Chapter 5

The Copycat Project: A Model of Mental Fluidity and Analogy-making

DOUGLAS HOFSTADTER and MELANIE MITCHELL

Copycat and Mental Fluidity

Copycat is a computer program designed to be able to discover insightful analogies, and to do so in a psychologically realistic way. Copycat's architecture is neither symbolic nor connectionist, nor was it intended to be a hybrid of the two (although some might see it that way); rather, the program has a novel type of architecture situated somewhere in between these extremes. It is an *emergent* architecture, in the sense that the program's top-level behavior emerges as a statistical consequence of myriad small computational actions, and the concepts that it uses in creating analogies can be considered to be a realization of "statistically emergent active symbols" (Chapter 26 of Hofstadter, 1985). The use of parallel, stochastic processing mechanisms and the implementation of concepts as distributed and probabilistic entities in a network make Copycat somewhat similar in spirit to certain connectionist systems. However, as will be seen, there are important differences, and we claim that the middle ground in cognitive modeling occupied by Copycat is at present the most useful level at which to attempt to understand the fluidity of concepts and perception that is so clearly apparent in human analogy-making.

Analogy problems in the Copycat domain

The domain in which Copycat discovers analogies is very small but surprisingly subtle. Not to beat around the bush for a moment, here is an example of a typical, rather simple analogy problem in the domain:

1. Suppose the letter-string ***abc*** were changed to ***abd***; how would you change the letter-string ***ijk*** in "the same way"?

Note that the challenge is essentially "Be a copycat" — that is, "Do the same thing as I did", where "same" of course is the slippery term. Almost everyone answers ***ijl***.[1] It is not hard to see why; most people feel that the natural way to describe what happened to ***abc*** is to say that *the rightmost letter was replaced by its alphabetic successor*; that operation can then be painlessly and naturally "exported" from the ***abc*** framework to the other framework, namely ***ijk***, to yield the answer ***ijl***. Of course this is not the only possible answer. For instance, it is always possible to be a "smart aleck" and to answer ***ijd*** (rigidly choosing to replace the rightmost letter by ***d***) or ***ijk*** (rigidly replacing all ***c***'s by ***d***'s) or even ***abd*** (replacing the whole structure blindly by ***abd***), but such "smart-alecky" answers are suggested rather infrequently, and when they are suggested, they seem less compelling to virtually everybody, even to the people who suggested them. Thus ***ijl*** is a fairly uncontroversial winner among the range of answers to this problem.

There is much more subtlety to the domain than that problem would suggest, however. Let us consider the following closely related but considerably more interesting analogy problem:

2. Suppose the letter-string ***aabc*** were changed to ***aabd***; how would you change the letter-string ***ijkk*** in "the same way"?

Here as in Problem 1, most people look upon the change in the first framework as *the rightmost letter was replaced by its alphabetic successor*. Now comes the tricky part: should this rule simply be transported rigidly to the other framework, yielding ***ijkl***? Although rigid exportation of the rule worked in Problem 1, here it seems rather crude to most people, because it ignores the obvious fact that the ***k*** is doubled. The two ***k***'s together seem to form a natural unit, and so it is tempting to change *both* of them, yielding the answer ***ijll***. Using the old rule literally will simply not give this answer; instead, under pressure, one "flexes" the old rule into a very closely related one, namely *replace the rightmost group by its alphabetic successor*. Here, the concept *letter* has "slipped", under pressure, into the related concept *group of letters*. Coming up with such a rule and corresponding answer is a good example of human mental "fluidity" (as contrasted with the mental rigidity that gives rise to ***ijkl***). There is more to the story of Problem 2, however.

Many people are perfectly satisfied with this way of exporting the rule (and the answer it furnishes), but some feel dissatisfied by the fact that the doubled ***a*** in ***aabc*** has been ignored. Once one focuses in on this consciously, it jumps to

1. Though the popularity of this answer can easily be predicted by one's intuition, we have carried out many surveys, both formal and informal, of people's answers to this and other problems. The results of the formal surveys are given in Mitchell, 1993.

mind easily that the ***aa*** and the ***kk*** play similar roles in their respective frameworks. From there it is but a stone's throw to "equating" them (as opposed to equating the ***c*** with the ***kk***), which leads to the question, "What then is the counterpart of the ***c***?" Given the already-established mapping of *leftmost* object (***aa***) onto *rightmost* object (***kk***), it is but a small leap to map *rightmost* object (***c***) onto *leftmost* object (***i***). At this point, we could simply take the successor of the ***i***, yielding the answer ***jjkk***.

However, few people who arrive at this point actually do this; given that the two crosswise mappings (***aa*** ⇔ ***kk***; ***c*** ⇔ ***i***) are an invitation to read ***ijkk*** in reverse, which reverses the alphabetical flow in that string, most people tend to feel that the conceptual role of alphabetical *successorship* in ***aabc*** is now being played by that of *predecessorship* in ***ijkk***. In that case, the proper modification of the ***i*** would not be to replace it by its successor, but by its alphabetical *predecessor,* yielding the answer ***hjkk***. And indeed, this is the answer most often reached by those people who consciously try to take into account *both* of the doubled letters. Such people, under pressure, have flexed the original rule into this variant of itself: *replace the leftmost letter by its alphabetic predecessor.* Another way of saying this is that a very fluid transport of the original rule from its home framework to the new one has taken place; during this transport, two concepts "slipped", under pressure, into neighboring concepts: *rightmost* into *leftmost,* and *successor* into *predecessor.* Thus, being a copycat — that is, "doing the same thing" — has proven to be a very slippery notion, indeed.

Mental fluidity: Slippages induced by pressures

Hopefully, the pathways leading to these two answers to Problem 2 — ***ijll*** and ***hjkk*** — convey a good feeling for the term "mental fluidity". There is, however, a related notion used above that still needs some clarification, and that is the phrase "under pressure". What does it mean to say "concept A *slips* into concept B *under pressure*"? It might help to spell out the intended imagery behind these terms. An earthquake takes place when subterranean structures are under sufficient pressure that something suddenly slips. Without the pressure, obviously, there would be no slippage. An analogous statement holds for pressures bringing about conceptual slippage: only under specific pressures will concepts slip into related ones. For instance, in Problem 2, pressure results from the doubling of the ***a*** and the ***k***; one could look upon the doubling as an "emphasis" device, making the left end of the first string and the right end of the second one stand out and in some sense "attract" each other. In Problem 1, on the other hand, there is nothing to suggest mapping the ***a*** onto the ***k*** — no pressure. In the absence of such pressure, it would make no sense at all to slip *leftmost* into *rightmost* and then to read ***ijk*** in reverse, which would in turn suggest a slippage of *successor* into *predecessor,* all of which would finally lead to the

downright bizarre answer ***hjk***. That would be *unmotivated* fluidity, which is not characteristic of human thought (except in humor, where higher-level considerations often *do* motivate all sorts of normally-unmotivated slippages).

Copycat is a thoroughgoing exploration of the nature of mental pressures, the nature of concepts, and their deep interrelationships, focusing particularly on how pressures can engender slippages of concepts into "neighboring" concepts. When one ponders these issues, many questions arise, such as the following ones: What is meant by "neighboring concepts"? How much pressure is required to make a given conceptual slippage likely? Just how big a slippage can be made — that is, how far apart can two concepts be and still be potentially able to slip into each other? How can one conceptual slippage create a new pressure leading to another conceptual slippage, and then another, and so on, in a cascade? Do some concepts resist slippage more than others? Can particular pressures nonetheless bring about a slippage of such a concept while another concept, usually more "willing" to slip, remains untouched? Such are the questions at the very heart of the Copycat project.

The intended universality of Copycat's microdomain

This project, which sprang out of two predecessors, Seek-Whence (Meredith, 1986) and Jumbo (Hofstadter, 1983a), has been under development since 1983. A casual glance at the project might give the impression that since it was specifically designed to handle analogies in a particular tiny domain, its mechanisms are not general. However, this would be a serious misconception. All the features of the Copycat architecture were in fact designed with an eye to great generality. A major purpose of this article is to demonstrate this generality by describing the features of Copycat in very broad terms, and to show how they transcend not just the specific microdomain, but even the very task of analogy-making itself. That is, the Copycat project is not about simulating analogy-making *per se*, but about simulating the very crux of human cognition: fluid concepts. The reason the project focuses upon analogy-making is that analogy-making is perhaps the quintessential mental activity where fluidity of concepts is called for, and the reason the project restricts its modeling of analogy-making to a specific and very small domain is that doing so allows the general issues to be brought out in a very clear way — far more clearly than in a "real-world" domain, despite what one might think at first.

Copycat's microdomain was designed to bring out very general issues — issues that transcend any specific conceptual domain. In that sense, the microdomain was designed to "stand for" other domains. Thus one is intended to conceive of, say, the *successor* (or *predecessor*) relation as an idealized version of *any* non-identity relationship in a real-world domain, such as "parent of", "neighbor of", "friend of", "employed by", "close to", etc. A *successor group* (*e.g.*,

abc) then plays the role of any conceptual chunk based on such a relationship, such as "family", "neighborhood", "community", "workplace", "region", etc. Of course, inclusion of the notion of *sameness* needs no defense; sameness is obviously a universal concept, much as is *opposite.* Although any real-world domain clearly contains many more than two basic types of relationship, two types (sameness plus one other one) already suffice to make an inexhaustible variety of structures of arbitrary complexity.

Aside from the idealized repertoire of *concepts* in the domain, there are also the *structures,* such as ***ijkk***, out of which problems are made. In particular, allowed structures are linear strings made from any number — usually a small number — of instances of letters of the alphabet. Thus one immediately runs into the *type/token distinction,* a key issue in understanding cognition. The alphabet can be thought of as a very simple "Platonic heaven" in which exactly 26 letter *types* permanently float in a fixed order; in contrast to this, there is a very rudimentary "physical world" in which any number of letter *tokens* can temporarily coexist in an arbitrary one-dimensional juxtaposition. In this extremely simple model of physical space, there are such physical relationships and entities as *left-neighbor, leftmost edge, group of adjacent letters,* and so on (as contrasted with such relationships and entities in the Platonic alphabet as *predecessor, alphabetic starting-point, alphabetic segment,* etc.). Both the Platonic heaven and the physical world of Copycat are very simple on their own; however, the psychological processes of perception and abstraction bring them into intimate interaction, and can cause extremely complex and subtle mental representations of situations to come about.

Copycat's alphabetic microworld is meant to be a tool for exploring general issues of cognition rather than issues specific to the domain of letters and strings, or domains restricted to linear structures with precise distances in them. Thus certain aspects specific to people's knowledge of letters and letter-strings — such as shapes, sounds, or cultural connotations of specific letters, or words that strings of letters might happen to form — have not been included in this microworld. Moreover, problems should not depend on arithmetical facts about letters, such as the fact that ***t*** comes exactly eleven letters after ***i***, or that ***m*** and ***n*** flank the midpoint of the alphabet. Arithmetical facts, while they are universal truths, are not common enough in analogy-making to be worthwhile modeling. This may seem to eliminate almost everything about the alphabet, but as Problems 1 and 2 show (and further problems will show even better), there is still plenty left to play with. Reference to the alphabet's *local* structure is fine; for example, it is perfectly legitimate to exploit the fact that ***u*** comes immediately after ***t***. It is also legitimate to exploit the fact that the Platonic alphabet has two distinguished members — namely, ***a*** and ***z***, its starting and ending points. Likewise, inside a string such as ***hagizk***, local relationships,

such as "the ***g*** is the right-neighbor of the ***a***", can be noticed, but long-distance observations, such as "the ***a*** is four letters to the left of the ***k***", are considered out of bounds.

Although arithmetical operations such as addition and multiplication play no role in the Copycat domain, numbers themselves — small whole numbers, that is — are included in the domain. Thus, Copycat is capable of recognizing not only that the structure ***fgh*** is a "successor group", but also that it consists of *three* letters. Just as the program knows the immediate neighbors of every letter in the alphabet, it also knows the successors and predecessors of small integers. Under the appropriate pressures, Copycat can even treat small integers as it does letters — it can notice relationships between numbers, can group numbers together, map them onto each other, and so on. However, generally speaking, Copycat tends to resist bringing numbers into the picture, unless there seems to be some compelling reason to do so — and *large* numbers, such as 5, are resisted even more strongly. The idea behind this is to reflect the relative ease humans have of recognizing pairs and perhaps trios of objects, but the relative insensitivity to such things as quintuples, let alone septuples and so on.

Finally, while humans tend to scan strings of roman letters from left to right, are much better at recognizing forwards alphabetical order than backwards alphabetical order, and have somewhat greater familiarity with the beginning of the alphabet than its middle or end, the Copycat program is completely free of these biases. This should not be regarded as a defect of the program, but a strength, because it keeps the project's focus away from domain-specific and nongeneralizable details.

A perception-based, emergent architecture for mental fluidity

When one describes the Copycat architecture in very abstract terms, the focus is not only on how it discovers mappings between situations, but also on how it perceives and makes sense of the miniature and idealized situations it is presented with. The present characterization will therefore read very much like a description of a computer model of *perception*. This is not a coincidence; one of the main ideas of the project is that even the most abstract and sophisticated mental acts deeply resemble perception. In fact, the inspiration for the architecture comes in part from a computer model of low-level and high-level auditory perception: the Hearsay II speech-understanding project (Erman *et al.*, 1980; Reddy *et al.*, 1976).

The essence of perception is the awakening from dormancy of a relatively small number of prior concepts — precisely the relevant ones. The essence of understanding a situation is very similar; it is the awakening from dormancy of a relatively small number of prior concepts — again, precisely the relevant ones — and applying them judiciously so as to identify the key entities, roles, and

relationships in the situation. Creative human thinkers manifest an exquisite selectivity of this sort — when they are faced with a novel situation, what bubbles up from their unconscious and pops to mind is typically a small set of concepts that "fit like a glove", without a host of extraneous and irrelevant concepts being consciously activated or considered. To get a computer model of thought to exhibit this kind of behavior is a great challenge.

Following this introductory section, there are six further main sections in this article. The second section is a description of the three main components of the architecture and their interactions. The third section deals with the notion of conceptual fluidity and shows how this architecture implements a model, albeit rudimentary, thereof. The fourth section tackles the seeming paradox of randomness as an essential ingredient of mental fluidity and intelligence. The fifth section views the Copycat program at a distance, summarizing thousands of runs on a few key problems in the letter-string microworld. The sixth section affords a close-up view of Copycat's workings, describing in detail the pathways followed by Copycat as it comes up with subtle answers to two particularly challenging analogy problems. The seventh section concludes the article with a discussion of the generality of Copycat's mechanisms.

The Three Major Components of the Copycat Architecture

There are three major components to the architecture: the Slipnet, the Workspace, and the Coderack. In very quick strokes, they can be described as follows. (1) The Slipnet is the site of all *permanent Platonic concepts*. It can be thought of, roughly, as Copycat's long-term memory. As such, it contains only concept *types*, and no *instances* of them. The distances between concepts in the Slipnet can change over the course of a run, and it is these distances that determine, at any given moment, what slippages are likely and unlikely. (2) The Workspace is the locus of *perceptual activity*. As such, it contains *instances* of various concepts from the Slipnet, combined into *temporary perceptual structures* (*e.g.*, raw letters, descriptions, bonds, groups, and bridges). It can be thought of, roughly, as Copycat's short-term memory or working memory, and resembles the global "blackboard" data-structure of Hearsay II. (3) Finally, the Coderack can be thought of as a "stochastic waiting room", in which small agents that wish to carry out tasks in the Workspace wait to be called. It has no close counterpart in other architectures, but one can liken it somewhat to an *agenda* (a queue containing tasks to be executed in a specific order). The critical difference is that agents are selected *stochastically* from the Coderack, rather than in a determinate order. The reasons for this initially puzzling feature will be spelled out and analyzed in detail below. They turn out to be at the crux of mental fluidity.

We now shall go through each of the three components once again, this time in more detail. (The finest level of detail — complete lists of algebraic formulas, numerical parameters, and their exact values — is not given here, but can be found in Mitchell, 1993.)

The Slipnet — Copycat's network of Platonic concepts

The basic image for the Slipnet is that of a network of interrelated concepts, each concept being represented by a *node* (caveat: what a concept is, in this model, is actually a bit subtler than just a pointlike node, as will be explained shortly), and each conceptual relationship by a *link* having a numerical length, representing the "conceptual distance" between the two nodes involved. The shorter the distance between two concepts is, the more easily pressures can induce a slippage between them.

Some of the main concepts in Copycat's Slipnet are: ***a***, ***b***, ***c***, ..., ***z***, *letter, successor, predecessor, .alphabetic-first, alphabetic-last, alphabetic position, left, right, direction, leftmost, rightmost, middle, string position, group, sameness group, successor group, predecessor group, group length, 1, 2, 3, sameness,* and *opposite.* In all, there are roughly 60 concepts.

The Slipnet is not static; it dynamically responds to the situation at hand as follows: Nodes *acquire* varying levels of activation (which can be thought of as a measure of relevance to the situation at hand), *spread* varying amounts of activation to neighbors, and over time *lose* activation by decay. Activation is not an on-and-off affair, but varies continuously. However, when a node's activation crosses a certain critical threshold, the node has a probability of jumping discontinuously into a state of *full* activation, from which it proceeds to decay. In sum, the activation — the perceived relevance — of each concept is a sensitive, time-varying function of the way the program currently understands the situation it is facing.

Conceptual links in the Slipnet adjust their lengths dynamically. Thus, conceptual distances gradually change under the influence of the evolving perception (or conception) of the situation at hand, which of course means that the current perception of the situation enhances the chance of certain slippages taking place, while rendering that of others more remote.

Conceptual depth

Each node in the Slipnet has one very important static feature called its *conceptual depth.* This is a number intended to capture the generality and abstractness of the concept. For example, the concept *opposite* is deeper than the concept *successor,* which is in turn deeper than the concept ***a***. It could be said roughly that the depth of a concept is how far that concept is from being directly perceivable in situations. For example, in Problem 2, the presence of

instances of ***a*** is trivially perceived; recognizing the presence of *successorship* takes a little bit of work; and recognition of the presence of the notion *opposite* is a subtle act of abstract perception. The further away a given aspect of a situation is from direct perception, the more likely it is to be involved in what people consider to be the *essence* of the situation. Therefore, once aspects of greater depth are perceived, they should have more influence on the ongoing perception of the situation than aspects of lesser depth.

Assignment of conceptual depths amounts to an *a priori* ranking of "best-bet" concepts. The idea is that a deep concept (such as *opposite*) is normally relatively hidden from the surface and cannot easily be brought into the perception of a situation, but that once it *is* perceived, it should be regarded as highly significant. There is of course no guarantee that deep concepts will be relevant in any particular situation, but such concepts were assigned high depth-values precisely because we saw that they tend to crop up over and over again across many different types of situations, and because we noticed that the best insights in many problems come when deep concepts "fit" naturally. We therefore built into the architecture a strong drive, if a deep aspect of a situation is perceived, to use it and to try to let it influence further perception of the situation.

Note that the hierarchy defined by different conceptual-depth values is quite distinct from abstraction hierarchies such as

$$poodle \Rightarrow dog \Rightarrow mammal \Rightarrow animal \Rightarrow living\ thing \Rightarrow thing.$$

These terms are all potential descriptions of a particular object at different levels of abstraction. By contrast, the terms ***a***, *successor*, and *opposite* are not descriptions of one particular *object* in Problem 2, but of various aspects of the situation, at different levels of abstraction.

