Making Sense of Analogies in Metacat

James B. Marshall
Douglas R. Hofstadter

Center for Research on Concepts and Cognition
Indiana University
Bloomington, Indiana 47408 USA
{marshall,dughof}@cogsci.indiana.edu

Abstract

This paper outlines the main ideas and objectives of the Metacat project, an extension of the Copycat computer model of analogy-making and high-level perception. The principal features of Metacat that allow it to make sense of analogies suggested to it by the user are described using a simple example.

Introduction

The Copycat computer model of analogy-making and high-level perception was originally developed by Hofstadter & Mitchell as a computational model of subcognitive mechanisms underlying human cognition, in which the notion of *fluid concepts* plays a central role. Copycat models the process of analogy-making within a stripped-down microworld of tiny, idealized situations represented as short strings of letters. For example, a typical Copycat problem is the following: "If **abc** changes to **abd**, how does **mrrjjj** change in an analogous way?" This microworld, though austere, harbors a surprisingly rich variety of subtle problems in which a wide range of answers is almost always possible—often including deeply elegant but non-obvious ones. For example, there are many defensible answers to the above problem, including **mrrkkk**, **mrrjjk**, **mrrjjd**, **mrrddd**, **mrrjjj** (in which only **c**'s are seen as changing), **mrsjjj**, **mrdjjj**, **mrrjjjj**, **mrrkkkk**, or even **abd** or **abbddd**. The apparent simplicity of Copycat's domain is deceptive, for it remains a formidable challenge to develop a computational model exhibiting a level of creative and flexible behavior comparable to that of humans even in this tiny, restricted domain of letter-strings.

Copycat discovers analogies between different situations by building up an understanding of the situations in terms of concepts that it understands about the letter-string world. Representations of these concepts are hard-wired into the program, yet they are not static entities with sharply defined boundaries. Rather, their boundaries are inherently fuzzy, overlapping each other to varying degrees and changing in response to competing contextual pressures that arise during the course of processing. The dynamic, "fluid" nature of Copycat's concepts is intended to model the extremely flexible human ability to perceive dissimilar things as being in fact "the same" when viewed at some appropriate level of description.

A detailed exposition of the Copycat program can be found in [Mitchell, 1993] and [Hofstadter and FARG, 1995]. In this paper, we give just a brief summary of Copy-

cat and then discuss in more detail recent work aimed at extending the model. The goal of the current project, dubbed Metacat, is to increase the program's "awareness" of its own behavior as it solves analogy problems, so that it may gain deeper insights into the analogies it makes.

The Copycat Model

When Copycat is given an analogy problem to work on, it starts out with the letter-strings in its *Workspace*, the architectural component of the program in which all perceptual processing occurs. Small, nondeterministic processing agents called *codelets* notice relations among the individual letters and build new structures around them, organizing them into a coherent high-level picture. All processing occurs through the collective actions of many codelets working in parallel, at different speeds, on different aspects of an analogy problem, without any centralized executive process controlling the course of events. The stochastic behavior of codelets is dynamically biased by the time-varying pattern of activation in the program's network of concepts, called the *Slipnet*, that it uses to build up an understanding of an analogy problem. In turn, this context-dependent pattern of conceptual activity in the Slipnet is itself an emergent consequence of codelet processing in the Workspace.

For example, in order to discover an answer to the problem "**abc** ⇒ **abd**; **mrrjjj** ⇒ **?**", codelets work together to build up a strong, coherent mapping between the *initial* string **abc** and the *target* string **mrrjjj**, and also between the initial string and the *modified* string **abd**. Within each letter-string, codelets attempt to build hierarchical *groups*, effectively organizing the strings (the raw perceptual data) into coherent, chunked wholes. In **mrrjjj**, for example, codelets might build the "sameness-groups" **rr** and **jjj**, causing the *sameness-group* concept in the Slipnet to become activated, which in turn makes it more likely for the program to regard **m** as a sameness-group of length one within the context of the other groups in its string. A higher-level "successor-group" comprised of **m**, **rr**, and **jjj** encompassing the entire string can then be seen based on the concept of *group-length* (*i.e.*, *1–2–3*) rather than on *letter-category*. Consequently, the letter-category-based successor-group **abc** can be mapped as a whole onto the length-based successor-group **mrrjjj**, representing the recognition of these strings as instances of the same concept, even though their surface resemblance is negligible. 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 **c** in **abc** and **jjj** in **mrrjjj**, supported by the concept-mappings *rightmost* ⇒ *rightmost* and *letter* ⇒ *group*, representing the idea that both objects are rightmost 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 perceive superficially-dissimilar situations as being identical at a deeper level.

