From Copycat to Metacat: Developing a Self-Watching Framework for Analogy-Making

James B. Marshall

Center for Research on Concepts and Cognition
Department of Computer Science
Indiana University
Bloomington, Indiana 47408 USA
marshall@cs.indiana.edu

Abstract

This paper describes Metacat, an extension of the Copycat computer model of fluid concepts, high-level perception, and analogy-making. Copycat models the complex interplay between concepts and perception that gives rise to the flexible human ability to see apparently-dissimilar situations as being "the same". A key feature of the architecture is the emergence of statistically robust high-level behavior from the interactions of many small, low-level, nondeterministic processing agents. Metacat focuses on extending the architecture in a way that permits it to create rich representations of the analogies it makes. This involves incorporating a long-term memory into the architecture, along with a "self-watching" ability, so that the program can recognize, remember, and recall patterns that occur in its own processing as it solves analogy problems. Using this higher-order "meta-level" information, analogies can be compared and contrasted in an insightful way, allowing Metacat to understand and explain its answers in a way that Copycat cannot.

1 Introduction

This paper describes Metacat, an extension of the Copycat computer model of analogy-making and high-level perception originally developed by Hofstadter & Mitchell [Hofstadter, 1984, Mitchell, 1993, Hofstadter and FARG, 1995]. The main goal of the Copycat project was to develop a computational model of cognition in which *fluid concepts* play a central role. Metacat builds on Copycat's fluid conceptual machinery by incorporating mechanisms for *self-watching* and *episodic memory* into Copycat's architecture. The goal of Metacat is to increase the program's "awareness" of its own behavior as it solves analogy problems, which

in turn allows it to gain deeper insights into the analogies it makes. 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 Copycat and then outline the main ideas of Metacat, illustrating them in more detail by way of several examples.

Copycat operates within a restricted, yet surprisingly rich, microdomain consisting of short strings of letters, where each letter-string can be thought of as representing some abstract, idealized situation. The program perceives analogies between different situations by building up an understanding of the situations in terms of various concepts about the letter-string world that it understands. Copycat's concepts are not static entities with sharply delineated boundaries. Rather, their boundaries are inherently fuzzy, overlapping each other to varying degrees, and changing according to the contextual pressures at hand. The dynamic, fluid nature of Copycat's concepts is intended to model the remarkably flexible human ability to perceive apparently dissimilar things as being in fact "the same" when viewed at some appropriate level of description.

2 The Copycat Model

Consider the following typical Copycat problem: "If **abc** changes to **abd**, how does **mrrjjj** change in an analogous way?" There are many defensible answers to this problem, including **mrrkkk**, **mrrjjk**, **mrrjjd**, **mrrddd**, **mrrjjj**, **mrsjjj**, **mrdjjj**, **mrrjjjj**, **mrrkkkk**, or even **abd** or **abbddd**. Clearly, some of these answers are more obvious than others, and the obvious ones may not be the most aesthetically satisfying ones, but there is no single demonstrably "correct" answer. In fact, a range of answers is possible for almost any imaginable problem in this microworld. The apparent simplicity of Copycat's domain is deceptive, for it remains a formidable challenge to develop a computational model capable of exhibiting a level of flexibility and creativity comparable to human behavior even in this tiny, restricted domain.

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 that serve to organize the letters 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.

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 (that is, 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 built based on the concept of *length* (*i.e.*, *1–2–3*) rather than on *letter-category* (*i.e.*, *m–r–j*). 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*, which represents 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 the same.

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 that 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*; that is, the way in which its strings are perceived in relation to one another. A particular interpretation implies a particular answer for the problem. To produce an answer, the slippages underlying the initial/target mapping are used by codelets to "translate" the rule describing the initial/modified change into a new rule that applies to the target string. 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 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. For many problems, the distribution of answers that it finds agrees reasonably well with the answers typically found by people. Furthermore, 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 (*i.e.*, the possible ways of building up the mappings between the strings). This is not too surprising, however, since Copycat was intended to be a model of the *subcognitive* mechanisms underlying cognition. 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 at the subcognitive level, 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 "watch" and 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 lower-level perceptual activity.