Likewise, conceptual depth is not the same as Gentner's notion of "abstractness" (Gentner, 1983). In Gentner's theory, attributes (*e.g.*, "the leftmost letter has value ***a***") are invariably less abstract than relations (*e.g.*, "the next-to-leftmost letter is the successor of the leftmost letter"), which are in turn invariably less abstract than relations between relations (*e.g.*, "*successor* is the opposite of *predecessor*"). This heuristic, based on syntactic structure, often agrees with our conceptual-depth hierarchy, but in Copycat certain "attributes" are considered to be conceptually deeper than certain "relations" — for example, *alphabetic-first* has a greater depth than *successor* because we consider the former to be less directly perceivable than the latter. (In the following chapter, we go into considerably more detail in contrasting Gentner's work with ours.)

Conceptual depth has a second important aspect — namely, the deeper a concept is, the more resistant it is (all other things being equal) to slipping into another concept. In other words, there is a built-in propensity in the program

to prefer slipping shallow concepts rather than deep concepts, when slippages have to be made. The idea of course is that insightful analogies tend to link situations that share a deep *essence,* allowing shallower features to slip if necessary. This basic idea can be summarized in a motto: *Deep stuff doesn't slip in good analogies.* There are, however, interesting situations in which specific constellations of pressures arise that cause this basic tendency to be overridden.

Activation flow and variable link-lengths

Some details about the flow of activation: (1) each node spreads activation to its neighbors according to their distance from it, with near neighbors getting more, distant neighbors less; (2) each node's conceptual-depth value sets its *decay rate,* in such a way that deep concepts always decay slowly and shallow concepts decay quickly. This means that, once a concept has been perceived as relevant, then, the deeper it is, the longer it will remain relevant, and thus the more profound an influence it will exert on the system's developing view of the situation — as indeed befits an abstract and general concept likely to be close to the essence of the situation.

Some details about the Slipnet's dynamical properties: (1) there are a variety of *link types,* and for each given type, all links of that type share the same *label*; (2) each label is itself a concept in the network; (3) every link constantly adjusts its length according to the activation level of its label, with high activation giving rise to short links, low activation to long ones. Stated another way: If concepts A and B have a link of type L between them, then as concept L's relevance goes up (or down), concepts A and B become conceptually closer (or further apart). Since this is happening all the time all throughout the network, the Slipnet is constantly altering its "shape" in attempting to mold itself increasingly accurately to fit the situation at hand. An example of a label is the node *opposite,* which labels the link between nodes *right* and *left,* the link between nodes *successor* and *predecessor,* and several other links. If the node *opposite* gets activated, all these links will shrink in concert, rendering the potential slippages they represent more probable.

The length of a link between two nodes represents the conceptual proximity or degree of association between the nodes: the shorter the link, the greater the degree of association, and thus the easier it is to effect a slippage between them. There is a probabilistic "cloud" surrounding any node, representing the likelihood of slippage to other nodes; the cloud's density is highest for near-neighbor nodes and rapidly tapers off for distant nodes. (This is reminiscent of the quantum-mechanical "electron cloud" in an atom, whose probability density falls off with increasing distance from the nucleus.) Neighboring nodes can be seen as being included in a given concept probabilistically, as a function of their proximity to the central node of the concept.

Concepts as diffuse, overlapping clouds

This brings us back to the caveat mentioned above: Although it is tempting to equate a concept with a pointlike node, a concept is better identified with this probabilistic "cloud" or halo *centered* on a node and extending outwards from it with increasing diffuseness. As links shrink and grow, nodes move into and out of each other's halos (to the extent that one can speak of a node as being "inside" or "outside" a blurry halo). This image suggests conceiving of the Slipnet not so much as a hard-edged network of points and lines, but rather as a space in which many diffuse clouds overlap each other in an intricate, time-varying way.

Conceptual proximity in the Slipnet is thus context-dependent. For example, in Problem 1, no pressures arise that bring the nodes *successor* and *predecessor* into close proximity, so a slippage from one to the other is highly unlikely; by contrast, in Problem 2, there is a good chance that pressures will activate the concept *opposite,* which will then cause the link between *successor* and *predecessor* to shrink, bringing each one more into the other's halo, and enhancing the probability of a slippage between them. Because of this type of context-dependence, concepts in the Slipnet are *emergent,* rather than explicitly defined.

The existence of an explicit core to each concept is a crucial element of the architecture. Specifically, slippability depends critically on the discrete jump from one core to another. Diffuse regions having no cores would not permit such discrete jumps, as there would be no specific starting or ending point. Even an explicit *name* attached to a coreless diffuse region could serve as a substitute for a core — it would permit a discrete jump. In any case, however, slippage requires each concept to be attached to some identifiable "place" or entity. One might liken the core of a concept to the official city limits of a large city, and the halo to the much vaguer metropolitan region surrounding the city proper, stretching out in all directions, and clearly far more subjective and context-dependent than the core.

It may be useful to briefly compare Copycat's Slipnet with connectionist networks. In localist networks, a concept is equated with a node rather than with a diffuse region centered on a node. In other words, concepts in localist networks lack halos. This lack of halos implies that there is no counterpart to slippability in localist networks. In distributed systems, on the other hand, there would seem to be halos, since a concept is equated with a diffuse region, but this is somewhat misleading. The diffuse region representing a concept is not explicitly centered on any node, so there is no explicit *core* to a concept, and in that sense no halo. But since slippability depends on the existence of discrete cores, there is no counterpart to slippability even in distributed connectionist models.

The lack of any explicit center to a concept would probably be found to be quite accurate if one could examine concepts on the neural level. However,

Copycat was not designed to be a neural model; it aims at modeling cognitive-level behavior by simulating processes at a subcognitive but superneural level. We believe that there is a subcognitive, superneural level at which it is realistic to conceive of a concept as having an explicit core surrounded by an implicit, emergent halo.

Another temptation might be to liken Copycat's context-dependent link-lengths to the changing of inter-node weights as a connectionist net adapts to training stimuli. One might even liken the effect of a label node in Copycat to a multiplicative connection (where some node's activation is used as a multiplicative factor in calculating the new weight of a link). To be sure, there is a mathematical analogy here, but conceptually there is a significant difference. As connectionist networks adapt and "learn" by changing their weights, there is no sense of departing from a norm and no tendency to return to an earlier state. By contrast, in Copycat, any changing of link-lengths takes place in response to a temporary context, and when that context is removed, the Slipnet tends to revert to its "normal" state. The Slipnet is thus "rubbery" or "elastic" in this sense; it responds to context but has a built-in tendency to "snap back" to its original state. We know of no corresponding tendency in connectionist networks.

Note that whereas the Slipnet changes over the course of a single run of Copycat, it does not retain changes from run to run, or create new permanent concepts. The program starts out in the same initial state on every run. Thus Copycat does not model *learning* in the usual sense. However, this project does concern learning, if that term is taken to include the notion of adaptation of one's concepts to novel contexts.

Although the Slipnet responds sensitively to events in the Workspace (described in a moment) by constantly changing both its "shape" and the activations of its nodes, its fundamental topology remains invariant. That is, no new structure is ever built, or old structure destroyed, in the Slipnet. The next subsection discusses a component of the architecture that provides a strong contrast to this type of topological invariance.

The Workspace — Copycat's locus of perceptual activity

The basic image for the Workspace is that of a busy construction site in which structures of many sizes and at many locations are being worked on simultaneously by independent crews, some occasionally being torn down to make way for new, hopefully better ones. (This image comes essentially from the biological cell; the Workspace corresponds roughly to the cytoplasm of a cell, in which enzymes carrying out diverse tasks all throughout the cell's cytoplasm are the construction crews, and the structures built up are all sorts of hierarchically-structured biomolecules.)

At the start of a run, the Workspace is a collection of unconnected raw data representing the situation with which the program is faced. Each item in the Workspace initially carries only bare-bones information — that is, for each letter token, just its alphabetic type is provided, as well as — for those letters at the very edges of their strings — the descriptor *leftmost* or *rightmost*. Other than that, all objects are absolutely barren. Over time, through the actions of many small agents "scouting" for features of various sorts (these agents, called "codelets", are described in the next subsection), items in the Workspace gradually acquire various *descriptions*, and are linked together by various *perceptual structures*, all of which are built entirely from concepts in the Slipnet.

The constant fight for probabilistic attention

Objects in the Workspace do not by any means all receive equal amounts of attention from codelets; rather, the probability that an object will attract a prospective codelet's attention is determined by the object's *salience*, which is a function of both the object's *importance* and its *unhappiness*. Though it might seem crass, the architecture honors the old motto "The squeaky wheel gets the oil", even if only probabilistically so. Specifically, the more descriptions an object has and the more highly activated the nodes involved therein, the more important the object is. Modulating this tendency is the object's level of unhappiness, which is a measure of how integrated the object is with other objects. An unhappy object is one that has few or no connections to the rest of the objects in the Workspace, and that thus seems to cry out for more attention. Salience is a dynamic number that takes into account both of these factors, and this number determines how attractive the object in question will appear to codelets. Note that salience depends intimately on both the state of the Workspace and the state of the Slipnet.

A constant feature of the processing is that pairs of *neighboring objects* (inside a single framework — *i.e.*, letter-string) are probabilistically selected (with a bias favoring pairs that include salient objects) and scanned for similarities or relationships, of which the most promising are likely to get "reified" (*i.e.*, realized in the Workspace) as inter-object *bonds*. For instance, the two ***k***'s in ***ijkk*** in Problem 2 are likely to get bonded to each other rather quickly by a *sameness* bond. Similarly, the ***i*** and the ***j*** are likely to get bonded to each other, although not as fast, by a *successorship bond* or a *predecessorship bond*.

The existence of differential rates of speed of bond-making is meant to reflect realities of human perception. In particular, people are clearly quicker to recognize two neighboring objects as identical than as being related in some abstract way. Thus the architecture has an intrinsic speed-bias in favor of sameness bonds: it tends to spot them and to construct them more quickly than it spots and constructs bonds representing other kinds of relationships. (How

the speeds of rival processes are dynamically controlled will be dealt with in more detail in the next subsection.)

Any bond, once made, has a dynamically varying *strength*, reflecting not only the activation and conceptual depth of the concept representing it in the Slipnet (in the case of ***kk***, the concept *sameness*, and in the case of ***ij***, either *successor* or *predecessor*) but also the prevalence of similar bonds in its immediate neighborhood. The idea of bonds is of course to start weaving unattached objects together into a coherent mental structure.

The parallel emergence of multi-level perceptual structures

A set of objects in the Workspace bonded together by a uniform "fabric" (*i.e.*, bond type) is a candidate to be "chunked" into a higher-level kind of object called a *group*. A simple example of a *sameness group* is ***kk***, as in Problem 2. Another simple group is ***abc***, as in Problem 1. This one, however, is a little ambiguous; depending on which direction its bonds are considered to go in, either it is perceived as having a left-to-right *successorship* fabric and is thus seen as a left-to-right *successor group*, or it is perceived as having a right-to-left *predecessorship* fabric and is thus seen as a right-to-left *predecessor group*. (It cannot be seen as both at once, although the program can switch from one vision to the other relatively easily.) The more salient a potential group's component objects and the stronger its fabric, the more likely it is to be reified.

Groups, just like more basic types of objects, acquire their own descriptions, salience values, and strengths, and are themselves candidates for similarity-scanning, bonding to other objects, and possibly becoming parts of yet higher-level groups. As a consequence, hierarchical perceptual structures get built up over time, under the guidance of biases emanating from the Slipnet. A simple example would be the successor (or predecessor) group ***ijkk*** in Problem 2, made up of three elements: the ***i***, the ***j***, and the short sameness group ***kk***.

Another constant feature of the processing is that pairs of objects in *different* frameworks (*i.e.*, strings) are probabilistically selected (again with a bias favoring salient objects) and scanned for similarities, of which the most promising are likely to get reified as *bridges* (or *correspondences*) in the Workspace. Effectively, a bridge establishes that its two end-objects are considered each other's counterparts — meaning either that they are intrinsically similar objects or that they play similar roles in their respective frameworks (or hopefully both).

Consider, for instance, the ***aa*** and ***kk*** in Problem 2. What makes one tempted to equate them? One factor is their intrinsic similarity — both are doubled letters (sameness groups of length 2). Another factor is that they fill similar roles, since one sits at the left end of its string, the other at the right end of its string. If and when a bridge gets built between them, concretely reifying this mental correspondence, it will be explicitly based on both these facts. The

fact that ***a*** and ***k*** are unrelated letters of the alphabet is simply ignored by most people. Copycat is constructed to behave similarly. Thus, the fact that ***aa*** and ***kk*** are both sameness groups will be embodied in an *identity mapping* (here, *sameness* ⇔ *sameness*); the fact that one is leftmost while the other is rightmost will be embodied in a *conceptual slippage* (here, *leftmost* ⇔ *rightmost*); the fact that nodes ***a*** and ***k*** are far apart in the Slipnet is simply ignored.

Whereas identity mappings are always welcome in a bridge, conceptual slippages always have to overcome a certain degree of resistance, the precise amount of which depends on the proposed slippage itself and on the circumstances. The most favored slippages are those whose component concepts not only are shallow but also have a high degree of overlap (*i.e.*, are very close in the Slipnet). Slippages between highly overlapping *deep* concepts are more difficult to build, but pressures can certainly bring them about.

Once any bridge is built, it has a *strength,* reflecting the ease of the slippages it entailed, the number of identity mappings helping to underpin it, and its resemblance to other bridges already built. The idea of bridges is of course to build up a coherent mapping between the two frameworks.

To form a clear image of all this hubbub, it is crucial to keep in mind that all the aforementioned types of perceptual actions — scanning, bond-making, group-making, bridge-building, and so forth (as well as all the spreading and decaying of activation and so on in the Slipnet) — take place in parallel, so that independent perceptual structures of all sorts, spread about the Workspace, gradually emerge at the same time, and all the biases controlling the likelihood of this concept or that one being brought to bear are constantly fluctuating in light of what has already been observed in the Workspace.

The drive towards global coherence and towards deep concepts

As the Workspace evolves in complexity, there is increasing pressure on new structures to be *consistent,* in a certain sense, with pre-existent structures, especially with ones in the same framework. For two structures to be consistent sometimes means that they are instances of the very same Slipnet concept, sometimes that they are instances of very close Slipnet concepts, and sometimes it is a little more complex. In any case, the Workspace is not just a hodgepodge of diverse structures that happen to have been built up by totally independent codelets; rather, it represents a coherent vision built up piece by piece by many agents all indirectly influencing each other. Such a vision will henceforth be called a *viewpoint.* A useful image is that of highly coherent macroscopic structures (*e.g.*, physical bridges) built by a colony of thousands of myopic ants or termites working semi-independently but nonetheless cooperatively. (The "ants" of Copycat — namely, codelets — will be described in the next subsection.)

There is constant competition, both on a local and a global level, among structures vying to be built. A structure's likelihood of beating out its rivals is determined by its *strength,* which has two facets: a context-independent facet (a contributing factor would be, for instance, the depth of the concept of which it is an instance) and a context-dependent facet (how well it fits in with the rest of the structures in the Workspace, particularly the ones that would be its neighbors). Out of the rough-and-tumble of many, many small decisions about which new structures to build, which to leave intact, and which to destroy comes a particular global viewpoint. Even viewpoints, however, are vulnerable; it takes a very powerful rival to topple an entire viewpoint, but this occasionally happens. Sometimes these "revolutions" are, in fact, the most creative decisions that the system as a whole can carry out.

As was mentioned briefly above, the Slipnet responds to events in the Workspace by selectively activating certain nodes. The way activation comes about is that any discovery made in the Workspace — creation of a bond of some specific type, a group of some specific type, etc. — sends a substantial jolt of activation to the corresponding concept in the Slipnet; the amount of time the effect of such a jolt will last depends on the concept's decay rate, which depends in turn on its depth. Thus, a deep discovery in the Workspace will have long-lasting effects on the activation pattern and "shape" of the Slipnet; a shallow discovery will have but transient effects. In Problem 2, for example, if a bridge is built between the groups ***aa*** and ***kk***, it will very likely involve an *opposite* slippage (*leftmost* ⇔ *rightmost*). This discovery will reveal the hitherto unsuspected relevance of the very deep concept *opposite,* which is a key insight into the problem. Because *opposite* is a deep concept, once it is activated, it will remain active for a long time and therefore exert powerful effects on subsequent processing.

It is clear from all this that the Workspace affects the Slipnet no less than the Slipnet affects the Workspace; indeed, their influences are so reciprocal and tangled that it is hard to tell the chicken from the egg.

Metaphorically, one could say that *deep concepts* and *structural coherency* act like strong magnets pulling the entire system. The pervasive biases favoring the realization of these abstract qualities in the Workspace imbues Copycat with an overall goal-oriented quality that *a priori* might seem surprising, given that the system is highly decentralized, parallel, and probabilistic, thus far more like a swarm of ants than like a rigid military hierarchy, the latter of which has more standardly served as a model for how to realize goal-orientedness in computer programs. We now turn to the description of Copycat's "ants" and how they are biased.

The Coderack — source of emergent pressures in Copycat

All acts of describing, scanning, bonding, grouping, bridge-building, destruction, and so forth in the Workspace are carried out by small, simple agents

called *codelets*. The action of a single codelet is always but a tiny part of a run, and whether any particular codelet runs or not is not of much consequence. What matters is the collective effect of many codelets.

There are two types of codelets: *scout codelets* and *effector codelets*. A scout merely looks at a potential action and tries to estimate its promise; the only kind of effect it can have is to create one or more codelets — either scouts or effectors — to follow up on its findings. By contrast, an effector codelet actually creates (or destroys) some structure in the Workspace.

Typical *effector* codelets do such things as: attaching a description to an object (*e.g.*, attaching the descriptor *middle* to the ***b*** in ***abc***); bonding two objects together (*e.g.*, inserting a *successor* bond between the ***b*** and ***c*** in ***abc***); making a group out of two or more adjacent objects that are bonded together in a uniform manner; making a bridge that joins similar objects in distinct strings (similarity being measured by proximity of descriptors in the Slipnet); destroying groups or bonds, and so on.

Before any such action can take place, preliminary checking-out of its promise has to be carried out by *scout* codelets. For example, one scout codelet might notice that the adjacent ***r***'s in ***mrrjjj*** are instances of the same letter, and propose a sameness bond between them; another scout codelet might estimate how well that proposed bond fits in with already-existing bonds; then an effector codelet might actually *build* the bond. Once such a bond exists, scout codelets might then check out the idea of subsuming the two bonded ***r***'s into a sameness group, after which an effector codelet could go ahead and actually build the group.

Each codelet, when created, is placed in the *Coderack*, which is a pool of codelets waiting to run, and is assigned an *urgency value* — a number that determines its probability of being selected from that pool as the next codelet to run. The urgency is a function of the estimated importance of that codelet's potential action, which in turn reflects the biases embodied in the current state of the Slipnet and the Workspace. Thus, for example, a codelet whose purpose is to seek instances of some lightly activated Slipnet concept will be assigned a low urgency and will therefore probably have to wait a long time, after being created, to get run. By contrast, a codelet likely to further a Workspace viewpoint that is currently strong will be assigned a high urgency and will thus have a good chance of getting run soon after being created.