Once a strong, coherent mapping has been built between the initial string and the modified string, another type of structure, called a *rule*, may get created based on this mapping, which succinctly describes the way in which the initial string changes into the modified string. There are often several 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. The configuration of structures in the Workspace collectively represents the way in which a given analogy problem is interpreted. A particular interpretation implies a particular answer for the problem. To produce an answer, the rule describing the way the initial string changes is translated into a new rule that applies to the target string, based on the slippages underlying the initial/target mapping. For example, if the **abc** ⇒ **abd** change is described according to the first rule above, and the abstract successor-group similarity between **abc** and **mrrjjj** has been noticed, then the rule will be translated as *Change length of rightmost group to successor*, yielding the answer **mrrjjjj**. On the other hand, if this deep similarity has not been noticed, the answers **mrrkkk**, **mrrjjk**, **mrrddd**, or **mrrkkd** may be found instead, depending on the rule chosen and whether or not **c** in **abc** is seen as corresponding to the **jjj** group or to just the rightmost letter **j** in **mrrjjj**.

As this example suggests, Copycat's stochastic processing mechanisms enable it to find a range of different answers for a given analogy problem. Copycat attaches a rough numerical measure of "quality" to the answers it finds, which, for many problems, corresponds reasonably well to human judgments of relative answer quality. But the program has very little awareness of how it actually finds the answers that it finds. It has almost no insight into its own processing mechanisms—fluid and flexible though they may be—which guide it through the "space" of possible interpretations of an analogy problem. This is not too surprising, however, given that Copycat was intended primarily as a model of subcognitive mechanisms. All of the nondeterministic codelet activity occurring in the Workspace—the building of bridges and groups, the making of slippages, and so on—is intended to represent perceptual activity carried out below the level of "conscious awareness". In contrast, the focus of Metacat is on developing mechanisms that support a higher "cognitive" level on top of Copycat's subcognitive level. To do this, Metacat needs to be able to remember what happens while its subcognitive mechanisms are building, destroying, and reconfiguring Workspace structures in pursuit of an answer to the problem at hand, and to build explicit representations of this activity.

Metacat's Objectives

Hofstadter has outlined several important objectives for the Metacat project [Hofstadter and FARG, 1995, Chapter 7]. First of all, the program should be able to explicitly characterize the *essence* of an answer—the core idea or cluster of ideas underlying the answer that fundamentally distinguishes it from other possible answers. The ability to perceive what a given answer is really "about" should enable the program to give at least a limited explanation of the answer's strengths and weaknesses compared to other answers it may have previously found. For example, the essence of the **mrrjjjj** answer described earlier lies in seeing both **abc** and **mrrjjj** as successor-groups, one based on the concept of *letter-category* and the other based on the concept of *group-length*. The recognition of this abstract similarity between the strings is what fundamentally distinguishes the answer **mrrjjjj** from other, more straightforward answers such as **mrrkkk**, **mrrjjk**, or **mrrddd**, in which the hidden "successorship fabric" of **mrrjjj** remains unnoticed.

The ability to compare and contrast answers, however, implies the ability to remember more than one at a time. In Copycat, answers are not retained after they are found. When Copycat discovers an answer to a problem, it simply reports the answer, along with the answer's numerical measure of quality, and then stops. No recollection of previously found answers is possible on subsequent runs of the program, 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 within a single analogy problem or across different problems. In contrast, Metacat should remember the answers it finds, along with characterizations of the key ideas involved, gradually building up in its memory a repertoire of experience on which it can draw when confronted with new situations.

