Metacat: A Program That Judges Creative Analogies in a Microworld

James B. Marshall[1]

Abstract. Metacat is a model of analogy-making and creativity that extends the earlier Copycat model by incorporating mechanisms that enable the program to compare and contrast idealized analogies in an insightful way, including analogies suggested to it by the user. This amounts to making analogies between analogies. This paper outlines the program's architecture, and presents several examples of the kinds of comparisons it is capable of making.

1 INTRODUCTION

The Copycat computer model of analogy-making was originally developed by Hofstadter and Mitchell as an investigation into the cognitive mechanisms underlying human creativity and perception [9, 8, 17]. Recent work on this project has focused on extending the model by incorporating mechanisms that enable the program to monitor its own behavior as it searches for answers to analogy problems [15]. This gives the program a much deeper level of insight into how it arrives at its answers, which in turn enables it to compare and contrast the analogies it makes on the basis of their similarities and differences—in other words, to make analogies between analogies. This paper describes the architecture of the program, called Metacat, and presents several examples of the kinds of comparisons between analogies it is capable of making.

Copycat and Metacat are part of a broader ongoing research program aimed at computationally modeling the psychological processes responsible for human creativity. This is, of course, a very ambitious goal, given the subtlety and complexity of human cognition. Over the years, several related projects have been developed by Hofstadter and his colleagues, all of which have involved building computer programs that operate in carefully-designed microworlds [9]. One such project, called Letter Spirit, uses an architecture similar to that of Copycat and Metacat, but applies it to the world of visual letter perception and design. Initial work on this project concentrated on the perception and categorization of *gridletters*, highly stylized letterforms of the lowercase roman alphabet drawn on a two-dimensional grid consisting of short line segments [16]. Designing a full set of gridletters from *a* to *z* in a single abstract, yet well-defined style is a challenging act of artistic creation. The current goal of the Letter Spirit project is to develop a program capable of perceiving the visual style common to an initial set of gridletters and then designing the rest of the alphabet in the same style, with the ultimate goal being for the program to create novel and interesting styles completely on its own. This very ambitious project is intended to model many of the most important aspects of creative artistic design.

[1] Computer Science Program, Pomona College, Claremont, California, USA, email: marshall@cs.pomona.edu

A key element of the Letter Spirit architecture is the "central feedback loop of creativity", in which the program not only *creates* new letterforms in a particular style, but also *judges the quality* of the letterforms it creates, in order to assess how well they actually reflect the desired style, possibly revising them as a result [16]. This continual cycle of creation, assessment, and revision is essential to the design process, and ought to play a key role in any faithful computer model of creativity. Recent work on Letter Spirit focuses on imbuing the program with this type of ability to step back and evaluate its own performance, and is closely related to the central issues of Metacat [18]. See [5] for a good overview of other work in AI on modeling creativity.

Copycat and Metacat operate in a *microdomain*—a tiny, idealized world explicitly designed to isolate the essential aspects of analogy-making and creativity by stripping away as many insignificant "real-world" details as possible. This act of idealization brings out the deep issues in stark relief, rendering them accessible to careful, controlled study. The raw material of the domain consists of 26 abstract objects, represented as lowercase letters for convenience, among which only three relations are meaningful: sameness, predecessorship, and successorship. All letters except ***a*** have an immediate predecessor, and all except ***z*** have an immediate successor. All other information pertaining to letters has been factored out, such as their shapes or semantic connotations. Analogies are stated in terms of short letter-strings, which can be thought of as idealized situations. For example, "***abc*** ⇒ ***abd***; ***mrrjjj*** ⇒ ***?***". Despite its apparent simplicity, this domain harbors an exceedingly rich variety of subtle analogy problems, in which many surprisingly creative and non-obvious answers are possible.

The Copycat architecture has been discussed at length elsewhere [9, 17], so details will be omitted here. Briefly, the program consists of a long-term memory of concepts about the letter-string world, called the *Slipnet*, together with a short-term memory for perceptual structures, called the *Workspace*. In the Workspace, small non-deterministic agents called *codelets* examine the letters of an analogy problem and build up structures around the letters representing a particular interpretation of the problem. Codelets look for sameness, successor, or predecessor relationships between letters, often chunking them together into *groups* based on a common relationship (for example, creating a "sameness group" from the three ***j***'s in ***mrrjjj***, or chunking the individual letters of ***abc*** into a "successor group"). The program's high-level behavior emerges in a bottom-up manner from the collective actions of many codelets working in parallel, in much the same way that an ant colony's high-level behavior emerges from the individual behaviors of the underlying ants, without any central executive directing the course of events.

Guiding the search for a mutually-consistent set of structures are

concepts in the Slipnet, which become activated to different degrees depending on the activity in the Workspace. This activation may spread to neighboring concepts, and strongly influences codelet decisions, resulting in top-down pressure that guides the program in its search for a good interpretation of a problem.

2 THE ARCHITECTURE OF METACAT

Since Metacat is an extension of the Copycat model, its architecture includes all of Copycat's main architectural components, such as the Workspace, the Slipnet, and the mechanisms for codelet processing. It also includes three new components: the *Episodic Memory*, the *Themespace*, and the *Temporal Trace*.