It is useful to draw a distinction between *bottom-up* and *top-down* codelets. Bottom-up codelets (or "noticers") look around in an unfocused manner, open to what they find, whereas top-down codelets (or "seekers") are on the lookout for a particular kind of phenomenon, such as successor relations or sameness groups. Codelets can be viewed as *proxies* for the pressures in a given problem. Bottom-up codelets represent pressures present in *all* situations (the desire to

make descriptions, to find relationships, to find correspondences, and so on). Top-down codelets represent specific pressures evoked by the specific situation at hand (*e.g.*, the desire, in Problems 1 and 2, to look for more successor relations, once some have already been discovered). Top-down codelets can infiltrate the Coderack only when triggered from "on high" — that is, from the Slipnet. In particular, activated nodes are given the chance to "spawn" top-down scout codelets, with a node's degree of activation determining the codelet's urgency. The mission of such a codelet is to scan the Workspace in search of instances of its spawning concept.

Pressures determine the speeds of rival processes

It is very important to note that the calculation of a codelet's urgency takes into account (directly or indirectly) numerous factors, which may include the activations of several Slipnet nodes as well as the strength or salience of one or more objects in the Workspace; it would thus be an oversimplification to picture a top-down codelet as simply a proxy for the particular concept that spawned it. More precisely, a top-down codelet is a proxy for one or more *pressures* evoked by the situation. These include *workspace pressures,* which attempt to maintain and extend a coherent viewpoint in the Workspace, and *conceptual pressures,* which attempt to realize instances of activated concepts. It is critical to understand that pressures, while they are very *real,* are not represented *explicitly* anywhere in the architecture; each pressure is spread out among urgencies of codelets, activations and link-lengths in the Slipnet, and strengths and saliences of objects in the Workspace. Pressures, in short, are implicit, emergent consequences of the deeply intertwined events in the Slipnet, Workspace, and Coderack.

Any run starts with a standard initial population of bottom-up codelets (with preset urgencies) on the Coderack. At each time step, one codelet is chosen to run and is removed from the current population on the Coderack. As was said before, the choice is probabilistic, biased by relative urgencies in the current population. Copycat thus differs from an "agenda" system such as Hearsay II, which, at each step, executes the waiting action with the highest estimated priority. The urgency of a codelet should not be conceived of as representing an estimated *priority*; rather, it represents the estimated relative *speed* at which the pressures represented by this codelet should be attended to. If the highest-urgency codelet were always chosen to run, then lower-urgency codelets would never be allowed to run, even though the pressures they represent have been judged to deserve *some* amount of attention.

Since any single codelet plays but a small role in helping to further a given pressure, it never makes a crucial difference that a particular codelet be selected; what really matters is that each *pressure* move ahead at roughly the

proper speed over time. Stochastic selection of codelets allows this to happen, even when judgments about the intensity of various pressures change over time. Thus allocation of resources is an emergent statistical result rather than a preprogrammed deterministic one. The proper allocation of resources could not be programmed ahead of time, since it depends on what pressures emerge as a given situation is perceived.

The shifting population of the Coderack

The Coderack would obviously dwindle rapidly to zero if codelets, once run and removed from it, were not replaced. However, replenishment of the Coderack takes place constantly, and this happens in three ways. Firstly, *bottom-up* codelets are continually being added to the Coderack. Secondly, codelets that run can, among other things, add one or more *follow-up* codelets to the Coderack before being removed. Thirdly, active nodes in the Slipnet can add *top-down* codelets. Each new codelet's urgency is assigned by its creator as a function of the estimated promise of the task it is to work on. Thus the urgency of a follow-up codelet is a function of the amount of progress made by the codelet that posted it, as gauged by that codelet itself, while the urgency of a top-down codelet is a function of the activation of the node that posted it. The urgency of bottom-up codelets is context-independent.

As a run proceeds, the population of the Coderack adjusts itself dynamically in response to the system's needs, as judged by previously-run codelets and by activation patterns in the Slipnet, which themselves depend on the current structures in the Workspace. This means there is a *feedback loop* between perceptual activity and conceptual activity, with observations in the Workspace serving to activate concepts, and activated concepts in return biasing the directions in which perceptual processing tends to explore. There is no top-level executive directing the system's activity; all acts are carried out by ant-like codelets.

The shifting population of codelets on the Coderack bears a close resemblance to the shifting enzyme population of a cell, which evolves in a sensitive way in response to the ever-changing makeup of the cell's cytoplasm. Just as the cytoplasmic products of certain enzymatic processes trigger the production of new types of enzymes to act further on those products, structures built in the Workspace by a given set of codelets cause new types of codelets to be brought in to work on them. And just as, at any moment, certain genes in the cell's DNA genome are allowed to be expressed (at varying rates) through enzyme proxies, while other genes remain essentially repressed (dormant), certain Slipnet nodes get "expressed" (at varying rates) through top-down codelet proxies, while other nodes remain essentially repressed. In a cell, the total effect is a highly coherent metabolism that emerges without any explicit top-down control; in Copycat, the effect is similar.

Note that though Copycat runs on a serial computer and thus only one codelet runs at a time, the system is roughly equivalent to one in which many independent activities are taking place in parallel and at different speeds, since codelets, like enzymes, work locally and to a large degree independently. The speed at which an avenue is pursued is an *a priori* unpredictable statistical consequence of the urgencies of the many diverse codelets pursuing that avenue.

The Emergence of Fluidity in the Copycat Architecture

Commingling pressures — the crux of fluidity

One of the central goals of the Copycat architecture is to allow many pressures to simultaneously coexist, competing and cooperating with one another to drive the system in certain directions. The way this is done is by converting pressures into flocks of very small agents (*i.e.*, codelets), each having some small probability of getting run. As was stated above, a codelet acts as a proxy for several pressures, all to differing degrees. All these little proxies for pressures are thrown into the Coderack, where they wait to be chosen. Whenever a codelet is given the chance to run, the various pressures for which it is a proxy make themselves slightly felt. Over time, the various pressures thus "push" the overall pattern of exploration different amounts, depending on the urgencies assigned to their codelets. In other words, the "causes" associated with the different pressures get advanced in parallel, but at different speeds.

There is a definite resemblance to classical time-sharing on a serial machine, in which any number of independent processes can be run concurrently by letting each one run a little bit (*i.e.*, giving it a "time slice"), then suspending it and passing control to another process, and so forth, so that bit by bit, each process eventually runs to completion. Classical time-sharing, incidentally, allows one to assign to each process a different speed, either by controlling the *durations* of its time slices or by controlling the *frequency* with which its time slices are allowed to run. The latter way of regulating speed is similar to the method used in Copycat; however, Copycat's method is probabilistic rather than deterministic (comments on why this is so follow in brief order).

This analogy with classical time-sharing is helpful but can also mislead. The principal danger is that one might get the impression that there are pre-laid-out *processes* to which time slices are probabilistically granted — more specifically, that any codelet is essentially a time slice of some preordained process. This is utterly wrong. In the Copycat architecture, the closest analogue to a classical process is a pressure — but the analogy is certainly not close. A pressure is nothing like a determinate sequence of actions; in very broad brushstrokes, a *conceptual* pressure can be portrayed as a concept (or cluster of closely related

concepts) trying to impose itself on a situation, and a *workspace* pressure as an established viewpoint trying to entrench itself further while keeping rival viewpoints out of the picture. Whereas classical processes are cleanly distinguishable from one another, this is not at all the case for pressures. A given codelet, by running, can advance (or hinder) any number of pressures.

There is thus no way of conceptually breaking up a run into a set of distinct foreordained processes each of which advances piecemeal by being given time slices. The closest one comes to this is when a series of effector codelets' actions *happen* to dovetail so well that the codelets *appear* to have been parts of some predetermined high-level construction process. However, what is deceptive here is that scattered amongst the actions constituting the visible "process", a lot of other codelets — certainly many scouts, and probably other effectors — have played crucial but less visible roles. In any case, there was some degree of luck because randomness played a critical role in bringing about this particular sequence of events. In short, although some large-scale actions tend to look planned in advance, that appearance is illusory; patterns in the processing are all *emergent.*

A useful image here is that of the course of play in a basketball game. Each player runs down the court, zigzagging back and forth, darting in and out of the enemy team as well as their own team, maneuvering for position. Any such move is simultaneously *responding* to a complex constellation of pressures on the floor as well as slightly *altering* the constellation of pressures on the floor. A move is thus fundamentally deeply ambiguous. Although the crowd is mostly concerned with the sequence of players who have the ball, and thus tends to see a localized, serial process unfolding, the players who seldom or never have the ball nonetheless play pivotal roles, in that they mold the globally-felt pressures that control both teams' actions at all moments. A tiny feint of the head or lunge to one side alters the probabilities of all sorts of events happening on the court, both near and far. After a basket has been scored, even though sports announcers and fans always try to account for the structure of the event in clean, spatially local, temporally serial terms (thus trying to impose a *process* on the event), in fact the event was in an essential way distributed all over space and time, amongst all the players. The event consisted of distributed, swiftly shifting pressures pushing *for* certain types of plays and *against* others, and impositions of locality and seriality, though they contain some truth, are merely ways of simplifying what happened for the sake of human consumption. The critical point to hold onto here is the *ambiguity* of any particular action en route to a basket; each action contributes to many potential continuations and cannot be thought of as a piece of some unique "process" coexisting with various other independent "processes" supposedly taking place on the court.

Much the same could be said for Copycat: an outside observer is free, after a run is over, to "parse" the run in terms of specific, discrete processes, and to attempt to impose such a vocabulary on the system's behavior; however, that parsing and labeling is not intrinsic to the system, and such interpretations are in no way unique or absolute, any more than in a basketball game. In other words, a long sequence of codelet actions can add up to what could be perceived, *a posteriori* and by an outsider, as a single coherent drive towards a particular goal, but that is the outsider's subjective interpretation.

The parallel terraced scan

One of the most important consequences of the commingling of multiple pressures is the *parallel terraced scan.* The basic image is that of many "fingers of exploration" simultaneously feeling out various potential pathways at different speeds, thanks to the coexistence of pressures of different strengths. These "fingers of exploration" are tentative probes made by scout codelets, rather than actual events realized by effector codelets. In the Workspace, there is only one *actual* viewpoint at any given time. However, in the background, a host of nearby variants of the actual viewpoint — *virtual* viewpoints — are constantly flickering probabilistically. If any virtual viewpoint is found sufficiently promising by scouts, then they create effector codelets that, when run, will attempt to realize that alternative viewpoint in the Workspace. This entails a "fight" between the incumbent structure and the upstart; the outcome is decided probabilistically, with the weights being determined by the strength of the current structure as opposed to the promise of the rival.

This is how the system's actual viewpoint develops with time. There is always a probabilistic "halo" of many *potential* directions being explored; the most attractive of these tend to be the *actual* directions chosen. Incidentally, this aspect of Copycat reflects the psychologically important fact that conscious experience is essentially unitary, although it is of course an outcome of many parallel unconscious processes.

A metaphor for the parallel terraced scan is provided by the image of a vast column of ants marching through a forest, with hordes of small scouts at the head of the column making small random forays in all directions (although exploring some directions more eagerly and deeply than others) and then returning to report; the collective effect of these many "feelers" will then determine the direction to be followed by the column as a whole. This is going on at all moments, of course, so that the column is constantly adjusting its pathway in slight ways.

The term "parallel terraced scan" comes from the fact that scouting expeditions are structured in a *terraced* way; that is, they are carried out in stages, each stage contingent upon the success of the preceding one, and probing a

little more deeply than the preceding one. The first stage is computationally cheap, so the system can afford to have many first-stage scouts probing in all sorts of directions, including quite unlikely directions. Succeeding stages are less and less cheap; consequently the system can afford fewer and fewer of them, which means it has to be increasingly selective about the directions it devotes resources to looking in. Only after a pathway has been deeply explored and found to be very promising are effector codelets created, which then will try to actually swerve the whole system down that pathway.

The constellation of top-down pressures at any given time controls the biases in the system's exploratory behavior, and also plays a major role in determining the actual direction the system will move in; ultimately, however, top-down pressures, no matter how strong, must bow to the reality of the situation itself, in the sense that prejudices alone cannot force inappropriate concepts to fit to reality. Top-down pressures must adapt when the pathways they have urged turn out to fail. The model is made explicitly to allow this kind of intermingling of top-down and bottom-up processing.

Time-evolving biases

At the very start of a run, the Coderack contains exclusively bottom-up similarity-scanners, which represent no situation-specific pressures. In fact, it is their job to make small discoveries that will then start generating such pressures. As these early codelets run, the Workspace starts to fill up with bonds and small groups and, in response to these discoveries, certain nodes in the Slipnet are activated. In this way, situation-specific pressures are generated and cause top-down codelets to be spawned by concepts in the Slipnet. Thus top-down codelets gradually come to dominate the Coderack.

At the outset of a run, the Slipnet is "neutral" (*i.e.*, in a standard configuration with a fixed set of concepts of low depth activated), meaning that there are no situation-specific pressures. At this early stage, all observations made in the Workspace are very local and superficial. Over the course of a run, the Slipnet moves away from its initial neutrality and becomes more and more biased toward certain organizing concepts — *themes* (highly activated deep concepts, or constellations of several such concepts). Themes then guide processing in many pervasive ways, such as determining the saliences of objects, the strengths of bonds, the likelihood of various types of groups to be made, and in general, the urgencies of all types of codelets.

It should not be imagined, incidentally, that a "neutral" Slipnet embodies no biases whatsoever; it certainly does (think of the permanent inequality of various nodes' conceptual depths, for instance). The fact that at the outset, a sameness group is likely to be spotted and reified faster than a successor group of the same length, for instance, represents an initial bias favoring sameness

over successorship. The important thing is that at the outset of a run, the system is more open than at any other time to *any* possible organizing theme (or set of themes); as processing takes place and perceptual discoveries of all sorts are made, the system loses this naïve, open-minded quality, as indeed it ought to, and usually ends up being "closed-minded" — that is, strongly biased towards the pursuit of some initially unsuspected avenue.

In the early stages of a run, almost all discoveries are on a very small, local scale: a primitive object acquires a description, a bond is built, and so on. Gradually, the scale of actions increases: small groups begin to appear, acquire their own descriptions, and so on. In the later stages of a run, actions take place on an even larger scale, often involving complex, hierarchically structured objects. Thus, over time there is a clear progression, in processing, from locality to globality.

Temperature as a regulator of open-mindedness

At the start of a run, the system is open-minded, and for good reason: it knows nothing about the situation it is facing. It doesn't matter all that much *which* codelets run, since one wants many different directions to be explored; hence decision-making can be fairly capricious. However, as swarms of scout codelets and local effector codelets carry out their jobs, that status gradually changes; in particular, as the system acquires more and more information, it starts creating a coherent viewpoint and focusing in on organizing themes. The more informed the system is, the more important it is that top-level decisions not be capriciously made. For this reason, there is a variable that monitors the stage of processing, and helps to convert the system from its initial largely bottom-up, open-minded mode to a largely top-down, closed-minded one. This variable is given the name *temperature.*

What controls the temperature is the *degree of perceived order* in the Workspace. If, as at the beginning of every run, no structures have been built, then the system sees essentially no order, which translates into a need for broad, open-minded exploration; if, on the other hand, there is a highly coherent viewpoint in the Workspace, then the last thing one wants is a lot of voices clamoring for irrelevant actions in the Workspace. Thus, temperature is essentially an inverse measure of the *quality of structure* in the Workspace: the more structures there are, and the more coherent they are with one another (as measured by their strengths), the lower the temperature. Note that although the overall trend is for temperature to wind up low at the end of a run, a *monotonic* drop in temperature is not typical; often, the system's temperature goes up and down many times during a run, reflecting the system's uncertain advances and retreats as it builds and destroys structures in its attempts to home in on the best way to look at a situation.

What the temperature itself controls is the *degree of randomness used in decision-making*. Decisions of every sort are affected by the temperature — which codelet to run next, which object to focus attention on, which of two rival structures should win a fight, and so on. Consider a codelet, for instance, trying to decide where to devote its attention. Suppose that Workspace object A is exactly twice as salient as object B. The codelet will thus tend to be more attracted to A than to B. However, the precise discrepancy in attractive power between A and B will depend on the temperature. At some mid-range temperature, the codelet will indeed be twice as likely to go for A as for B. However, at very *high* temperatures, A will be hardly any more attractive than B to the codelet. By contrast, at very *low* temperatures, the probability of choosing A over B will be much greater than two to one. For another example, consider a codelet trying to build a structure that is incompatible with a currently existing strong structure. Under low-temperature conditions, the strong structure will tend to be very stable (*i.e.*, hard to dislodge), but if the temperature should happen to rise, it will become increasingly susceptible to being swept away. In "desperate times", even the most huge and powerful structures and worldviews can topple.

The upshot of all this is that at the start of a run, the system explores possibilities in a wild, scattershot way; however, as it builds up order in the Workspace and simultaneously homes in on organizing themes in the Slipnet, it becomes an increasingly conservative decision-maker, ever more deterministic and serial in its style. Of course, there is no magic crossover point at which nondeterministic parallel processing turns into deterministic serial processing; there is simply a gradual tendency in that direction, controlled by the system's temperature.

Note that the notion of temperature in Copycat differs from that in simulated annealing, an optimization technique sometimes used in connectionist networks (Kirkpatrick, Gelatt, & Vecchi, 1983; Hinton & Sejnowski, 1983; Smolensky, 1983). In simulated annealing, temperature is used exclusively as a top-down randomness-controlling factor, its value falling monotonically according to a predetermined, rigid "annealing schedule". By contrast, in Copycat, the value of the temperature reflects the current quality of the system's understanding, so that temperature acts as a *feedback mechanism* that determines the degree of randomness used by the system. Thus, the system itself controls the degree to which it is willing to take risks.

Long after the concept of temperature had been conceived and implemented in the program, it occurred to us that temperature could serve an extra, unanticipated role: the *final* temperature in any run could give a rough indication of how good the program considered its answer to be (the lower the temperature, of course, the more desirable the answer). The idea is simply that the quality of an answer is closely correlated with the amount of strong, coherent

structure underpinning that answer, and temperature is precisely an attempt to measure that quantity. From the moment we realized this, we kept track of the final temperatures of all runs, and those data provided some of the most important insights into the program's "personality", as will be apparent when we discuss in detail the results of runs.

Overall trends during a run

In most runs, despite local fluctuations here and there, there is a set of overall tendencies characterizing how the system evolves in the course of time. These tendencies, although they are all tightly linked together, can be roughly associated with different parts of the architecture, as follows.

- In the Slipnet, there is a general tendency for the initially activated concepts to be *conceptually shallow*, and for concepts that get activated later to be increasingly *deep*. There is also a tendency to move from *no themes* to *themes* (*i.e.*, clusters of highly activated, closely related, high-conceptual-depth concepts).
- In the Workspace, there is a general tendency to move from a state of *no structure* to a state with *much structure*, and from a state having *many local, unrelated objects* to a state characterized by *few global, coherent structures*.
- As far as the processing is concerned, it generally exhibits, over time, a gradual transition from *parallel* style toward *serial* style, from *bottom-up* mode to *top-down* mode, and from an initially *nondeterministic* style toward a *deterministic* style.