In addition to remembering the answers it finds, Metacat should also keep track of patterns that occur in its own processing while it is trying to discover new answers. As it works on an analogy problem, it should create an explicit sequential trace of its own behavior as it searches through the space of possible interpretations leading to different answers. This type of memory is of a more short-term, temporal nature than that just described for the answers themselves. Such a *self-watching* ability would enable Metacat not only to remember the important events that led it to find an answer, but also to recognize when it has fallen into a repetitive or otherwise unproductive pattern of behavior. Recognizing that it is in a "rut" should enable it to subsequently "jump out of the system" by explicitly focusing on ideas other than the ones that seem to be leading it nowhere. This type of self-awareness pervades human cognition. People can easily pay attention to patterns in their own thinking; see for example [Chi et al., 1989, VanLehn et al., 1992].

Once Metacat has the ability to size up the answers it finds in terms of their essential features, it ought to be able to evaluate other answers suggested to it by some outside agent. In other words, Metacat should not only be able to come up with answers to analogy problems on its own, it should also be able to justify answers on their own terms, even if the program itself didn't come up with them. This constitutes an ability to work "backwards" from a given answer toward an insightful characterization of the answer, in order to understand why it makes sense. Once an answer has been understood in this way, it could then be compared and contrasted with other answers that the program has either itself discovered previously, or been shown by someone else.

The Metacat Model

The Metacat architecture includes all of Copycat's main architectural components, such as the Workspace, the Slipnet, and the mechanisms that support distributed, nondeterministic codelet processing. In addition, new architectural components have been incorporated into the model, and mechanisms for building bridges and creating rules have been extended and generalized. These components provide a general framework in which to address the objectives outlined in the previous section.

Unlike Copycat, Metacat incorporates a memory for its answers, which allows it to remember more than one answer over the course of a run. Whenever it finds a new answer, instead of simply stopping, Metacat pauses to display the answer along with the Workspace structures representing the interpretation of the problem. This information is packaged together and stored in Metacat's memory, after which the program continues searching for alternative answers to the problem. Gradually over time, a series of answers accumulates in memory, each one representing a different way of making sense of the analogy problem at hand.

The most important type of auxiliary information stored with answers consists of structures called *themes*. Themes reside in Metacat's *Themespace*, and represent key concepts underlying the mappings created between letter-strings. Collections of themes serve as high-level characterizations of Metacat's answers, and provide a basis on which to compare and contrast answers with each other. Themes are comprised of Slipnet concepts, and assume time-varying levels of activation ranging from -100 to $+100$, depending on the extent to which the ideas they represent are present or absent in a particular configuration of Workspace structures.

As an example, consider the structures shown in Figure 1. This figure shows the status of the Workspace during a run on the problem "**abc** ⇒ **abd**; **xyz** ⇒ **?**", in which the answer **wyz** has already been suggested to the program by the user. In this run, Metacat is attempting to justify the **wyz** answer by searching for an overall interpretation of the problem in which this particular answer makes sense. The Workspace shows the various structures that have been built after several hundred codelets have run[1], including horizontal bridges comprising the **abc** ⇒ **abd** and **xyz** ⇒ **wyz** mappings, and vertical bridges comprising the mapping between **abc** and **xyz**. Concept-mappings supporting the vertical bridges can also be seen (concept-mappings for the horizontal bridges are not displayed). Immediately above and below the Workspace are rules describing how **abc** changes to yield the string **abd** (the top rule), and how **xyz** changes to yield **wyz** (the bottom rule).