2.1 The Episodic Memory

Unlike Copycat, Metacat is able to remember its problem-solving experiences over time. When the program discovers a new answer, it pauses temporarily to display the answer along with the Workspace structures that gave rise to it, instead of simply quitting. Together, these structures represent a way of interpreting the analogy problem that yields the answer just found. This information is then packaged together into an *answer description* and stored in Metacat's Episodic Memory, after which the program continues searching for alternative answers to the problem. Gradually, over time, a series of answer descriptions accumulates in memory, each one containing much more information than just the answer string itself.

The most important information stored in an answer description consists of structures called *themes*, which represent the key ideas underlying the answer. The themes in an answer description provide a basis for comparing and contrasting the answer to other answers stored in memory. Furthermore, Metacat may be reminded of similar answers it has encountered in the past if the themes associated with a newly-discovered answer, acting as a memory retrieval cue, match those of some previously stored answer description sufficiently well. The pattern of themes in an answer description serves as an index under which the answer can be stored and retrieved from memory.

2.2 The Themespace

Themes are created in Metacat's *Themespace*, and consist of pairs of Slipnet concepts. For example, a theme representing the idea of alphabetic-position symmetry between ***a*** and ***z*** is composed of the Slipnet concepts *Alphabetic-Position* and *opposite*. In some ways, themes act like ordinary Workspace structures. They are not initially present in the Themespace; rather, they are created during the course of a run, in response to structure-building activity in the Workspace.

In other ways, however, themes act like Slipnet concepts. They can take on various levels of activation, depending on the extent to which the ideas they represent play a role in the program's current interpretation of the problem. A theme's activation may decay over time, and may be influenced by the activation levels of other themes. Like Slipnet concepts, themes can, under certain conditions, exert strong *top-down pressure* on perceptual activity occurring in the Workspace. In fact, themes can assume both positive and negative levels of activation. Positively-activated themes encourage the building of Workspace structures compatible with the themes, while negatively-activated themes encourage the creation of alternative structures.

2.3 The Temporal Trace

In addition to the Themespace and Episodic Memory, Metacat's architecture includes a separate short-term memory called the *Temporal Trace* (or just the *Trace* for short) that serves as the focus for self-monitoring. Like the Themespace, the Temporal Trace accumulates information over the course of a single run. The Trace stores an explicit temporal record of the most important *processing events* that occur during the course of solving an analogy problem. Examples of such events include recognizing a key idea pertaining to the problem (by noticing the strong activation of a theme or concept), hitting a snag (such as attempting to take the successor of ***z***, as in the problem *"**abc** ⇒ **abd**; **xyz** ⇒ **?**"*), or discovering a new answer. Of course, a large number of events of all "sizes" occur during the processing of almost any analogy problem, ranging from local "micro-events" such as individual codelet actions to global "macro-events" such as the discovery of new answers. However, only those events above a threshold level of importance get represented in the Trace. This allows Metacat to filter out all but the most significant events, giving the program a very selective, high-level view of what it is doing.

Once processing events have been explicitly represented in the Trace, they are themselves subject to examination by codelets. This allows Metacat to perceive patterns in its own behavior in much the same way that Copycat perceives patterns in its letter-strings—via codelets looking for relationships among perceptual structures. In Metacat's case, these perceptual structures include both Workspace structures and the "reified" processing events in the Trace. When a new answer is found, an answer description can be formed by examining the temporal record to see which events contributed to the answer's discovery.

This approach is similar in flavor to work on derivational analogy, in which the trace of a problem-solving session is stored in memory for future reference, together with a series of annotations describing the conditions under which each step in the solution was taken [3, 20]. In Metacat's case, however, the information in the Trace is used as the basis for constructing an abstract description of the answer found, rather than being permanently stored itself.

3 ANSWER JUSTIFICATION

Unlike Copycat, Metacat is capable of evaluating the relative strengths and weaknesses of analogies suggested to it by an outside agent, in addition to judging the analogies it makes on its own. In other words, Metacat can not only *discover* answers to analogy problems, it can also *justify* answers on their own terms, even if the program itself didn't come up with them. This amounts to "working backwards" from a given answer toward an insightful characterization of it, in order to understand why it makes sense. Once an answer has been understood in this way, it can be compared and contrasted with other answers that the program has either discovered previously itself, or been shown by someone else.

This type of "hindsight understanding" presents little difficulty for humans. People who are asked to solve the problem *"**abc** ⇒ **abd**; **mrrjjj** ⇒ **?**"*, for example, may not think of the answer ***mrrjjjj***, even when given an unlimited amount of time. However, as soon as this answer is suggested to them, they have no trouble seeing why it makes sense, even though they didn't think of it themselves. In a similar vein, suggesting the somewhat "tongue-in-cheek" answer ***abd*** usually elicits a few chuckles from people, indicating that they can see how it "makes sense", although few people give this answer on their own [17]. This is not to say that *every* suggested answer can be

readily understood in retrospect (for example, a person might never figure out the justification for an answer such as ***mssjjj***), but for many non-obvious answers, no additional explanation beyond just the answer itself is needed.