The Intimate Relation between Randomness and Fluidity

It may seem deeply counterintuitive that randomness should play a central role in a computational model of intelligence. However, careful analysis shows that it is inevitable if one believes in any sort of parallel, emergent approach to mind.

Biased randomness gives each pressure its fair share

A good starting point for such analysis is to consider the random choice of codelets (biased according to their urgencies) from the Coderack. The key notion, stressed in earlier sections, is that the urgency attached to any codelet represents the estimated proper *speed* at which to advance the pressures for which it is a proxy. Thus it would make no sense at all to treat higher urgencies as higher *priorities* — that is, always to pick the highest-urgency codelets first. If one were to do that, then lower-urgency codelets would never get run at all, so

the effective speeds of the pressures they represent would all be zero, which would totally defeat the notion of commingling pressures, the parallel terraced scan, and temperature.

A more detailed analysis is the following. Suppose we define a "grass-roots" pressure as a pressure represented by a large swarm of low-urgency codelets, and an "elite" pressure as one represented by a small coterie of high-urgency codelets. Then a policy to select high-urgency codelets most of the time would arbitrarily favor elite pressures. In fact, it would allow situations wherein any number of grass-roots pressures could be entirely squelched by just *one* elite pressure — even if the elite pressure constituted but a small fraction of the *total* urgency (the sum of the urgencies of all the codelets in the Coderack at the time), as it most likely would. Such a policy would result in a very distorted image of the overall makeup of the Coderack (*i.e.*, the distribution of urgencies among various pressures). In summary, it is imperative that during a run, low-urgency codelets get mixed in with higher-urgency codelets, and in the right proportion — namely, in the proportions dictated by urgencies, no more and no less. As was said earlier, only by using probabilities to choose codelets can the system achieve (via statistics) a *fair* allocation of resources to each pressure, even when the strengths of various pressures change as processing proceeds.

Randomness and asynchronous parallelism

One might well imagine that the need for such randomness (or biased nondeterminism) is simply an artifact of this architecture's having been designed to run on a sequential machine; were it redesigned to run on parallel hardware, then all randomness could be done away with. This turns out to be not at all the case, however. To see why, we have to think carefully about what it would mean for this architecture to run on parallel hardware. Suppose that there were some large number of parallel processors to which tasks could be assigned, and that each processor's speed could be continuously varied. It is certainly not the case that one could assign *processes* to *processors* in a one-to-one manner, since, as has been stressed, there is no clear notion of "process" in this architecture. Nor could one assign one *pressure* to each processor, since codelets are not univalent as to the pressures that they represent. The only possibility would be to assign a processor to every single codelet, letting it run at a speed defined by that codelet's urgency. (Note that this requires a very large number of co-processors — hundreds, if not thousands. Moreover, since the codelet population varies greatly over time, the number of processors in use at different times will vary enormously. However, on a conceptual level, neither of those poses a problem in principle.)

Now notice a crucial consequence of this style: since all the processors are running at speeds that are completely independent of one another, they are

effectively carrying out *asynchronous* computing, which means that relative to one another, the instants at which they carry out actions in the (shared) Workspace are totally decoupled — in short, entirely random relative to one another. This is a general fact: asynchronous parallelism is inseparable from processors' actions being random relative to one another (as pointed out in Hewitt, 1985). Thus parallelism provides no escape from the inherent randomness of this architecture. When it runs on serial hardware, some *explicit* randomizing device is utilized; when it runs on parallel hardware, the randomness is *implicit*, but no less random for that.

The earlier image of the swiftly-changing panorama of a basketball game may help to make this necessary connection between asynchronous parallelism and randomness more intuitive. Each player might well feel that the snap decisions being made constantly inside their own head are anything but random — that, in fact, their decisions are rational responses to the situation. However, from the point of view of *other* players, what any one player does is not predictable — a player's mind is far too complex to be modeled, especially in real time. Thus, because all the players on the court are complex, independent, asynchronously-acting systems, each player's actions *necessarily* have a random (*i.e.*, unpredictable) quality from the point of view of all the other players. And obviously, the more unpredictable a team seems to its opponents, the better.

A seeming paradox: Randomness in the service of intelligence

Even after absorbing all these arguments, one may still feel uneasy with the proposition that greater intelligence can result from making *random* decisions than from making *systematic* ones. Indeed, when the architecture is described this way, it sounds nonsensical. Isn't it always wiser to choose the *better* action than to choose at *random*? However, as in so many discussions about mind and its mechanisms, this appearance of nonsensicality is an illusion caused by a confusion of levels.

Certainly it would seem extremely counterintuitive — in fact, downright nonsensical — if someone suggested that a melody-composition program (say) should choose its next note by throwing dice, even weighted dice. How could any global coherence come from such a process? This objection is of course totally valid — good melodies cannot be produced in that way (except in the absurd sense of millions of monkeys plunking away on piano keyboards for trillions of years and coming up with "Blue Moon" once in a blue moon). But our architecture in no way advocates such a coarse type of decision-making procedure!

The choice of next note in a melody is a *top-level* macro-decision, as opposed to a low-level act of "micro-exploration". The purpose of micro-exploration is to efficiently explore the vast, foggy world of possibilities lying ahead without getting

bogged down in a combinatorial explosion; for this purpose, randomness, being equivalent to non-biasedness, is the *most efficient* method. Once the terrain has been scouted out, much information has been gained, and in most cases some macroscopic pathways have been found to be more promising than others. Moreover — and this is critical — the more information that has been uncovered, the more the temperature will have dropped — and the lower the temperature is, the less randomness is used. In other words, the more confidently the system believes, thanks to lots of efficient and fair micro-scouting in the fog, that it has identified a particular promising pathway ahead, the more certain it is to make the macro-decision of picking that pathway. Only when there is tight competition is there much chance that the favorite will not win, and in such a case, it hardly matters since even after careful exploration, the system is not persuaded that there is a clear best route to follow.

In short, in the Copycat architecture, hordes of random forays are employed on a microscopic level when there is a lot of fog ahead, and their purpose is precisely to get an evenly-distributed sense of what lies out there in the fog rather than simply plunging ahead blindly, at random. The foggier things are, the more unbiased should be the scouting mission, hence the more randomness is called for. To the extent that the scouting mission succeeds, the temperature will fall, which in turn means that the well-informed macroscopic decision about to be taken will be made *non*-randomly. Thus, randomness is used *in the service of,* and not in opposition to, intelligent nonrandom choice.

A subtle aspect of this architecture is that there are all shades between complete randomness (much fog, high temperature) and complete determinism (no fog, low temperature). This reflects the fact that one cannot draw a clean, sharp line between micro-exploratory scouting forays and confident, macroscopic decisions. For instance, a smallish, very local building or destruction operation carried out in the Workspace by an effector codelet working in a mid-range temperature can be thought of as lying somewhere in between a micro-exploratory foray and a well-informed macroscopic decision.

As a final point, it is interesting to note that non-metaphorical fluidity — that is, the physical fluidity of liquids like water — is inextricably tied to random microscopic actions. A liquid could not flow in the soft, gentle, *fluid* way that it does, were it not composed of tiny components whose micro-actions are completely random relative to one another. This does not, of course, imply that the top-level action of the fluid *as a whole* takes on any appearance of randomness; quite the contrary! The flow of a liquid is one of the most nonrandom phenomena of nature that we are familiar with. This does not mean that it is by any means *simple*; it is simply familiar and natural-seeming. Fluidity is an emergent quality, and to simulate it accurately requires an underlying randomness.

Copycat's Performance: A Forest-level Overview

The statistically emergent robustness of Copycat

Now that the architecture of the Copycat program has been laid out, we can take a tour through the program's performance on a number of problems in its letter-string microworld. As was discussed earlier, Copycat's microworld was designed to isolate, and thus to bring out very clearly, some of the essential issues in high-level perception and analogy-making in general. The program's behavior on the problems presented here demonstrates how it deals with these issues, how it responds to variations in pressures, and how it is able, starting from exactly the same state on each new problem, to fluidly adapt to a range of different situations.[2] (The program's performance on a much larger set of problems and some comparisons with people's performance on the same problems are given in Mitchell, 1993.)

On any given run on a particular problem, the program settles on a specific answer; however, since the program is permeated with nondeterminism, different answers (to the same problem) are possible on different runs. The nondeterministic decisions the program makes (*e.g.*, which codelet to run next, which objects a codelet should act on, etc.) are all at a microscopic level, compared with the macroscopic-level decision of what answer to produce on a given run. Every run is different at the microscopic level, but statistics lead to far more deterministic behavior at the macroscopic level. For example, there are a huge number of possible routes (at the microscopic level of individual codelets and their actions) the program can take to arrive at the solution ***ijl*** to Problem 1, and a large number of micro-biases tend to push the program down one of those routes rather than down one of the huge number of possible routes to ***ijd***. Thus in this problem, at a macroscopic level, the program is very close to being deterministic: it gets the answer ***ijl*** almost all the time.

The phenomenon of macroscopic determinism emerging from microscopic nondeterminism is often demonstrated in science museums by means of a contraption in which several thousand small balls are allowed to tumble down, one by one, through a regular grid of horizontal pins that run between two parallel vertical plexiglass sheets. Each ball, as it falls, bounces helter-skelter off various pins, eventually winding up in one of some 20 or 30 adjacent equal-sized bins forming a horizontal row at the bottom. As the number of balls that have

2. The current version of the Copycat program can deal only with problems whose initial change involves a replacement of at most one letter (*e.g.*, ***abc*** ⇒ ***abd***, or ***aabc*** ⇒ ***aabd***; of course the ***answer*** can involve a change of more than one letter, as in ***aabc*** ⇒ ***aabd***; ***ijkk*** ⇒ ***ijll***). This is a limitation of the program as it now stands; in principle, the letter-string domain is much larger. But even given this limitation, a very large number of interesting problems can be formulated, requiring considerable mental fluidity. (For a good number of examples of such problems, see Hofstadter, 1984b or Mitchell, 1993.)

fallen increases, the stacks of balls in the bins grow. However, not all bins are equally likely destinations, so different stacks grow at different rates. In fact, the heights of the stacks in the bins at the bottom gradually come to form an excellent approximation to a perfect gaussian curve, with most of the balls falling into the central bins, and very few into the edge bins. This reliable buildup of the mathematically precise gaussian curve out of many unpredictable, random events is fascinating to watch.

In Copycat, the set of bins corresponds to the set of different possible answers to a problem, and the precise pathway an individual ball follows, probabilistically bouncing left and right many times before "choosing" a bin at the bottom, corresponds to the many stochastic micro-decisions made by the program (at the level of individual codelets) during a single run. Given enough runs, a reliably repeatable pattern of answer frequencies will emerge, just as a near-perfect gaussian curve regularly emerges in the bins of a "gaussian pinball machine".

Copycat's "personality" is revealed through bar graphs

We present these patterns in the form of bar graphs, one for each problem, giving the frequency of occurrence (representing surface appeal) and the average end-of-run temperature (representing quality) for each different answer. For each problem, a bar graph is given, summarizing 1,000 runs of Copycat on that problem. The number 1,000 is somewhat arbitrary; after about 100 runs on each problem, the basic statistics do not change much. The only difference is that as more and more runs are done on a given problem, certain bizarre and improbable "fringe" answers, such as ***ijj*** in Problem 1 (see Figure V–1), begin to appear very occasionally; if 2,000 runs were done on Problem 1, the program would give perhaps one or two other such answers, each once or twice. This allows the bar graphs to make a very important point about Copycat: even though the program has the potential to get strange and crazy-seeming answers, the mechanisms it has allow it to steer clear of them almost all of the time. It is critical that the program (as well as people) be allowed the *potential* to follow risky (and perhaps crazy) pathways, in order for it to have the flexibility to follow *insightful* pathways, but it also has to avoid following bad pathways, at least most of the time.

In the bar graph of Figure V–1, each bar's height gives the relative frequency of the answer it corresponds to, and printed above each bar is the actual number of times that answer was given. The average final temperature appears below each bar. The frequency of a given answer can be thought of as an indicator of how *obvious* or *immediate* that answer is, given the biases of the program. For example, ***ijl***, produced 980 times, is much more immediate to the program than ***ijd***, produced 19 times, which is in turn much more obvious than the strange answer ***ijj***, produced only once. (To get the latter answer, the program decided to replace the rightmost letter by its predecessor rather than

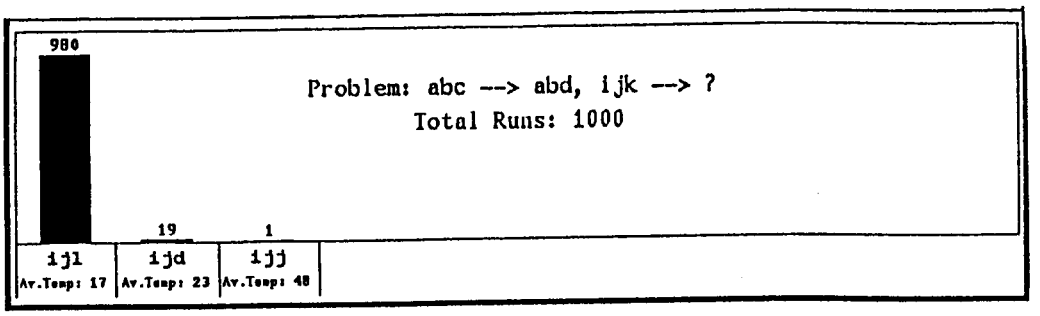


*Figure V–1. Bar graph summarizing 1,000 runs of the Copycat program on the analogy problem "**abc** ⇒ **abd**; **ijk** ⇒ ?".*

its successor. This slippage is always possible in principle, since *successor* and *predecessor* are linked in the Slipnet. However, as can be seen by the rarity of this answer, it is extremely unlikely in this situation: under the pressures evoked by this problem, *successor* and *predecessor* are almost always considered too distant for a slippage to be made between them.)

Although the frequencies shown in Figure V–1 seem quite reasonable, it is not intended that they should precisely reproduce the frequencies one would find if this problem were posed to humans, since, as we said earlier, the program is not meant to model all the domain-specific mechanisms people use in solving these letter-string problems. Rather, what is interesting here is that the program does have the potential to arrive at very strange answers (such as ***ijj***, but also many others), yet manages to steer clear of them almost all the time.

As we said earlier, the average final temperature of an answer can be thought of as the program's own assessment of that answer's *quality*, with lower temperatures meaning higher quality. For instance, the program assesses ***ijl*** (average final temperature 17) to be of somewhat higher quality than ***ijd*** (temperature 23), and of much higher quality than ***ijj*** (temperature 48).

One can get a sense for what a given numerical value of temperature represents by seeing how various sets of perceptual structures built by the program affect the temperature. This will be illustrated in the next section, when a detailed set of screen dumps from a run of Copycat is presented. Roughly speaking, an average final temperature below 30 indicates that the program was able to build a fairly strong, coherent set of structures — that it had, in some sense, a reasonable "understanding" of what was going on in the problem. Higher final temperatures usually indicate that some structures were weak, or perhaps that there was no coherent way of mapping the initial string onto the target string.

The program decides probabilistically when to stop running and produce an answer, and although it is much more likely to stop when the temperature is low, it sometimes stops before it has had an opportunity to build all appropriate structures. For example, there are runs on Problem 1 in which the program

stops before the target string has been grouped as a whole; the answer is still often ***ijl***, but the final temperature is higher than it would have been if the program had continued. This kind of run increases the average final temperature for this answer. The lowest possible temperature for answer ***ijl*** is about 7, which is about as low as the temperature ever gets.[3]

Systematically studying the effects of variant problems

Systematic studies can be done in which a given problem is slightly altered in various ways. Each such variant tampers with the pressures that the original problem evokes, and one can expect effects of this to show up in the bar graph for that problem. For instance, Problem 2, discussed above, is a variant of Problem 1 in which the doubling of letters shifts the "stresses" in the strings ***abc*** and ***ijk***; one might expect this to make the ***aa*** and the ***kk*** far more salient and more similar to each other than the ***a*** and ***k*** were in Problem 1, thus pushing towards a crosswise mapping in which the two double letters correspond.

In Figure V–2, one sees that despite the pressure towards a crosswise mapping, the "Replace rightmost group by successor" answer (***ijll***) is still the most common answer and the "Replace rightmost letter by successor" answer (***ijkl***) is second, indicating the lingering appeal of the straightforward *leftmost* ⇒ *leftmost, rightmost* ⇒ *rightmost* view, even here. However, the pressure is felt to some extent: ***jjkk*** makes a good showing and ***hjkk*** has some representatives too, as well as having by far the lowest average temperature. (This is to be contrasted with the results on Problem 1: note that in 1,000 runs, the program *never* gave an answer involving a replacement of the leftmost letter.) The answers on the fringe here include ***jkkk*** (which is similar to ***jjkk***, but results from grouping the *two* leftmost letters — a far-fetched and, to most people, unappealing way of "parsing" the string); ***ijkd*** and ***ijdd*** (both based on the rule "Replace the rightmost letter by ***d***", but flexed in different ways because of different bridges built from the ***c***); ***ijkk*** (replacing all ***c***'s by ***d***'s); and ***djkk*** (replacing the ***i*** by a ***d*** instead of by its successor or predecessor).

Another variant on Problem 1 is the following:

3. Suppose the letter-string ***abc*** were changed to ***abd***; how would you change the letter-string ***kji*** in "the same way"?

Here a literal application of the original rule ("Replace rightmost letter by successor") would yield ***kjj***, which ignores an abstract similarity between ***abc***

3. There is a problem with the way temperature is calculated in the program as it now stands. As can be seen, the answer ***ijd*** has an average final temperature almost equal to that of ***ijl*** (even though it is much less frequent), whereas most people feel it is a far worse answer. This, along with other problems with the current program, is discussed in detail in Mitchell, 1993.

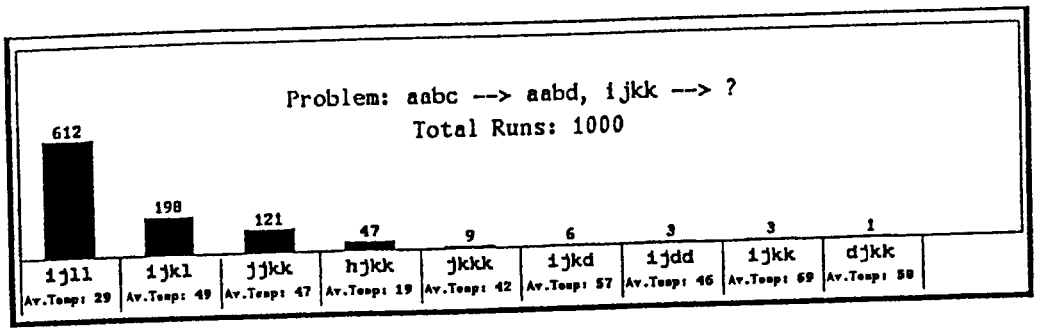


*Figure V–2. Bar graph summarizing 1,000 runs of the Copycat program on the analogy problem "**aabc** ⇒ **aabd**; **ijkk** ⇒ ?".*

and ***kji***. An alternative many people prefer is ***lji*** ("Replace the leftmost letter by its successor"), which is based on seeing both strings in terms of a *successorship* fabric, in one string running to the right and in the other one to the left; thus there is a slippage from the concept *right* to the concept *left*, which in turn gives rise to the "cousin" slippage *rightmost* ⇒ *leftmost*. Another answer given by many people is ***kjh*** ("Replace the rightmost letter by its predecessor"), in which one string is seen as having a *successorship* fabric and the other as having a *predecessorship* fabric (both viewed as moving in the same spatial direction), thus involving a slippage of the concept *successor* into the concept *predecessor*.

