly with human behavior on a dataset involving multiple fonts and style-filled letterforms (0.9821 on PSYCH). Error-making is reasonably similar in nature to human error-making, suggesting that the role model captures important facets of the underlying representations that people use in letter categorization. The fact that the model is able to perform well in both the NORMALS and FONTS conditions shows its level of sophistication.

Our look at theoretical category representations and letter categorization strongly suggests that the usual approaches to recognition, based on templates and local features, have fundamental flaws. The role model provides a working implementation of a different theory of category representation and recognition of tokens — the role hypothesis.

There is another critical difference between the role model and its simpler counterparts (including Netrec+). The role model's output is more than a mere category label. In addition to providing a categorization, the role model returns a *parsing* of any given letter in terms of roles. The parts that the role model identifies in a letterform can be used in further processing. Our intention has always been to use the role model as the perceptual module of a gridfont-design system — Letter Spirit — that applies analogical processing to the task of large-scale design. The ability to refer to a given letterform's perceptual structure and to map it analogically to other categories is possible only if a letterform can be successfully seen as filling some letter conceptualization's roles. Our strong desire to create a model based on the role hypothesis ends up providing Letter Spirit with a very strong model of letter perception based closely on human cognition.

Results from the data analysis reported above serve to justify the role model in several ways. The results of Palmer [1977] and our follow-up experiment on parsings discussed in Chapter 7 suggest some guidelines for taking a structured approach to letter recognition and parsing letterforms into parts. The discovery that humans consider more than one category when presented with ambiguous letterforms and that humans can respond to the exact same stimulus presented at different times with two or more different answers suggests that the overall control principle be a nondeterministic one allowing a number of answers to be at least superficially considered in parallel. The various tasks in each possible matching, including the buildup of parts and the filling of roles with parts, probably also happen in parallel. We feel that the central finding of this inquiry is that modeling the human process of letter recognition is an essential — rather than simply an attractive — ingredient of a successful approach to letter recognition.

CHAPTER NINE

Contributions of This Research

It is safe to say that Letter Spirit is the most ambitious project yet undertaken at the Center for Research on Concepts and Cognition. Although it was dreamed up in the early 80's, and its time has truly yet to come, Letter Spirit has finally been moved off the back burner and directly over the fire. The work in this thesis — focused on letter perception, letter conception, and letter understanding — is a crucial milestone on the way to a complete model of gridfont design.

The long road to the implementation of the Letter Spirit architecture began with Hofstadter's work on the Jumbo program in the years 1982–83. Jumbo served as an initial spark of a program — a preliminary test of a new architecture, and a hint of things to come. Over the years 1986–1990, Melanie Mitchell's impressive realization of the ideas of Copycat proved that the mechanisms of Jumbo, now in mature form, were fundamentally reasonable and were applicable to complex high-level cognitive problems. Bob French's Tabletop program, co-evolving with Copycat, emphasized the generality of the newly fleshed-out parallel terraced scan, as well as the importance of slippability and high-level perception. Together, Copycat and Tabletop represent a new approach to cognitive modeling, in which the fluid nature of concepts is the most central concern. Cognitive scientists would do well to focus on this conceptual fluidity when building their models.

Implementation of Copycat and Tabletop helped to move the once-frightening Letter Spirit project within the realm of the thinkable. It is thanks to the groundbreaking work on those programs that Letter Spirit is finally taking shape.

Part I of this thesis was devoted to explaining the aims and goals of Letter Spirit at the highest of levels. There, a general architecture for Letter Spirit is painted in broad brushstrokes. At the outset, we introduced our theory of letter conceptualization — the role hypothesis — based on the idea that letter concepts are made up of constituent roles. The role hypothesis served to guide Letter Spirit's approach to letter design and perception. Comparisons between Letter Spirit and other models of typeface design and creativity served to emphasize the importance we place on perception. A central tenet of this thesis is that perception is absolutely essential to the "central feedback loop of creativity". This emphasis has forced us to concern ourselves deeply with letter perception. For in order to design letters within the implicit esthetic guidelines dictated by an emerging style, one must first be able to *see* and *understand* letters. Design without perception, though possible and even the norm in AI, bears little or no relation to human creativity, and as such has no importance for cognitive science.

Our absolute insistence on the need for a deep model of letter perception made it necessary that the first implemented module of Letter Spirit be the Examiner, a model of letter recognition based on the role hypothesis. This has now been fully implemented. The role hypothesis has proven to yield a very strong model of letter recognition, as it clearly explains much of human letter-recognition behavior (especially with regard to error-making). The mechanisms developed in Copycat and Tabletop have shown their mettle once again, this time applied to a task apparently at a less abstract level of cognition than analogy-making.

Hofstadter, Mitchell, and French have claimed that the mechanisms of analogical thinking have much in common with the mechanisms of perception [Hofstadter et al., 1987; Hofstadter and FARG, 1995]. Work on the role model supports these claims by successfully applying the same general mechanisms that have been shown to work in models of analogical thinking to a lower-level categorization and perception task.

There remains much work to be done on Letter Spirit. Work on the Imaginer and the Drafter has only just begun. The baton will soon be passed to John Rehling, who will hopefully push the CRCC theme of conceptual fluidity further than it has yet gone.

9.1 The role model and human data

The results reported in Part II of this thesis point out fundamental shortcomings in approaches to letter recognition that do not deal with letters as realizations of roles in the same way as people do. Establishing the importance of roles as units of representation is an empirical step that confirms the architecture of Letter Spirit in general and the role model in particular.

Among the critical portions of the recognition process in the role model are these: building preliminary parts in a bottom-up fashion from low-level data, adjusting parts under top-down influence from roles, noting aspects of style — norm violations — that result from filling roles with particular parts, evaluating prospective filled role-sets as matches of different letter categories, tracking the strengths of activation of competing categories, and searching for new perceptual parsings if initial attempts at recognition result in only weak categorization.

Correlations between role-model and human-subject performance on common datasets show that the role model captures many of the essential features of human letter perception. Category-level correlation over the 544-token PSYCH dataset is 0.9821 ($p < .0001$). As is evident from the correlation, letter perception in the role model is very similar to letter perception by people. This is especially relevant as the correlation was made over a large and varied dataset. Specific analysis of error-making confirms this claim. Together, the results of comparing the role model to humans provide strong evidence that roles are an important part of people's underlying representations of letters.

Unlike many recognition programs, the role model does not simply perform very well on one style, only to crash and burn on others. It was designed specifically to handle a huge variety of styles. Tests using large datasets show that we have captured at least some of the perceptual fluidity exhibited by people. The role model is able to recognize both standard and stylistically eccentric letterforms. Though its performance degrades when presented with weird letterforms, so does the performance of people (to a lesser extent, however). In any case, the role model is certainly a more flexible model of letter recognition than all others that we are aware of.

The role model's overall success suggests that a "structured" approach to letter perception may help to close the gap in performance between people and machine recognition systems. Most OCR programs take a "flat" approach to recognition — almost certainly the main reason for their brittle performance. We feel that the time is ripe for OCR models based on confirmed psychological theories of representation. The role model provides a particular implementation of one such theory — our role hypothesis.

A distinct advantage that the role model holds over simpler models lies in its end result. The role model returns not only the usual category label, but also a parsing of a letterform in terms of its constituent parts. Part-level parsings correspond to role-level conceptualizations of letters. Further processing — for instance, the extraction of style information, or the analogical design of other letterforms — is possible only with the kind of structural information that the role model provides. The ability to parse a letterform into natural parts corresponding to roles is of critical importance to the future design phase of Letter Spirit.

9.2 Weaknesses in the role model

9.2.1 Limitations to flexibility

Though the role model exhibits great flexibility in its performance and shows a clear ability to correctly recognize many stylistically-varied letterforms, it still has several weaknesses. Because we have subjected the role model to rigorous testing on hundreds of letterforms run thousands of times, we are in a position to note some of these weaknesses with an eye toward future improvement of the model. The kind of probing we have done on our model is reminiscent of the Turing-test approach to probing a possibly-intelligent program [Turing, 1950], the idea being that such a program can be successfully judged only when subjected to sufficiently varied probes. Understanding how well any model captures specific psychological behavior is possible only through extensive testing using systematically varied input. With this in mind, we tested the role model on letterforms varying in a vast number of ways. We also made systematic changes to key system parameters to see what effects this would have on the system.

One side effect of all this testing is a clear picture of some of the role model's current weaknesses — weaknesses that it is important to report. Some of these weaknesses involve the ways in which letterforms, when varying, become unexpectedly unrecognizable. (These were reported in Chapter 5 and are summarized below.) The role model is sometimes unable to handle letterforms that people can easily recognize. Letters that feature the following kinds of properties are especially hard for the role model:

- letterforms with "thick" parts composed of regular patterns of quanta — visual textures — or closed shapes that should be seen as "solid";

- letterforms in which a quantum is swung like a gate, causing what would have been a closed circuit to become open;
- letterforms with one or more gaps where more standardly there would be a continuous line segment — gaps of the sort that people have no trouble filling in mentally with "invisible quanta";
- letterforms that fall squarely in the middle between two different role-sets of the same letter category, causing "fratricidal" fights (this is only rarely a problem, actually);
- letterforms with particularly sparse parts.

Many of these problems should be fairly easy to address without requiring a deep revision of the role model. Future local patches to the role model should enhance its ability to recognize these kinds of letters.