When Metacat runs in "justify mode", it takes a problem together with an answer supplied by the user and attempts to discover a way of interpreting the problem in which the given answer makes sense. To do so, it begins by building up perceptual structures among the letter-strings, as usual. This "bottom-up" approach, however, may lead it to build an inconsistent interpretation of the problem that does not support the answer in question. Nevertheless, examining different parts of this interpretation may suggest new ideas to try out. Metacat can explicitly focus on these ideas, represented as patterns of themes, by clamping the themes with strong positive activation. The resulting top-down pressure forces the program to reorganize its interpretation of the problem in accordance with these ideas, leading to a new—and perhaps more coherent—way of looking at things.

For example, when Metacat is asked to justify the answer ***wyz*** to the problem *"**abc** ⇒ **abd**; **xyz** ⇒ **?**"*, it typically begins by building straightforward mappings in which all the strings are seen as going in the same direction. In addition, it may create a "top rule" describing the ***abc*** ⇒ ***abd*** mapping as *Change letter-category of rightmost letter to successor* and a "bottom rule" describing the ***xyz*** ⇒ ***wyz*** mapping as *Change letter-category of leftmost letter to predecessor*. This state of affairs is shown in Figure 1.

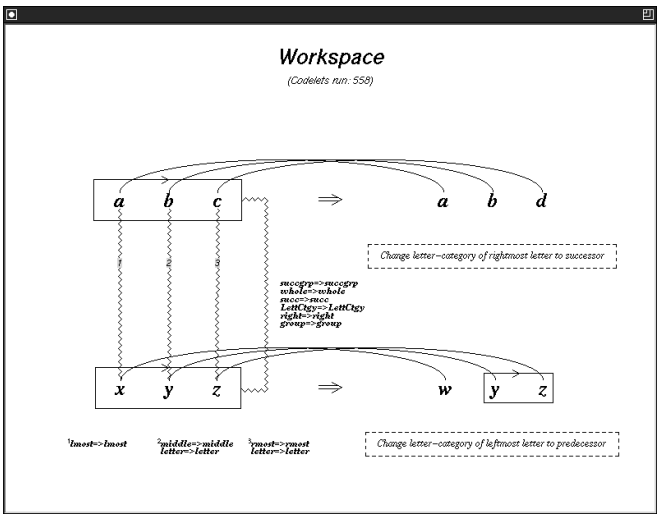

Figure 1. An inconsistent interpretation of the answer ***wyz***.

Although each of the three string mappings making up this interpretation is locally consistent when considered in isolation, together they do not make sense at a global level. The letters ***c*** and ***x*** are not seen as corresponding to each other, 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). Comparing the two *rules* to each other, however, suggests the idea of *rightmost–leftmost* symmetry, as well as *successor–predecessor* symmetry. This idea can be captured by a set of themes such as *String-Position: opposite*, *Direction: opposite*, and *Group-Type: opposite*. Metacat can explore the ramifications of this idea by clamping the associated themes at full activation in the Themespace. The resulting positive thematic pressure strongly promotes the creation of new structures compatible with the idea of mapping ***abc*** and ***xyz*** onto each other in a crosswise fashion, and significantly weakens

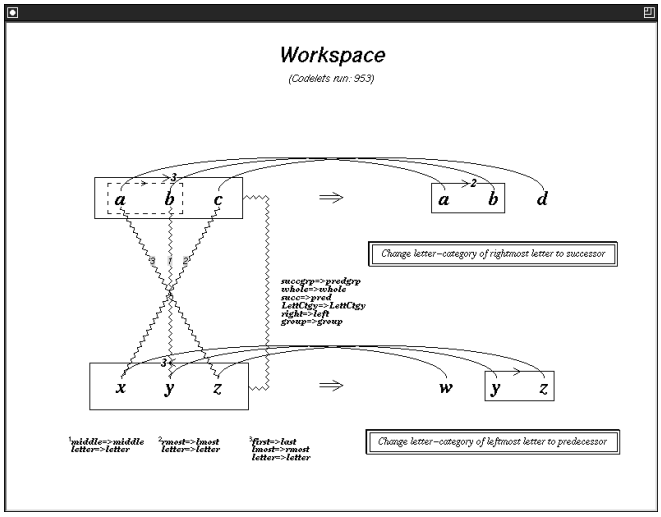

Figure 2. The final consistent interpretation of ***wyz***.

existing structures incompatible with this idea, such as the ***a–x*** and ***c–z*** bridges. The net effect is that the original mapping between ***abc*** and ***xyz*** shown in Figure 1 is swiftly reorganized by codelets into a new mapping consistent with the activated themes.

Figure 2 shows the final, globally consistent interpretation, in which ***c*** and ***x*** are seen as corresponding. In addition, the previously unnoticed alphabetic-position symmetry between the letters ***a*** and ***z*** has been identified as a result of the increased attention focused on these objects by top-down thematic pressure. Consequently, the final answer description for ***wyz*** includes an *Alphabetic-Position: opposite* theme.

