Chapter 5, from Douglas Hofstadter & the Fluid Analogies Research Group, *Fluid Concepts and Creative Analogies*, Basic Books, 1995.

liferating in
id them. My
ese kinds of
t at Indiana
batical, and
l a technical
opycat work.
was my way
tal halos and
in a natural
ly, codelets,
a natural fit,
was starting
l years down
one person.
nie Mitchell
terest in the
tle to go on,
, I tentatively
te.

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