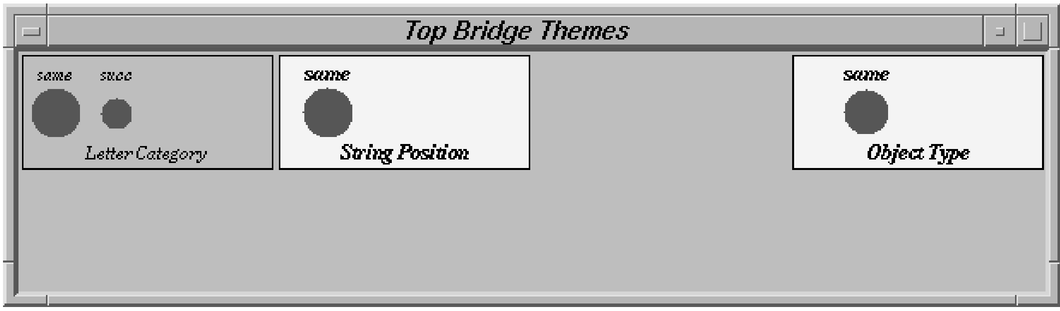
At the top of the figure is shown a portion of the Themespace containing four "top-bridge" themes characterizing the horizontal **abc** ⇒ **abd** bridges. These themes are shown as circles of differing sizes according to the activation level of each theme. The first two themes reflect the ideas of letter-category sameness and letter-category successorship within the **abc** ⇒ **abd** mapping. The **a**–**a** and **b**–**b** bridges both involve the idea of letter-category sameness, while the **c**–**d** bridge involves the idea of successorship. On the other hand, all bridges map objects of identical string-position (*e.g.*, *leftmost* ⇒ *leftmost*) and object-type (*e.g.*, *letter* ⇒ *letter*) onto each other, so the string-position and object-type sameness themes are highly active. The latter two themes serve as an abstract characterization of the **abc** ⇒ **abd** mapping. Other sets of themes in the Themespace characterize other Workspace structures in a similar fashion.

Thus, themes are first and foremost representational structures. But under certain conditions, when highly activated, they can also exert powerful *top-down pressure* on Metacat's processing mechanisms, strongly biasing the stochastic behavior of codelets in favor of particular outcomes. Active themes can be regarded as Metacat's way of "seizing on" certain key ideas implicit in an analogy problem and making them explicit, driving the program toward an interpretation of the problem organized around these ideas.

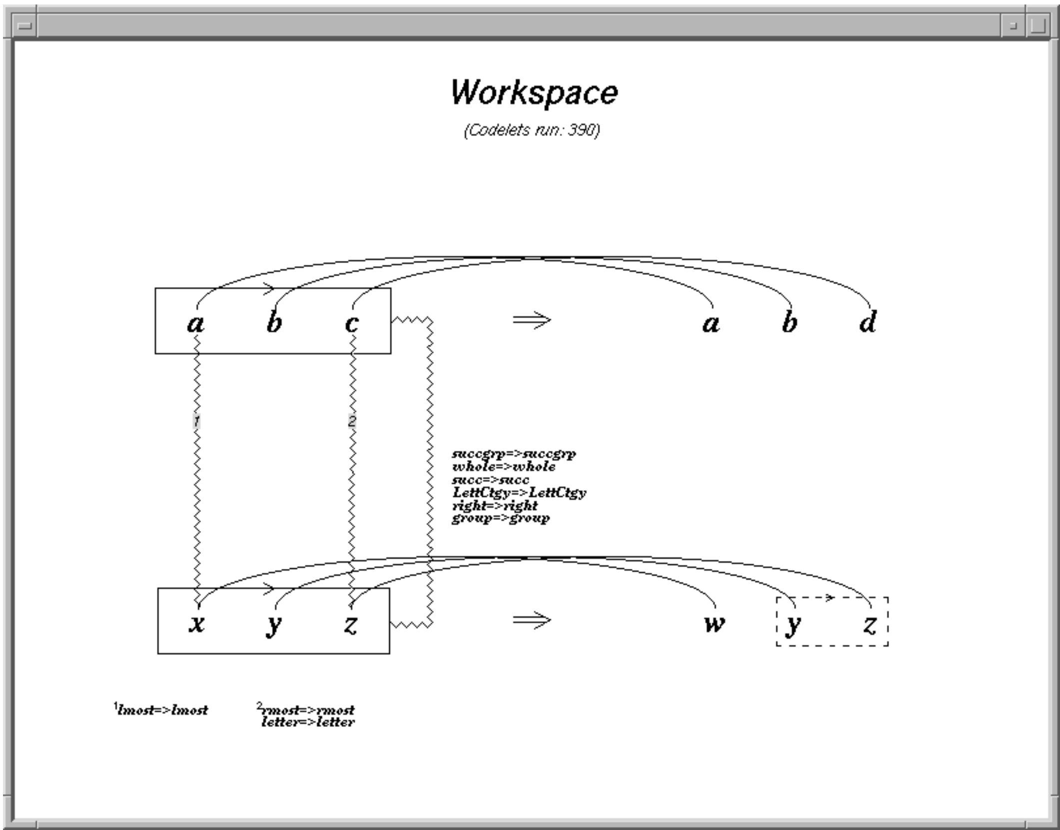
In the Figure 1 example, Metacat has perceived **abc** and **xyz** as successor-groups going in the same direction (left-to-right). This is represented by the vertical **a**–**x** and **c**–**z** bridges, which are supported by the concept-mappings *leftmost* ⇒ *leftmost* and *rightmost* ⇒ *rightmost*, respectively. However, this way of interpreting the situation doesn't make sense, because **c** and **x** are not seen as corresponding to each other (since there is no bridge between them), yet they are both identified by the rules as being the objects that change in their respective strings (the **c** to its successor and the **x** to its predecessor).