4 ANSWER COMPARISON

As an example of how the similarities and differences between analogy problems can be understood in terms of themes, consider again the answer ***wyz*** just described for the problem *"**abc** ⇒ **abd**; **xyz** ⇒ **?**"*. This answer relies on an interpretation of the problem in which ***abc*** and ***xyz*** are seen as going in opposite directions, ***abc*** and ***abd*** are seen as going in the same direction, and ***abc*** ⇒ ***abd*** is described by the rule *Change letter-category of rightmost letter to successor*. At the crux of this interpretation lies the alphabetic-position symmetry of the letters ***a*** and ***z***, which provides the justification for perceiving ***abc*** and ***xyz*** as "mirror images" of each other. These ideas can be represented by a collection of structures that includes *Alphabetic-Position: opposite* and *String-Position: opposite* themes describing the relationship between ***abc*** and ***xyz***, a *String-Position: identity* theme describing the relationship between ***abc*** and ***abd***, and the aforementioned rule. Together, these structures constitute ***wyz***'s answer description in memory.

In contrast, Figure 3 shows Metacat's Workspace after it has found the answer ***xyd***. In this interpretation of the problem, ***abc*** and ***xyz*** are seen as going in the same direction, with letters in identical string positions corresponding to each other. The strings ***abc*** and ***abd*** map onto each other in a similar fashion, as shown by the horizontal bridges across the top, and the ***c*** in ***abc*** is seen as changing literally to ***d***, as indicated by the rule *Change letter-category of rightmost letter to 'd'*. These are the essential ingredients of the answer ***xyd***, and they can be explicitly represented by an answer description that includes *String-Position: identity* themes and the above rule. The

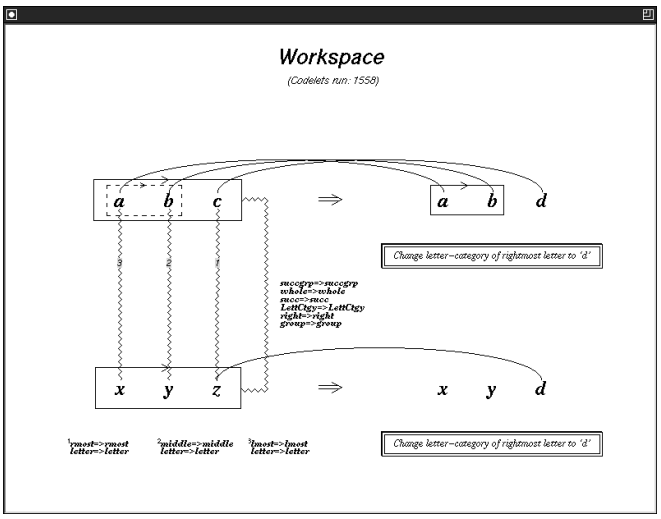

Figure 3. An interpretation yielding the answer ***xyd***.

idea of alphabetic-position symmetry does not arise, so there is no *Alphabetic-Position: opposite* theme involved.

Now consider the problem *"**rst*⇒*rsu; xyz*⇒*?"*, which is similar in many respects to the problem *"**abc*⇒*abd; xyz*⇒*?"*. In particular, the answers ***xyu*** and ***wyz*** are possible, based on many of the same considerations that arose in the earlier problem. The answer ***xyu*** depends in part on seeing ***rst*** and ***xyz*** as going in the *same* direction, while the answer ***wyz*** depends on seeing these strings as going in *opposite* directions. However, in this problem there is far less justification for seeing ***rst*** and ***xyz*** as mirror images of each other, 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 seeing ***abc*** and ***xyz*** (or ***rst*** and ***xyz***) as going in opposite directions, both involve seeing ***abc*** and ***abd*** (or ***rst*** and ***rsu***) as going in the same direction, and both involve viewing the ***abc*** ⇒ ***abd*** (or ***rst*** ⇒ ***rsu***) change abstractly rather than literally. The diminished justification for the answer ***wyz*** in this 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 previous problem. In short, ***xyd*** and ***xyu*** play essentially *identical* roles in their respective 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 farfetched, is certainly possible for the problem *"**abc*⇒*abd; xyz*⇒*?"*. Seeing this answer depends on noticing the abstract symmetry between ***abc*** and ***xyz***, and yet—somewhat ironically—taking a very literal-minded view of the way in which ***c*** changes to ***d***. In this case, doing "the same thing" to ***xyz*** involves changing its *leftmost* letter literally to ***d***. The answer ***uyz*** for the problem *"**rst*⇒*rsu; xyz*⇒*?"* arises in a similar manner, except that here there is 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 ***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 just 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 some of the information stored in their answer descriptions (for the sake of clarity, not all of the information is shown). These descriptions bring out clearly the similarities and differences among the various possible 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 the ***abc*** ⇒ ***abd*** change is perceived abstractly or literally (as indicated by the rule involved). The descriptions of ***xyd*** and ***xyu*** are identical, revealing the underlying similarity between these two literal-minded answers. The difference between the two ***wyz*** answers lies in the presence or absence of the idea of alphabetic-position symmetry. Furthermore, the *way* in which these answers differ is precisely the same as the way in which ***dyz*** differs from ***uyz***.