A different sort of weakness in the model is related to the amount of plasticity that the program allows in a letterform's role-filling parts. The role model is easily able to handle many types of norm violations. Metaphorically speaking, its plastic roles are usually able to "bend" to encompass particular grid-bound parts. R-role slippages are also properly handled in most cases. However, if too many minor changes are required *at once* — especially if all of the roles of a particular role-set must bend a great deal — the role model breaks down. Clearly, humans' roles are even more plastic and flexible than the role model's.

This is not to say that the role model is extremely brittle. What is under consideration is not whether the model can act flexibly *at all* — as we have demonstrated, it can surely do that. The question is, rather, *how much* the role model can bend its roles versus how much people seem to be able to bend theirs. People are still better. . . by a long shot.

9.2.2 Learning roles

One aspect of cognition that the role model does not address is learning. We have come under fire in the past for presenting a model in which roles are "spoon-fed" to the system. In fact, some have gone so far as to call our work "doomed to succeed" [David Stork, personal communication]. We can attest to the fact that the spoon-feeding of roles is indeed a tedious and a time-consuming process. So, wouldn't it be better to have a system that *learns* the roles?

In a word, yes. The problem is, we see no current learning methods that would allow the kind of roles that we have implemented to be learned. Having roles emerge as distributed representations in the hidden layer of a connectionist network, for example, is a problem that is, at the moment, beyond connectionist capabilities. Research in connectionism that aims to address these issues (*i.e.*, reference, part–whole hierarchies, analogical mappings, and so on) is at a very early stage of the game [Pollack, 1989; Hinton, 1990; Chalmers, 1990; Regier, 1992; Meeden, 1994; Blank, 1995]. It remains to be seen whether roles can be learned, made explicit, and made available for further processing in connectionist systems.

Both the Netrec+ model and Stork's GridFont network ignore roles and role-level processing in the hopes that this level of processing can be "skipped over" during the recognition and design tasks, the idea being that learned distributed representations in the hidden layer should suffice. Results show, to the contrary, that although low-level, statistical approaches to categorization can indeed work to a limited extent, they suffer from flaws that are extremely serious. These flaws have to do with ignoring the level of roles.

This is not to say that we believe roles cannot be learned. However, naïve approaches that assume that backpropagation (or some similar algorithm) will cause roles to emerge in very simple three-layer networks *without specific training subtasks* (which, if well-designed, might *force* the system to process roles) or *without specific* *architectural features* (which likewise might force a network to process roles) is misguided. Before any progress can be made along these lines, these vague ideas about training subtasks, better representations, or more sophisticated architectures would require a great deal of fleshing-out.

In the meantime, the role model shows that non-brittle, context-sensitive roles (regardless of whether they are learned or not) can be used to put top-down pressure on the categorization process. In its favor, the role model has many features in common with subsymbolic approaches (of which connectionism is a distinguished member). These features impart an unprecedented amount of flexibility to the model, setting it apart from more traditional symbolic-AI approaches.

9.2.3 Self-watching

Possibly the most important way in which the role model could be improved has to do with *self-watching*. The way that roles are loosened according to a rigid codelet-count–based schedule is unacceptably mechanical. A more reasonable system of role loosening would require the system to be able to watch its own behavior and adjust its processing accordingly. This would involve a far greater degree of self-regulation than is now possible. Figuring out just how to have a codelet-based system track its own behavior at a reasonably high level is a current research issue at CRCC [Marshall, 1993; Hofstadter and FARG, 1995, Chapter 7]. Self-watching and high-level regulation are deeply tied to questions of how much plasticity to allow in a system, when to allow particularly radical slippages into the picture, and what the appropriate level for tracking processing throughout a run is.

9.3 Cellular processing, perception, and the parallel terraced scan

The role model's distributed asynchronous parallelism, like Jumbo's and Copycat's, was inspired by the self-organizing activity that occurs in a biological cell [Hofstadter, 1984]. In a cell, all activity is carried out by large numbers of enzymes of various sorts, distributed throughout the cytoplasm. Random motion in the cell's cytoplasm results in substrates (like amino acids or glucose, for example) being encountered by enzymes. Enzymes can build larger structures (such as proteins or carbohydrate chains) out of these substrates. Local enzymatic actions work together in a long series without any global supervision to form complex structures. The end result is that many complex structures are built independently and asynchronously in various locations throughout the cell. The moment-to-moment needs of the cell, standing in direct relation to the kinds of structure that have been built so far, serve to influence the current enzyme population, regulating it from within.

In the role model, codelets play the part of enzymes. All processing is carried out by codelets that probabilistically choose what to work on, building perceptual structure as they run. Chains of codelets working throughout the Visual Focus build up complex structures. Just as in a cell, the codelet population self-regulates according to the moment-to-moment needs of the system. The presence or absence of particular structures influences the codelet populations controlling processing from within.

It is possible to push the cellular processing metaphor even further. Consider that genes inside DNA are selectively turned on and off by regulatory elements in a cell — as a function of chemical context, or stage of processing inside the cell. The Conceptual Memory could be likened to the DNA of a cell, with active roles and wholes corresponding to genes currently being expressed in the cell. Just as these genes serve to control enzyme production in a cell, active roles and wholes control processing in the role model by causing top-down codelets to be posted to the Coderack.

The metaphor of the cell provides general processing principles that, properly abstracted, can be carried to a perceptual architecture. Distributed parallelism, emergent processing, local structure-building, self-organization, and self-regulation are all central to our model of perception. Many of these mechanisms were inspired by the cell metaphor.

In the role model, the tradeoff between exploration and exploitation is regulated by the parallel terraced scan, which allows paths to be explored in proportion to their estimated promise. Since the estimates change as processing proceeds, the system dynamically adjusts its search strategy over time in a highly context-sensitive way. A parallel investigation of many possibilities to different levels of depth and with different speeds is the result. The parallel terraced scan has much in common with the k-armed-bandit problem and related solutions to exploration and exploitation commonly found in genetic algorithms.

The role model draws on ideas both from the fairly recent subsymbolic paradigm and from the classical symbolic paradigm. This mix of approaches results in two key properties of the model: 1) a great flexibility inherent in its processing, and 2) the ability to create structures that are easily used in high-level cognitive tasks such as analogical mapping. Flexibility results from the use of fine-grained parallelism, spreading activation, and self-organizing emergent structure — all properties associated with subsymbolic systems. Structure-building arises naturally from the symbolic paradigm, in which explicit reference and hierarchical structure play central roles. This mix of philosophies results in an interesting sort of intermediary model, one that may help to close the gap between symbolic and subsymbolic systems.

9.4 The importance of modeling concepts

The main philosophical and scientific quest underpinning work on Copycat, Tabletop, and Letter Spirit is to come to better understand concepts. Our research aims to provide a scientific basis for the study of concepts and to help to more rigorously define their central role in cognition. Concepts are the building-blocks of cognition. Their fluid, associative nature is what imbues human perception and categorization with their marvelous flexibility.

At a high philosophical level, the question we are asking in this thesis is: *What makes human cognition so flexible?* The specific form this question takes with regard to the role model is: *How is it that we can see such a large variety of diverse letterforms as members of the same category?* The answer we propose is: We can perceive letterforms so flexibly precisely because of the context-sensitive, associative, malleable nature of our concepts.

Like its predecessors, the role model is an attempt to capture some of the essential features of fluid concepts in a working computer model. Our goal is to better understand the process by which concepts and percepts interact. Making actual working models that we can systematically and thoroughly test is an important part of putting flesh onto some rather ancient philosophical bones. High-flown philosophical ideas, when confronted with real-world implementation constraints, must often undergo a process of evolution and change as they are made concrete. Very serious implementation issues are raised throughout the development of any program as ambitious and complex as the role-model, as philosophical and psychological theories clash with the real world and the inevitable brittleness of computational hardware. It is gratifying to have seen the role hypothesis evolve from a philosophical and psychological theory to a working computer model — a model that we can validate empirically and test extensively.

This work has pushed the level of conceptual modeling at CRCC to a deeper level than in the past, as it has delved *inside* concepts to try and understand what they are made of. (Prior research was focused more on how nearby concepts overlap; the current research is focused more on the detailed nature of the "conceptual halo" of *single* concepts.) One of our research goals in this work is to try and understand the hierarchical relationship between roles and wholes. This work models one way in which "sub"-concepts work together (under the constraints of inter-concept relations) to form larger concepts (*e.g.*, how individual roles and r-roles together define a whole). A related goal is to develop an implementation of the way concepts are defined in terms of sets of norms. The role model of perception sheds some light on the relationship between norms, concepts defined by sets of norms, activation of concepts according to their norms, and conceptual plasticity.

The role model, together with Copycat and Tabletop, explores just what "active symbols" are about, and helps to suggest how active symbols can be computationally realized. The process of high-level perception, whereby an initially unstructured collection of sensory data organizes in a bottom-up way under top-down pressure from the conceptual level, has been shown in this work to play a key role in what seem at first to be simple categorization tasks no less than in analogical thinking of a more exalted variety.

Letter Spirit's eventual aim is to model a large-scale creative act using statistically emergent fluid concepts so that we may come to better understand the central mechanisms of creativity. Thus, a complete implementation of Letter Spirit will be of interest both as an experiment in imparting creativity to a machine and as a model of human creativity. This thesis, with its focus on perception, is a critical milestone along the road towards Letter Spirit. Throughout this work, we have claimed that human creativity is deeply intertwined with perception. Our emergent model of letter perception using fluid concepts is thus a central and fundamental dimension of the complete Letter Spirit project.

