Beyond Copycat:
Incorporating Self-Watching into a Computer Model of High-Level Perception and Analogy-Making

James B. Marshall and Douglas R. Hofstadter

Center for Research on Concepts and Cognition
& Department of Computer Science
Indiana University
Bloomington, IN 47408

marshall@cs.indiana.edu
dughof@cogsci.indiana.edu

Abstract

This paper summarizes recent work on extending the architecture of the Copycat analogy-making program to include the ability to monitor certain aspects of its own processing as it solves analogy problems. It discusses several important weaknesses of the Copycat model and outlines current efforts aimed at overcoming these limitations.

Introduction

This paper discusses recent continuing work on the Copycat project, a stochastic computer model of fluid concepts, high-level perception, and analogy-making (Hofstadter & Mitchell, 1994). Copycat perceives analogies between short strings of letters, which can be thought of as representing abstract situations in an idealized microworld. An example of such an analogy might be "If **abc** changes to **abd**, how does **iijjkk** change in an analogous way?" An interesting feature of such problems is that there is no single "right" answer; rather, a range of answers is always possible for each problem. For the previous example, some possible answers might be **iijjll**, **iijjdd**, **iijjkl**, **iijjkd**, **iijjkk**, or even **abd** or **aabbdd**. Of course, for most analogy problems, some answers are consistently judged by people to be better than others. But a surprisingly large number of defensible answers can usually be found for each problem. Furthermore, for some problems, the answers judged to be the "best" are not at all the most "obvious" ones. Despite its apparent simplicity, Copycat's microworld actually exhibits a remarkable degree of richness and subtlety, and constitutes an ideal "laboratory" in which to study analogy-making and high-level perception.

A detailed exposition of the Copycat model can be found in (Mitchell, 1993). Here we shall give just a brief overview, drawing attention to several important limitations. Recent work has successfully addressed some of these limitations, but more remains to be done. The following sections summarize the progress that has been made so far, and outline the central ideas behind present efforts to extend the Copycat model in a way that will allow it to understand the analogies it makes in a much more sophisticated capacity than currently possible.

The Copycat Model

The central theme underlying the Copycat architecture is the idea of nondeterministic, stochastic processing distributed among a large number of small computational agents called *codelets* that work on different aspects of an analogy problem simultaneously, although at different speeds, thereby achieving a kind of "differential parallelism". All processing in the model occurs through the collective actions of many codelets working together over time, without any higher-level "executive" process supervising or controlling the overall course of events. The model's high-level "macroscopic" behavior emerges as a consequence of many fine-grained "microscopic" events. Thus, Copycat lies firmly within the paradigm of "emergent computation". At the same time, however, it incorporates many ideas from the more traditional paradigm of "symbolic AI", inhabiting a kind of middle ground between these two opposites. Blank *et al.* (1992) give some further perspectives on the emergent--symbolic spectrum and Copycat's relationship to it.

In order to discover an answer to an analogy problem such as **abc** -> **abd**; **iijjkk** -> **?**, codelets work together to build up a strong, coherent mapping between the *initial string* **abc** and the *target string* **iijjkk**, and also between the initial string **abc** and the *modified string* **abd**. Codelets also build hierarchical *groups* within strings, which serve to organize the raw perceptual data (*i.e.*, the letters) into a coherent, chunked whole. For example, in the string **iijjkk**, codelets might build three "sameness-groups" **ii**, **jj**, and **kk**, and then a higher-level "successorship" group comprised of these three groups encompassing the entire string. The distributed nature of codelet processing interleaves the chunking process with the mapping process, and as a result each process influences and drives the other.

A mapping consists of a set of *bridges* between corresponding letters or groups that play respectively similar roles in different strings. Each bridge is supported by a set of *concept-mappings* that together provide justification for perceiving the objects connected by the bridge as corresponding to one another. For example, a bridge might be built between **a** in **abc** and the group **ii** in **iijjkk**, supported by the concept-mappings *leftmost => leftmost* and *letter => group*, which represents the idea that both objects are leftmost in their strings, and that one is a letter and the other a group. Non-identity concept-mappings such as *letter => group* are called *slippages*, and form the basis of Copycat's ability to flexibly perceive superficially-dissimilar situations as being in fact "the same" at some appropriate level of description.