Problem/Answer	Themes	Rule
abc ⇒ ***abd***; ***xyz*** ⇒ ***wyz***	*Alphabetic-Position: opposite* *String-Position: opposite*	Abstract
rst ⇒ ***rsu***; ***xyz*** ⇒ ***wyz***	*String-Position: opposite*	Abstract
abc ⇒ ***abd***; ***xyz*** ⇒ ***xyd***	*String-Position: identity*	Literal
rst ⇒ ***rsu***; ***xyz*** ⇒ ***xyu***	*String-Position: identity*	Literal
abc ⇒ ***abd***; ***xyz*** ⇒ ***dyz***	*Alphabetic-Position: opposite* *String-Position: opposite*	Literal
rst ⇒ ***rsu***; ***xyz*** ⇒ ***uyz***	*String-Position: opposite*	Literal

Table 1. Six answers and their associated answer descriptions.

This example gives the flavor of how Metacat's answer descriptions enable it to compare and contrast idealized analogies in an insightful way, by analyzing the themes and other information stored in these descriptions. 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, Metacat's answers can be retrieved from memory on the basis of their stored descriptions. For example, suppose that Metacat finds the answer ***xyd*** to the problem *"**abc*⇒*abd; xyz*⇒*?"*. If it has previously encountered the answer ***xyu*** to the problem *"**rst*⇒*rsu; xyz*⇒*?"*, finding ***xyd*** may remind it of the ***xyu*** answer it has already seen—based on the strong similarity between the themes characterizing ***xyd*** and the stored description of ***xyu***—prompting Metacat to "comment" on the similarity between the two answers.

4.1 Program-generated commentary

As Metacat works on an analogy problem, it displays a running commentary in English of its ideas and observations about the problem and about its own "train of thought". This narrative, which appears in Metacat's *Comment Window*, is not an event-by-event transcription of the information in the Temporal Trace, although it corresponds closely to the chain of events recorded there. Rather, it simply consists of messages generated by codelets in a variety of circumstances as they go about their business. Essentially, this amounts to the program "thinking out loud" while it works on a problem. When Metacat hits a snag, for instance, it reports this fact and briefly explains why the snag has occurred. Upon discovering a new answer, it states its opinion of the answer's quality, and mentions any other answers that happen to "come to mind" as a result. The program also mentions when it is getting "frustrated" by a lack of progress. Furthermore, if it hits on some new idea to try, it gives a brief assessment of the progress achieved, in retrospect, as a result of focusing on the idea. The program can also comment on the similarities and differences between various answers, if asked to do so by the user.

Figure 4 illustrates the type of commentary typically generated by the program during a run. This example shows a run of the problem *"**abc*⇒*abd; xyz*⇒*?"* in which the program hits the ***z*** snag a couple

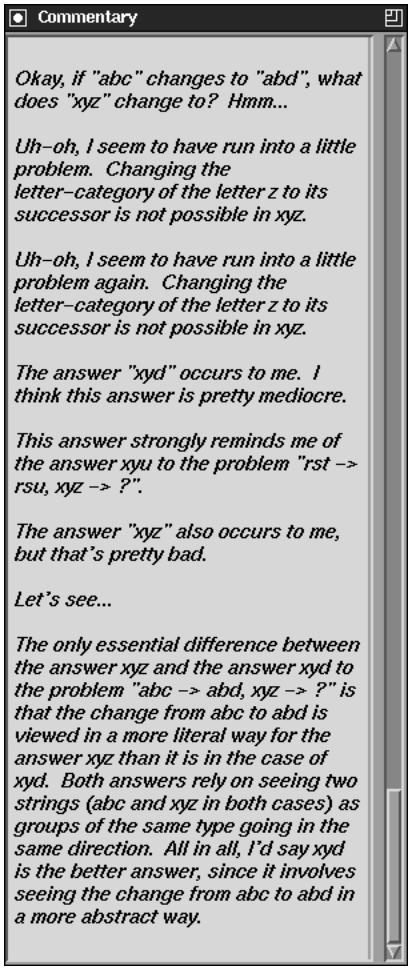

Figure 4. Metacat's commentary for a run of *"**abc** ⇒ **abd**; **xyz** ⇒ **?**"* in which it found the answers ***xyd*** and ***xyz***.

of times and then answers ***xyd***. As it happens, the answer ***xyd*** reminds the program of a similar answer to a different problem that it has already encountered. Continuing on, the program then finds the "do-nothing" answer ***xyz***, based on the rule *Change letter-category of letter 'c' to 'd'*. This rule is even more literal-minded than the rule *Change letter-category of rightmost letter to 'd'*. At this point, prompted by the user, the program compares the answer ***xyz*** to the answer ***xyd***, expressing a preference for the latter answer.