3 The Main Objectives of Metacat

In Chapter 7 of [Hofstadter and FARG, 1995], Hofstadter outlines several important objectives for Metacat. Here we briefly review some of these objectives and then discuss in greater detail how they are addressed in the current Metacat architecture.

3.1 Comparing and contrasting answers

The central objective of Metacat is to increase the program'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. Metacat should be able to

recognize and explicitly represent the most important issues involved in an answer. For example, the essence of the **mrrjjjj** answer for the problem "**abc** ⇒ **abd**; **mrrjjj** ⇒ **?**" lies in seeing both **abc** and **mrrjjj** as *successor-groups*, one based on the idea of *letter-category* and the other based on the idea of *group-length*. The recognition of such an 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. Although Copycat rates **mrrjjjj** substantially higher on its numerical answer-quality scale than it rates the other answers, it nevertheless remains unable to explain *why* it judges this answer to be better than the others. It simply has no insight into what makes **mrrjjjj** a better answer. Metacat needs to be able to create rich characterizations of its answers that can serve as a basis for comparing and contrasting them in an insightful way.

The ability to compare answers based on their key underlying similarities and differences essentially amounts to the ability to make analogies between analogies, since each answer itself represents an analogy between a particular set of situations (*i.e.*, letter-strings). With rich enough answer characterizations, Metacat ought to be able to notice when two answers resemble one another *in the same way* that two *other* answers resemble each other, thereby creating a higher-order "meta-level" analogy involving four "first-order" analogies.

3.2 Remembering answers

In order to compare and contrast answers, Metacat necessarily needs to be able 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 quits. 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 within a single analogy problem or across different problems. In contrast, Metacat should remember the answers it finds, along with the characterizations of their key underlying ideas, gradually building up in its memory a repertoire of experience on which it can draw when confronted with new situations. A new analogy problem, if sufficiently similar to some other problem already stored in memory, should remind Metacat of the previously encountered problem. Once activated, the previous situation can then be compared with the current one, possibly influencing the interpretation of the current situation as a consequence.

3.3 Self-watching

In addition to remembering the individual answers it finds, Metacat should 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 [Chi et al., 1989, VanLehn et al., 1992].

3.4 Understanding a given answer

Finally, once Metacat has the ability to characterize the answers it finds in terms of their key underlying ideas, it ought to be able to characterize 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 can then be compared and contrasted with other answers that the program has either itself discovered previously, or been shown by someone else.

4 From Copycat to Metacat

The remainder of this paper describes the general framework in which the preceding ideas are addressed in the current Metacat architecture. Metacat is an extension of the Copycat model—not an alternative model designed to supplant it. Consequently, Metacat's architecture includes all of Copycat's main architectural components, such as the Workspace, the Slipnet, and the mechanisms that support distributed, nondeterministic codelet processing. Furthermore, the mechanisms for building bridges and creating rules have been greatly extended and generalized in Metacat. Unlike Copycat, however, Metacat incorporates a memory for its answers, which allows it to remember several different answers during a single run on a given analogy problem. Whenever it finds a new answer, instead of simply

quitting, Metacat pauses temporarily to display the answer along with the groups, bridges, rules, slippages, and other Workspace structures that gave rise to it. Together these structures represent a way of interpreting the analogy problem that yields the answer just found. All of this information, including the problem itself, is then 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 containing much more information than just the answer string itself. Each stored answer represents a different way of looking at or making sense of an analogy problem.

The most important auxiliary information stored with answers, however, consists of other types of structures called *themes*. Themes reside in Metacat's *Themespace*, and represent key concepts underlying the mappings created between letter-strings. They serve as the basis for Metacat's high-level characterizations of its answers, which it uses to compare and contrast answers with each other. Themes are comprised of Slipnet concepts, and can take on various levels of *activation*, depending on the extent to which the ideas they represent are present or absent in a particular configuration of Workspace structures.