In addition to bridges and groups, another type of structure is necessary to produce an answer to an analogy problem. Once a mapping has been built between the initial string and the modified string (*i.e.*, between **abc** and **abd**), a *rule* based on this mapping must be created that succinctly captures the way in which the initial string changes. Of course, there are many possible ways of describing this change, some more abstract than others. For example, two possible rules for **abc** -> **abd** are *Change letter-category of rightmost letter to successor* and *Change letter-category of rightmost letter to d*. Different ways of looking at the initial-modified change, combined with different ways of building the initial-target mapping, give rise to different answers. A bridge from the letter **c** in **abc** to the group **kk** in **iijjkk**, based on a *letter => group* slippage, may yield the answer **iijjll** or **iijjdd**, depending on the rule used to describe **abc** -> **abd**. On the other hand, a bridge from **c** to the rightmost *letter* **k** in **iijjkk** may instead yield **iijjkl** or **iijjkd** as an answer, again depending on the rule. To produce an answer, the slippages underlying the mapping

between **abc** and **iijjkk** are used by codelets to "translate" the rule describing **abc** -> **abd** into a new rule that applies to **iijjkk**, such as *Change letter-category of rightmost group to successor*.

Weaknesses

All of this structure-building activity by codelets occurs in Copycat's *Workspace*, where the strings themselves reside. The model incorporates a simple numerical measure of overall Workspace "coherence", called *temperature*, which reflects, at any given moment, the amount and quality of structures built so far. The final temperature of the Workspace when an answer is found can therefore be interpreted as an indication of the answer's overall quality. Furthermore, this measure of answer quality agrees well with the relative judgments of answer quality given by people for a wide range of Copycat problems. For example, Copycat's rating of the answer **iijjll** in the above problem will tend to be much higher than for "weirder" answers such as **iijjkd**.

Unfortunately, such a stark, numerical measure is extremely crude, and reflects a fundamental weakness of the current model: its almost complete lack of any in-depth understanding of the answers it finds. Copycat is unable to explain *why* it considers particular answers to be good or bad. The reason is that Copycat's processing mechanisms focus almost exclusively on perceiving patterns and relationships in the perceptual *data* (the letter strings), while ignoring patterns that occur in its own *processing* when solving an analogy problem. Thus, although it may discover an insightful answer for some problem, it lacks any internal representation or knowledge of the underlying process that led it to discover that answer--knowledge that could provide a basis for explaining the answer's relative strengths or weaknesses, thereby permitting a much richer assessment of its quality. Copycat's lack of any such "self-watching" ability stands in marked contrast to people, who are typically able to give an account of why they consider one answer to be better or worse than another for a particular analogy problem. An interesting related phenomenon, dubbed the *self-explanation effect*, has been studied recently in the context of students learning to solve physics problems from worked-out examples (Chi *et al.*, 1989; VanLehn *et al.*, 1992).

Another weakness of the model is the fact that answers are not retained after they are found. When Copycat discovers an answer to a problem, it simply reports its answer, along with the answer's final temperature, and then stops. On subsequent runs of the same problem, no recollection of previous answers is possible, so there is no way for the program to bring its past experience to bear on its current situation. This makes comparison of different answers impossible, either for the same problem or among different problems. Furthermore, any type of learning that might occur over multiple runs is impossible--although learning itself was never intended to be a central focus of the project, since the notion of learning to make "better" Copycat analogies is not entirely clear.

Yet another limitation of the model concerns the creation of rules. Initial work on Copycat focused on the mapping process between the initial and target strings, and paid relatively little attention to the creation of rules describing the change from the initial string to the modified string. That is, the first phase of development concentrated on perceiving similarity between strings via bridges and slippages, rather than on perceiving and characterizing differences between strings via rules. Accordingly, severe restrictions were placed on what types of changes were allowed to the initial string. In fact, at most a

single letter-category change to a single letter was allowed. Thus, **abc** -> **abd** was allowed, but more general changes--even as simple as **abc** -> **cba** or **abc** -> **abcc**, in which more than one letter changes or the length of the string changes--were not. Under such rigid restrictions, creating an initial-modified mapping and abstracting a rule based on it is essentially trivial, since such a mapping is always one-to-one, and always involves just one possible type of change. Developing more robust mechanisms for perceiving and characterizing differences between arbitrary strings was postponed to a later phase of the project.