From this example, it may appear that Metacat possesses a sophisticated linguistic ability. However, it must be stressed that this is not the case. In fact, the program's ability to "speak" arises purely from a flexible set of prefabricated phrase-templates that get filled in and combined in complicated ways, according to context. For example, in the run shown in Figure 4, the explanation of the snag is generated on the basis of the concepts and Workspace structures involved in the snag—namely, the Slipnet concept *Letter-Category*, the letter ***z***, the Slipnet concept *successor*, and the Workspace string ***xyz***. As an added touch, the second time the program hits the snag, it inserts the word "again", on account of the fact that a previous snag event exists in the Temporal Trace. In addition, the program uses canned phrases to describe various numerical measures, such as the overall quality of an answer (*e.g.*, "pretty mediocre", "pretty bad") or the degree of reminding of one answer by another (*e.g.*, "strongly"). See [15] for a detailed discussion of Metacat's mechanisms for generating English commentary. Furthermore, no type of linguistic *interaction* with the program is possible. For instance, asking the program to compare two answers is accomplished simply by clicking on graphical icons associated with the answers.

Metacat's English-language veneer, although deceptive in a certain sense, is not intended to deceive. Rather, it is intended simply to show the various things that happen during the course of a run, in a somewhat whimsical but very user-friendly fashion. In the case of comparing two answers, it is intended to summarize the various parallels and distinctions between the answers that are perceived by the program, in an easy-to-understand way. Answers are compared on the basis of their underlying *conceptual representations*, which consist of the themes and Slipnet concepts stored in answer descriptions. Metacat's ability to compare answers at this representational level is what really counts, not its ability to generate English summaries of these comparisons.

That said, it is worth adding that not all of the words used by the program are completely devoid of semantic content. To be sure, many of them are (*e.g.*, "okay", "think", "mediocre", "I", "me", and so on). However, some of them, such as "letter", "letter-category", "groups", "successor", and "direction", reflect concepts that the program *does* understand—in a more genuine and quite defensible sense—about its letter-string world. These words correspond to Slipnet concepts, which become activated to different degrees according to the perceptual context at hand, and are thus *grounded* within the program's microworld. See [9, Chapter 6] for a fuller discussion of this point.

The following is a sampling of Metacat's explanations of the similarities and differences between some of the analogies in Table 1. To generate these explanations, the program was first run (in justify mode) on each of the answers, and then asked to compare them. The figures show the output generated by the program.

In Figure 5, the program compares the answers ***wyz*** and ***xyd*** to the problem *"**abc** ⇒ **abd**; **xyz** ⇒ **?**"*, and explains why it considers ***wyz*** to be the better analogy. The phrase "a richer set of ideas" refers to the fact that ***wyz***'s answer description contains more themes than ***xyd***'s description.

The answer wyz to the problem "abc -> abd, xyz -> ?" is based on seeing abc and xyz as symmetric predecessor and successor groups going in opposite directions, and on seeing alphabetic-position symmetry between the strings, while the answer xyd is based on seeing abc and xyz as groups of the same type going in the same direction. In xyd's case, the idea of seeing alphabetic-position symmetry between abc and xyz does not arise. Another key difference between the answers is that the change from abc to abd is viewed in a more abstract way for the answer wyz than it is in the case of xyd. All in all, I'd say wyz is the better answer, since it is based on a richer set of ideas.

Figure 5. ***abc*** ⇒ ***abd***; ***xyz*** ⇒ ***xyd*** versus ***abc*** ⇒ ***abd***; ***xyz*** ⇒ ***wyz***

The next three figures illustrate answer comparison between different problems, namely, *"**abc** ⇒ **abd**; **xyz** ⇒ **?**"* and *"**rst** ⇒ **rsu**; **xyz** ⇒ **?**"*. In Figure 6, the program explains why it considers the answers ***xyd*** and ***xyu*** to be fundamentally the same analogy. As the program notes, the rules giving rise to these answers are very similar, since they both involve changing the rightmost letter in a literal-minded way. The program assigns a rating of "pretty mediocre" to each answer, based on the low degree of abstractness of the answers' underlying themes and rules.

In Figure 7, the two ***wyz*** answers are compared. In this case, the program recognizes the essential difference between these analogies—namely, the presence of alphabetical symmetry in one but not the other—despite the superficial similarity of the two answers.

The answer xyd to the problem "abc –> abd, xyz –> ?" is essentially the same as the answer xyu to the problem "rst –> rsu, xyz –> ?". Both answers rely on seeing two strings (abc and xyz in one case and rst and xyz in the other) as groups of the same type going in the same direction. Furthermore, the change from abc to abd is viewed in essentially the same way as the change from rst to rsu. All in all, I'd say they're both pretty mediocre answers.