At some point, codelets may com-

[1] The dotted **yz** group is a "tentative" structure, which has been proposed by codelets but has not actually been built yet.

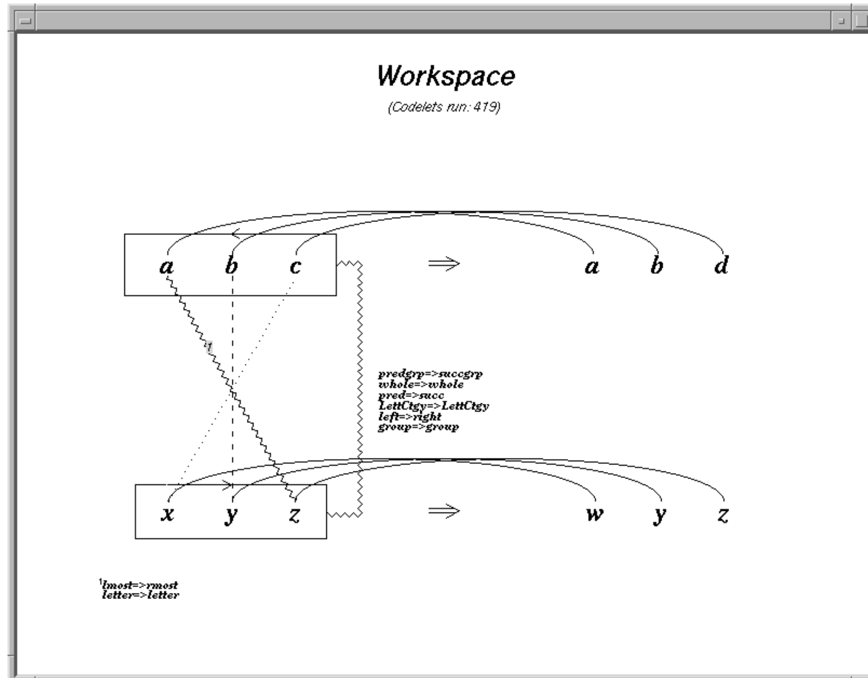


Change letter–category of rightmost letter to successor



Change letter–category of leftmost letter to predecessor

Figure 1: An inconsistent interpretation of the answer **wyz**



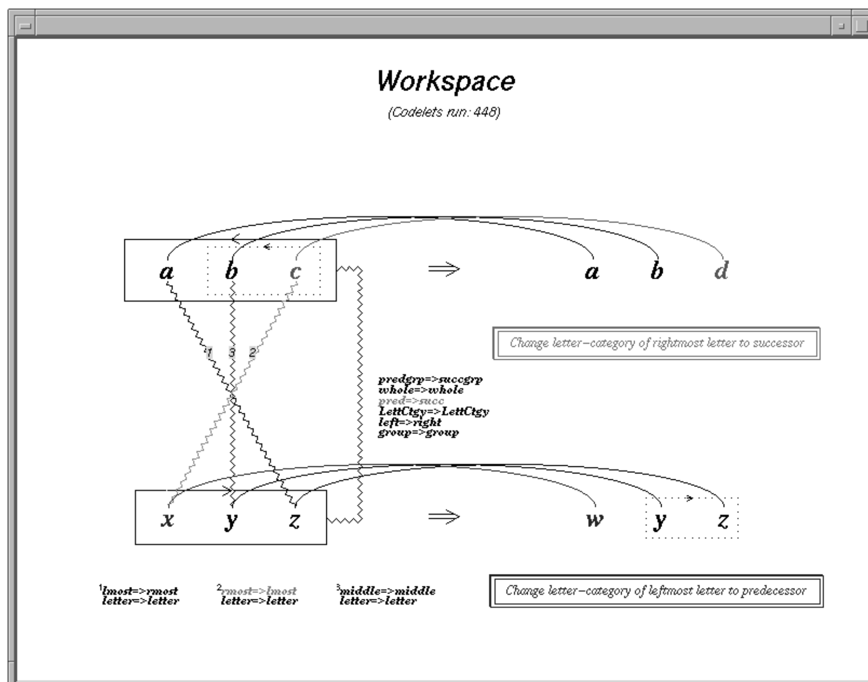


Figure 2: The final consistent interpretation of **wyz**

pare the two rules and notice that taken together, they imply the concept-mappings *rightmost* ⇒ *leftmost* and *successor* ⇒ *predecessor*. These concept-mappings suggest the idea of mapping the strings **abc** and **xyz** onto each other in a *crosswise* fashion, so that one group is viewed as a successor-group and the other is viewed as a predecessor-group, with the rightmost letter of one corresponding to the leftmost letter of the other, and vice versa. This idea can be succinctly characterized by a set of "vertical-bridge" themes representing string-position and group-direction symmetry. These themes are then clamped at full activation, strongly promoting the creation of new structures compatible with the idea of a crosswise mapping and greatly weakening existing structures incompatible with this idea. The net effect is that the original vertical mapping shown in Figure 1 is swiftly reorganized by codelets into a new mapping consistent with the activated themes. The top image in Figure 2 shows the Workspace shortly after the themes have been clamped. The original **a**–**x** and **c**–**z** bridges have been destroyed and new bridges are being built in their place. A quite different