From Copycat to Metacat

Recent work on Copycat, now in its second stage of development, has focused on these weaknesses. The central objective of the current phase of the project, dubbed "Metacat", is to increase the model's understanding of its answers to the point where it can give at least a limited explanation of an answer's strengths and weaknesses relative to other answers it has previously found. This requires not only remembering answers, but also the ability to compare and contrast different answers to a single problem, or even answers to different problems.

Step One: Generalizing the Rule Representations

To this end, the mechanisms for building a mapping between the initial string and the modified string, and then abstracting a rule based on it, have been generalized and extended to handle arbitrary strings. This is an important step toward increasing both the model's generality and its ability to understand the answers it finds, since understanding the strengths and weaknesses of an answer depends quite heavily on the ability to perceive differences between the initial and modified strings in very flexible ways.

To build a rule, codelets examine the concept-mappings underlying the bridges that have been built between the initial and modified strings. They attempt to abstract out common patterns from these concept-mappings, particularly from among the slippages, and then build a high-level description of the mapping based on the patterns they have noticed. Changes to objects in the initial string can be described either "intrinsically" or "extrinsically". An intrinsic change involves just a single object changing in some way, such as **c** changing to its successor in **abc** -> **abd**. An extrinsic change, however, involves two or more objects that change relative to each other. An example would be describing two objects as exchanging or swapping some particular characteristic, such as letter-category or position, as in **abcd** -> **dbca**. A sampling of some rules that Metacat is currently able to build is shown below:

- **abc** -> **cba**
  Reverse direction of string.
- **abc** -> **cba**
  Swap positions of leftmost letter and rightmost letter.
- **abc** -> **abcd**
  Increase length of string by one.
- **abc** -> **abdd**
  Change letter-category of rightmost letter to successor;
  Increase length of rightmost letter by one.

- **abc -> dba**
  Reverse direction of string;
  Change letter-category of rightmost letter to successor.
- **abc -> aabbcc**
  Increase lengths of all objects in string by one.
- **abc -> aaa**
  Change letter-categories of all objects in string to a.
- **aabccc -> aaabcc**
  Swap lengths of leftmost group and rightmost group.
- **eeqee -> qeeq**
  Swap letter-categories and lengths of all objects in string.
- **abc -> bbbcccddd**
  Change lengths of all objects in string to three;
  Change letter-categories of all objects in string to successor.