APPENDIX A: GRIDFONTS AND DATASETS

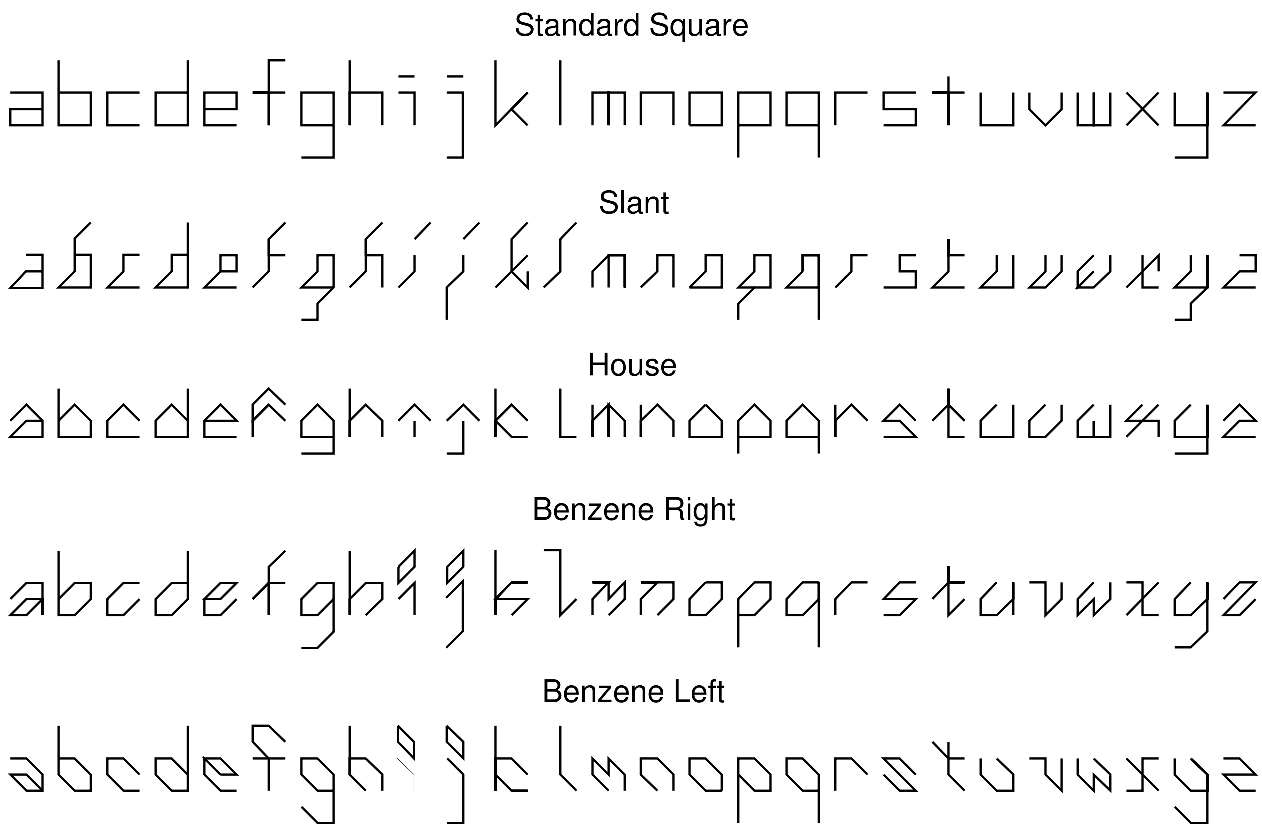


Figure 9.1: The five complete gridfonts used to train the GridFont network of Grebert *et al.* See Chapter 2.

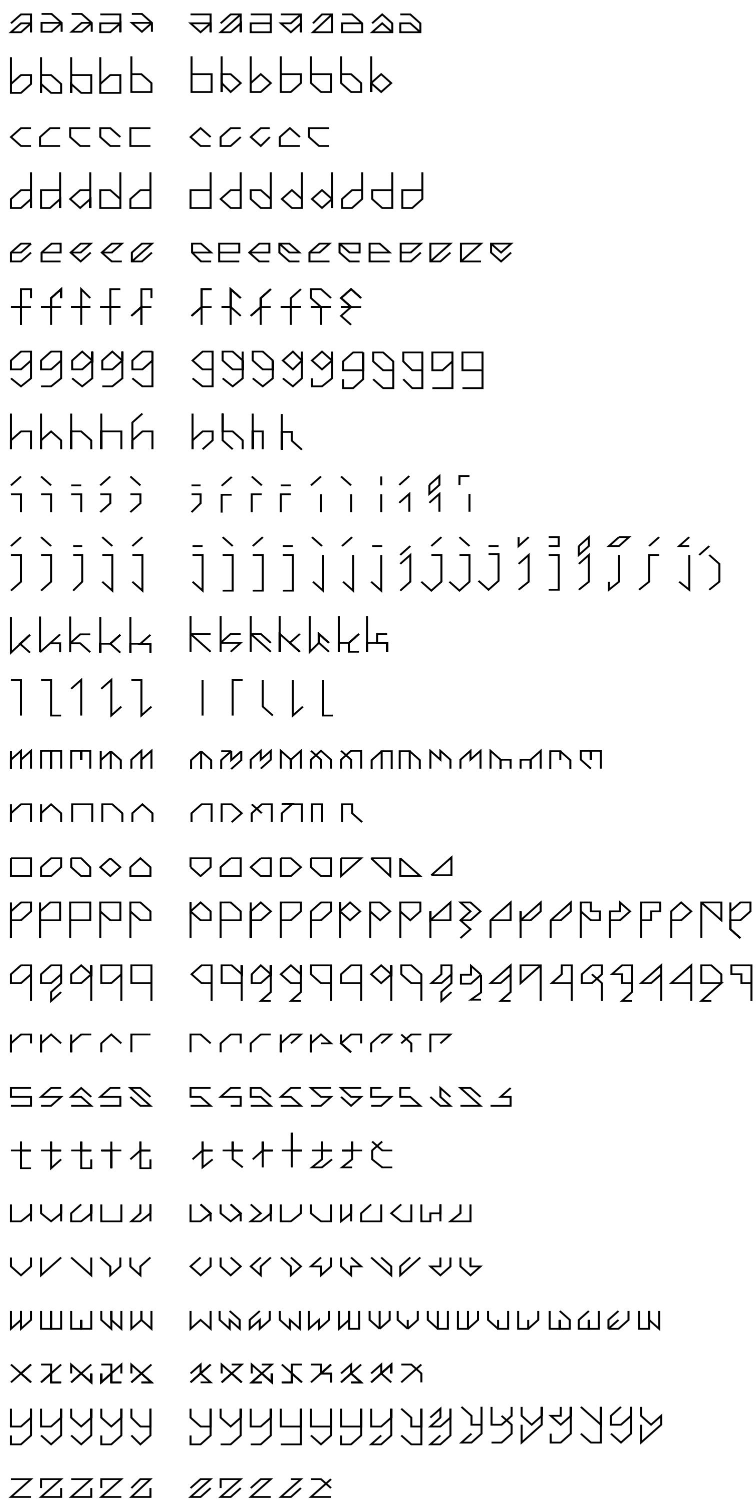


Figure 9.2: The NORMALS data set. Each of these letters is meant to be a strong member of its intended category. The first five tokens in each letter category made up the TRAIN data set.