As can be seen in Figure V–3, there are three answers that predominate, with ***kjh*** being the most common (and having the lowest average final temperature), and ***kjj*** and ***lji*** almost tying for second place (the latter being a bit less common, but having a much lower average final temperature). The answer ***kjd*** comes in a very distant fourth, and then there are two "fringe" answers with but one instance apiece: ***dji*** (an implausible blend of insight and rigidity in which the opposite spatial direction of the two successor groups ***abc*** and ***kji*** was seen, but instead of the leftmost letter being replaced by its successor, it was replaced by a ***d***— and yet, notice the relatively low temperature on this answer, indicating that a strong set of structures was built!), and ***kji*** (reflecting the literal-minded rule "Replace ***c*** by ***d***", where there are no ***c***'s in ***kji***), which has a very high temperature of 89, indicating that on this run, almost no structures were built before the program chanced, against very high odds, to stop.

How hidden concepts emerge from dormancy

Consider now the following problem, which involves a very different set of pressures from those in the previous problems:

4. Suppose the letter-string ***abc*** were changed to ***abd***; how would you change the letter-string ***mrrjjj*** in "the same way"?

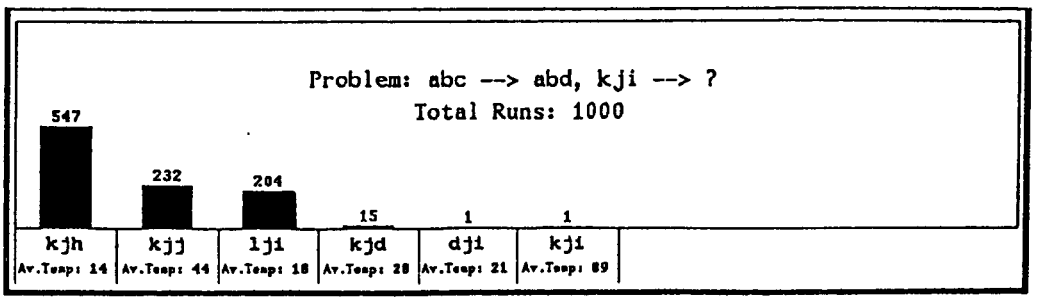


*Figure V–3. Bar graph summarizing 1,000 runs of the Copycat program on the analogy problem "**abc** ⇒ **abd**; **kji** ⇒ ?".*

There is a seemingly reasonable, straightforward solution: ***mrrkkk***. Most people give this answer, reasoning that since ***abc***'s rightmost letter was replaced by its successor, and since ***mrrjjj***'s rightmost "letter" is actually a *group* of ***j***'s, one should replace *all* the ***j***'s by ***k***'s. Another possibility is to take the phrase "rightmost letter" literally, thus replacing only the rightmost ***j*** by ***k***, giving ***mrrjjk***. However, neither answer is very satisfying, since neither takes into account the salient fact that ***abc*** is an alphabetic sequence (*i.e.*, a successor group). This fabric of ***abc*** is an appealing and seemingly central aspect of the string, so one would like to use it in making the analogy, but there is no obvious way to do so. No such fabric seems to weave ***mrrjjj*** together. So either (like most people) one settles for ***mrrkkk*** (or possibly ***mrrjjk***), or one looks more deeply. But where to look, when there are so many possibilities?

The interest of this problem is that there happens to be an aspect of ***mrrjjj*** lurking beneath the surface that, once recognized, yields what many people feel is a deeply satisfying answer. If one ignores the *letters* in ***mrrjjj*** and looks instead at *group lengths,* the desired successorship fabric is found: the lengths of groups increase as "1–2–3". Once this hidden connection between ***abc*** and ***mrrjjj*** is discovered, the rule describing the change ***abc*** ⇒ ***abd*** can be adapted to apply to ***mrrjjj*** as "Replace the length of the rightmost group by its successor", yielding "1–2–4" at the abstract level, or, more concretely, ***mrrjjjj***.

Thus this problem demonstrates how a previously irrelevant, unnoticed aspect of a situation can emerge as relevant in response to pressures. The crucial point is that the process of perception is not just about deciding which *clearly apparent* aspects of a situation should be ignored and which should be taken into account; it is also about the question of how aspects that were initially considered to be irrelevant — or rather, that were initially so far out of sight that they were not even recognized as being irrelevant! — can *become* apparent and relevant in response to pressures that emerge as the understanding process is taking place.

Sometimes, given certain pressures, a concept that one initially had no idea was germane to the situation will emerge seemingly from nowhere and turn out to be exactly what was needed. In such cases, however, one should not feel upset about not having suspected its relevance at the outset. In general, far-out ideas (or even ideas *slightly* past one's defaults) ought not continually occur to people for no good reason; in fact, a person to whom this happens is classified as crazy or crackpot.

Time and cognitive resources being limited, it is vital to resist nonstandard ways of looking at situations without strong pressure to do so. You don't check the street sign at the corner, every time you go outdoors, to reassure yourself that your street's name hasn't been changed. You don't worry, every time you sit down for a meal, that perhaps someone has filled the salt shaker with sugar. You don't worry, every time you start your car, that someone might have stuck a potato in its tailpipe or attached a bomb to its chassis. However, there are pressures — such as receiving a telephone threat on your life — that would make such a normally unreasonable suspicion start to seem reasonable. (These ideas overlap with Kahneman & Miller's 1986 treatment of counterfactuals, and are also closely related to the frame problem in artificial intelligence, as discussed in McCarthy & Hayes, 1969.)

Not only is pressure needed to evoke a dormant concept in trying to make sense of a situation, but the concepts brought in are often clearly related to the source of the pressure. For example, when one looks carefully at Problem 4 (as we will do in the next main section), one can see how certain aspects of it create pressures that, acting in concert, stand a decent chance of evoking the concept of *group length*. Some critical aspects of the story (not in any particular order) are these: (1) once successor relations have been noticed in ***abc***, there arises a top-down pressure to look for them in ***mrrjjj*** as well; (2) once the ***rr*** and ***jjj*** sameness groups have been perceived in ***mrrjjj***, the normally dormant concept *length* becomes weakly active and lingers in the background; (3) the perception of these sameness groups leads to top-down pressure to perceive other parts of the same string as sameness groups as well, and the only way this can be done is the unlikely possibility of perceiving the ***m*** as a sameness group consisting of *only one letter*; and (4) after standard concepts have failed to yield progress in making sense of the situation at hand, resistance to bringing in nonstandard concepts decreases.

People occasionally give the answer ***mrrkkkk***, replacing *both* the letter category *and* the group length of the rightmost group by their successors (***k*** and 4, respectively). Despite its interest, this answer confounds aspects of the two situations. What counts in establishing the similarity of ***abc*** and ***mrrjjj*** is their shared successorship fabric. In ***mrrjjj***, that fabric has nothing to do with the specific letters ***m***, ***r***, and ***j***; the letter sequence ***m–r–j*** is merely acting as a *medium* in which the *numerical* sequence "1–2–3" can be expressed. It is thus misguided

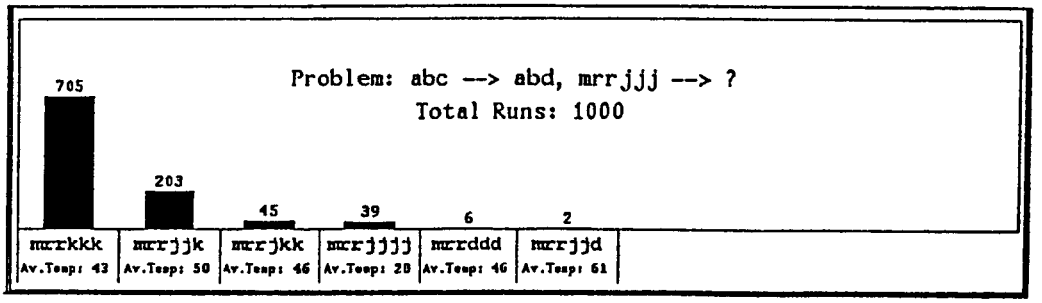


*Figure V–4. Bar graph summarizing 1,000 runs of the Copycat program on the analogy problem "**abc** ⇒ **abd**; **mrrjjj** ⇒ ?".*[4]

to focus on the *alphabetic* level of ***mrrjjj*** when one has just established that the essence of that string, in this context, is not its letters but its higher-level *numerical* structure. If the relations between lengths are perceived, then the answer at the level of lengths is 1–2–4. Translated back into the language of the carriers, this yields ***mrrjjjj***. Additionally converting the four ***j***'s into ***k***'s is gilding the lily: it simply blends the alphabetic view with the numerical view in an inappropriate manner.

People have also proposed answers such as ***mrryyyy***, where the three ***j***'s are replaced by four copies of an arbitrary letter — here, ***y***. The reasoning is that since the successorship fabric in ***mrrjjj*** has nothing to do with the specific letters ***m***, ***r***, and ***j***, it doesn't matter *which* letter-value is used to replace the ***j***'s. Such reasoning is too sophisticated for the current version of Copycat, which does not have the concept "arbitrary letter" — but even if Copycat could produce such an answer, we would still argue that ***mrrjjjj*** is the best answer to this problem. The letters ***m***, ***r***, and ***j*** serve as the medium for expressing the message "1–2–3", and we feel the most elegant solution is one that *preserves* that medium in expressing the modified message "1–2–4". Otherwise, why wouldn't an answer involving *total* replacement of the medium, such as ***uggyyyy***, be just as good as, if not better than, ***mrryyyy***?

As can be seen in Figure V–4, by far the most common answer is the straightforward ***mrrkkk***, with ***mrrjjk*** coming in a fairly distant second. For Copycat, these are the two most immediate answers; however, the average final temperatures associated with them are fairly high, because of the lack of any coherent structure tying together the target string as a whole.

Next come two answers with roughly equal frequencies: ***mrrjkk***, a rather silly answer that comes from grouping only the rightmost two ***j***'s in ***mrrjjj*** and viewing this group as the object to be replaced; and ***mrrjjjj***. The average final

4. The differences between the frequencies given in this figure and those given for the same problem in Mitchell & Hofstadter, 1990b are due to several improvements in the program — in particular, improvements in the way letter-groups and bridges are constructed.

temperature associated with this answer is much lower than that of the other answers, which shows that the program assesses it to be the most satisfying answer, though far from the most immediate. As in many aspects of real life, the immediacy or obviousness of a solution is by no means perfectly correlated with its quality. The other two answers produced in this series of runs, ***mrrddd*** and ***mrrjjd***, come from replacing either a letter or a group with ***d***'s, and are on the fringes.

In Problem 4, the successorship fabric is between group lengths rather than between letters, and is thus not immediately apparent. A simple variant on Problem 4 involves a successorship fabric both at the level of letters and at the level of lengths:

5. Suppose the letter-string ***abc*** were changed to ***abd***; how would you change the letter-string ***rssttt*** in "the same way"?

The strong pressures evoked in Problem 4 by the lack of any alphabetical fabric are missing in this variant, and the effect on Copycat can be seen in the bar graph given in Figure V–5: the program gave the *length* answer ***rssttttt*** only once in 1,000 runs (as contrasted with 39 instances of ***mrrjjjj*** in Problem 4). In Problem 5, the program is much more satisfied with the *letter-level* answer (***rssuuu***), which dominates and has a relatively low average final temperature. The other answers are similar to the answers given in the previous problem (plus there are a few additional answers based on strange groupings of the target string).[5]

Paradigm shifts in a microworld

A different set of issues comes up in the following problem:

6. Suppose the letter-string ***abc*** were changed to ***abd***; how would you change the letter-string ***xyz*** in "the same way"?

Naturally, the focus is on the letter ***z***. One immediately feels challenged by its lack of successor — or more precisely, by the lack of a successor to *Platonic* ***z***, the abstract concept (as opposed to the instance thereof found inside the string ***xyz***). Many people, eager to *construct* a successor to Platonic ***z***, invoke the commonplace notion of circularity, thus conceiving of ***a*** as the successor of ***z***, much as January can be considered the successor of December, the digit '0' the successor of '9', an ace the successor of a king, or, in music, the note A the successor of G. This would yield ***xya***.

Invoking circularity in this way to deal with Problem 6 is a small type of creative leap, and not to be looked down upon. However, the general notion of

5. The current version of Copycat is not able to create two simultaneous bonds between two given objects (*e.g.*, both the *alphabetical* and *numerical* successorship bonds between ***r*** and ***ss***), so the program is at present unable to get what many people consider to be the best answer — namely, ***rssuuuu***.

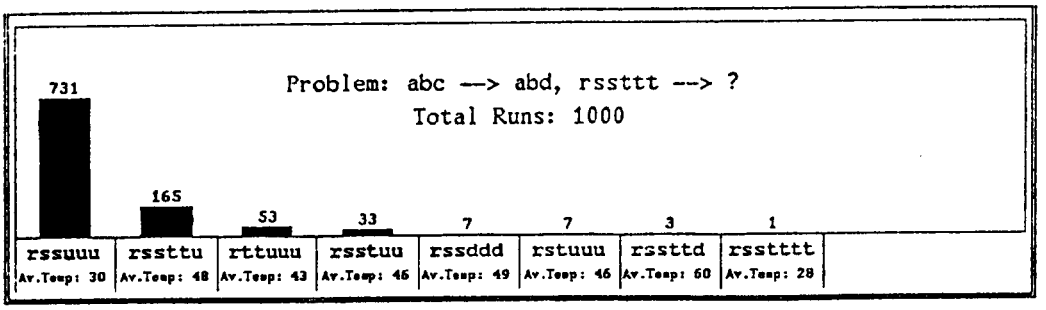


*Figure V–5. Bar graph summarizing 1,000 runs of the Copycat program on the analogy problem "**abc** ⇒ **abd**; **rssttt** ⇒ ?".*

circularity is not available to the program, as it is to people, for borrowing and insertion into the alphabet world. In fact, for important reasons, it is strictly stipulated that Copycat's alphabet is linear and just stops dead at *z*, immutably. We deliberately set this roadblock, because one of our main goals from the very conception of the project was to model the process whereby people deal with impasses.

In response to the *z*'s lack of successor, quite a variety of thoughts can and do occur to people, such as: replace the *z* by nothing at all, thus yielding the answer ***xy***. Or replace the *z* by the literal letter ***d***, yielding answer ***xyd***. (In other circumstances, such a resort to literality would be considered a rigid-minded and rather crude maneuver, but here it suddenly appears quite fluid and certainly reasonable.) Other answers, too, are possible, such as ***xyz*** itself (since the *z* cannot move farther along, just leave it alone); ***xyy*** (since you can't take the *successor* of the *z*, why not take its *predecessor*, which seems like second best?); ***xzz*** (since you can't take the successor of the *z* itself, why not take the successor of the letter sitting next to it?); and many others.

However, there is one particular way of looking at things that, to many people, seems like a genuine insight, whether or not they come up with it themselves. Essentially this is the idea that ***abc*** and ***xyz*** are "mirror images" of each other, each one being "wedged" against its own end of the alphabet. This would imply that the *z* in ***xyz*** corresponds not to the ***c*** but to the ***a*** in ***abc***, and that it is the ***x*** rather than the *z* that corresponds to the ***c***. (Of course, the ***b*** and the ***y*** are each other's counterparts as well.) Underlying these *object* correspondences (*i.e.*, bridges) is a set of three conceptually parallel slippages: *alphabetic-first* ⇒ *alphabetic-last, rightmost* ⇒ *leftmost,* and *successor* ⇒ *predecessor.* Under the profound conceptual reversal represented by these slippages, the raw rule flexes exactly as it did in Problem 2 — namely, into *replace the leftmost letter by its alphabetic predecessor.* This yields the answer ***wyz***, which many people (including the authors) consider elegant and superior to all the other answers proposed above. More than any other answer, it seems to result from *doing the same thing* to ***xyz*** as was done to ***abc***.

Note how similar and yet how different Problems 2 and 6 are. The key idea in both of them is to effect a double reversal (*i.e.*, to reverse one's perception of the target string both spatially and alphabetically). However, it seems considerably easier for people to come up with this insight in Problem 2, even though in that problem there is no "snag", as there is in Problem 6, serving to *force* a search for radical ideas. The very same insight is harder to come by in Problem 6 because the cues are subtler; the resemblance between ***a*** and ***z*** lurks far beneath the surface, whereas the resemblance between ***aa*** and ***kk*** is quite immediate.

In a sense, answer ***wyz*** to Problem 6 seems like a miniature "conceptual revolution" or "paradigm shift" (Kuhn, 1970), whereas answer ***hjkk*** to Problem 2 seems elegant but not nearly as radical. Any model of mental fluidity and creativity must faithfully reflect this notion of distinct "levels of subtlety". We will return to these issues of snags, cues, radical perceptual shifts, and levels of subtlety in the next main section, where we discuss the way in which Copycat succeeds, at least occasionally, in carrying out this miniature paradigm shift.

As can be seen in Figure V–6, the most common answer by far is ***xyd***, for which the program decides that if it can't replace the rightmost letter by its *successor*, the next best thing is to replace it by a ***d***. This is also an answer that people frequently give when told the ***xya*** avenue is barred.

A distant second in frequency, but the answer with the lowest average final temperature, is ***wyz***, which, as was said above, is based on simultaneous spatial and alphabetic reversal in perception of the target string. This discrepancy between rank-order by obviousness and rank-order by quality is characteristic of problems where creative insight is needed. Clearly, brilliance will distinguish itself from mediocrity only in situations where deep ideas are elusive.

To bring about pressures that get the idea of the double reversal to bubble up, radical measures must be taken upon encountering the "*z*-snag" (the moment when the attempt to take the successor of *z* fails). These include sharply focusing attention upon the trouble spot, and raising the temperature from its rather low value just before the *z*-snag is hit to its maximum possible value of 100, opening up a far wider range of possible avenues of exploration. Only with the special combination of a sharp focus of attention and an unusually "broad-minded" attitude could ***wyz*** ever be found. (We will discuss all this in more detail in the next section.)

The next answer, ***yyz***, reflects a view that sees the two strings as mapping to each other in a crosswise fashion, but ignores their opposite alphabetic fabrics; thus, while it considers the *leftmost* letter as the proper one in ***xyz*** to be changed, it clings to the notion of replacing it by its *successor*, since the letter changed in ***abc*** was replaced by *its* successor. (It is Problem 6's analogue to the answer ***jjkk*** in Problem 2.) Although this view seems somewhat inconsistent, like a good

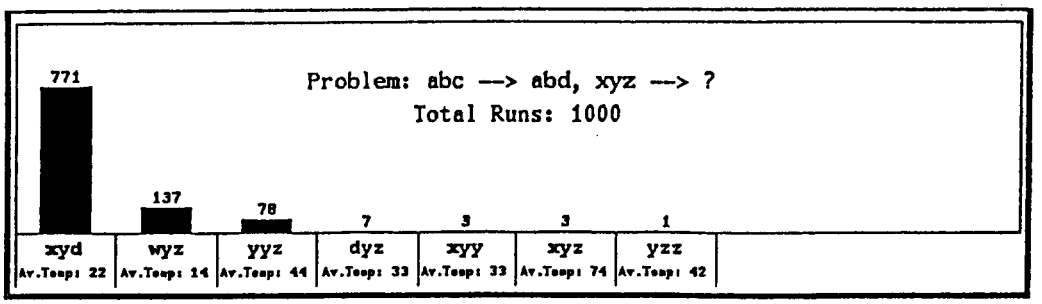


*Figure V–6. Bar graph summarizing 1,000 runs of the Copycat program on the analogy problem "**abc** ⇒ **abd**; **xyz** ⇒ ?".*

idea carried out only halfway, people very often come up with it. Indeed, such blends and half-completed trains of thought are very characteristic of human cognition (Hofstadter & Moser, 1989 gives examples of many types of cognitive blends and discusses their origin).