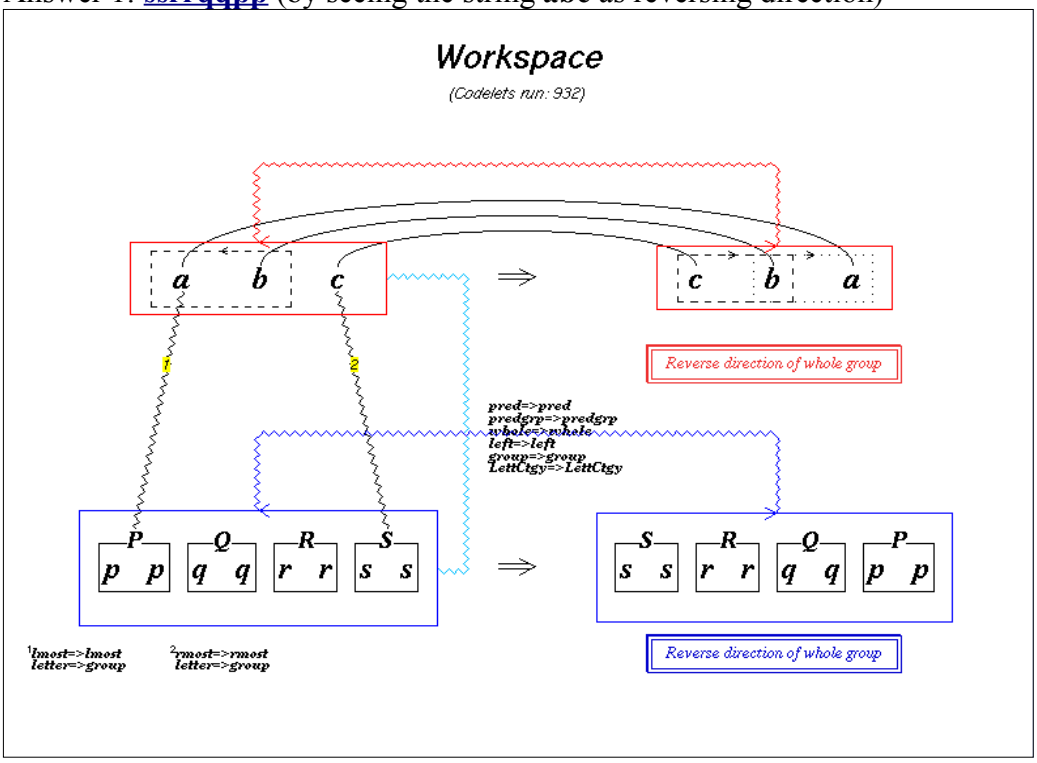
Step Two: Incorporating a Memory

In addition to its generalized rule capability, Metacat is now able to find and remember many different answers during a single run on a given problem. Whenever it finds a new answer, rather than simply stopping, it pauses to display the answer, along with the groups, bridges, concept-mappings, and rule that gave rise to it. Furthermore, all of this answer-specific information, including the strings themselves, is then packaged together and stored in memory, after which the program continues searching for alternative answers to the problem. Gradually, over time, a repertoire of answers builds up in memory, each one containing much more information than just the answer string itself. Each stored answer represents a different way of perceiving or making sense of a particular analogy problem.

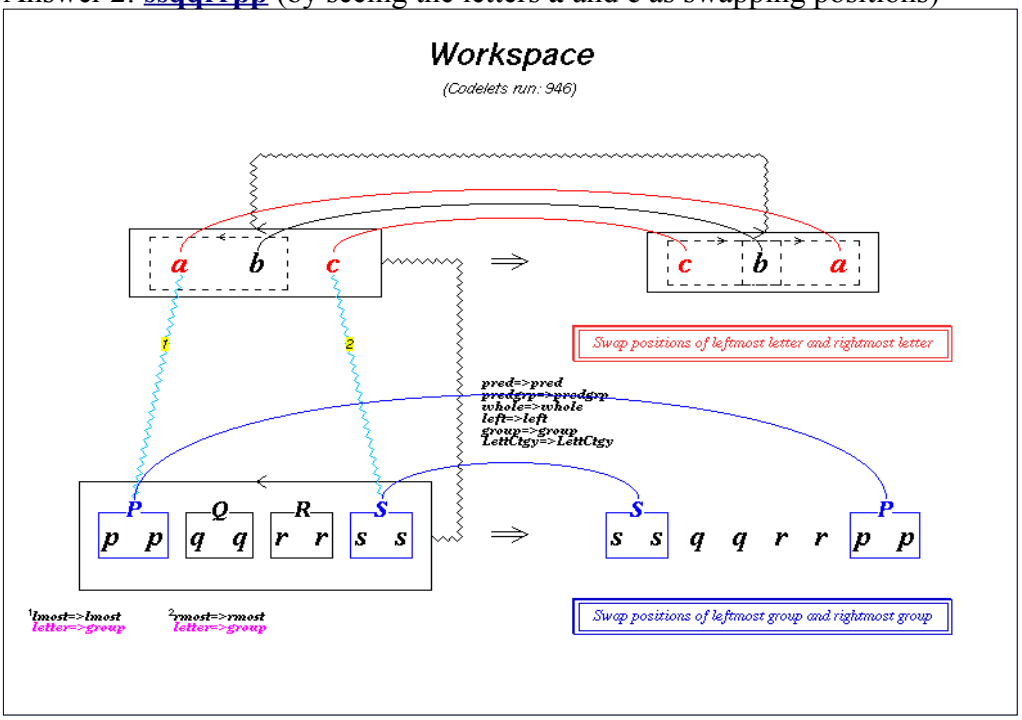
As an example, consider the problem **abc** -> **cba**; **ijkl** -> **?**. Metacat might first find the answer **lkji**, which is supported by the rule Reverse direction of string, based on a bridge from the right-directed string **abc** to the left-directed string **cba**. All of this information is stored along with the answer **lkji** in memory. Later during the same run, the program might find the answer **ljki**, which is supported by the rule Swap positions of leftmost letter and rightmost letter, based on a bridge from **a** in **abc** to **a** in **cba**, and a bridge from **c** in **abc** to **c** in **cba**. This new answer is likewise stored in memory, along with its associated structures. Later, the program might find the same answer **ljki** again, but this time for a different reason. The supporting structures this time would include the rule Swap letter-categories of leftmost letter and rightmost letter, based on a bridge from **a** in **abc** to **c** in **cba**, and a bridge from **c** in **abc** to **a** in **cba**.

