Tabletop: An Emergent, Stochastic Model of Analogy-Making

Robert M. French and Douglas R. Hofstadter
Center for Research on Concepts and Cognition
Indiana University
510 North Fess
Bloomington, Indiana 47408
e-mail: french@cogsci.indiana.edu, dughof@cogsci.indiana.edu

Abstract

This paper describes Tabletop, a computer progam that models human analogy-making in a micro-world consisting of a small table covered with ordinary table objects. We argue for the necessity, even in this simple domain, of an architecture that builds its own representations by means of a continual interaction between an associative network of fixed concepts (the *Slipnet*) and simple low-level perceptual agents (*codelets*), that relies on local processing and (simulated) parallelism, and that is fundamentally stochastic. Several problems solved by the Tabletop program are used to illustrate these principles.

Introduction

Tabletop, the program described in this paper, is a computer model of analogy-making. Unlike many analogy programs, it does not attempt to discover analogies between political situations, concepts in science, or plots in literature. Rather, it operates in a microdomain, the Tableworld, consisting of ordinary table objects on an ordinary table. Imagine there are two people, Henry and Eliza, seated at a table facing each other. They play the following game: Henry touches some object on the table and says to Eliza, "Do this!" Eliza must respond by doing "the same thing". In other words, she must find the object that, from her vantage point, seems to correspond most closely to the object that Henry touched. In a sense, this is analogy-making at its most prosaic. And yet, the psychological mechanisms that underlie human analogy-making in this microdomain are, we believe, of the same kind and complexity as in any real-world situation.

Competing pressures

A human Eliza intuitively takes into account a number of factors when determining her choice. These factors include: the positions of the objects on the table, the category to which they belong, their size, and their functional association with other objects (for example, cups and saucers are related in a way that cups and plates are not). Each of these factors exerts some pressure on her decision. Some pressures, such as the ones just mentioned (except for position), are relatively context-independent. Other pressures are evoked by the situation itself and cannot reasonably be anticipated either by a human Eliza or by a program simulating Eliza. Rather, they emerge gradually as the person or program develops a representation of the configuration of objects on the table.

Consider the three table configurations in Figure 1. In the simplest situation (Figure 1*a*), the two competing pressures are category membership and position on the table. In this situation, the chances are good that Eliza would touch the cup in the diagonally opposite corner of the table. But when both the touched cup and the glass directly opposite it are surrounded by several objects, this changes the pressures (Figure 1*b*). The similar groups of objects around the touched cup and the glass generate contextual pressures that make it considerably more likely that Eliza will touch the glass instead of the cup. This probability is strongly enhanced by the fact that a glass is conceptually similar to a cup. Now what if we modified these pressures further, by replacing the glass by a saucer? Would this shift the balance of pressures to favor the isolated cup? Probably, but not necessarily; this is because even though cups and saucers, strictly speaking, do not belong to the same category, they are conceptually related as "objects that are used together", in much the same way that knives and forks are normally thought of as being used together. Finally, what if the saucer were replaced by a salt shaker (Figure 1*c*)? This time the pressures seem sufficiently in favor of the isolated cup that Eliza would probably touch it.

Tabletop is an attempt to model the way humans respond to these (and other) pressures in the Tableworld. We have argued in detail elsewhere [Chalmers, French, & Hofstadter 1991] that a satisfactory computer model of analogy-making must be able to build its own representation of the situation that it is faced with. We have also argued against the possibility of a separate "representation module" that would act as a preprocessor to an "analogy-making module". In accordance with these precepts, the processes of representation-building