As an example of how the similarities and differences between analogy problems can be characterized in terms of Metacat's themes, consider the problem "**abc** ⇒ **abd**; **xyz** ⇒ **?**". In the letter-string domain, **a** has no predecessor and **z** has no successor. The alphabet is explicitly designed not to "wrap around" from **z** back to **a**, so a straightforward answer based on taking the successor of **z** in **xyz** is impossible. One is forced to adopt a different strategy as a result of this constraint. 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 "mirror images" of each other starting at opposite ends of the alphabet, with **abc** going to the right based on the idea of successorship and **xyz** going to the left based on predecessorship. 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.

Figure 1 shows Metacat's final Workspace configuration after it has found the answer **xyd**. In this interpretation of the problem, the strings **abc** and **xyz** are seen as mapping onto each other in a straightforward, left-to-right fashion: letters with identical string-positions are linked by vertical bridges supported by the concept-mappings *leftmost* ⇒ *leftmost*, *middle* ⇒ *middle*, and *rightmost* ⇒ *rightmost*. Furthermore, the strings **abc** and **abd** are mapped onto each other in a similar fashion, as shown by the horizontal bridges across the top. The literal-minded way of viewing the **abc** ⇒ **abd** mapping is represented by the rule *Change letter-category of rightmost letter to d*. In the absence of any conceptual slippages, this rule gets applied unchanged to **xyz**, yielding the answer **xyd**. This interpretation of the problem relies on (1) seeing **abc** and **xyz** as going in the same direction; (2) seeing **abc** and **abd** as going in the same direction; and (3) seeing **abc** as changing to **abd** in a literal way rather than in a more abstract way (*i.e.*, such as *Change letter-category of rightmost letter to suc-*