These ideas are illustrated more clearly in the following three images, which are screendumps of Metacat's Workspace taken from a single run of the program on the slightly different problem **abc** -> **cba**; **ppqqrrss** -> **?**. Each image shows an answer that the program found for this problem, each based on one of the three ways described above of interpreting **abc** -> **cba**. The information displayed in each image can be thought of as an "explanation" of the answer just found, and forms the basis of a representation for that answer in memory.

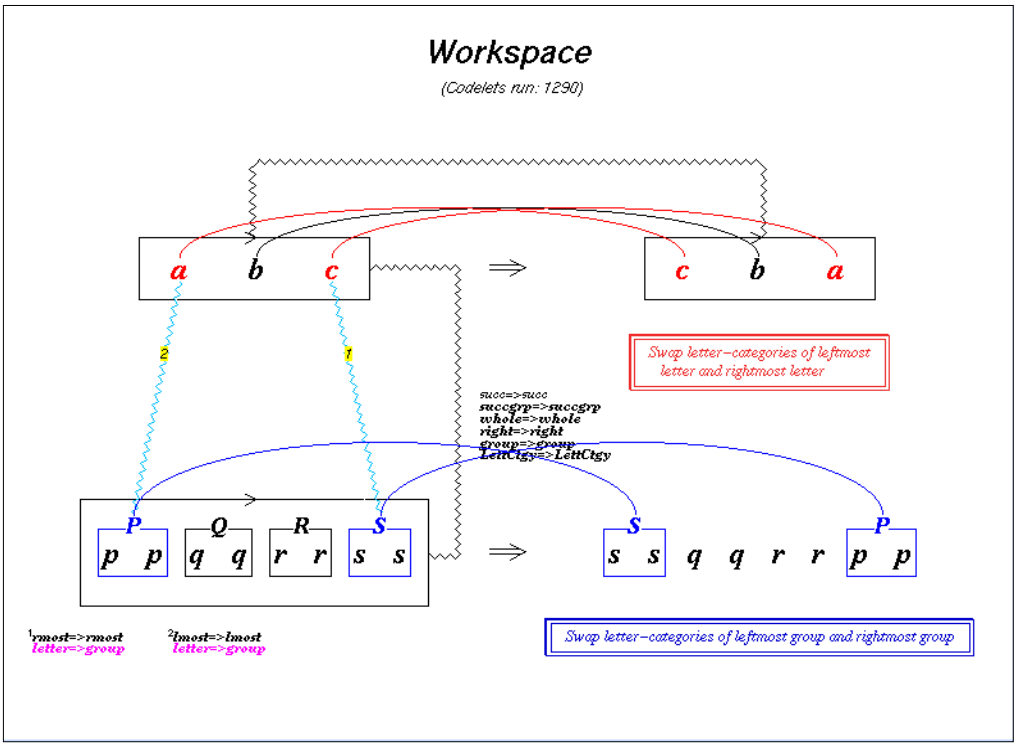
- Answer 1: **ssrrqqpp** (by seeing the string **abc** as reversing direction)



- Answer 2: **ssqqrrpp** (by seeing the letters **a** and **c** as swapping positions)