The other four answers are much farther out on the low-frequency fringes. The answer ***dyz*** (much like ***dji*** in Problem 3) is a highly implausible blend of insight and simple-mindedness, the insight being the subtle perception of the abstract symmetry linking ***abc*** and ***xyz***, and the simple-mindedness being the extremely concrete and unimaginative way of conceiving the ***abc*** ⇒ ***abd*** change. Amusingly, this answer is self-descriptive, in that ***dyz*** can be pronounced "dizzy". Indeed, some people find this answer so dizzy in its style of thought that it evokes laughter. In Hofstadter *et al.* (1989), this and several other Copycat analogies are mapped onto real-world jokes, and are thereby used to suggest a theory of "slippage humor", one of whose tenets is that there is a continuum running from sensible through "sloppy" answers and winding up in "dizzy" answers, where "sloppy" and "dizzy" can be given semi-precise definitions in terms of the degree of consistency with which conceptual slippages are carried out.

The answer ***xyy*** allows that the two strings are to be perceived in opposite *alphabetic* directions (thus a *successor* ⇒ *predecessor* slippage), yet refuses to give up the idea that the strings have the same *spatial* direction; it thus insists on changing the *rightmost* letter, as was done to ***abc***. It is amusing to note that ***ijj*** — Problem 1's analogue to this answer[6] — was produced one time in 1,000 runs, even without the pressure of a snag.

6. It is ironic that a claim of analogousness of answers to different Copycat problems (such as the offhand remark made in the text that ***ijj*** in Problem 1 is "the analogue" to ***xyy*** in Problem 6) comes across as objective and unproblematic to most people, despite the fact that many people express doubt about the notion of "rightness" or "wrongness" of letter-string analogies. The fact is, most people *do* have a strong intuitive sense of right and wrong analogies — it's just that when the psychological context is "Solve this analogy puzzle", they put their guard up and become wary of any claims, whereas when the context is "commentary on our program's behavior", they lower their guard and go with their intuitions, without even realizing the change in their attitude.

The answer ***xyz***, whose very high temperature of 74 indicates that the program did not "like" it at all, comes from interpreting the ***abc*** ⇒ ***abd*** change as "Replace ***c*** by ***d***". (This is not nearly as clever as positing that the ***z*** might serve as its *own* successor, which is an entirely different way of justifying this same answer, and one that people fairly often suggest. In fact, when ***xya*** is barred, ***xyz*** is what people come up with most often.) Note that ***ijk***, the analogous answer[6] to Problem 1, was *never* produced. It takes the "desperation" caused by the *z*-snag to allow such strange ideas any chance at all.

Finally, answer ***yzz*** is a peculiar, almost pathological, variant of the above-discussed answer ***yyz***, in which the ***x*** and ***y*** in ***xyz*** are grouped together as one object, which is then replaced *as a whole* by its "successor" (the successor of each letter in the group). Luckily, it was produced only once in 1,000 runs, and was considered a poor answer.

In Problem 6, pressure for a crosswise mapping (leading to the answer ***wyz***) comes both from the existence of an impasse and from the possibility of an appealing way out of that impasse — namely, a high-quality bridge linking instances of the two "distinguished" Platonic letters, ***a*** and ***z***. Suppose that the impasse was retained while the appeal of the "escape route" was greatly reduced — what would be the effect on Copycat's behavior? The following variant explores that question.

7. Suppose the letter-string ***rst*** were changed to ***rsu***; how would you change the letter-string ***xyz*** in "the same way"?

As Figure V–7 shows, ***wyz*** was produced on only 1 percent of the runs, whereas in Problem 6 it was given on almost 14 percent of the runs. Here, there is very little to suggest building a crosswise bridge, because the ***r*** and ***z*** have almost nothing in common, aside from the rather irrelevant fact that the ***r*** is leftmost in its string and the ***z*** rightmost in *its* string — hardly a powerful reason to make an ***r–z*** bridge.

For some perspective, compare this to Problem 1. How much appeal is there to the idea of mapping the leftmost letter of ***abc*** onto the rightmost letter of ***ijk***? Such a crosswise ***a–k*** bridge would result either in the answer ***hjk*** (characterized at the beginning of this article as "unmotivated fluidity"), or possibly in ***jjk*** or ***djk***. However, in 1,000 runs on Problem 1, Copycat never produced any of those answers, nor have we ever run into a human who has suggested any of them as an answer to Problem 1. (Actually, one person once *did* propose ***hjk*** in response to Problem 1, but this was under the influence of having just seen the ***wyz*** answer to Problem 6.) In colloquial terms, answers to Problem 1 based on a crosswise mapping seem completely "off the wall".

In Problem 7, of course, things are different, because there is, after all, a snag, and hence a kind of "desperation". The various emergency measures —

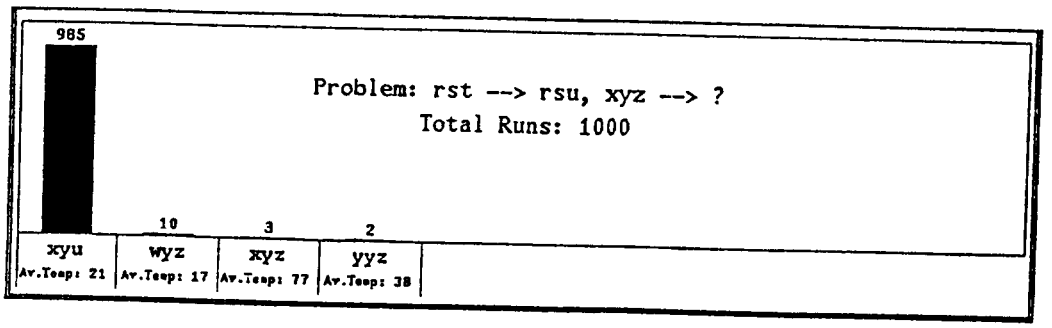


*Figure V–7. Bar graph summarizing 1,000 runs of the Copycat program on the analogy problem "**rst* ⇒ *rsu; xyz* ⇒ *?**".*

especially the persistent high temperature — make the normally unappealing *r–z* bridge a bit more tempting, and so, once in a while, it gets built. From that point on, the whole paradigm shift goes through exactly as in Problem 6, and *wyz* is the outcome.

In some sense, Problem 7 lies halfway between Problems 1 and 6, so answer *wyz* in Problem 7 represents an intermediate stage between unmotivated and motivated fluidity. It is most gratifying to us that Copycat responds to the different constellations of pressures in these problems in much the way that our intuition feels it ought to.

Families of problems as a "miniature Turing Test"

It cannot be overestimated how critical we feel this method of probing Copycat through various families of subtly related problems is. When we began running Copycat on a large number of problems, we had no clear idea of what its performance would be, and we were, frankly, somewhat nervous. To us, the experience of watching Copycat reacting to each new problem had the feel of a kind of "miniature Turing Test", in the sense that each new problem posed to Copycat was like a question and answer in the Turing Test that would inevitably bring out some new and unanticipated aspect of the program's personality (this analogy to the Turing Test is further discussed in French, 1995; see also the Epilogue of this book). Copycat's mechanisms were truly put to the test by the families of problems we challenged it with, and by and large it came through with flying colors. A thorough discussion of those tests is found in Chapters 4 and 5 of Mitchell (1993).

The bar graphs just presented suggest the range of Copycat's abilities, and show how diverse constellations of pressures affect its behavior. They show the degree to which the program exhibits rudimentary fluid concepts able to adapt to different situations in a microworld that, though idealized, captures much of the essence of real-world analogy-making. They also reveal, by displaying *bad* analogies Copycat makes, some of the program's flaws and weaknesses. But they

also show that though Copycat has the *potential* to get far-fetched answers — a potential essential for flexibility — it still manages to *avoid* them almost all the time, which shows its robustness.

It is important to emphasize once again that our goal is not to model specifically how people solve these letter-string analogy problems (it is clear that the microworld involves only a very small fraction of what people know about letters and might use in solving these problems), but rather to propose and model mechanisms for fluid concepts and analogy-making *in general.* These mechanisms, described earlier on, will be illustrated in detail in the next section, which follows the temporal progression of Copycat as it solves two problems. First we take a particular run of the program on Problem 4 and present it through a series of screen dumps; then we move to Problem 6 and discuss the abstract pathway followed by almost all runs that lead to answer ***wyz***.

Copycat's Performance: A Tree-level Close-up

A problem where perception plays a crucial role is chosen as a focus

We now illustrate the mechanisms described in this article by presenting a detailed set of screen dumps from a single run of Copycat on Problem 4. As was discussed above, this problem has a seemingly reasonable, straightforward solution, ***mrrkkk***, but neither this answer nor the more literal ***mrrjjk*** is very satisfying, since neither reflects an underlying successorship structure in ***mrrjjj*** analogous to that in ***abc***. Such a successorship fabric can be found only if relationships between group lengths are perceived in ***mrrjjj***. But how can the notion of *group length,* which in most problems remains essentially dormant, come to be seen as relevant by Copycat?

Length is certainly in the halo of the concept *group,* as are other concepts, such as *letter category* (*e.g.*, ***j*** for the group ***jjj***), *string position* (*e.g., rightmost*), and *group fabric* (*e.g., sameness*). Some of these concepts are more closely associated with *group* than others; in the absence of pressure, the notion of *length* tends to be fairly far away from *group* in conceptual space. Thus in perceiving a group such as ***rr***, one is virtually certain to notice its letter category (namely, ***r***), but not very likely to notice, or at least attach importance to, its length (namely, 2). However, since the concept *length* is in *group*'s halo, there is some chance that lengths will be noticed and used in trying to make sense of the problem. One might, for instance, consciously notice a group's length at some point, but if this doesn't turn out useful, *length*'s relevance will diminish after a short while. (This might happen in the variant problem "***abc*** ⇒ ***abd***; ***mrrrrjj*** ⇒ ?".) This dynamic aspect of relevance is very important: even if a new concept is at some point brought in as relevant, it is counterproductive to continue spend-

ing much of one's time exploring avenues involving that concept if none seems promising.

The story in quick strokes

One way Copycat arrives at ***mrrjjjj*** is now sketched (of course, since the program is nondeterministic, there are many possible routes to any given answer). The input consists of three "raw" strings (here, ***abc***, ***abd***, and ***mrrjjj***) with no preattached bonds or preformed groups; it is thus left to the program to build up perceptual structures constituting its understanding of the problem in terms of concepts it deems relevant.

On most runs, the groups ***rr*** and ***jjj*** get built (the program tends to see sameness groups quite fast). Each group's letter category (***r*** and ***j***, respectively) is noted, as the concept *letter category* is relevant by default. Although there is some chance for length to be noticed when a group is made, it is low, as *length* is only weakly associated with *group*. Once ***rr*** and ***jjj*** are made, *sameness group* becomes very relevant, which creates top-down pressure to describe other objects, especially in the same string, as sameness groups, if possible. The only way to do this here is to describe the ***m*** as a sameness group having just one member. But this is resisted by a strong opposing pressure: a single-member group is an intrinsically weak and far-fetched construct. It would be disastrous if Copycat were willing to bring in unlikely notions such as single-member groups without strong pressure: it would then waste huge amounts of time exploring ridiculous avenues in every problem. However, the prior existence of two other strong sameness groups in the same string, coupled with the system's unhappiness at its failure to incorporate the lone ***m*** into any large, coherent structure (revealed by a persisting high temperature), pushes against this intrinsic resistance.

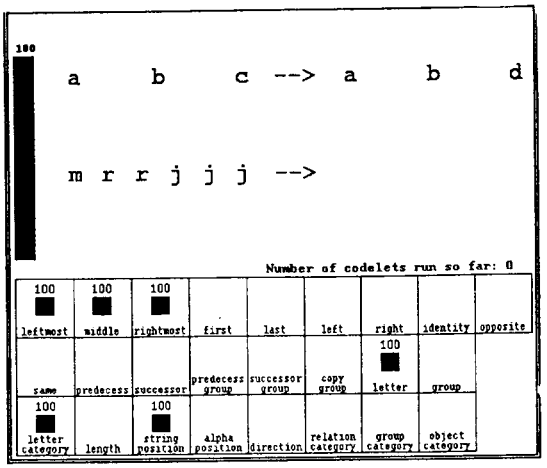
These opposing pressures fight; the outcome is decided only as a statistical result of probabilistic decisions made by a large number of codelets. If the ***m*** chances to be perceived as a single-letter sameness group, that group's length will very likely be noticed (single-letter groups are noteworthy precisely because of their abnormal length), making *length* more relevant in general, and thus increasing the probability of noticing the other two groups' lengths. Moreover, *length,* once brought into the picture, has a good chance of staying relevant, since descriptions based on it turn out to be useful. (Without reinforcement, a node's activation decays over time. Thus, for instance, had the target string been ***mrrrrjj***, *length* might get brought in at some point, but it would not turn out useful, so it would likely fade back into obscurity.)

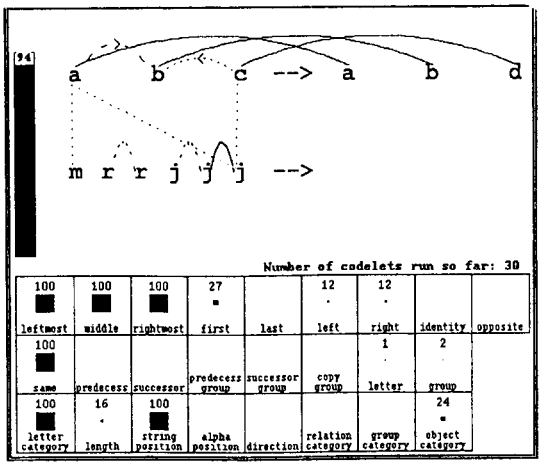
In ***mrrjjj***, once lengths are seen, the (numerical) successor relations among them might be spotted by bottom-up codelets, ever-present in the Coderack, continually seeking new relations in the Workspace. (Note that such

spontaneous bottom-up noticing could happen only in a parallel architecture where many types of properties can be continually being looked for at once without a need for explicit prompting.) Another way, perhaps more likely, that the noticing of successor relations in ***mrrjjj*** could occur is through top-down pressure caused by the already-seen successor relations in ***abc***. In any case, as soon as the numerical successorship relations are seen and a much more satisfying view of ***mrrjjj*** begins to emerge, interest in the groups' letter categories fades and *length* becomes their most salient aspect. Thus the crux of finding this solution lies in the triggering of the concept *length*.

Screen dumps tell the story in detail

Figure V–8 is a series of screen dumps from a run of Copycat, showing one way it arrives at the answer ***mrrjjjj*** (note that this answer is not very typical: according to Figure V–4, this answer is given only about 4 percent of the time).





1. The problem is presented. Temperature, shown on a "thermometer" (at the left), is at its maximum of 100, since no structures have yet been built. At the bottom, some Slipnet nodes are displayed. (Note: links are not shown. Also, due to limited space, many nodes are not shown, *e.g.*, those for ***a***, ***b***, etc.) A black square represents a node's current activation level (the numerical value, between 0 and 100, is shown above the square).

Nodes here displayed include *leftmost, middle,* and *rightmost* (the possible *string positions* of objects in the Workspace); *first* and *last* (the distinguished *alphabetic positions* of Platonic letters ***a*** and ***z***); *left* and *right* (the possible *directions* for bonds and groups); *identity* and *opposite* (two of the possible relations between concepts); *same, predecessor,* and *successor* (the possible *bond categories* for bonds between Workspace objects); *predecessor group, successor group,* and *copy-group* [1] (the various *group categories*); *letter* and *group* (the possible *object categories* for Workspace objects); and in row 3, nodes representing these various categories of descriptions, including *length.*

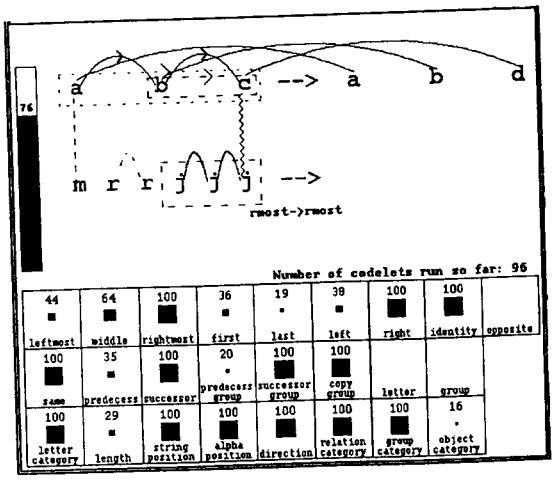
Every letter comes with some preattached descriptions: its *letter category* (*e.g.*, ***m***), its *string position* (*leftmost, middle, rightmost,* or none — *e.g.*, the fourth letter in ***mrrjjj*** has no string-position description), and its *object category* (*letter,* as opposed to *group*). These nodes start out highly activated.

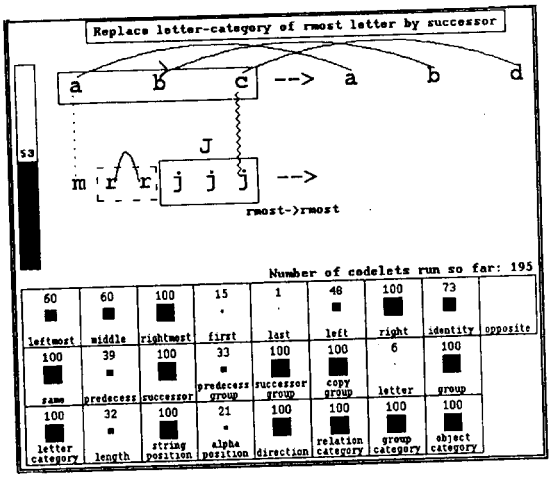
2. The 30 codelets so far run have begun exploring many possible structures. *Dotted* lines and arcs represent structures in early stages of consideration; *dashed* lines and arcs represent structures in more serious stages of consideration; finally, *solid* lines and arcs represent structures actually built, which can thus influence temperature as well as the building of other structures. Various bonds and bridges between letters are being considered (*e.g.*, the dotted ***a–j*** bridge, which is based on the relatively long *leftmost–rightmost* Slipnet link; being implausible, it won't be pursued much further).

Bridges connecting letters in ***abc*** with their counterparts in ***abd*** have been built by bottom-up codelets, as has a ***j–j*** sameness bond at the right end of ***mrrjjj***; this latter discovery activated the node *same,* resulting in top-down pressure (*i.e.*, new codelets) to seek instances of sameness elsewhere.