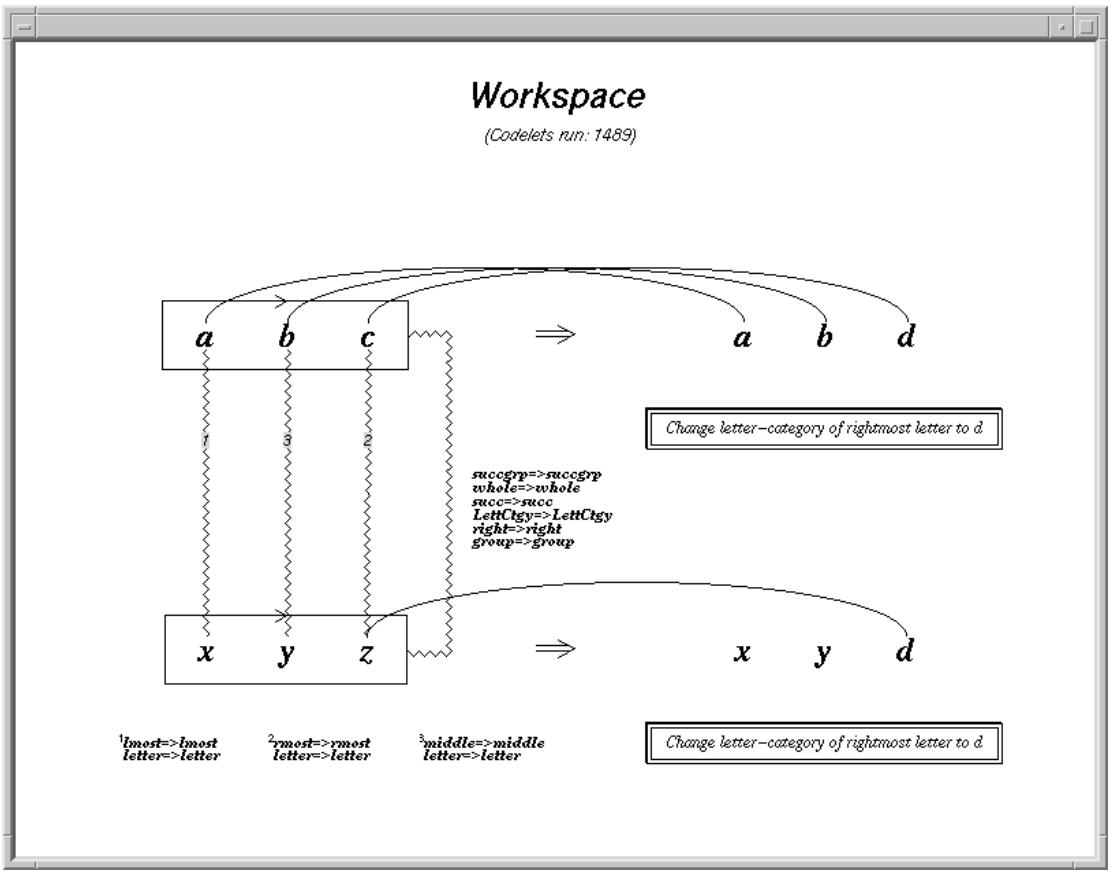

Figure 1: *Metacat's interpretation of the problem "***abc** ⇒ **abd***;* **xyz** ⇒ **?***" yielding the answer* **xyd***, showing the various Workspace structures involved.*

cessor). These are the central ideas underlying the answer **xyd**. Metacat represents these ideas explicitly as themes in its Themespace (not shown). The first idea is represented by a "vertical" *Direction:Same* theme; the second idea by "top" *Direction:Same* theme; and the third idea by a *Rule:Literal* theme. These themes get created in the Themespace—and then activated—by structure-building activity occurring in the Workspace as Metacat gradually builds up its interpretation of the problem. For example, as more vertical bridges are built in support of a same-direction mapping between **abc** and **xyz**, the vertical *Direction:Same* theme becomes more strongly activated. Once the **xyd** answer has been found, the most strongly-activated themes get stored in memory along with the answer, and together they constitute the answer's high-level *thematic characterization*.

In contrast, Figure 2 shows the Workspace after the answer **wyz** has been found. In this interpretation, **abc** and **xyz** are mapped onto each other in a crosswise fashion. The strings are thus seen as going in opposite directions, one starting with **a** and the other starting with **z**. In addition, a *first* ⇒ *last* slippage supports the **a**–**z** bridge, which represents the idea that the alphabetic-first letter **a** is seen as corresponding to the alphabetic-last letter **z**. The **abc** ⇒ **abd** mapping, however, is the same as before, except that it is now described by the more abstract rule *Change letter-category of rightmost letter to successor*. The crosswise vertical mapping causes this rule to be translated as *Change letter-category of leftmost letter to predecessor*, yielding the answer **wyz**. The crux of interpreting the problem this way involves (1) noticing the alphabetic-position symmetry between the letters **a** and **z**; (2) seeing **abc** and **xyz** as going in opposite directions, which is a consequence of (1); (3) seeing **abc** and **abd** as going in the same direction; and (4) seeing **abc** as changing to **abd** in an abstract, non-literal way. The thematic characterization of **wyz** thus consists of the vertical themes *Alphabetic-position:Opposite* and *Direction:Opposite*, the top theme *Direction:Same*, and the rule theme *Rule:Abstract*.

A related problem is "**rst** ⇒ **rsu**; **xyz** ⇒ **?**". Essentially the same arguments that applied in the previous problem can be applied to this problem, yielding the answers **xyu** and **wyz**. Seeing the answer **xyu** is based in part on seeing **rst** and **xyz** as going in the same direction, while the answer **wyz** depends on seeing them as going in opposite directions. However, in this problem there really is 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. Indeed, the presence or absence of alphabetic-position symmetry is the crucial difference between the two **wyz** answers. Everything else about them is the same: both involve (1) seeing **abc** and **xyz** (or **rst** and **xyz**) as going in opposite directions; (2) seeing **abc** and **abd** (or **rst** and **rsu**) as going in the same direction; and (3) seeing the **abc** ⇒ **abd** (or **rst** ⇒ **rsu**) change in an abstract, non-literal way. The relative lack of justification for seeing the answer **wyz** in the second problem tends to diminish its overall quality. While arguably better than **xyu**, **wyz** is not nearly as superior to **xyu** as was **wyz** to **xyd** in the first problem. In short, **xyd** and **xyu** play essentially *identical* roles in their respective