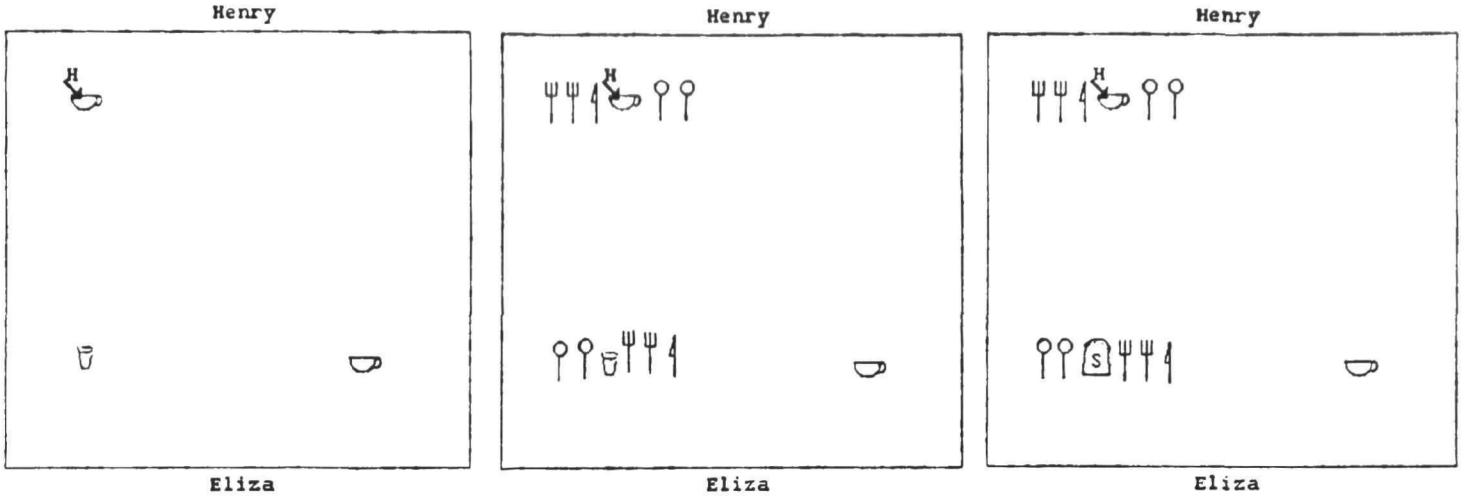

Figure 1*a* Figure 1*b* Figure 1*c*
Various competing pressures in three table configurations

and correspondence-finding are completely integrated in Tabletop. This represents a significant departure from current analogy-making programs [e.g., Burstein & Adelson 1987; Kedar-Cabelli 1988; Falkenhainer et al. 1989; Thagard et al. 1991] with the exception of Copycat [Mitchell 1990; Hofstadter & Mitchell 1991].

We will begin our description of Tabletop with a brief discussion of its architecture. We then explain why we believe this type of architecture to be essential to models of analogy-making. We conclude with several runs of the program to illustrate how it develops representations of situations and uses them to make analogies. We argue that preprogramming explicit mechanisms to handle the kinds of difficulties posed by these situations is psychologically implausible. Rather, implicit pressures should emerge that accomplish the same things as explicit mechanisms. Our architecture allows this to happen.

High-level perception and the architecture of Tabletop

We define high-level perception as that stage of perceptual processing where concepts begin to play a significant role. High-level perception ranges from our ability to recognize objects to our capacity to grasp grasp relationships among objects and, ultimately, includes our ability to understand entire complex situations, such as a love affair or a war [Hofstadter 1985; Chalmers, French, & Hofstadter 1991]. Tabletop simulates this range of high-level perception in a number of ways, including how it scans the table looking for objects and groups of objects, how it discovers relationships among objects and, finally, how it gradually builds up a coherent view of a complete situation.

Three central features of the Tabletop architecture provide the basis for its high-level perceptual abilities. Before illustrating several runs of the program we will briefly focus our discussion on these features, which are:

- continual interaction between the associative concept network (the Slipnet)and many low-level perceptual agents (codelets) whose job it is to examine and structure the Tableworld;
- local processing and (simulated) parallelism;
- stochastic mechanisms.

There are other features — most significantly, computational temperature mechanisms [Mitchell & Hofstadter 1989] — that distinguish Tabletop from other models of analogy-making. However, in this paper we will restrict ourselves to a discussion of the three features above.

Interaction between the Slipnet and codelets

Tabletop, like Copycat [Hofstadter 1984; Mitchell 1990; Hofstadter & Mitchell 1991] before it, has an associative concept network, called the *Slipnet*, and a *Workspace* filled with simple perceptual agents, called codelets, responsible for observing the Tableworld and building the structures needed to understand it. The division of the program into these two aspects reflects the traditional philosophical distinction between types and tokens: the Slipnet contains Platonic concepts, while the Workspace contains instances of those concepts, which codelets observe and manipulate. Examples of codelets include agents that look for groups of objects on the table; agents that look for neighbors of a particular object; agents that, given

a particular group of objects, look for the same type of group elsewhere on the table; and so on. These perceptual agents are low-level observers and builders in the sense that they do not have a global view of the table at any time.