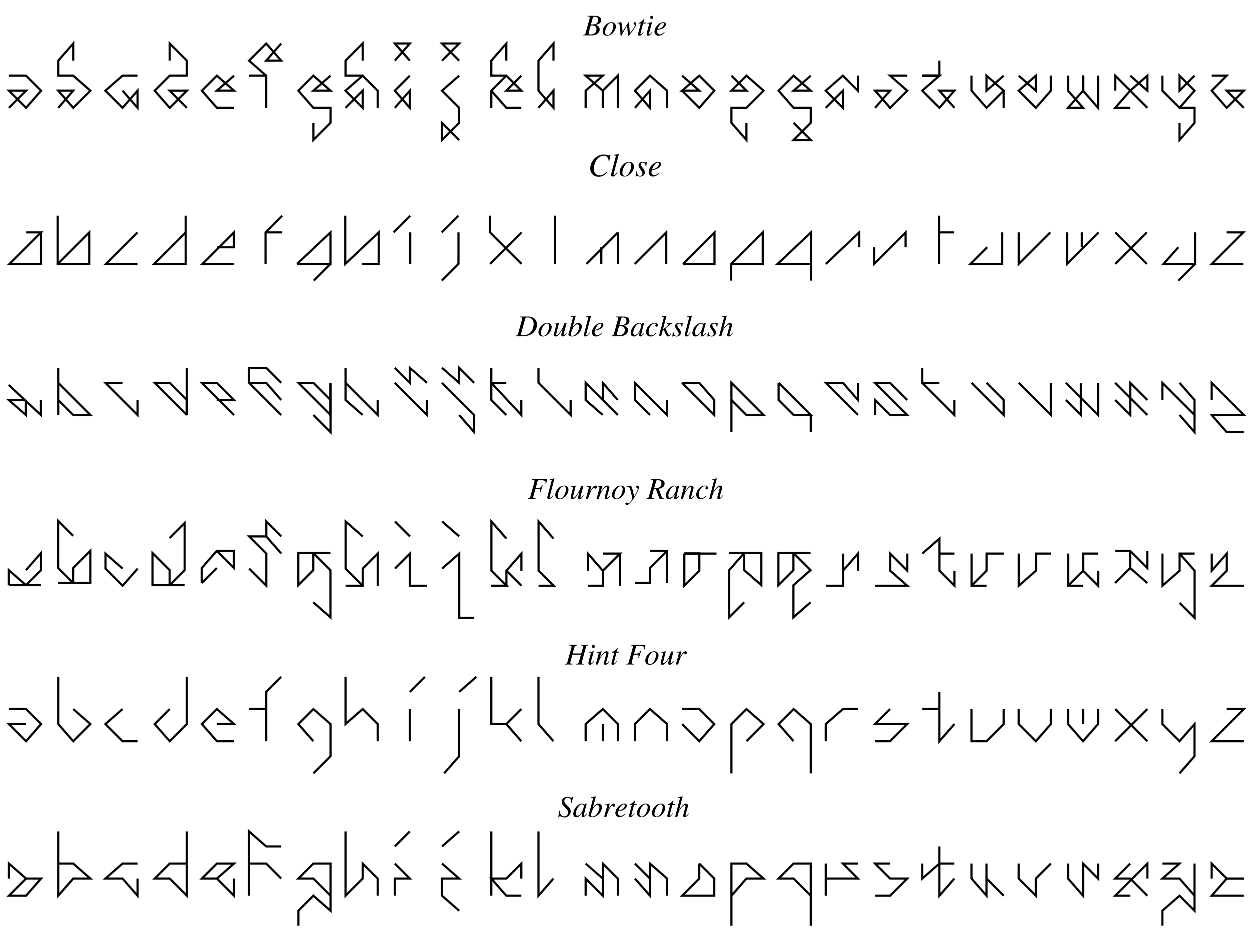


Figure 9.3: The six gridfonts that we used in the psychology experiment, making up the data set FONTS.

Omnipage in default mode		Omnipage after training on test set	
s aH ffla a	23% (3/13)	a A a a a a a	46% (6/13)
bibbb bibbibb	75% (9/12)	bbbbb bbbbbbb	100%
CCCU: tGG(C	0% (0/10)	ccccc ccccc	100%
dsiddl rJdiSodd[	46% (6/13)	ddddd ddddddd	100%
sevex eeeeeeeazes	63% (10/16)	eeeee eeeeeeeeeee	100%
f 4 t f 8 f f 5r $	36% (4/11)	fffff TAffff	82% (9/11)
99Eg9 9S9SS99999	7% (1/15)	ggggg gggggggggg	100%
hlqlahh loCh h	56% (5/9)	hhhhh hhh h	100%
7 7 r r r		111 w 7 7 r r r —	
/_ V n A ff o =	0% (0/15)	/ _ V n A ff o =	0% (0/15)
]]]III0JJJ0]BJXI')	0% (0/23)	]]]IIIjJJJ0]BJXI')	4% (1/23)
klzkSl kiLl	23% (3/13)	kkkkk k kkkk	77% (10/13)
ll111 IFtlL	30% (3/10)	lllll I llll	90% (9/10)
mmrW mD nmm	26% (5/19)	mmmmm mm nmm	47% (9/19)
nrn nno n rs	45% (5/11)	nnnnn n n n	73% (8/11)
zooon 97naDgvosa	29% (4/14)	ooooo ooooooosoo	93% (13/14)
PPPPS PPPPPPPP PFPI9P	0% (0/23)	ppppp pppppppppd ppppp	78% (18/23)
SialrlQ SD E489 1451d49E	0% (0/23)	qqqqq qqeSqqqqqedvqqqqqqq	83% (19/23)
rerer rnrrzF z	43% (6/14)	rerer rrrrzF r	57% (8/14)
	0% (0/16)		0% (0/16)
ttttvt tt++$it	67% (8/12)	tf ttt tttt$tt	83% (10/12)
UUGLJ BU2UL<]LrlS	0% (0/15)	UUuuS LA uUuUuuuu	53% (8/15)
Qvnvv veovvw	40% (6/15)	vVvVv <JR VvvVVv	40% (6/15)
WSL INW H9SWWkWWWWU92aSS	0% (0/22)	WwwwW wwwwWwWwWWwwwwww	71% (15/22)
X7bG>M M(R2ZXAM(y)	0% (0/13)	X7xxM xxxXxxxx	69% (9/13)
YY?YY YYYIYY5EtB?MSEUvS	0% (0/21)	YYyYY YYYyYYYYYYyyYYyy	29% (6/21)
zzzzf	40% (4/10)	ZZZzS	10% (1/10)
Overall % correct	24%	Overall % correct	65%

Table 9.1: How Omnipage, a top-of-the-line, commercial OCR program fares on the NORMALS dataset of Figure 9.2. The NORMALS were scanned in using a high-quality 600dpi scanner after being reduced to a standard 12pt size. The first and third columns are actual ASCII output from the program and can (in general) be "lined up" with the Figure. Note however, that some characters are omitted and others split into two. In the "Default" condition, Omnipage used its standard OCR toolset to try and recognize characters from the scanned image. In the "Trained" condition, Omnipage was trained by the user on each glob of ink that it could not understand in its default condition. It is surprising that even in the trained condition — in which the program has the potential to be helped *on every character* — Omnipage's performance results in such a low accuracy. Omnipage was not able to recognize any of the FONTS characters in either condition.

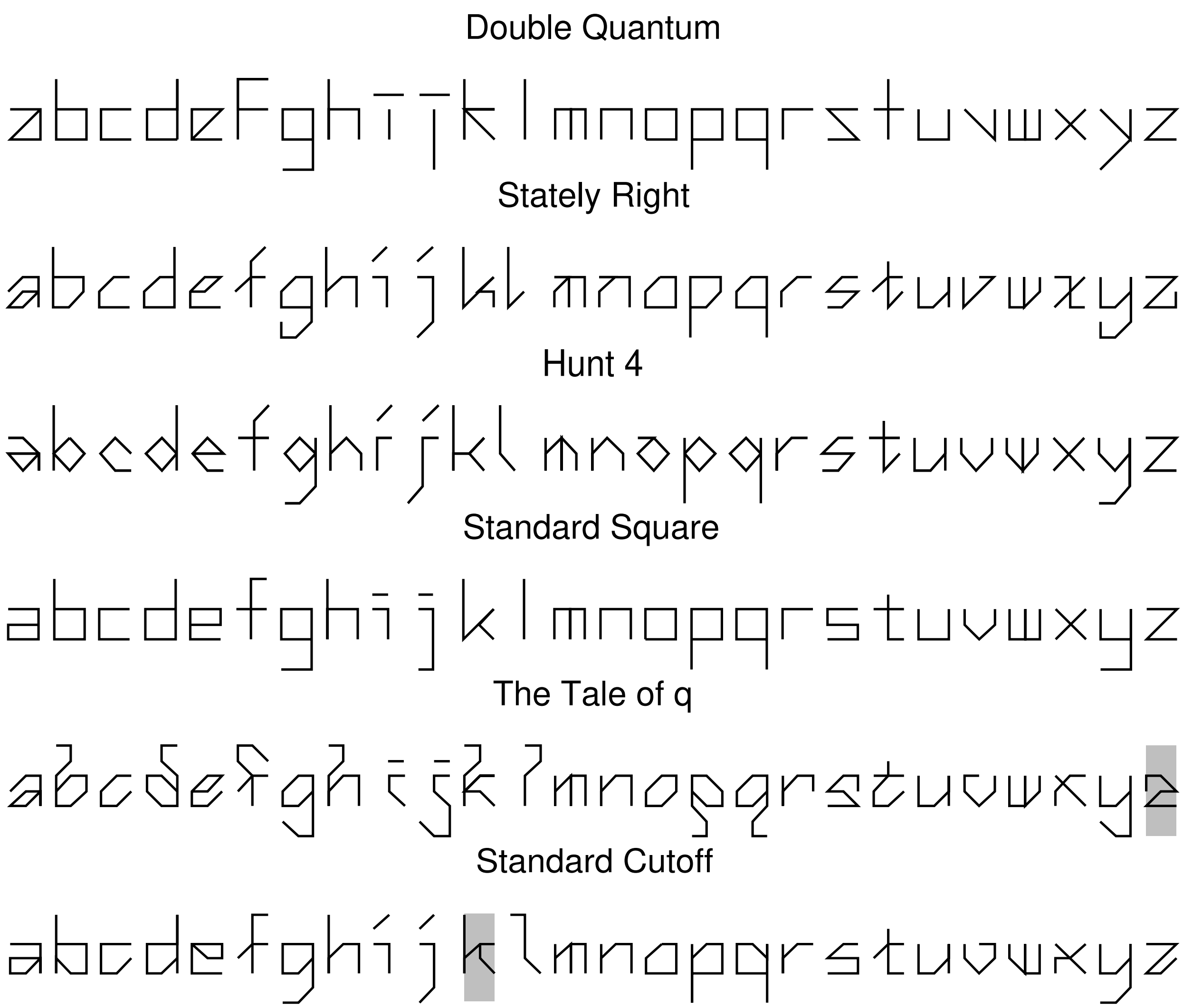


Figure 9.4: The role model's performance on six particular gridfonts. Letters in each gridfont that the role model does not recognize correctly over 50% of the time are displayed with a shaded background. Together with the twelve fonts shown in the next two figures, this collection of actual gridfonts gives some idea of the role model's flexibility.