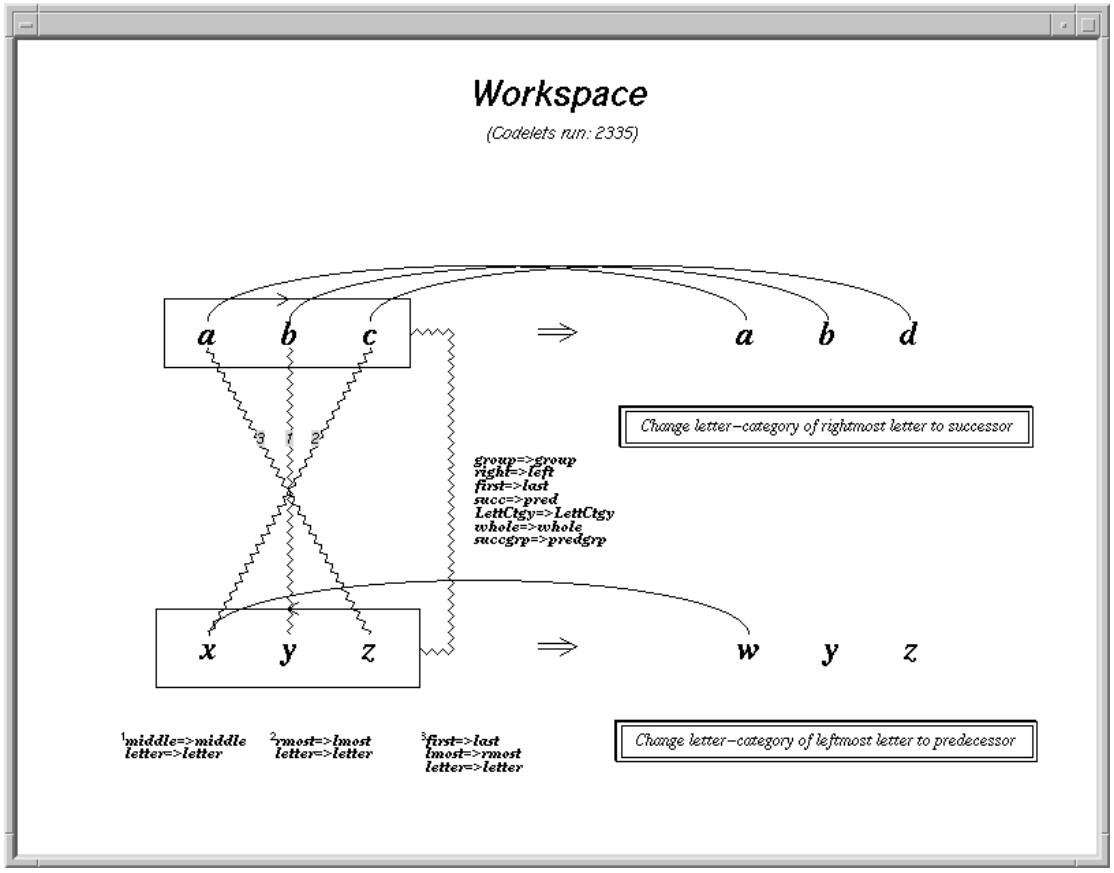

Figure 2: *Metacat's Workspace after having found the answer* **wyz**

Problem/Answer	Vertical Themes	Top Themes	Rule Theme
abc ⇒ **abd**; **xyz** ⇒ **wyz**	*Alpha-Pos:Opposite* *Direction:Opposite*	*Direction:Same*	*Rule:Abstract*
rst ⇒ **rsu**; **xyz** ⇒ **wyz**	*Direction:Opposite*	*Direction:Same*	*Rule:Abstract*
abc ⇒ **abd**; **xyz** ⇒ **xyd**	*Direction:Same*	*Direction:Same*	*Rule:Literal*
rst ⇒ **rsu**; **xyz** ⇒ **xyu**	*Direction:Same*	*Direction:Same*	*Rule:Literal*
abc ⇒ **abd**; **xyz** ⇒ **dyz**	*Alpha-Pos:Opposite* *Direction:Opposite*	*Direction:Same*	*Rule:Literal*
rst ⇒ **rsu**; **xyz** ⇒ **uyz**	*Direction:Opposite*	*Direction:Same*	*Rule:Literal*

Table 1: *Six answers and their associated thematic characterizations.*

problems, and are thus of comparable quality, while the two **wyz** answers are quite *different*, even though on the surface they appear to be identical.