The Slipnet and the Workspace continually interact with one another. The Slipnet is reminiscent of a typical spreading-activation concept network [Collins & Loftus 1975] with a number of differences, possibly the most significant being that the distances between concepts in the Slipnet vary according to pressures evoked by the situation (and conveyed to the Slipnet by codelets). In a reciprocal manner, the numbers and types of active perceptual agents are governed by the activations of concepts in the Slipnet.

Local processing and (simulated) parallelism

At no time does any codelet have a global view of the table. At the start of a run, the program knows only which object was touched by Henry and where on the table that object is located. The initial codelets that run may be thought of as scouts examining the table for information (e.g., finding an object in a particular location, discovering the neighbor of a particular object, etc.) and looking for various structures (e.g., groups of objects) that might be useful to its subsequent understanding of the Tableworld. Depending on their relation to the touched object, certain areas of the table are given preference and will generally be examined first. This can be thought of as a visual scan of the table, a scan with top-down control, one in which the salience of objects preferentially determines which objects will be attended to first, and, significantly, one in which some objects, or even groups of objects, might sometimes be altogether ignored.

Consider, for example, *group-finding* scout codelets. Suppose that the object touched is in the left-hand corner of the far side of the table: then these codelets will preferentially (but not deterministically) look in the corner of the table diagonally opposite the touched object to determine if there are any groups of objects there. This bias is built into the program deliberately in order to imitate typical human behavior.

It is important to note that group-finding codelets, like all codelets, neither have an overview of the table that would allow them to immediately spot all of the groups of objects, nor do they conduct a systematic search for groups, say from the upper left to the lower right-hand corner of the table. Instead, they search the table probabilistically. Codelets are biased to look at different parts of the table with different probabilities. Factors influencing the probability of an area being looked at include the presence of objects known to be conceptually similar to the touched object and being diagonally or directly opposite the touched object.

One might object that such a probabilistic search technique might allow the program to overlook some structure that could be useful later. This is true, but there is a trade-off — namely, it allows the program to focus its resources on avenues that seem most promising. The ultimate reason for this strategy is to make the program psychologically realistic; in particular, its mechanisms should remain valid in scaled-up situations. True real-world situations are generally far more complex than these simple caricatural situations, and brute-force search techniques are entirely inappropriate. There have to be strong pressures that serve to focus resources on promising avenues. Any probabilistic search technique is a risk-taking strategy and as such, sacrifices guaranteed success for greater efficiency.

Active concepts in the Slipnet dispatch agents to look for instances of themselves. For example, if the "group" concept is active, codelets will be dispatched that search for groups in the Tableworld. If one of these codelets succeeds in finding a group, this causes more activation to be sent to the concept "group" in the Slipnet. This amounts to a signal to the system to hunt for even more groups; thus more codelets are sent out to look for groups. However, as the rate of group-finding falls (i.e., the rate of change of activation of "group" decreases), proportionately fewer codelets are sent out to look for groups. Eventually, the program concludes that there are probably no more groups to be found on the table.

This process is rather like a person who wants to pick all of the apples from a tree. Initially, it is easy to see the apples. She will look first in the most salient places on the tree (e.g., on the outer, leafy branches). The sight of apples remaining in the tree will encourage her to look for more. As the number of apples in the tree gets smaller, she will start to look in less likely places (e.g., on the lower branches, closer to the main trunk). Gradually, as fewer and fewer apples remain, it becomes harder to determine if there are any left at all. Ultimately, there will come a point when she will peer into the tree several times and decide that she has gotten all the apples. But did she *really* get them all? Probably, but not necessarily. Ultimately, though, the energy (to say nothing of the eye-strain) involved in attempting to find any additional apples becomes prohibitive, and she stops looking.

The group-finding codelets find groups on the table in a similar way. It is true that some groups may be missed, but, on the other hand, the program does not expend needless energy covering

every last square centimeter of the table to make sure that every single group is found, even those that have very little chance of playing any role in a potential analogy. In such a local approach, as more group-finding codelets fail to find groups, fewer of these codelets run, until finally none run at all. The program has then "decided" — without any given codelet seeing the whole table, any more than the person could take in the entire apple tree at one time — that it has found all of the groups on the table, or at least all groups worth looking for.