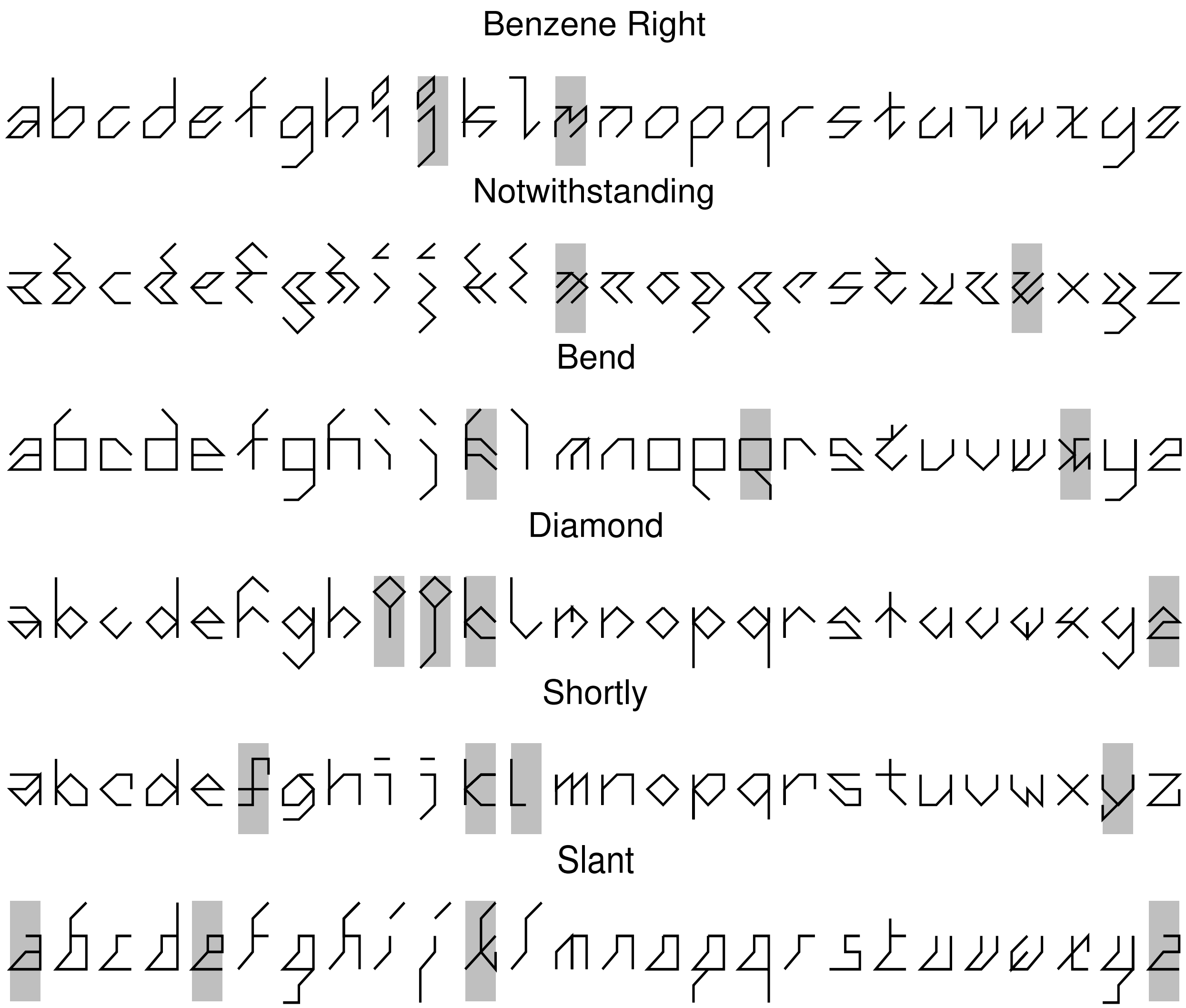



Figure 9.5: The role model's performance on six more particular gridfonts. Letters in each gridfont that the role model does not recognize correctly over 50% of the time are displayed with a shaded background. Together with the six fonts shown on the previous page and the six shown on the next page, this collection of actual gridfonts gives some idea of the role model's flexibility.

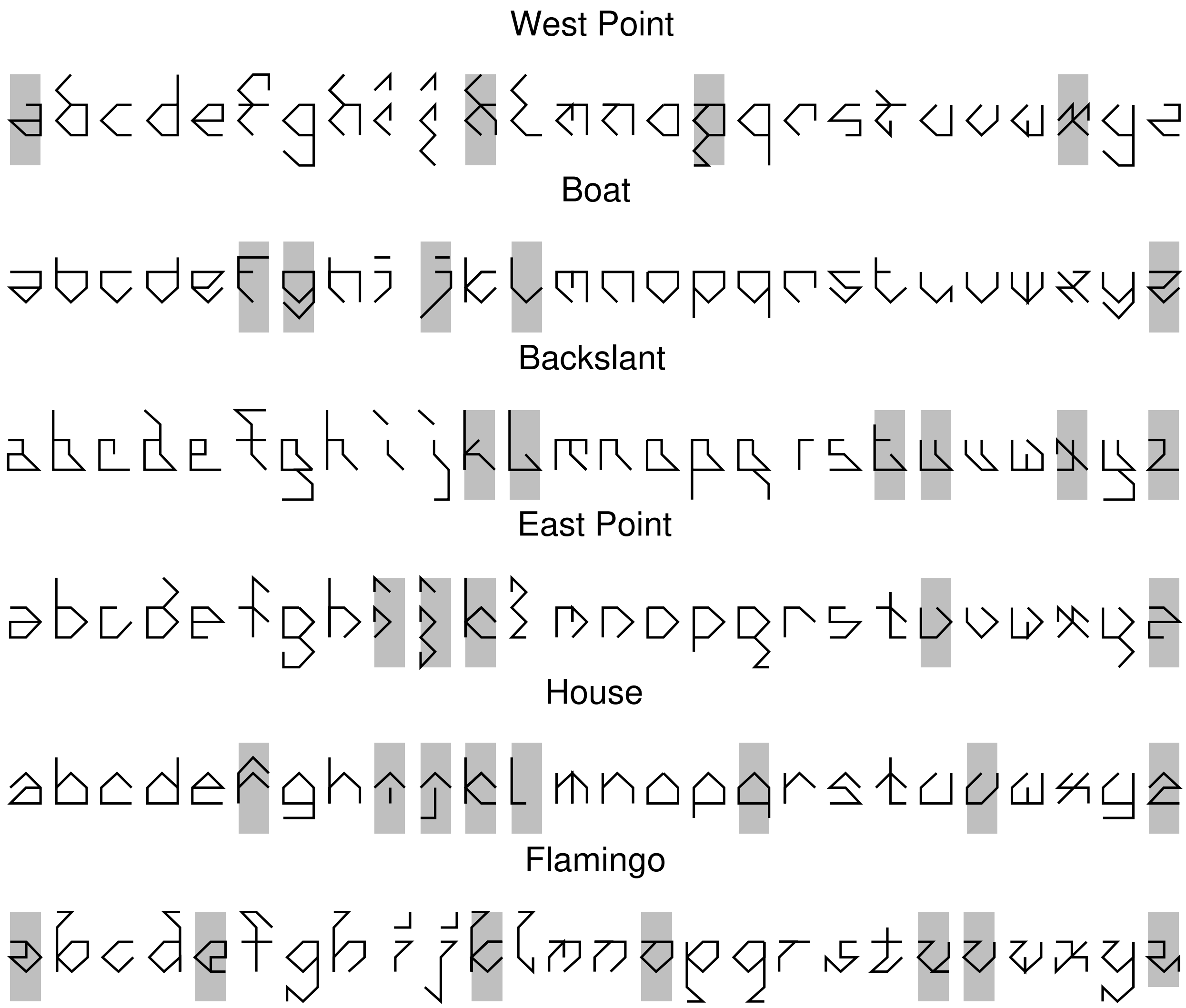


Figure 9.6: The role model's performance on six more particular gridfonts. Letters in each gridfont that the role model does not recognize correctly over 50% of the time are displayed with a shaded background. Together with the twelve fonts shown in the previous two figures, this collection of actual gridfonts gives some idea of the role model's flexibility.

APPENDIX B: INTERNET SITES

Tools that have been developed at CRCC for the Letter Spirit project include the Gridfont program and the 'a'tabase. The Gridfont program is an X application that allows a human to design, store, and manipulate gridfonts. It greatly facilitates our own observations of the creative process in humans. The 'a'tabase is a collection of gridbound 'a's, all of which can be given ratings by people, representing their strength as 'a's. Its purpose is to facilitate exploration of the vast space of potential gridfont 'a's in order to deepen our understanding of abstract letter-concepts — particularly, what makes for strong and weak category membership.

Many papers have been published about the Letter Spirit project. The core ideas found in most of these papers have been interwoven into this thesis. Nevertheless, if you are inclined to further pursue information about Letter Spirit, you might want to look at some or all of the following:

- Douglas Hofstadter & Gary McGraw (1995). Letter Spirit: Esthetic Perception and Creative Play in the Rich Microcosm of the Alphabet, Chapter 9 of *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, New York.

- Gary McGraw (1995). Emergent Perception of Gridletters. In Evans, M., editor, *Proceedings of the 1995 Midwest AI and Cognitive Science Society Conference*, pages 63–67.

- Gary McGraw, John Rehling, & Robert Goldstone (1994). Roles in letter perception: Human data and computer models. Technical Report 90, Center for Research on Concepts and Cognition, 510 North Fess, Bloomington, IN 47405, July 1994. Submitted for journal publication.