In addition to these four answers, there are two other possibilities worth mentioning. The answer **dyz**, although perhaps a bit far-fetched, is certainly possible for the problem "**abc** ⇒ **abd**; **xyz** ⇒ **?**". Seeing this answer depends on noticing the abstract "mirror image" symmetry between **abc** and **xyz**, yet—somewhat ironically—taking a very literal-minded view of the way in which **c** changes to **d**. Thus, making the "analogous" change to **xyz** involves changing its leftmost letter simply to **d**. The answer **uyz** for the problem "**rst** ⇒ **rsu**; **xyz** ⇒ **?**" arises in a similar manner, except that here there is of course no good reason to see **rst** and **xyz** as mirror images of each other in the first place. Just as for the two **wyz** answers, the key difference between the answers **dyz** and **uyz** lies in the presence or absence of the idea of alphabetic-position symmetry. In other words, the *way* in which the two **wyz** answers are analogous to each other is exactly like the way in which the **dyz** and **uyz** answers are analogous to each other. Here we have a simple example of a "meta-level" analogy in the letter-string microworld.

Table 1 shows these six answers along with their associated thematic characterizations. These characterizations bring out very clearly the similarities and differences among the various answers to the two problems. For example, it is clear from examining the themes that the crucial distinction between the first **wyz** answer and **dyz** is whether **abc** ⇒ **abd** is perceived abstractly or literally (as indicated by the rule theme). The thematic characterizations of **xyd** and **xyu** are identical, revealing the deep underlying similarity between these two literal-minded answers. The difference between the two **wyz** answers rests on the presence or absence of the idea of alphabetic-position oppositeness. Furthermore, the *way* in which these answers differ is precisely the same as the way in which **dyz** differs from **uyz**.

This example, although somewhat simplified, gives the flavor of how Metacat's thematic

characterizations allow it to compare and contrast its answers in an insightful way, even to the point of being able to see higher-level analogies among the analogies it makes. Such an ability lies far beyond that of Copycat, which has only a crude numerical measure of "quality" available as a basis for answer comparison. In addition, answers can be retrieved from memory on the basis of their stored thematic characterizations. For example, suppose that Metacat finds the answer **xyu** for the "**rst** ⇒ **rsu**; **xyz** ⇒ **?**" problem. If it has previously encountered the answer **xyd** for the "**abc** ⇒ **abd**; **xyz** ⇒ **?**" problem, finding **xyu** may remind it of the **xyd** answer it has already seen—based on the strong similarity between **xyu**'s thematic characterization and the stored characterization of **xyd**—prompting Metacat to "comment" on the similarity between the two answers.

As mentioned earlier, themes take on varying levels of activation during the course of processing. At any given moment, a theme's activation level represents an estimate of the importance of its role in characterizing the program's understanding of the situation at hand. 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 subcognitive processing mechanisms, strongly biasing the stochastic behavior of codelets in favor of particular outcomes. For example, in the "**abc** ⇒ **abd**; **xyz** ⇒ **?**" problem, a highly active *Direction:Opposite* vertical theme will strongly promote the creation of structures that support the idea of mapping **abc** onto **xyz** in a crosswise fashion—and will suppress structures that are incompatible with this idea. The creation of *rightmost* ⇒ *leftmost* or *leftmost* ⇒ *rightmost* vertical bridges, for instance, will become extremely likely, whereas *leftmost* ⇒ *leftmost* or *rightmost* ⇒ *rightmost* bridges will be inhibited. Active themes can be thought of 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 key ideas. Different configurations of active themes in the Themespace will guide Metacat to different interpretations of an analogy problem, which consequently may cause different answers to be discovered for the problem.

In fact, themes can assume both positive and negative levels of activation (ranging from −100 to +100). Positively-activated themes exert "positive thematic pressure" as just described, encouraging the building of Workspace structures compatible with the themes. Negatively-activated themes, on the other hand, exert "negative thematic pressure". Their effect is to *discourage* the creation of compatible structures, promoting instead the creation of structures *incompatible* with the themes. For example, a strongly-negative *Direction:Same* vertical theme will discourage the creation of a straightforward left-to-right vertical mapping between **abc** and **xyz**. This in turn will "push" the program into other regions of "interpretation space", encouraging it to explore alternative ways of creating this mapping.