This might seem somewhat strange at first blush. Why not allow the program to search the table *systematically* to make sure that it has gotten *all* instances of groups of objects? The answer is that such a brute-force technique would lead to unacceptable and psychologically unrealistic performance in scaled-up situations. Tabletop's strategy of having a multitude of small processes running in parallel provides an efficient focusing mechanism that allows the program's attention to be probabilistically shifted to promising areas of exploration.

A fundamental assumption behind Tabletop is that high-level cognitive processes are the emergent result of myriad low-level, quasi-independent processes. It is our belief that it is the interaction of these low-level processes that give cognition its flexibility. It is therefore important to attempt, insofar as possible, to model analogy-making by relying on low-level processes to gradually build up the high-level structures necessary to solve an analogy problem.

In Figure 2, we give some examples of Tabletop configurations where a particular structure — in this case, a diagonal correspondence — unexpectedly blocks a very reasonable answer. There are, however, no special "blockage-checking mechanisms" built into Tabletop to deal with such situations. It would be unreasonable to have a mechanism that invariably looked for blockages of this type. This illustrates a general principle about Tabletop — namely, the lack of special-purpose mechanisms. Building in a raft of diverse special-purpose mechanisms all of which invariably get "wheeled out" for every situation not only is psychologically implausible but also poses severe problems of scaling-up. Rather than routinely invoking special-purpose mechanisms, Tabletop evokes context-dependent pressures that accomplish the same results but without brute-force techniques.

Justification for stochastic mechanisms

One of the most surprising features of the Tabletop architecture is that, at all levels, almost all of the processes are stochastic. Hofstadter and Mitchell have argued for the necessity of stochastic mechanisms [Hofstadter & Mitchell 1991] in emergent architectures modeling cognitive processes. Two significant advantages of stochastic mechanisms for exploration and structure-building are the following:

- they allow, on average, promising avenues to be explored before less promising ones;
- they also, on occasion, allow improbable paths to be explored, whereas in normal heuristic search, these paths might never be chosen at all.

In the language of traditional artificial intelligence, the stochastic mechanisms in Tabletop are reminiscent of probabilistic heuristic search. In deterministic heuristic search, if the program must chose between action A, which will give an estimated result of 60, and action B, which will give 40, the program will invariably select action A. In probabilistic heuristic search, by contrast, A would be selected 60 percent of the time, B 40 percent.

Why is this sensible? Shouldn't the program *always* explore the most promising paths first? The answer is: no, sometimes it should not. Tabletop problems in particular, and analogy problems in general, have the property that often several solutions of different levels of quality exist. Pathways that don't look promising at the outset may conceal very high-quality answers, and conversely, pathways that look very promising at the outset may lead to mediocre answers. A strategy that always pursued the most promising avenues until it found a solution would therefore miss any hidden high-quality solutions (if they existed). On the other hand, it would not be wise to totally ignore the estimated promise of pathways. Standard AI strategies would deterministically explore a vast number of possible pathways, leading to many, if not all, answers, and then would select the best answer. This type of strategy is appealing; however, once again, it poses severe scaling-up problems in real-world situations. Tabletop's strategy is poised between these two deterministic types of strategy — it will *sometimes* choose less promising routes, but the less promising the route, the less often it will be chosen. This probabilistic technique leaves all routes at least theoretically open and so, on some runs of the program, the hidden gems (if they exist) will be uncovered, although on most runs the most obvious solutions will be chosen. This non-deterministic strategyavoids any combinatorial explosion while at the same time avoiding the trap of always following the most obvious pathways. Note that in the microdomain we could have opted for a brute-force solution; however, since our research aims for psychological realism, we eschewed this technique.