- Gary McGraw, John Rehling & Robert Goldstone (1994). Letter perception: Toward a conceptual approach, In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 613–618, Atlanta, GA, August 1994.

- Gary McGraw & Douglas R. Hofstadter (1993). Letter Spirit: An architecture for creativity in a microdomain. In *Advances in Artificial Intelligence: Third Congress of the Italian Association for Artificial Intelligence* (AI*IA 93), P. Torasso (ed.), pages 65–70, Torino, October 26–8, 1993.

- Gary McGraw & Douglas R. Hofstadter (1993). Perception and creation of alphabetic style. In *Artificial Intelligence and Creativity: Papers from the 1993 Spring Symposium*, AAAI Technical Report SS-93-01, AAAI Press.

- Gary McGraw & Douglas R. Hofstadter (1993). Perception and creation of diverse alphabetic styles. In *Artificial Intelligence and Simulation of Behaviour Quarterly*, Number 85, pages 42–49, Autumn 1993. University of Sussex, UK.

- Gary McGraw & Daniel Drasin (1993). Recognition of gridletters: Probing the behavior of three competing models. In *Proceedings of the Fifth Midwest AI and Cognitive Science Conference*, pages 63–67, April 1993.

- Douglas Hofstadter & Gary McGraw (1993). Letter Spirit: An emergent model of the perception and creation of alphabetic style. Technical Report 68, Center for Research on Concepts and Cognition, 510 North Fess, Bloomington, IN 47405, January 1993.

- Gary McGraw (1992). Letter Spirit: Recognition and Creation of Letterforms Based on Fluid Concepts. Technical Report 61, Center for Research on Concepts and Cognition, 510 North Fess, Bloomington, IN 47405, June 1992.

Many of the reports listed above are available by anonymous **ftp** from:

`ftp.cogsci.indiana.edu`.

Also available on the CRCC ftp site is a large (and growing) database of gridfonts. The gridfonts entered range from some fairly standard gridfonts to those completely awash in style. Our database can be used by researchers interested in exploring gridfonts and/or testing their own recognition models against ours. An X-windows application serving as a front end for the ASCII gridfont database may also be retrieved. It makes creation of a gridfont easier by providing a set of tools for entering and editing a gridfont.

Also available is the source code for the role model. The code was developed under Chez Scheme but should compile successfully under any standard Scheme implementation. The code has a graphical interface component that uses the SXM hooks developed by Motorola [Zuckerman, 1992]. Although SXM is not yet publicly available, the code will compile and run (without graphics) under Scheme. For information on obtaining a graphical version of the role model (or setting up a demo over the net) contact me via e-mail [`gem@cogsci.indiana.edu`].

Another way to get to the items listed above is through the World Wide Web. CRCC has a homepage set up at URL:

<`http://www.cogsci.indiana.edu`>

The CRCC Web pages also highlight some of the other work done at CRCC as well as providing links to CRCC researchers and other labs.

REFERENCES

Adams, D. (1986). A Dialogue of Forms: Letters and Digital Font Design. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA.

Biederman, I. (1987). Recognition by components: A theory of human image understanding. *Psychological Review*, 94(2):115–147.

Blank, D. (1994). Learning to see visual analogies. Unpublished manuscript: Indiana University, Department of Computer Science, Ph.D. thesis proposal.

Blank, D. (1995). A distributed representation of multiple objects in a visual scene. In Evens, M., editor, *Proceedings of the 1995 Midwest Artificial Intelligence and Cognitive Science Society Conference*, pages 143–147. Huntsville, AL: Intergraph Corporation.

Blank, D., Meeden, L., and Marshall, J. (1992). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In Dinsmore, J., editor, *The Symbolic and Connectionist Paradigms*, pages 113–148. Lawrence Erlbaum Associates, Hillsdale, NJ.

Blesser, B., Shillman, R., Cox, C., Kuklinski, T., Ventura, J., and Eden, M. (1973). Character recognition based on phenomenological attributes. *Visible Language*, 7(3):209–223.

Boden, M. (1977). *Artificial Intelligence and Natural Man.* Basic Books, New York.

Boden, M. (1990). *The Creative Mind: Myths and Mechanisms.* Basic Books, New York.

Boles, D. and Clifford, J. (1989). An upper- and lowercase alphabetic similarity matrix, with derived generation similarity values. *Behavior Research Methods, Instruments, & Computers*, 21(6):579–586.

Bouma, H. (1971). Visual recognition of isolated lower-case letters. *Vision Research*, 11:459–474.

Carter, K. (1984). IMP — a system for computer-aided typeface design. Technical report, Computer Laboratory, University of Cambridge.

Chalmers, D. (1990). Syntactic transformations on distributed representations. *Connection Science*, 2:53–62.

Chalmers, D. (1993). Connectionism and compositionality: Why Fodor and Pylyshyn were wrong. *Philosophical Psychology*, 6:305–319.

Chalmers, D., French, R., and Hofstadter, D. (1992). High-level perception, representation, and analogy: A critique of artificial intelligence methodology. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 4(3):185–211. Reprinted as Chapter 4 in *Fluid Concepts and Creative Analogies*.

Cox, C., Blesser, B., and Eden, M. (1971). The application of type font analysis to automatic character recognition. Technical report, MIT, Research Laboratory of Electronics, Cambridge, MA.

Cutler, A. (1982). *Slips of the Tongue and Language Production*. Mouton, Berlin.

Dartnall, T., editor (1994). *Artificial Intelligence and Creativity*. Kluwer Academic Publishers, Boston.

Dell, G. (1986). A spreading activation theory of retrieval in sentence production. *Psychological Review*, 93(3).

Dowker, A. (1995). Estimation strategies of four groups. *Mathematical Cognition*, 1(1).

Dybvig, R. (1987). *The Scheme Programming Language*. Prentice-Hall, New Jersey.

Erman, L., Hayes-Roth, F., Lesser, V., and Raj Reddy, D. (1980). The Hearsay II speech-understanding system: Integrating knowledge to resolve uncertainty. *Computing Surveys*, 12(2):213–253.

Fauconnier, G. (1985). *Mental Spaces: Aspects of Meaning Construction in Natural Language*. MIT Press, Cambridge, Mass.

Fodor, J. and Pylyshyn, Z. (1988). Connectionism and cognitive architecture: A critical analysis. In Pinker, S. and Mehler, J., editors, *Connections and Symbols*. MIT Press, Cambridge, MA. Reprinted from *Cognition: International Journal of Cognitive Science*, Volume 28 (1988).

Forrest, S., editor (1991). *Emergent Computation*. MIT Press, Cambridge, MA.

French, R. (1992). *Tabletop: An Emergent Stochastic Computer Model of Analogy-making*. PhD thesis, University of Michigan, Ann Arbor, Michigan.

French, R. (1995). *The Subtlety of Sameness: A Theory and Computer Model of Analogy-making*. MIT Press/Bradford Books, Cambridge, MA. In press.

Fromkin, V. (1980). *Errors in Linguistic Performance: Slips of the Tongue, Ear, Pen, and Hand*. Academic Press, New York.

Fukushima, K. (1989). Analysis of the process of visual pattern recognition by the Neocognitron. *Neural Networks*, 2:413–420.

Fukushima, K. (1992). Character recognition with neural networks. *Neurocomputing*, 4:221–233.

Gaillat, G. and Berthod, M. (1979). Panorama des techniques d'extraction de traits caractéristiques en lecture optique des caractères. *Revue Technique Thomson-CSF*, 11(4):943–959.

Gentner, D. (1983). Structure-mapping: A theoretical framework for analogy. *Cognitive Science*, 7(2):155–170.

Geyer, L. (1977). Recognition and confusion of the lowercase alphabet. *Perception & Psychophysics*, 22(5):487–490.

Gibson, E. (1969). *Principles of Perceptual Learning and Development*. Meredith Corporation, New York.

Gibson, E. (1971). Perceptual learning and the theory of word perception. *Cognitive Psychology*, 2:351–368.

Gilmore, G., Hersh, H., Caramazza, A., and Griffin, J. (1979). Multidimensional letter similarity derived from recognition errors. *Perception & Psychophysics*, 25(5):425–431.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Addison-Wesley, Reading, MA.

Goldstone, R., Medin, D., and Gentner, D. (1991). Relations, attributes, and the non-independence of features in similarity judgments. *Cognitive Psychology*, 23(2):222–264.

Grebert, I., Stork, D., Keesing, R., and Mims, S. (1991a). Connectionist generalization for production: An example from GridFont. In *Proceedings of the 1991 International Joint Conference on Neural Networks (IJCNN)*.

Grebert, I., Stork, D., Keesing, R., and Mims, S. (1991b). Connectionist generalization for production: An example from GridFont. Technical Report CRC-TR-91-20, Ricoh California Research Center, Menlo Park, CA.

Grebert, I., Stork, D., Keesing, R., and Mims, S. (1991c). Network generalization for production: Learning and producing styled letterforms. In *Proceedings of the Neural Information Processing Systems Conference (NIPS) 1991*.

Grebert, I., Stork, D., Keesing, R., and Mims, S. (1992). Connectionist generalization for production: An example from GridFont. *Neural Networks*, 5.

Guyon, I., Poujaud, I., Personnaz, L., and Dreyfus, G. (1989). Comparing different neural network architectures for classifying handwritten digits. In *Proceedings of the International Joint Conference on Neural Networks (IJCNN-89)*, pages II, 127–132, Washington, D.C.