Some nodes have become lightly activated via spreading activation (*e.g.*, the node *first,* via activation from the node ***a*** [not shown]). The slight activation of *length* comes from its weak association with *letter category* (letters and numbers form linear sequences and are thus similar; numbers are associated with *length*). The temperature has fallen in response to the structures so far built. It should be pointed out that many fleeting explorations are constantly occurring (*e.g.*, "Are there any relations of interest between the ***m*** and its neighbor ***r***?"), but they are not visible here.

1. Note that the term *copy-group,* in this set of screen dumps and captions, means *sameness group,* in the terminology of the text.





3. The successorship fabric of ***abc*** has been observed, and two rival groups based on it are being considered: ***bc*** and ***abc***. Although the former got off to an early lead (it is dashed while the latter is only dotted), the latter, being intrinsically a stronger structure, has a higher chance of actually getting built.

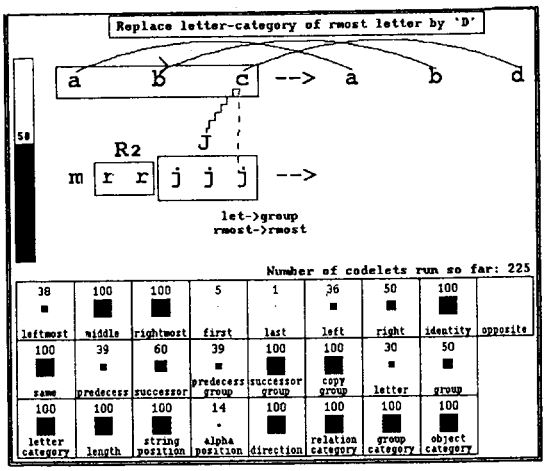
Exploration of the crosswise ***a–j*** bridge was aborted, since it was (probabilistically) judged to be too weak to merit further consideration. A more plausible ***c–j*** bridge has been built (jagged vertical line); its reason for existence (namely, both letters are *rightmost* in their respective strings) is given beneath it, in the form of an identity mapping.

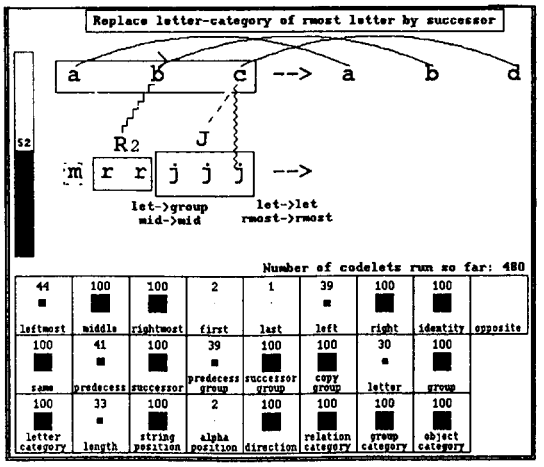
Since *successor* and *sameness* bonds have been built, these nodes are highly active; they in turn have spread activation to *successor group* and *copy-group* (*i.e., sameness group*), which creates top-down pressure to look for such groups. Indeed, a ***jjj*** copy-group is being strongly considered (dashed box). Also, since *first* was active, *alphabetic position* became highly active (a probabilistic event), making alphabetic-position descriptions likely to be considered.

4. Groups ***abc*** and ***jjj*** have been built (the bonds between their letters still exist, but for the purposes of graphical simplicity are no longer being displayed). An ***rr*** copy-group is being considered. The already-built copy-group ***jjj*** strongly supports this potential move, which accelerates it, in the sense that codelets investigating the potential structure will be assigned higher urgency values.

Meanwhile, a *rule* (shown at the top of the screen) has been constructed to describe how ***abc*** changed. The current version of Copycat assumes that the example change involved the replacement of exactly one letter, so rule-building codelets fill in the template "Replace ____ by ____", choosing probabilistically from descriptions that the program has attached to the changed letter and its replacement, with a probabilistic bias toward choosing more abstract descriptions (*e.g.*, usually preferring *rightmost letter* to *c*).

Since the nodes *first* and *alphabetic position* didn't turn out useful, they have faded. Also, although *length* received additional activation from *group*, it is still not very activated, and so lengths are still unlikely to be noticed.





5. Now, some 225 codelets into the run, the letter-to-*letter* ***c–j*** bridge has been defeated by the stronger letter-to-*group* ***c–J*** bridge, although the former possibility still lurks in the background. Meanwhile, an ***rr*** copy-group has been built whose length (namely, 2) happened to be noticed (a probabilistic event); therefore, a "2", along with the group's letter category (namely, ***r***), is displayed just above the group. *Length* is now fully active, and for this reason the "2" is a salient Workspace object (indicated by boldface).

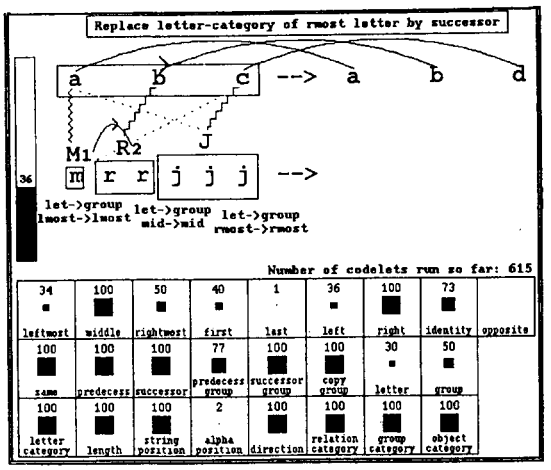
A new rule, "Replace the letter category of the rightmost letter by ***d***", has replaced the old one at the top of the screen. Although this rule is weaker than the previous one, fights between rival structures (including rules) are decided probabilistically, and this one simply happened to win. However, its weakness has caused the temperature to go up.

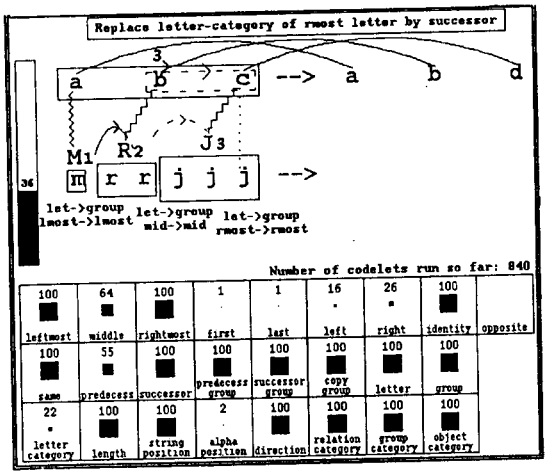
If the program were to stop now (which is quite unlikely, since a key factor in the program's probabilistic decision when to stop is the temperature, which is now quite high), the rule would be adapted for application to the string ***mrrjjj*** as "Replace the letter category of the rightmost *group* by ***d***" (the ***c–J*** bridge establishes that the role of *letter* in ***abc*** is played by *group* in ***mrrjjj***), yielding the answer ***mrrddd*** (an answer that Copycat does indeed produce, on rare occasions).

6. The previous, stronger rule has been restored (again the result of a fight having a probabilistic outcome), but at the same time, the strong ***c–J*** bridge also happened to get defeated by its weaker rival, the ***c–j*** bridge. As a consequence, if the program were to stop at this point, its answer would be ***mrrjjk***. This, incidentally, would also have been the answer in screen dump #4.

In the Slipnet, the activation of *length* has decayed a good deal, since the length description given to ***rr*** wasn't found to be useful. In the Workspace, the diminished salience of the ***rr***'s length description "2" is represented by the fact that the "2" is no longer in boldface.

The temperature is still fairly high, since the program is having a hard time making a single, coherent structure out of ***mrrjjj***, something that it did easily with ***abc***. That continuing difficulty, combined with strong top-down pressure from the two copy-groups that have been built inside ***mrrjjj***, makes it now somewhat tempting for the system to flirt with the *a priori* extremely unlikely idea of making a single-letter copy-group (this flirtation is represented by the dashed rectangle around the letter ***m***).

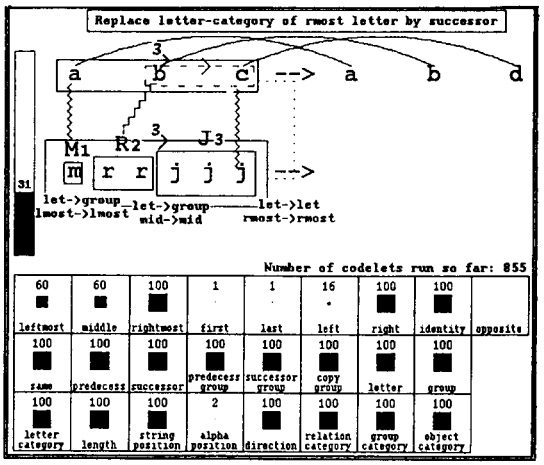


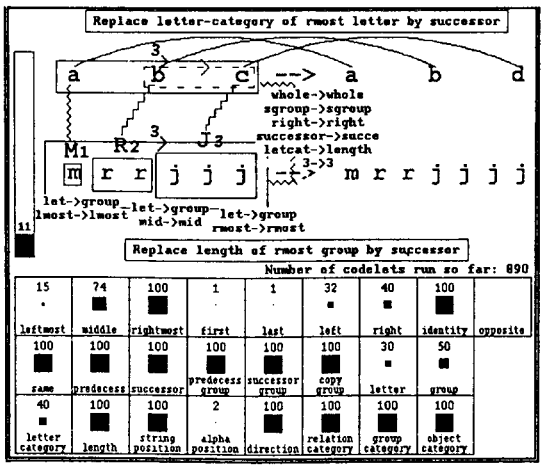


7. As a result of these combined pressures, the *a priori* extremely unlikely single-letter copy-group ***m*** happened to get built, and its length of 1, being very noteworthy, has been attached to the group as a description. A successorship bond between that "1" and its right-neighbor "2" has already been built; all of this is helping *length* to stay active. A consistent trio of *letter* ⇒ *group* bridges has now been made, and as a result of these promising new structures, the temperature has fallen to the relatively low value of 36, which in return helps to lock in this emerging view.

If the program were to halt in this screen dump or in the following one, it would produce the answer ***mrrkkk***, which is its most frequent answer.

8. As a result of *length*'s continued activity, length descriptions have been attached to the remaining two groups in the problem (***jjj*** and ***abc***), and a successorship bond between the "2" and the "3" (for which there is much top-down pressure coming from both ***abc*** and the emerging view of ***mrrjjj***) is being considered (dashed arc). *Letter category* has decayed, indicating that it hasn't lately been of use in building structures.





9. The "2"–"3" bond was built, whereupon a very abstract, high-level numerical successor group involving the group lengths in ***mrrjjj*** was perceived (large solid rectangle surrounding the three copy-groups). Also, a bridge (dotted vertical line to the right of the two strings) is being considered between strings ***abc*** and ***mrrjjj*** in their entireties.

Ironically, just as these sophisticated ideas seem to be converging toward a highly insightful answer, a small renegade codelet, totally unaware of the global momentum, has had some good luck: its bid to knock down the ***c–J*** bridge and replace it with a ***c–j*** bridge was accepted. Of course, this is a setback on the global level. If the program were forced to stop at this point, it would answer ***mrrjjk*** — the same rather dull answer as it would have given in screen dumps #6 and #4. However, at either of those stages, that answer would have been far more excusable than it is now, since the program hadn't yet made the subtle discoveries it has now made about the structure of ***mrrjjj***. It would seem a shame for the program to have gotten this far and then to "drop the ball" and answer in a relatively primitive way. However, 31 is a high enough temperature that there is a good chance that the program will get back on track and be allowed to explore the more abstract avenue to its logical conclusion.

10. Indeed, the aberrant bridge from the ***c*** was quickly destroyed and replaced by a rebuilt ***c–J*** bridge, in keeping with the emerging sophisticated view. Also, the high-level bridge between ***abc*** and ***mrrjjj*** as wholes, which in the previous screen dump was merely a dotted candidate, has now been promoted through the dashed state and actually built. Its six component concept-mappings, including identity mappings (such as *right* ⇒ *right*, meaning that both strings are seen as flowing rightwards) as well as conceptual slippages (such as *letter category* ⇒ *length*, meaning that letters are mapped onto numbers — specifically, onto group lengths), are listed in the middle of the screen, somewhat obscuring the bridge itself.

The original rule has been translated, according to the slippages, for application to the target string ***mrrjjj***. The translated rule, "Replace the length of the rightmost group by its successor", appears just above the Slipnet, and the answer ***mrrjjjj*** at the right. The very low final temperature of 11 reflects the program's unusually high degree of satisfaction with this answer.

Although the preceding run may look quite smooth, there were many struggles involved in coming up with this answer: it was hard not only to make a single-letter group, but also to bring the notion of *length* into the picture, to allow it to persist long enough to trigger the noticing of all three group lengths, and to build bonds between the group lengths. The program, like people, usually gives up before all these hurdles can be overcome, and gives one of the more obvious answers. Arriving at the deeper answer ***mrrjjjj*** requires not only the insights brought about by the strong pressures in the problem, but also a large degree of patience and persistence in the face of uncertainty.

The moral of all this is that in a complex world (even one with the limited complexity of Copycat's microworld), one never knows in advance what concepts may turn out to be relevant in a given situation. This dilemma underscores the point made earlier: it is imperative not only to avoid dogmatically open-minded search strategies, which entertain all possibilities equally seriously, but also to avoid dogmatically closed-minded search strategies, which in an ironclad way rule out certain possibilities *a priori*. Copycat opts for a middle way, in which it quite literally takes calculated risks all the time — but the *degree* of risk-taking is carefully controlled. Of course, taking risks by definition opens up the potential for disaster — and indeed, disaster occurs once in a while (as was evidenced by some of the far-fetched answers displayed earlier). But this is the price that must be paid for flexibility and the potential for creativity.

People, too, occasionally explore and even favor peculiar routes. Copycat, like people, has to have the potential to concoct strange and unlikely solutions in order to be able to discover subtle and elegant ones like ***mrrjjjj***. To rigidly close off any routes *a priori* would necessarily remove critical aspects of Copycat's flexibility. On the other hand, the fact that Copycat so rarely produces strange answers demonstrates that its mechanisms manage to strike an effective balance between open-mindedness and closed-mindedness, imbuing it with both flexibility and robustness.

Hopefully, these screen dumps have made clearer the fundamental roles of nondeterminism, parallelism, non-centralized and simple perceptual agents (*i.e.*, codelets), the interaction of bottom-up and top-down pressures, and the reliance on statistically emergent (rather than explicitly programmed) high-level behavior. Large, global, deterministic decisions are never made (except perhaps towards the end of a run). The system relies instead on the accumulation of small, local, nondeterministic decisions, none of which alone is particularly important for the final outcome of the run. As could be seen in the screen dumps, large-scale effects occur only through the statistics of the lower levels: the ubiquitous notion of a "pressure" in the system is really a shorthand for the statistical effects over time of a large number of actions carried out by codelets, on the one hand, and of activation patterns of nodes in the Slipnet, on the other.

As was seen in the screen dumps, as structures are formed and a global interpretation coalesces, the system gradually makes a transition (via gradually falling temperature) from being quite parallel, random, and dominated by bottom-up forces to being more serial, deterministic, and dominated by top-down forces. We believe that such a transition is characteristic of high-level perception in general.

The importance of such a transition in degree of risk-taking was confirmed by two experiments we performed on the model (described in Mitchell, 1993). Temperature was clamped throughout a run, in one experiment at a very high value and in the other at a very low value. In each experiment, 1,000 runs were made. In neither test did the hobbled Copycat ever come up with the answer *mrrjjjj* — not even once.

Micro-anatomy of a paradigm shift

We now turn our attention from Problem 4 to Problem 6, and give a sketch of how Copycat can, on occasion, come up with the answer *wyz*. It turns out to be a surprisingly intricate little tale. The reason for this is that it is an attempt to show in slow motion how a human mind, under severe pressure, can totally transform its perception of a situation in a blinding flash (colloquially termed the "Aha!" phenomenon). Since such paradigm shifts are often found at the core of deeply creative acts, one should expect their microstructure to be very complex (otherwise, the mystery of creativity would long ago have been revealed and put on a mass-produced microchip). Indeed, the challenge of getting Copycat to produce *wyz properly* — faithfully to what we believe really goes on in a human mind at the most informative subcognitive level of description — has, from the very outset, been the central inspiration in guiding the development of the Copycat architecture.

The very cursory justification for *wyz* given in the previous section lies at far too high and coarse-grained a level to be informative about the mental mechanisms responsible for paradigm shifts. The following detailed story, by contrast, evolved hand in hand with the architecture itself, and is intended not only as a description of Copycat, but hopefully as an accurate description of the underpinnings of a typical paradigm shift in a human mind. (An annotated series of screen dumps of a particular run on Problem 6 is given in Mitchell & Hofstadter, 1990a.)

Emergency measures convert a serious snag into a set of exploratory pressures

Things start out essentially analogously to a typical run on Problem 1, in terms of bonding, grouping, bridge-building, and such — that is, both source and target strings come quite quickly to be perceived as successor groups, and the raw rule "Replace rightmost letter by its successor" is effortlessly produced.

Everything thus proceeds pretty smoothly up to the point of trying to take the successor of *z*, which is impossible. This serious snag causes several coordinated "emergency measures" to be taken:

- the *physical* trouble spot — here, the instance of *z* in the Workspace — is highlighted, in the sense that its salience is suddenly pumped up so high that, to codelets, it becomes the most attractive object in the entire Workspace;
- the *conceptual* trouble spot — here, the node *z* in the Slipnet — is highlighted, in the sense that a huge jolt of activation is pumped into it, and as a consequence, its halo broadens and intensifies, meaning that related concepts are more likely to be considered, at least fleetingly;
- the temperature is pumped up to its maximum value of 100 and temporarily clamped there, thus encouraging a broader and more open-minded search;
- the high temperature enables previously dormant "breaker" codelets to run, whose purpose is to arbitrarily break structures that they find in the Workspace, thus reducing the system's attachment to a viewpoint already established as being problematic.

Note the generality of these "impasse-handling" mechanisms: they have nothing to do with this snag itself, with the particular problem, with the alphabetic domain, or even with analogy-making! The reason for this is of course that running into an impasse is a critical and common event that any cognitive system must be capable of dealing with. To be sure, no set of mechanisms can be guaranteed to resolve *all* snags (otherwise we would be dealing with omniscience, not intelligence). The best that can be hoped for is that the impasse itself can be "read" as a source of *cues* — possibly very subtle ones — that may launch tentative forays down promising new avenues. A "cue", in the Copycat architecture, is essentially the creation of a *pressure* that pushes for exploration along a certain direction. Thus the idea of *interpreting the snag as a source of pressures* is the philosophy behind the four mechanisms above, especially the first two.

Although these emergency measures are not powerful enough to guide Copycat to coming up with ***wyz*** all that often, when it does get there, it does so essentially according to the following scenario.