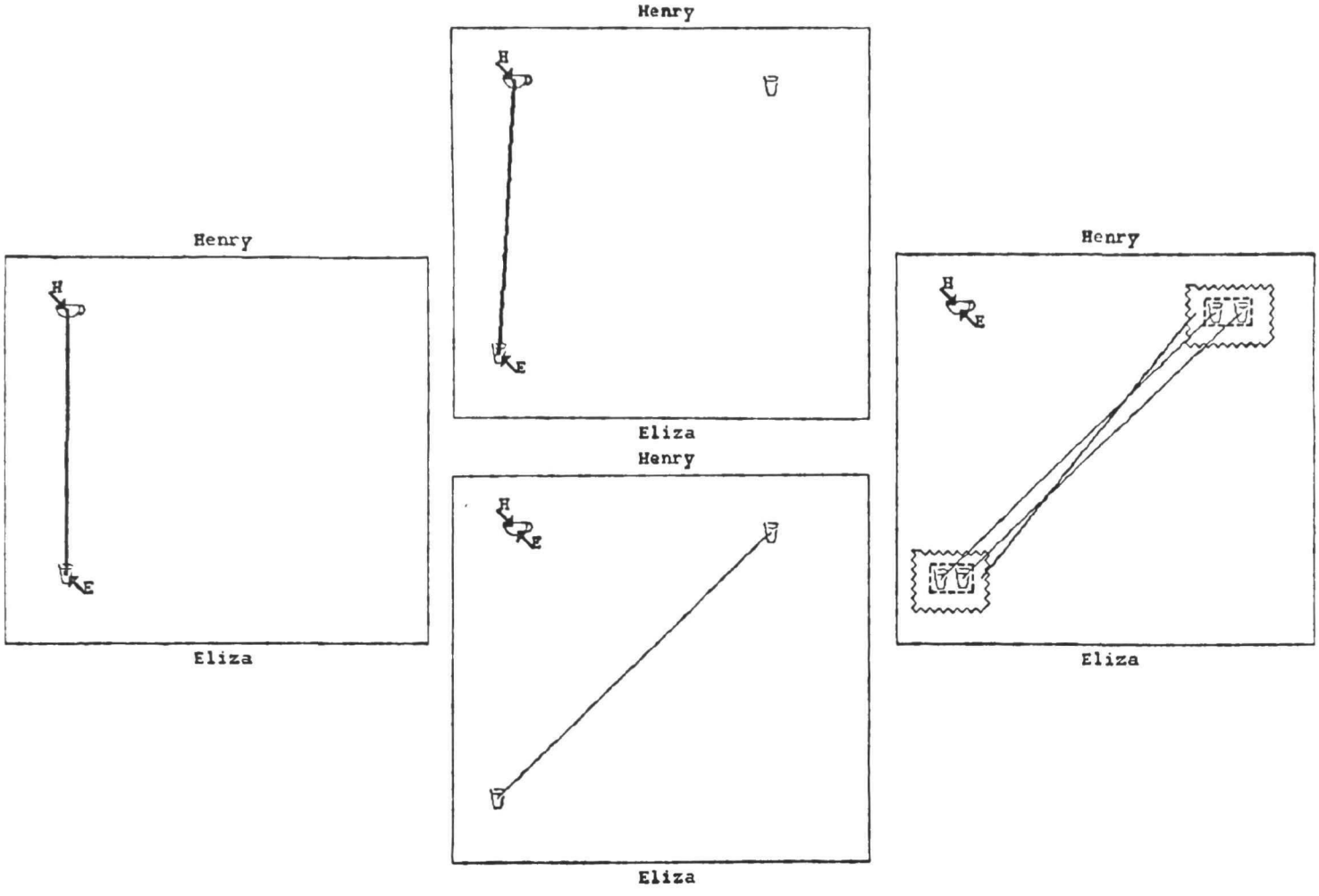

Figure 2*a* Figure 2*b*, parts *i* and *ii* Figure 2*c*
Three Tabletop problems (note the two solutions to the second problem)

Three analogy problems illustrating these mechanisms

In the second section of this paper we discussed some of the competing pressures in the Tabletop world that are involved in deciding which table objects and relationships among objects are examined and which structures are built in order to arrive at an answer to the "Do this!" challenge. These pressures include object category; whether or not an object is in a group; what the neighbors of an object are; whether or not the object is on the edge of a group; whether there are other similar or identical objects elsewhere on the table, especially if they are in salient positions with respect to the touched object (in particular, diagonally or directly opposite it); whether an object or group of objects is close to a particularly salientobject or group of objects; and so on. These groupings, descriptions, and relations are discovered only after the program has begun scanning the table. Descriptions (neighbors, in a group or not, etc.) are gradually attached to the various objects, groups of objects, and correspondences.

In order for the program to give an answer, it needs to have built up one or more mappings between objects or groups of objects across the table (called *correspondences*). However, the erection of such structures is not separated in time from the build-up of other representations (descriptions, groupings, etc.); rather, it is part and parcel of a single high-level perceptual process, in which correspondences can become elements of descriptions of objects and, conversely, descriptions become the underpinning of new correspondences. This fusion of representation-building with correspondence-building marks a fundamental difference with many other current analogy-making programs.