Haber, R. and Haber, L. (1981). Visual components of the reading process. *Visible Language*, XV(2):147–182.

Hinton, G. (1990). Mapping part–whole hierarchies into connectionist networks. *Artificial Intelligence*, 46:47–75.

Hinton, G. and Sejnowski, T. (1986). Learning and relearning in Boltzmann machines. In Rumelhart, D. and McClelland, J., editors, *Parallel Distributed Processing, Volume 1*. MIT Press, Cambridge, MA.

Hinton, G., Williams, K., and Revow, M. (1992). Adaptive elastic models for hand-printed character recognition. In Moody, J., Hanson, S., and Lippmann, R., editors, *Advances in Neural Information Processing Systems, Volume 4*, San Mateo, CA. Morgan Kaufmann. Also presented at the Cognitive Science Society Conference 1991, and posted to Neuroprose.

Hock, H., Tromley, C., and Polmann, L. (1988). Perceptual units in the acquisition of visual categories. *Journal of Experimental Psychology: Learning, Memory, and Cognition*, 14:75–84.

Hofstadter, D. (1979). *Gödel, Escher, Bach: an Eternal Golden Braid*. Basic Books, New York.

Hofstadter, D. (1983). The architecture of Jumbo. In *Proceedings of the International Machine Learning Workshop*, pages 161–170, Urbana, IL: University of Illinois. Expanded and reprinted as Chapter 2 of *Fluid Concepts and Creative Analogies*.

Hofstadter, D. (1984). The Copycat project: An experiment in nondeterminism and creative analogies. Technical Report 755, Artificial Intelligence Laboratory, MIT.

Hofstadter, D. (1985). *Metamagical Themas: Questing for the Essence of Mind and Pattern.* Basic Books, New York. See especially Chapters 10, 12, 13, 23, 24, and 26.

Hofstadter, D. (1987). Fluid analogies and human creativity. Technical Report 16, Center for Research on Concepts and Cognition, Indiana University, Bloomington, IN.

Hofstadter, D. (1994). How could a Copycat ever be creative? In Dartnall, T., editor, *Artificial Intelligence and Creativity*, pages 405–424. Kluwer, The Netherlands.

Hofstadter, D. (1995). On seeing a's and seeing as. *Stanford Humanities Review*, 4(2):109–121.

Hofstadter, D. and Dennett, D., editors (1981). *The Mind's I: Fantasies and Reflections on Self and Soul.* Bantam, New York.

Hofstadter, D. and FARG (1995). *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought.* Basic Books, New York. "FARG" is an acronym for the "Fluid Analogies Research Group".

Hofstadter, D. and McGraw, G. (1995). Letter Spirit: Esthetic perception and creative play in the rich microcosm of the alphabet. In Hofstadter, D. and FARG, editors, *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*, pages 407–466. Basic Books, New York.

Hofstadter, D., Mitchell, M., and French, R. (1987). Fluid concepts and creative analogies: A theory and its computer implementation. Technical Report 18, Center for Research on Concepts and Cognition, Indiana University, Bloomington, IN.

Holland, J. (1975). *Adaption in Natural and Artificial Systems.* University of Michigan Press, Ann Arbor, MI. Reissued by MIT Press, 1992.

Holland, J., Holyoak, K., Nisbett, R., and Thagard, P. (1986). *Induction: Processes of Inference, Learning, and Discovery.* MIT Press, Cambridge, Mass.

Holland, J. H. (1986). Escaping brittleness: The possibilities of general-purpose learning algorithms applied to parallel rule-based systems. In Michalski, R., Carbonell, J., and Mitchell, T., editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, pages 593–623. Morgan Kaufmann.

Indurkhya, B. (1992). *Metaphor and Cognition: An Interactionist Approach.* Kluwer, Norwell, MA.

Jackendoff, R. (1995). *Patterns in the Mind: Language and Human Nature.* Basic Books, New York.

Jacobs, A., Nazir, T., and Heller, O. (1989). Perception of lowercase letters in peripheral vision: A discrimination matrix based on saccade latencies. *Perception & Psychophysics*, 46(1):95–102.

Johnson-Laird, P. (1988). Freedom and constraint in creativity. In Sternberg, R., editor, *The Nature of Creativity*, pages 202–219. Cambridge University Press, New York.

Johnson-Laird, P. (1989). Jazz improvisation: A theory at the computational level. Technical report, MRC Applied Psychology Unit, Cambridge, UK.

Kahan, S., Pavlidis, T., and Baird, H. (1987). On the recognition of printed characters of any font and size. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-9(2):274–288.

Kahneman, D. and Miller, D. (1986). Norm theory: Comparing reality to its alternatives. *Psychological Review*, 93:136–153.

Keren, G. and Baggen, S. (1981). Recognition models of alphanumeric characters. *Perception & Psychophysics*, 29(3):234–246.

Kirkpatrick, S., C. Gelatt, J., and Vecchi, M. (1983). Optimization by simulated annealing. *Science*, 220:671–680.

Knuth, D. (1982). The concept of a meta-font. *Visible Language*, XVI(1):3–27.

Knuth, D. (1986). *The METAFONTbook.* Addison-Wesley, Reading, Mass.

Kolodner, J. (1993). *Case-Based Reasoning.* Morgan Kaufmann, San Mateo, CA.

Kuhn, T. (1970). *The Structure of Conceptual Revolutions.* University of Chicago Press, Chicago. Second edition.

Kurzweil, R. (1990). *The Age of Intelligent Machines.* MIT Press, Cambridge. MA.

Lam, L. and Suen, C. (1988). Structural classification and relaxation matching of totally unconstrained handwritten zip-code numbers. *Pattern Recognition*, 21(1):19–31.

Langley, P., Simon, H., Bradshaw, G., and Zytkow, J. (1987). *Scientific Discovery: Computational Explorations of the Creative Process.* MIT Press, Cambridge, Mass.

Le Cun, Y., Boser, B., Denker, J., Henderson, D., Howard, R., Hubbard, W., and Jackel, L. (1990). Handwritten digit recognition with a back-propagation network. In Touretsky, D., editor, *Advances in Neural Information Processing Systems, Volume 2*, pages 396–404, San Mateo, CA. Morgan Kaufmann.

Lee, M. and Oldham, W. (1990). Font recognition by a neural network. *International Journal of Man–Machine Studies*, 33:41–61.

Lee, Y. (1991). Handwritten digit recognition using k-nearest-neighbor, radial-basis function, and backpropagation neural networks. *Neural Computation*, 3:440–449.

Lenat, D. (1982). AM: Discovery in mathematics as heuristic search. In Davis, R. and Lenat, D., editors, *Knowledge-Based Systems in Artificial Intelligence*, pages 1–225. McGraw-Hill, New York.

Lenat, D. (1983a). Eurisko: A program that learns new heuristics and domain concepts. *Artificial Intelligence*, 21(1,2):61–98.

Lenat, D. (1983b). Theory formation by heuristic search. *Artificial Intelligence*, 21(1,2):31–59.

Mantas, J. (1986). An overview of character recognition methodologies. *Pattern Recognition*, 19(6):425–430.

Mantas, J. (1987). Methodologies in pattern recognition and image analysis — a brief survey. *Pattern Recognition*, 20(1):1–6.

Marshall, J. (1993). A self-watching cognitive architecture for high-level perception and analogy-making. Technical Report 100, Center for Research on Concepts and Cognition, Indiana University, Bloomington, IN.

McClelland, J. and Rumelhart, D. (1981). An interactive-activation model of context effects in letter perception: Part 1, An account of basic findings. *Psychological Review*, 88(5):375–407.

McClelland, J. and Rumelhart, D., editors (1986). *Parallel Distributed Processing: Volumes I and II.* MIT Press, Cambridge, MA.

McCorduck, P. (1991). *Aaron's Code: Meta-art, Artificial Intellgence, and the Work of Harold Cohen.* Freeman, New York.

McDermott, D. (1981). Artificial Intelligence meets natural stupidity. In Haugeland, J., editor, *Mind Design*. MIT Press, Cambridge, Mass. First appeared in *SIGART Newsletter*, No. 57, April 1976.

McGraw, G. (1992). Letter Spirit: Recognition and creation of letterforms based on fluid concepts. Technical Report 61, Center for Research on Concepts and Cognition, 510 North Fess, Bloomington, IN, 47408.

McGraw, G. (1995). Emergent perception of gridletters. In Evens, M., editor, *Proceedings of the 1995 Midwest AI and Cognitive Science Society Conference*, pages 63–67, Huntsville, AL. Intergraph Corporation.

McGraw, G. and Drasin, D. (1993). Recognition of gridletters: Probing the behavior of three competing models. In *Proceedings of the Fifth Midwest AI and Cognitive Science Society Conference*, pages 63–67.

McGraw, G. and Hofstadter, D. R. (1993a). Perception and creation of alphabetic style. Technical Report SS-93-01, American Association of Artificial Intelligence. In *Artificial Intelligence and Creativity: Papers from the 1993 AAAI Spring Symposium*.

McGraw, G. and Hofstadter, D. R. (1993b). Perception and creation of diverse alphabetic style. *Artificial Intelligence and Simulation of Behaviour Quarterly*, 85:42–49.

McGraw, G., Rehling, J., and Goldstone, R. (1994a). Letter perception: Toward a conceptual approach. In *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*, pages 613–618, Atlanta, GA.