Figure 6. ***abc*** ⇒ ***abd***; ***xyz*** ⇒ ***xyd*** versus ***rst*** ⇒ ***rsu***; ***xyz*** ⇒ ***xyu***

The answer wyz to the problem "abc –> abd, xyz –> ?" is based in part on seeing alphabetic–position symmetry between abc and xyz. In contrast, in the case of the answer wyz to the problem "rst –> rsu, xyz –> ?", the idea of seeing alphabetic–position symmetry between rst and xyz does not arise. All in all, I'd say the first wyz is the better answer, since it is based on a richer set of ideas.

Figure 7. ***abc*** ⇒ ***abd***; ***xyz*** ⇒ ***wyz*** versus ***rst*** ⇒ ***rsu***; ***xyz*** ⇒ ***wyz***

The answer dyz to the problem "abc –> abd, xyz –> ?" is based in part on seeing alphabetic–position symmetry between abc and xyz. In contrast, in the case of the answer uyz to the problem "rst –> rsu, xyz –> ?", the idea of seeing alphabetic–position symmetry between rst and xyz does not arise. The answer dyz, however, seems incoherent to me, since it involves seeing abstract similarities between abc and xyz (seeing abc and xyz as symmetric predecessor and successor groups going in opposite directions, and seeing alphabetic–position symmetry between the strings), while at the same time viewing the change from abc to abd in a more literal way. The answer uyz also seems incoherent, since it involves seeing abstract similarities between rst and xyz (seeing rst and xyz as symmetric predecessor and successor groups going in opposite directions), while at the same time viewing the change from rst to rsu in a more literal way. Overall, though, I'd say uyz is the better answer, because it doesn't seem quite as incoherent as dyz.

Figure 8. ***abd*** ⇒ ***abd***; ***xyz*** ⇒ ***dyz*** versus ***rst*** ⇒ ***rsu***; ***xyz*** ⇒ ***uyz***

In Figure 8, the program compares the answers ***dyz*** and ***uyz***, each of which involves a somewhat incoherent blend of abstract and literal-minded perspectives. Just as in the earlier ***wyz***/***wyz*** case, the program identifies the presence or absence of alphabetical symmetry as the fundamental difference between these two analogies. It also notes their peculiar incoherence, expressing a preference for ***uyz***. The reason is that since ***abc*** and ***xyz*** are completely symmetric in every way, while ***rst*** and ***xyz*** are not, changing ***xyz*** to ***dyz*** seems an even sillier thing to do than changing ***xyz*** to ***uyz***, at least in the program's judgment. Thus it considers ***dyz*** to be even more incoherent than ***uyz***.

4.2 Reminding

Closely related to answer comparison is the phenomenon of reminding, in which one answer may trigger the spontaneous retrieval from memory of other answers that are in some way similar. This may happen whenever a new answer is discovered (or justified) by the program. When a new answer is found, the answer description created from the information in the Trace acts as an index into memory, causing other stored answer descriptions to become activated in proportion to their degree of similarity. Similarity between answer descriptions is determined by a numerical measure from 0 to 100 called the *distance*, which measures the amount of overlap of the answer descriptions' themes and concepts. If the activation level of an answer description exceeds some threshold, Metacat will be reminded of the answer, with the activation level corresponding to the strength of recall.

In addition to remembering its answers, Metacat also remembers

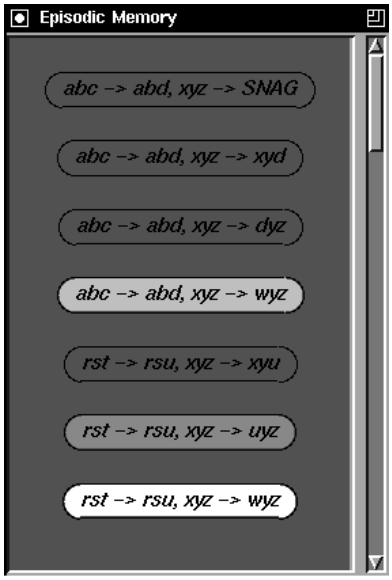

Figure 9. Six answer descriptions and one snag description stored in Metacat's Episodic Memory.

the snags that it encounters while solving problems on its own. On hitting a snag for the first time, the program creates an abstract *snag description* that characterizes the situation (in addition to creating a new snag event in the Temporal Trace), which it then stores in memory. Like answer descriptions, snag descriptions consist of themes and Workspace structures (*i.e.*, those directly responsible for causing the snag), and are used by the program in comparing and contrasting answers with one another.

Figure 9 shows an example of the state of Metacat's memory upon discovering the answer ***wyz*** to the problem "***rst*** ⇒ ***rsu***; ***xyz*** ⇒ ***?***", after having encountered several other answers to this problem and to the problem "***abc*** ⇒ ***abd***; ***xyz*** ⇒ ***?***". In addition, a snag description for the latter problem exists. The activation levels of answers are indicated by shades of grey, ranging from white for the most strongly-activated answers to dark grey for completely dormant answers—so that the less activated an answer is, the more it appears to fade into the background of Metacat's memory. In the present situation, the ***wyz*** answer just found is (not surprisingly) the most strongly activated. As can be seen, this answer partially activates the other ***wyz*** answer, and, to a lesser extent, ***uyz***. The other answers, however, are too distant from ***wyz*** to be recalled. As a result, the program reports in its Comment Window that the newly-found answer "somewhat" reminds it of the other ***wyz*** answer, and "vaguely" reminds it of ***uyz***. These terms are chosen on the basis of the activation levels of the answer descriptions.