- Answer 3: **ssqqrrpp** (by seeing the letters **a** and **c** as swapping letter-categories)



In each case, the actual answer is the string in the lower-right corner. The rule describing the interpretation of **abc** -> **cba** is shown in red, along with the structures that support this interpretation. The translated rule, describing the analogous way in which **ppqqrrss** changes, is shown in dark blue, along with the structures that change according to the translated rule. Bridges between the initial and target strings link letters or groups that are perceived as playing analogous roles in their respective strings. Bridges between letters or groups that change in an analogous fashion are highlighted in light blue. Finally, the concept-mappings associated with the bridges in the initial-target mapping are listed for each bridge. Of these, the conceptual slippages involved in translating the rule are highlighted in magenta.

Step Three: Self-Watching

The most important type of explanatory information that will be stored with answers, however, won't be the rules, bridges, or other supporting structures as just described. Rather, new types of structures, representing important events that occur during processing, will be created from time to time in Metacat's Workspace as the program works on an analogy problem. These structures, called themes, might represent, for instance, the occurrence of important slippages, or the recognition of abstract concepts involved in making an analogy. They will explicitly represent patterns and relationships perceived, not among Metacat's perceptual input (*i.e.*, the letter strings), but rather within Metacat's own processing of that input, and will form the basis of the program's "self-watching" ability. Thus, whenever Metacat discovers a new answer to a problem, it will have ready access to an explicit temporal trace of the important themes that led up to the discovery of the answer, which can then be stored along with the answer's other

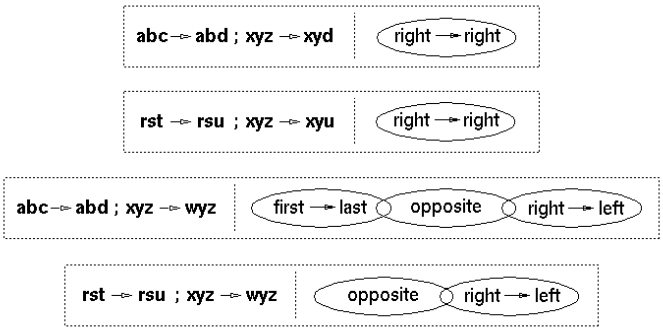
supporting structures in memory. In some ways, this idea is similar in flavor to work on derivational analogy, in which a system stores temporal traces of a problem-solving session in memory for use in analogous situations that may arise later, so as to improve the system's level of performance (Carbonell, 1986). As mentioned earlier, however, the focus in Metacat is not on learning to make better analogies, but rather on being able to explain why one analogy is judged to be more compelling than another.

A Simple Example

As a quick sketch of how themes will allow the similarities and differences between analogy problems to be perceived and contrasted, consider the problem **abc** -> **abd**; **xyz** -> **?**. This problem has been discussed at length elsewhere (Mitchell, 1993), so we summarize briefly here. In the letter-string microworld, **a** has no predecessor and **z** has no successor, so a straightforward answer based on taking the successor of **z** in **xyz** is impossible. One way out is simply the literal-minded answer **xyd**. On the other hand, if the symmetry between the "opposite" letters **a** and **z** is noticed, then the answer **wyz** suggests itself, based on seeing **abc** and **xyz** as starting at opposite ends of the alphabet, with **abc** going in the "successor" direction and **xyz** going in the opposite "predecessor" direction. This answer is very elegant, and most people see it as being strongly analogous to **abd**, even though it is not at all obvious at first.

A related problem is **rst** -> **rsu**; **xyz** -> **?**. Essentially the same arguments as before can be applied to this problem, yielding the answers **xyu** and **wyz**. Seeing **xyu** as the answer is based on seeing **rst** and **xyz** as both going in the same direction, while the answer **wyz** is based on seeing them as going in opposite directions. However, there is really no compelling justification for seeing **rst** and **xyz** as going in opposite directions, unlike in the previous case of **abc** and **xyz**, with their strong **a**-**z** symmetry. Thus, **wyz** seems weaker as an answer to the problem **rst** -> **rsu**; **xyz** -> **?** than it does as an answer to **abc** -> **abd**; **xyz** -> **?**. On the other hand, **xyd** and **xyu** seem like equally valid (if mediocre) answers for their respective problems. How might Metacat be able to explain these differences?