McGraw, G., Rehling, J., and Goldstone, R. (1994b). Roles in letter perception: Human data and computer models. Technical Report 90, Center for Research on Concepts and Cognition, 510 North Fess, Bloomington, IN, 47405.

Medin, D., Goldstone, R., and Gentner, D. (1993). Respect for similarity. *Psychological Review*, 100:254–278.

Meeden, L. (1994). *Towards Planning: Incremental Investigations into Adaptive Robot Control*. PhD thesis, Indiana University, Department of Computer Science, Bloomington, IN.

Meehan, J. (1976). *The Metanovel: Writing Stories by Computer*. PhD thesis, Yale University, New Haven, Connecticut.

Meehan, J. (1978). TALE-SPIN, An interactive program that writes stories. *Natural Language*, 5:91–98.

Meredith, M. (1986). *Seek-Whence: A Model of Pattern Perception.* PhD thesis, Indiana University, Bloomington, IN. Also available as Technical Report No. 214, Computer Science Department, Indiana University.

Meredith, M. (1991). Data modeling: A process for pattern induction. *Journal of Experimental and Theoretical Artificial Intelligence (JETAI)*, 3:43–68.

Minsky, M. (1985). *The Society of Mind.* Simon and Schuster, New York.

Minsky, M. and Papert, S. (1969). *Perceptrons: An Introduction to Computational Geometry.* MIT Press, Cambridge, MA. Third printing, 1988.

Mitchell, M. (1990). *Copycat: A Computer Model of High-level Perception and Conceptual Slippage in Analogy-making.* PhD thesis, University of Michigan, Ann Arbor, Michigan.

Mitchell, M. (1993). *Analogy-making as Perception.* MIT Press/Bradford Books, Cambridge, MA.

Mori, S., Yamamoto, K., and Yasuda, M. (1984). Research on machine recognition of handprinted characters. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-6(4):386–405.

Nanard, M., Nanard, J., Gandara, M., and Porte, N. (1989). A declarative approach for font design by incremental learning. In Andre, J. and Hersch, R., editors, *Raster Imaging and Digital Typography*, pages 71–82. Cambridge University Press, Cambridge.

Naus, M. and Shillman, R. (1976). Why a Y is not a V: A new look at the distinctive features of letters. *Journal of Experimental Psychology: Human Perception and Performance*, 2(3):394–400.

Neisser, U. (1967). *Cognitive Psychology.* Appleton-Century-Crofts, New York.

Newell, A. and Simon, H. (1963). GPS, A program that simulates human thought. In Feigenbaum, E. and Feldman, J., editors, *Computers and Thought.* McGraw-Hill, New York.

Newell, A. and Simon, H. (1981). Computer science as empirical enquiry: Symbols and search. In Haugeland, J., editor, *Mind Design.* MIT Press, Cambridge, Mass. First appeared in *CACM*, No. 19, March 1976.

Norman, D. (1981). Categorization of action slips. *Psychological Review*, 88(1):1–15.

Norman, D. (1988). *The Psychology of Everyday Things*. Basic Books, New York.

Palmer, S. (1977). Hierarchical structure in perceptual representation. *Cognitive Psychology*, 9:441–474.

Palmer, S. (1978). Structural aspects of visual similarity. *Memory & Cognition*, 6(2):91–97.

Pollack, J. (1988). Recursive auto-associative memory: Devising compositional distributed representations. In *Proceedings of the Tenth Annual Conference of the Cognitive Science Society*, pages 33–39. Lawrence Erlbaum Associates.

Pollack, J. (1989). Implications of recursive distributed representations. In Touretzky, D., editor, *Advances in Neural Information Processing Systems I*, pages 527–536. Morgan Kaufmann.

RACTER, Chamberlain, W., and Etter, T. (1984). *The Policeman's Beard is Half-Constructed: Computer Prose and Poetry*. Warner Software/Books, New York.

Regier, T. (1991). Learning spatial concepts using a partially-structured connectionist architecture. Technical Report TR-91-050, International Computer Science Institute, Berkeley, California.

Regier, T. (1992). *The Acquistion of Lexical Semantics for Spatial Terms: A Connectionist Model of Perceptual Categorization*. PhD thesis, International Computer Science Institute, Berkeley, CA. Available as technical report TR-92-062.

Rich, E. (1983). *Artificial Intelligence*. McGraw-Hill, New York.

Ritchie, G. and Hanna, F. (1990). AM: A case study in AI methodology. In Partridge and Wilks, editors, *The Foundations of AI: A Sourcebook*. Cambridge University Press, New York.

Rosch, E. and Lloyd, B. (1978). *Cognition and Categorization*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Rowe, J. (1991). *Emergent Creativity: A Computational Study*. PhD thesis, Computer Science Department, University of Exeter, UK.

Rowe, J. and Partridge, D. (1991a). Creativity: A survey of AI approaches. Technical Report R 214, Department of Computer Science, University of Exeter, UK.

Rowe, J. and Partridge, D. (1991b). Creativity: Learning flexible representations through emergent memory. Technical Report R 215, Department of Computer Science, University of Exeter, UK.

Ruggles, L. (1983). Letterform design systems. Technical Report STAN-CS-83-971, Computer Science Department, Stanford University, Palo Alto, California.

Sanocki, T. (1986). *Visual Knowledge Underlying Letter Perception: A Font-Specific Approach*. PhD thesis, University of Wisconsin, Department of Psychology, Madison, Wisconsin.

Sanocki, T. (1987). Visual knowledge underlying letter perception: Font-specific, schematic tuning. *Journal of Experimental Psychology*, 13(2):267–278.

Sanocki, T. (1991). Effects of early common features on form perception. *Perception & Psychophysics*, 50(5):490–497.

Sanocki, T. and Oden, G. (1991). Perceptual adjustments on representations of familiar patterns: Change over time and relational features. *Perception & Psychophysics*, 50:28–44.

Schank and Riesbeck, C. (1981). *Inside Computer Understanding*. Lawrence Erlbaum Associates, Hillsdale, NJ.

Schyns, P. and Murphy, G. (1995). The ontogeny of part representation in object concepts. In Medin, D., editor, *The Psychology of Learning and Motivation*. Academic Press. In press.

Searle, J. (1980). Minds, brains, and programs. *The Behavioral and Brain Sciences*, 3:417–57.

Shiffrin, R. and Schneider, W. (1977). Controlled and automatic human information processing: II. Perceptual learning, automatic attending, and a general theory. *Psychological Review*, 84:127–190.

Shillman, R., Kuklinski, T., and Blesser, B. (1976). Psychophysical techniques for investigating the distinctive features of letters. *International Journal of Man–Machine Studies*, 8(2):195–205.

Smith, E. and Medin, D. (1981). *Categories and Concepts*. Harvard University Press, Boston.

Smolensky, P. (1988). On the proper treatment of connectionism. *Behavioural and Brain Sciences*, 11:1–74.

Smolensky, P. (1990). Tensor product variable binding and the representation of symbolic structures in connectionist systems. *Artificial Intelligence*, 46:159–216.

Suen, C., Nadal, C., Legault, R., Mai, T., and Lam, L. (1992). Computer recognition of unconstrained handwritten numerals. *Proceedings of the IEEE*, 68(4):469–487.

Takano, Y. (1989). Perception of rotated forms: A theory of information types. *Cognitive Psychology*, 21:1–59.

Tappert, C., Suen, C., and Wakahara, T. (1990). The state of the art in on-line handwriting recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(8):787–808.

Townsend, J. (1971). Alphabetic confusion: A test of models for individuals. *Perception & Psychophysics*, 9(6):449–454.

Townsend, J. and Ashby, F. (1982). Experimental test of contemporary mathematical models of visual letter recognition. *Journal of Experimental Psychology*, 8:834–864.

Treisman, A. (1986). Properties, parts, and objects. In Boff, K., Kaufman, L., and Thomas, J., editors, *Handbook of Perception and Human Performance, Volume II: Cognitive Processes and Performance*. John Wiley and Sons, New York.

Treisman, A. and Gelade, G. (1980). A feature-integration theory of attention. *Cognitive Psychology*, 12(12):97–136.

Treisman, A. and Paterson, R. (1984). Emergent features, attention, and object perception. *Journal of Experimental Psychology: Human Perception and Performance*, 10:12–31.

Turing, A. (1950). Computing machinery and intelligence. *Mind*, 52(236):433–460.

Turner, S. (1991). A case-based model of creativity. In *Proceedings of the Thirteenth Annual Conference of the Cognitive Science Society*, pages 933–937.

Turner, S. (1992). *MINSTREL: A Computer Model of Creativity and Storytelling.* PhD thesis, UCLA, Los Angeles, California.

Turner, S. (1994). *The Creative Process: A Computer Model of Storytelling.* Lawrence Erlbaum Associates, Hillsdale, NJ.

Tversky, A. (1977). Features of similarity. *Psychological Review*, 84:372–352.

Tversky, A. and Hutchinson, J. (1986). Nearest neighbor analysis of psychological spaces. *Psychological Review*, 93(1):3–22.

Tversky, B. and Hemenway, K. (1984). Objects, parts, and categories. *Journal of Experimental Psychology: General*, 113(2):169–193.

Vickery, R. (1983). *Sharing Architecture*. University of Virginia Press, Charlottesville, VA.

Weizenbaum, J. (1976). *Computer Power and Human Reason: From Judgment to Calculation*. Academic Press, New York.

Zuckerman, J. (1992). *The Scheme/X/Motif Manual*. Motorola, Inc.