interpretation of the analogy problem is now emerging. The bottom image shows the final interpretation, in which **c** and **x** are seen as corresponding. In this way, themes allow Metacat to effectively work backwards from a given answer to a high-level understanding of why the answer makes sense, even if the program would be hard-pressed on its own to come up with the answer in question.

Conclusion

To summarize, themes in Metacat can be viewed as a medium through which ideas made explicit at the "cognitive" level can actively influence and guide the course of processing at the "subcognitive" level. By strongly activating different patterns of themes in the Themespace, the program can explicitly focus on different high-level ideas as it works on understanding an analogy problem. Furthermore, once an answer has been understood, its associated themes represent a characterization of the key ideas underlying the answer, which can subsequently be used as the basis for comparing and contrasting the answer with other answers encountered previously.

Acknowledgements

This research is supported in part by Sun Microsystems Co. Academic Equipment Grant #EDUD-NAFO-960418 and by grants to the Center for Research on Concepts and Cognition from the College of Arts and Sciences of Indiana University.

References

[Chi et al., 1989] 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 and FARG, 1995] Hofstadter, D. R. and FARG (1995). *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*. Basic Books, New York.

[Mitchell, 1993] Mitchell, M. (1993). *Analogy-making as Perception*. MIT Press/Bradford Books, Cambridge, MA.

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