The ability to steer away from certain interpretations of an analogy problem, by negatively activating the themes characterizing their key ideas, offers a way for Metacat to avoid falling into mindlessly-repetitive patterns of behavior, or at least to be able to "jump out of

the system" when it does end up falling into one. Copycat is plagued by such "loopy" behavior on certain problems, for it has no way of noticing patterns in its own processing. When it first tries to solve "**abc** ⇒ **abd**; **xyz** ⇒ **?**", for example, it almost invariably perceives **abc** and **xyz** as going in the same direction. This is certainly a reasonable predisposition. However, such an interpretation of the situation leads inevitably to an attempt to take the successor of **z**, since under this interpretation **c** and **z** are seen as corresponding. This attempt fails, and Copycat "hits a snag". Rather than falling back on seeing **abc** ⇒ **abd** literally, which would give the answer **xyd**, it typically tries to reconfigure the vertical **abc**–**xyz** mapping, breaking bridges or other structures in the process. Unfortunately, it usually just ends up rebuilding the same left-to-right mapping and consequently hitting the snag again, sometimes going round and round in circles hitting the snag over and over, until it finally stumbles on the rule *Change letter-category of rightmost letter to d*. In fact, Copycat hits the snag an average of nine times per run on this problem—sometimes even as many as twenty or thirty times on certain runs. This is quite unlike typical human behavior. People tend to "get the message" after attempting some unsuccessful strategy a few times. They are able to recognize that their strategy isn't working and that they should try something different.

Themes offer a way to address this problem. In addition to storing answers in its memory, Metacat maintains, seperately, an explicit temporal record (called the *Trace*) of the important *events* that occur while it works on some analogy problem. One type of important event that may occur during a run, of course, is the discovery of an answer. But hitting a snag is also important, and Metacat notes this type of event in its temporal trace as well. Another type of "failure event", although less dramatic than hitting a snag, is the case in which the program has simply ceased to make any progress in its attempt to build up a strong, coherent interpretation of the analogy problem at hand. At some point a "no progress" threshold is reached, and the situation is explicitly noted in the trace.

Whenever an event is added to the Trace, the themes most active at the time of the event are also noted along with it. These themes serve as the event's thematic characterization. In the case of an answer event, the answer and its characterization are also stored in Metacat's answer memory, as described earlier. In the case of a snag event, the snag's thematic characterization represents a way of interpreting the analogy problem that leads to failure. If Metacat continues to hit the same snag several times in succession, a series of failure events gets created in the Trace, all with very similar thematic characterizations. Since these processing events are now represented in the Trace as explicit, tangible structures, they are subject to examination and manipulation by codelets in a natural way. That is, the thematic similarity between failure events in the Trace can be noticed by codelets in much the same way that similarity between letters or groups in the Workspace is noticed. By monitoring its own behavior in this way, Metacat can *recognize* when it is stuck in an ongoing, repetitive pattern of behavior. Furthermore, once a particular thematic configuration has been recognized as