As it works on these problems, Metacat will notice interesting events that happen along the way--the activation of the concept of Opposite, for instance, or the recognition of the symmetric relationship between **a** and **z**. When it finds an answer, it will store a representation of the answer in memory, incorporating into this representation the important events or themes involved in the discovery of the answer. A schematic diagram shows the contents of Metacat's memory after having found the four answers to the two problems discussed above.



Each of the four structures represents an analogy problem that the program has solved. The stored information includes the problem itself, the particular answer found, and the themes that gave rise to the answer. The first two structures represent the "literal" answers **xyd** and **xyu**, each one based on the theme of mapping the initial and target strings onto each other in the same direction (*i.e.*, right). The third structure represents the answer **wyz** to the problem **abc** -> **abd**; **xyz** -> **?**. The themes underlying this answer include the theme of alphabetical symmetry (i.e., the symmetry between the alphabetical-first letter **a** and the alphabetical-last letter **z**), and the theme of directional symmetry (*i.e.*, the symmetry between the right-directed string **abc** and the left-directed string **xyz**). The fourth structure, representing the answer **wyz** to the problem **rst** -> **rsu**; **xyz** -> **?**, is similar, except here only themes representing directional symmetry are present, since in this problem no alphabetical-opposite slippage between **r** and **z** can occur.

Given these four answer-representations in memory, the stage is set for comparing and contrasting them based on the thematic information they contain. Clearly, the important difference between the two **wyz** answers is the absence of alphabetical-symmetry themes in the **rst** case. Herein lies the difference in quality of **wyz** as an answer to these two problems. Furthermore, the thematic characterizations of the "literal" answers **xyd** and **xyu** are identical, reflecting the absence of qualitative differences between them. In fact, it is even possible to see a kind of "meta-level" analogy between the weak answers **xyd** and **xyu** that is qualitatively much stronger than the corresponding analogy between the two **wyz** answers, even though in the latter case the answers are identical, and are of higher quality individually than the answers in the former case.

Conclusion

Enriching Metacat's understanding of its answers by incorporating higher-order thematic information gleaned from self-watching should enable it to perceive abstract similarities and differences among the analogies it makes. It should be able to apply the same processing mechanisms that it now uses to perceive relationships in its perceptual input to the more abstract task of perceiving relationships among the answers that it finds, comparing and contrasting them in a much more interesting way than currently possible. In short, it should eventually be able to make analogies between analogies. Endowing Copycat with a sophisticated self-watching capability forms the central theme of present efforts to extend and refine the model, and is a logical next step along the road to understanding and capturing the full richness of high-level perception and analogy-making in a computational framework.

References

Blank, D., Meeden, L., and Marshall, J. (1992). Exploring the symbolic/subsymbolic continuum: A case study of RAAM. In Dinsmore, J. (Ed.), *The Symbolic and Connectionist Paradigms: Closing the Gap* (pp. 113-148). Lawrence Erlbaum.

Carbonell, J. (1986). Derivational analogy: A theory of reconstructive problem solving and expertise acquisition. In Michalski, R., Carbonell, J., and Mitchell, T. (Eds.), *Machine Learning: An Artificial Intelligence Approach, Volume II* (pp. 371-392). Morgan Kaufmann.

Chi, M., Bassok, M., Lewis, M., Reimann, P., and Glaser, R. (1989). Self-explanations: How students study and use examples in learning to solve problems. *Cognitive Science*, **13**, 145-182.

Hofstadter, D., and Mitchell, M. (1994). The Copycat project: A model of mental fluidity and analogy-making. In Holyoak, K. and Barnden, J. (Eds.), *Advances in Connectionist and Neural Computation Theory, Volume 2: Analogical Connections* (pp. 31-112). Ablex.

Mitchell, M. (1993). *Analogy-Making as Perception*. Cambridge: MIT Press/Bradford Books.

VanLehn, K., Jones, R., and Chi, M. (1992). A model of the self-explanation effect. *The Journal of the Learning Sciences*, **2**, 1-59.