In Figure 2 we illustrate three separate Tabletop problems. In all three problems, Tabletop is capable of producing more than one distinct answer, but in the first and third problems,

there is one answer that swamps the others in frequency, so it is almost as if there is just one answer. However, in the second problem, there are two distinct answers both of which are produced with fairly high frequency. These three problems illustrate how an unanticipated pressure can emerge that will significantly bias the program's responses.

In the first problem, the program, like people, will usually touch the glass opposite the cup that Henry touched. This is quite straightforward.

In the second problem, a single glass is added in the upper right-hand corner of the table. Even though one would not *a priori* expect it to affect the description of any of the objects (e.g., it is not the neighbor of any object; it is not part of a group of objects; it is not in a particularly salient location on the table with respect to the touched object, etc.), its presence nonetheless significantly biases the choice of the second object to be touched. As the program runs, it sometimes builds a correspondence between the two glasses (Figure 2*b*, part *ii*). When it does so, the new pressure that this creates will often be enough to cause Eliza to touch the very cup that Henry touched. Of course, had the second glass not been on the table, such a choice would seem rather strange.

In the final example, there are two glasses in the lower left-hand corner and two in the upper right-hand corner of the table. The correspondence between these two *groups* of glasses is very salient and very strong, usually strong enough to cause the program to touch the cup that Henry touched.

Thirty-one human subjects were given these three problems and, like the program, in the second configuration, more subjects touched the cup in the upper left-hand corner than in the first configuration (40% vs. 33%); in the final configuration, twice as many touched the touched cup (66% vs. 33%).

Conclusion

We have described a microdomain for computer analogy-making consisting of ordinary objects on an ordinary table. This world has a decidedly more "real-world" feel to it than the only other program of its kind, Copycat [Hofstadter & Mitchell 1991; Mitchell 1990]. We argue that it is necessary, even in this simple domain, to use a model that builds its own representations by means of a continual interaction between an associative network of fixed concepts and simple low-level perceptual agents, that relies on local processing and (simulated) parallelism, and that is fundamentally stochastic.

Acknowledgments

Thanks to David Chalmers for his helpful comments on this paper.

Bibliography

Anderson, J. R. (1983), *The architecture of cognition.* Cambridge, MA: Harvard University Press.

Burstein, M. & Adelson, B. (1987). Mapping and integrating partial mental models. In *Proceedings of the ninth annual conference of the Cognitive Science Society.* Hillsdale, NJ: Lawrence Erlbaum Associates.

Chalmers, D. J., French, R. M., & Hofstadter, D. R. (1991). *High-level perception, representation, and analogy: A critique of artificial intelligence methodology.* CRCC Technical Report 49-1990, Center for Research on Concepts and Cognition, Bloomington, Indiana. Also submitted to *Journal of Experimental and Theoretical Artificial Intelligence.*

Collins, A. M. & Loftus, E. F. (1975). A spreading activation theory of semantic memory. *Psychological Review,* **82** 407–428.

Falkenhainer. B., Forbus, K. D., & Gentner, D. (1989). The structure-mapping engine. *Artificial Intelligence,* **41** (1), 1–63.

Kedar-Cabelli, S. (1988). Towards a computational model of purpose-directed analogy. In A. Prieditis (ed.), *Analogica.* Los Altos, CA: Morgan Kaufmann.

Hofstadter, D. R. (1984). *The Copycat project: An experiment in nondeterminism and creative analogies.* AI Memo No. 755, Massachusetts Institute of Technology, Cambridge, MA.

Hofstadter, D. R. (1985). Analogies and roles in human and machine thinking. In *Metamagical themas,* 547–603. New York: Basic Books.

Hofstadter, D. R. & Mitchell, M. (1991). *An overview of the Copycat project.* CRCC Technical Report 51-1991, Center for Research on Concepts and Cognition, Bloomington, Indiana.

Mitchell, M. (1990). *Copycat: A computer model of high-level perception and conceptual slippage in analogy-making.* Ph.D. dissertation, University of Michigan, Ann Arbor, Michigan.

Thagard, P., Holyoak, K. J., Nelson, G., & Gochfeld, D. (1991) (in press). Analog retrieval by constraint satisfaction. *Artificial Intelligence.*