The spotlight focused on Platonic *z* has the effect of making all concepts in *z*'s halo — including the closely-related concept *alphabetic-last* — somewhat more likely to be looked at by description-building codelets. The probability is thus significantly increased that the instance of *z* in the Workspace will get explicitly described as *alphabetic-last.*

Note that in most problems — even ones that involve one or more instances of the letter *z* — there is little or no reason to pay attention to the notion *alphabetic-last,* and therefore, this conceptually deep neighbor of Platonic *z* usually remains — and *should* remain — dormant (as it did in Problems 1–5). After all, as was brought out in the discussion of Problem 4, it is extremely crucial to avoid cluttering up the processing with all sorts of extraneous interfering notions, no matter how conceptually deep they may be. But now, under the emergency measures, unusual avenues are more likely to at least be "sniffed out" a short ways.

If the description *alphabetic-last* does indeed get attached to the *z*, which is a dicey matter, then a further boost is given to the node *alphabetic-last,* as the system has deemed it potentially relevant. So now that *alphabetic-last* (part of the halo of Platonic *z*) has been lifted considerably out of dormancy, concepts in *its* halo will in turn receive more activation, which means codelets will tend to pay more attention to them (probabilistically speaking). One such neighbor-concept is *alphabetic-first,* which is now briefly given the chance to show its relevance. Obviously, if there were no instance of ***a*** in the problem, *alphabetic-first* would be found to be completely irrelevant and would soon decay back to dormancy, but since there *is* an ***a*** inside ***abc***, it has a fair chance of getting explicitly described as *alphabetic-first,* in much the same way as the *z* in ***xyz*** got described as *alphabetic-last.*

If both these descriptions get attached — and that is a big "if" — then both letters become even more salient than before; in fact, they almost cry out to be mapped onto each other — not because the system can anticipate the great insight that such a mapping will bring, but simply because both letters are so salient! Once the system tries it out, however, the great appeal of the tentative mapping instantly becomes apparent. Specifically, a pair of conceptual slippages are entailed in the act of "equating" the ***a*** with the *z* (*i.e.,* building an ***a***–*z* bridge): *alphabetic-first* ⇒ *alphabetic-last,* and *leftmost* ⇒ *rightmost.*

How resistance to a deep slippage is overcome — a tricky matter

Although normally the deep slippage of *alphabetic-first* into *alphabetic-last* would be quite valiantly resisted (recall the motto given earlier, "Deep stuff doesn't slip in good analogies"), here a special circumstance renders it a bit more probable: the companion would-be slippage *rightmost* ⇒ *leftmost* is of the same type — in particular, each of these slippages involves slipping a concept representing an extremity into its opposite concept. These two would-be slippages are thus conceptually parallel, so that each one on its own reinforces the other's plausibility. This fact helps to overcome the usual resistance to a deep slippage. (Incidentally, this is the kind of subtlety that was not apparent to us before the computer implementation was largely in place; only at that point

were Copycat's flailings and failures able to give us pointers as to what kinds of additional mechanisms were needed.)

Another fact that helps overcome the usual resistance to the deep slippage in this bridge is that *any* two slippages, whether parallel or not, provide more justification for building a bridge than either one alone would. Altogether, then, there is a fairly good chance that this bridge, once tentatively suggested, will actually get built. Once this critical step has taken place, essentially it's all downhill from there. This is why we have paid particularly close attention to the pathway via which such a bridge can emerge.

Locking-in of a new view

The first thing that is likely to happen as a result of an ***a–z*** bridge getting built is that the temperature will get unclamped from its value of 100. In general, what unclamps the temperature is the construction of any strong structure different from those that led up to the snag — in other words, a sign that an alternative way of looking at things may be emerging — and this bridge is a perfect example of such a structure. Like most actions in Copycat, the unclamping of temperature is probabilistic. In this case, the stronger the novel structure is, the more likely it is to trigger the unclamping. Since the ***a–z*** bridge is both novel and very strong, unclamping is virtually assured, which means that the temperature falls drastically right after the bridge is built. And when the temperature falls, decisions tend to get more deterministic, which means that the emerging new view will tend to get supported. In short, there is a powerful kind of *locking-in* effect that is triggered by the discovery of an ***a–z*** bridge. This is a critical effect.

Another aspect of locking-in is the following idea. The building of this first bridge involving the simultaneous slippage of two concepts into their opposites sends a burst of activation into the very deep concept *opposite*; as a result, all pairs of concepts connected via links labeled *opposite* are drawn much closer together, facilitating the slippage of one into the other. Such slippages will still not happen without reason, of course, but now they will be much easier to make than in ordinary circumstances. Thus in a sense, making *one* bridge based on conceptual opposites sets a tone making it easier to make *more* of them. The emerging theme of the concept *opposite* can fairly be characterized as a kind of "bandwagon".

Given all this, one of the most likely immediate consequences of the crosswise ***a–z*** bridge is the building of the "mirror" crosswise bridge connecting the ***c*** with the ***x***. It, too, depends on the slippage between *leftmost* and *rightmost*, and is thus facilitated; in addition, once built, it strongly reinforces the emerging relevance of the concept *opposite*. Moreover, the temperature will fall significantly because this bridge, too, will be very strong. Thanks to all of this, the

locking-in effect may by now be so strong that it will be hard to stop the momentum towards building a completely new view of the situation.

The reversals taking place become a near-stampede at this point, with significant pressure emerging to flip the direction of the fabric of the group ***xyz*** from *rightwards* to *leftwards,* which means also that the perceived fabric itself would switch from *successor* to *predecessor.* Thus at this point, Copycat has carried out both a spatial and an alphabetical reversal of its vision of ***xyz***. The paradigm shift has been completed. At this point, Copycat is ready to translate the raw rule, and, as was said above, the result is the new rule *replace the leftmost letter by its alphabetic predecessor,* which yields the answer ***wyz***.

It must be stressed that all the multifarious activity just described — shifting degrees of activation of various key concepts; deep slippages; interrelated spatial and conceptual reversals — all this takes place in a flash in a human mind. There is no hope of making out all the details of this paradigm shift (or any other) in one's own mind through mere introspection. In fact, it has taken the authors several years to settle on the above account, which represents our current best stab at the true story's intimate details.

How hard is it to make this paradigm shift?

As was pointed out a moment ago, the motto "Deep stuff doesn't slip in good analogies" is violated by the answer ***wyz***, in that *alphabetic-first* is a deep concept and yet is allowed to slip into *alphabetic-last* here. This is one reason that makes it so hard for many people to discover it on their own. Yet many people, when they are shown this answer, appreciate its elegance and find it very satisfying. Problem 6 is thus a circumstance where a constellation of pressures can occasionally overcome the powerful natural resistance expressed by the motto; in fact, making such a daring move results in what many people consider to be a deep and insightful analogy.

There is an important irony here. In particular, even though slippages tend to be (and should be) *resisted* in proportion to their depth, once a very deep slippage has been made, then it tends to be (and should be) *respected* in proportion to its depth. We consider this to be characteristic of creative breakthroughs in general. More specifically, we consider the process of arriving at answer ***wyz*** to be very similar, on an abstract level, to the process whereby a full-scale conceptual revolution takes place in science (Kuhn, 1970).

Now we come back to the point raised in our earlier discussion of Problem 6 about "levels of subtlety" of answers. Specifically, we claimed above that, because finding the answer ***wyz*** to Problem 6 is far subtler *for people* than finding the similar answer ***hjkk*** to Problem 2, any model of mental fluidity should respect this difference in levels of subtlety. Yet when one compares the bar graphs for these problems, one discovers that ***wyz*** was found far more often than ***hjkk*** was

found (a ratio of 137 to 47, when both problems were run 1,000 times). This seems to completely contradict the claim that the former answer is subtler than the latter. How can one account for this unexpected ratio?

There are two basic factors that explain it. The first has to do with the fact that there was a snag in one problem and no snag in the other. In attacking Problem 6, Copycat was *forced* to look for solutions other than taking the successor of the rightmost letter, because that route turned out to be impossible. By contrast, all sorts of superficially attractive routes led directly to solutions in Problem 2. There was no snag that prevented any attractive route from being taken all the way to its natural conclusion. Had all or most of the easy routes been barred, then of course ***hjkk*** would have constituted a much larger percentage of the answers found.

The second factor is that the average *length of time* taken to find various solutions (measured in terms of number of codelets run) is a key notion. This fact is not apparent, because average run-lengths are unfortunately not represented in the bar graphs. When Copycat came up with ***hjkk*** in Problem 2, it was essentially always a relatively direct process involving no backtracking or getting stuck for a while in a loop. To be specific, the average number of codelets taken to get ***hjkk*** was 899. By contrast, the average number of codelets taken to get ***wyz*** was 3,982 — over four times as long. The reason for this is that in most runs, the program came back time and time again to the standard way of looking at ***xyz***, and thus hit the snag over and over again: it was stuck in a kind of rut. This means that on runs where Copycat was lucky enough to come across the double reversal, by the time it did so it had usually tried out all sorts of other pathways in vain beforehand. In this sense of *time needed to make the discovery*, ***wyz*** was an extremely elusive answer for the program, whereas ***hjkk*** was not at all elusive. In sum, ***wyz*** was indeed far subtler for Copycat than ***hjkk*** was, as ought to have been the case.

Conclusion: The Generality of Copycat's Mechanisms

The crucial question of scaling-up

As was stated at the outset, the Copycat project was never conceived of as being dependent in any essential way on specific aspects of its small domain, nor even on specific aspects of analogy-making *per se*. Rather, the central aim was to model the emergence of insightful cognition from fluid concepts, focusing on how slippages can be engendered by pressures.

One of the key questions about the architecture, therefore, is whether it truly is independent of the small domain and the small problems on which it now works. It would certainly be invalidated if it could be shown to depend on

few instances of those concepts that appear in a typical problem. However, from the very conception of the project, every attempt has been made to ensure that Copycat would not succumb to a combinatorial explosion if the domain were enlarged or the problems became bigger. In some sense, Copycat is a caricature of genuine analogy-making. The question is, what makes a caricature faithful? What is the proper way to construct a cognitive model that will scale up?

Shades of gray and the mind's eye

Real cognition of course occurs in the essentially boundless real world, not in a tiny artificial world. This fact seems to offer the following choice to would-be "cognition architects": either have humans scale down all situations by hand in advance into a small set of sharp-edged formal data-structures, so that a brute-force architecture can work, or else let the computer effectively do it instead — that is, use a heuristic-based architecture that at the outset of every run makes a sharp and irreversible cut between concepts, pathways, and methods of attack that might eventually be brought to bear during that run, and ones that might not. There seems to be no middle ground between these two types of strategy, because either you must be willing to give *every* approach a chance (the brute-force approach), or you must choose some approaches while *a priori* filtering others out (the "heuristic-chop" approach).

The only way out would seem to involve a notion of "shadedness", in which concepts, facts, methods of attack, objects, and so on, rather than being ruled "out" or "in" in a black-and-white way, would be present in shades of gray — in fact, shades of gray that change over time. At first glance, this seems impossible. How can a concept be invoked only *partially*? How can a fact be neither fully ignored nor fully paid attention to? How can a method of attack be merely "sort of" used? How can an object fall somewhere in between being considered "in the situation" and being considered "not in the situation"?

Since we believe that these "shades of gray" questions lie at the crux of the modeling of mind, they merit further discussion. A special fluid quality of human cognition is that often, solutions to a problem — especially the most ingenious ones, but even many ordinary ones — seem to come from far outside the problem as conceived of originally. This is because problems — or more generally, *situations* — in the real world do not have sharp definitions; when one is in, or hears about, a complex situation, one typically pays no conscious attention to the question of what counts as "in" the situation and what counts as "out" of it. Such matters are almost always vague, implicit, and intuitive.

choose where to have the mind's eye "look". When one directs one's gaze at what one feels is the situation's core, only a few centrally located things will come into clear focus, with more tangential things being less and less clear, and then at the peripheries there will be lots of things of which one is only dimly aware. Finally, whatever lies beyond the field of vision seems by definition to be outside of the situation altogether. Thus "things" in the mind's eye are definitely shaded, both in terms of how clear they are, and in terms of how aware one is of them.

The very vague term "thing" was used deliberately above, with the intent of including both *abstract Platonic concepts* and *concrete specific individuals* — in fact, to blur the two notions, since there is no hard-and-fast distinction between them. To make this clearer, think for a moment of the very complex situation that the Watergate affair was. As you do this, you will notice (if you followed Watergate at all) that all sorts of different events, people, and themes float into your mind with different degrees of clarity and intensity. To make this even more concrete, turn your mind's eye's gaze to the Senate Select Committee, and try to imagine each different senator on that committee. Certainly, if you watched the hearings on television, some will emerge vividly while others will remain murky. Not just *Platonic abstractions* like "senator" are involved, but many *individual* senators have different degrees of mental presence as you attempt to "replay" those hearings in your mind. Needless to say, the memory of anyone who watched the Watergate hearings on television is filled to the brim both with Platonic concepts of various degrees of abstractness (ranging from "impeachment" to "coverup" to "counsel" to "testimony" to "paper shredder") and with specific events, people, and objects at many levels of complexity (ranging from the "Saturday night massacre" to the Supreme Court, from Maureen Dean to the infamous 18½-minute gap, and all the way down to the phrase "expletive deleted" and even Sam Ervin's gavel, with which every session of the committee was rapped to order). When one conjures up one's memories of Watergate, all of these "things" have differential degrees of mental presence, which change as one's mind's eye scans the "scene".

Note that in the preceding paragraph, all the "things" mentioned were carefully chosen so that readers — at least readers who remember Watergate reasonably well — would give them unthinking acceptance as genuine "parts" of Watergate. However, now consider the following "things": England, France, communism, socialism, the Viet Nam War, the Six-Day War, the Washington Monument, the *New York Times*, Spiro Agnew, Edward Kennedy, Howard Cosell, Jimmy Hoffa, Frank Sinatra, Ronald Reagan, the AFL–CIO, General Electric, [illegible] University, helicopters, keys,

to try to draw a sharp line. One is forced to accept the fact that for a model of a mind to be at all realistic, it must be capable of imbuing all concrete objects and individuals, as well as all abstract Platonic concepts, with shaded degrees of mental presence — and of course, those degrees of presence must be capable of changing over time.

Like a real eye, the mind's eye can be attracted by something glinting in the peripheries, and shift its gaze. When it does so, things that were formerly out of sight altogether now enter the visual field. By a series of such shifts, "things" that were totally outside of the situation's initial representation can eventually wind up at the very center of attention. This brings us back, finally, to that special fluid quality of human thought whereby initially unsuspected notions occasionally wind up being central to one's resolution of a problem, and reveals how intimately such fluidity is linked with the various "shades of gray" questions given above.

Copycat's shaded exploration strategy

Let us thus return to the list of "shades of gray" questions: How can a concept be invoked only *partially*? How can a fact be neither fully ignored nor fully paid attention to? How can a method of attack be merely "sort of" used? How can an object fall somewhere in between being considered "in the situation" and being considered "not in the situation"? These questions were not asked merely rhetorically; in fact, it was precisely to respond to the challenges that they raise that the probabilistic architecture of Copycat was designed.

Copycat's architecture has in common with brute-force architectures the fact that every possible concept, fact, method, object, and so on is *in principle* available at all times;[7] on the other hand, it has in common with heuristic-chop architectures the fact that out of all available concepts, facts, methods, objects, and so on, only a few will get very intensely drawn in at any given moment, with most being essentially dormant and an intermediate number having a status somewhere in between. In other words, virtually all aspects of the Copycat architecture are riddled by shades of gray instead of by hard-edged, black-and-white cutoffs. In particular, *activation* (with continuous values rather than a binary on/off distinction) is a mechanism that gives rise to shadedness in the Slipnet, while *salience* and *urgency* serve similar purposes in the Workspace and Coderack, respectively. These are just three of a whole family of related "shades-of-gray mechanisms" whose entire *raison d'être* is to defeat the scaling-up problem.

7. Note that the claim is not that every single concept imaginable to humans is available, but simply that all concepts *within the system's dormant repertoire* are in principle ac-

An architecture thus pervaded by shades of gray has the very attractive property that although no concept or object or pathway of exploration is ever *strictly* or *fully* ruled out, only a handful of them are at any time *seriously* involved. At any given moment, therefore, the system is focusing its attention on just a small set of concepts, objects, and pathways of exploration. However, this "searchlight of attention" can easily shift under the influence of new information and pressures, allowing *a priori* very unlikely concepts, objects, or pathways of exploration to enter the picture as serious contenders.

The chart below summarizes the various mechanisms in the Copycat architecture that incorporate shades of gray in different ways. In it, the term "shaded" should be understood as representing the opposite of a binary, black/white distinction; it often means that one or more *real numbers* are attached to each entity of the sort mentioned, as opposed to there being an on/off distinction. The term "dynamic" means that the degree of presence — the "shade", so to speak — can change with time.

Shades of gray in the Slipnet

- shaded, dynamic presence of Platonic concepts (via dynamic activation levels)
- shaded, dynamic conceptual proximities (via dynamic link-lengths)
- shaded, dynamic spreading of activation to neighbor concepts (giving rise to "conceptual halos")
- shaded conceptual depths of nodes
- shaded decay rates of concepts (determined by conceptual depths)
- shaded, dynamic emergence of abstract themes (stable activation patterns of interrelated conceptually deep nodes)

Shades of gray in the Workspace

- shaded, dynamic number of descriptions for any object
- shaded, dynamic importance of each object (via activation levels of descriptors in Slipnet)
- shaded, dynamic unhappiness of each object (determined by degree of integration into larger structures)
- shaded, dynamic presence of objects (via dynamic salience levels)
- shaded, dynamic tentativity of structures (via dynamic strengths)

- shaded, dynamic emergence of pressures (via urgencies of codelets and shifting population of Coderack)
- shaded, dynamic degree of willingness to take risks (via temperature)
- shaded, dynamic mixture of deterministic and nondeterministic modes of exploration
- shaded, dynamic mixture of parallel and serial modes of exploration
- shaded, dynamic mixture of bottom-up and top-down processing

There is one further aspect of shadedness in Copycat that is not localized in a single component of the architecture, and is somewhat subtler. This has to do with the fact that, over time, higher-level structures emerge, each of which brings in new and unanticipated concepts, and also opens up new and unanticipated avenues of approach. In other words, as a run proceeds, the field of vision broadens out to incorporate new possibilities, and this phenomenon feeds on itself: each new object or structure is subject to the same perceptual processes and chunking mechanisms that gave rise to it. Thus there is a spiral of rising complexity, which brings new items of ever-greater abstraction into the picture "from nowhere", in a sense. This process imbues the Copycat architecture with a type of fundamental unpredictability or "openness" (Hewitt, 1985) that is not possible in an architecture with frozen representations. The ingredients of this dynamic unpredictability form an important addendum to the list of shades of gray given above.

Dynamic emergence of unpredictable objects and pathways

- creation of unanticipated higher-level perceptual objects and structures
- emergence of *a priori* unpredictable potential pathways of exploration (via creation of novel structures at increasing levels of abstraction)
- creation of large-scale viewpoints
- competition between rival high-level structures

By design, none of the mechanisms in the lists presented above has anything in the least to do with the size of the situations that Copycat is currently able to deal with, or with the current size of Copycat's Platonic conceptual repertoire. Note, moreover, that none of them has anything whatsoever to do with the subject matter of the Copycat domain, or even with the task of analogy-making *per se*. Yet these mechanisms and their emergent consequences — especially commingling pressures and the parallel terraced scan — are what Copycat is *truly* about. This is the underpinning of our belief in the cognitive generality of the Copycat architecture.