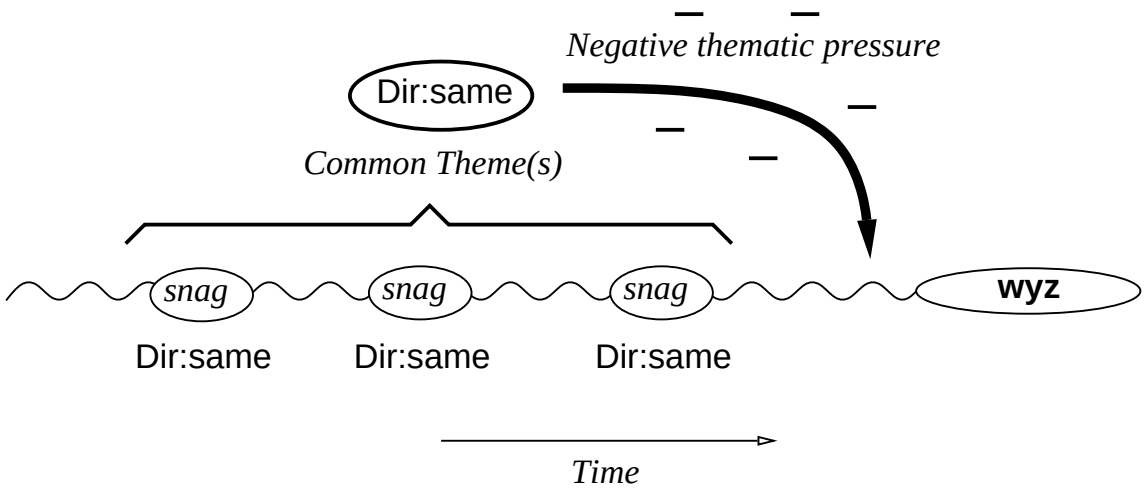

Figure 3: *Schematic diagram showing how themes common to several snag events can trigger negative thematic pressure, subsequently steering Metacat away from the problematic interpretation (i.e., seeing* **abc** *and* **xyz** *as going in the same direction), and eventually towards alternative interpretations.*

leading to failure, the themes comprising it can be clamped with strong negative activation, effectively steering the program away from the unproductive interpretation leading to the snag. In this way, Metacat can both recognize and subsequently break out of, its repetitive behavior. Figure 3 shows this idea schematically.

Metacat's thematic framework also allows it to effectively work backwards from a given answer to an understanding of why the answer makes sense, even if the program hasn't actually discovered the answer on its own. For example, when the program is given the answer **wyz** to the problem "**abc** ⇒ **abd**; **xyz** ⇒ **?**" and asked to justify it, it typically begins by building straightforward, left-to-right mappings between the strings, seeing all four of them as going in the same direction. Seeing **abc**–**abd** and **xyz**–**wyz** in this way may prompt it to create the "top" rule *Change letter-category of rightmost letter to successor* and the "bottom" rule *Change letter-category of leftmost letter to predecessor*. However, the same-direction mapping between **abc** and **xyz** is inconsistent with this way of looking at things, because the rightmost letter in **abc** does not map onto the leftmost letter in **xyz**. Comparing the two rules to each other, however, suggests the idea of *rightmost/leftmost* symmetry, as well as *successor/predecessor* symmetry. As a result, Metacat strongly activates vertical-mapping themes representing these ideas, causing the vertical mapping between **abc**

and **xyz** to be reorganized in a crosswise fashion. Seeing **abc** and **xyz** in this way yields an overall interpretation that makes sense.

5 Summary

Themes in Metacat can be viewed as a kind of intermediate level between "subcognitive" structure-building activity in the Workspace and higher-level "cognitive" activity associated with events in the Trace. They provide a medium through which ideas made explicit at the cognitive level (*i.e.*, the explicit thematic characterizations of events in the Trace) can actively influence and guide the course of subcognitive processing. In effect, themes are a "tool" used by the cognitive level to explore the ramifications of particular ideas. By strongly activating different patterns of themes and observing the results, the program can systematically try out different ideas as it solves analogy problems, some of which may lead it to answers, others of which may lead it to confusion. Furthermore, once an answer has been found, the themes associated with the answer represent a characterization of the key ideas and events that led to the answer's discovery. This characterization can subsequently be used as the basis for comparing and contrasting the answer with other previously-encountered answers.

To summarize, enriching Metacat's understanding of its answers by incorporating higher-order thematic information gleaned through self-watching enables it to perceive abstract similarities and differences among the analogies it makes. By applying the same processing mechanisms used to perceive relationships in its perceptual input to the more abstract task of perceiving relationships among the key ideas and events involved in the process of answer discovery, Metacat is able to recognize and possibly avoid certain types of behavior in its own processing, and to create rich representations of the answers it eventually finds. Metacat's thematic machinery provides a general framework in which to address the idea of self-watching, and represents 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.

6 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, 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. R. (1984). The Copycat Project: An experiment in nondeterminism and creative analogies. AI Memo 755, MIT Artificial Intelligence Laboratory.

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

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

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.