Storing snag descriptions in memory enables Metacat to "appreciate" certain answers in ways that otherwise would not be possible. For example, consider the problem "***eqe*** ⇒ ***qeq***; ***abbbc*** ⇒ ***?***". In this problem, ***eqe*** gets turned "inside-out", an idea that can be captured by the rule *Swap letter-categories of all objects in string*. However, it is not so easy to do "the same thing" to ***abbbc***, since three different letter-categories are involved, instead of just two. One creative way out of this quandary is to reperceive ***abbbc*** as *1–3–1* and then swap the *group-lengths* rather than the letter-categories, yielding the answer ***aaabccc***. Unfortunately, Metacat is unable to get this answer on its own, because it is incapable of perceiving ***eqe*** or *1–3–1* as single, unified chunks, due to the absence of predecessor, successor,

or sameness relations between the adjacent parts. Consequently, it is unable to connect the idea of *letter* to the idea of *number* at a global level, and thus never sees these ideas as playing analogous roles in ***eqe*** and ***abbbc***. Instead, the program ends ends up repeatedly trying to swap the letter-categories of ***a***, ***bbb***, and ***c***, hitting a snag each time. On the other hand, if the answer ***aaabccc*** is provided to Metacat by the user, the program can make sense of it, although it still considers the connection between *letter* and *number* to be an "unjustified" idea.

The same is true for the answer ***aaabaaa*** to the related problem *"**eqe** ⇒ **qeq**; **abbba** ⇒ **?**"*. Metacat can (almost) make sense of it, but cannot get it on its own. However, there is a crucial difference between ***aaabaaa*** and ***aaabccc***. In the problem *"**eqe** ⇒ **qeq**; **abbba** ⇒ **?**"*, there is no good reason to view ***abbba*** as *1–3–1*, since swapping letter-categories is perfectly feasible. That is, no snag arises in this problem. In a sense, then, the answer ***aaabccc*** is the better analogy, since seeing ***abbbc*** as *1–3–1* provides an elegant and creative way around a snag, while seeing ***abbba*** as *1–3–1* is unnecessary. Metacat can make this observation, but it can only do so if it knows that the problem *"**eqe** ⇒ **qeq**; **abbbc** ⇒ **?**"* usually leads to a snag. If it has tried this problem on its own, it will know this, because the appropriate snag description will exist in memory. Conversely, if it is shown the answer ***aaabccc*** without having first attempted the problem itself, it will remain unaware of the possibility of a snag arising. In this way, snag descriptions enable the program to perceive subtle distinctions between certain analogies.

The following experiment illustrates this behavior. First, Metacat's memory was cleared in order to reset the program to a "tabula rasa" state. It was then shown the analogy ***eqe*** ⇒ ***eqeq***; ***abbba*** ⇒ ***aaabaaa*** and asked to justify it. At the end of the run, the program created an answer description for ***aaabaaa***, which it then stored in memory. Next, the program was shown the analogy ***eqe*** ⇒ ***eqeq***; ***abbbc*** ⇒ ***aaabccc***. At the end of the second run, the program reported that ***aaabccc*** strongly reminded it of the first answer, ***aaabaaa***. At this point, the program had not yet attempted to solve *"**eqe** ⇒ **qeq**; **abbbc** ⇒ **?**"* on its own, and therefore did not know that it normally gives rise to a snag. When asked to compare these two analogies, the program reported that it saw essentially no differences between them. Figure 10 shows the program's commentary.

The program was then reset to a tabula rasa state and given the problem *"**eqe** ⇒ **qeq**; **abbbc** ⇒ **?**"* to work on its own (*i.e.*, with no answer provided). After attempting unsuccessfully to swap the letters of ***abbbc*** a couple of times, the program settled on the more literal-minded answer ***qeeeq***. However, this resulted in the creation of a snag description for this problem in memory. Next, the program was shown the analogy ***eqe*** ⇒ ***eqeq***; ***abbba*** ⇒ ***aaabaaa***, as before, and asked to justify it, after which it was shown ***eqe*** ⇒ ***eqeq***; ***abbbc*** ⇒ ***aaabccc***. This time, the program reported that ***aaabccc*** reminded it only vaguely of ***aaabaaa***, indicating that it perceived the analogies as being quite different—although still recognizably related. The program's commentary is shown in Figure 11.

The answer aaabaaa to the problem "eqe –> qeq, abbba –> ?" is essentially the same as the answer aaabccc to the problem "eqe –> qeq, abbbc –> ?". Both answers rely on seeing two strings (eqe and abbba in one case and eqe and abbbc in the other) as going in the same direction, and on viewing one of the strings in terms of letters and the other in terms of numbers (although there is no good reason for doing so). Furthermore, the change from eqe to qeq is viewed in essentially the same way in both cases. All in all, I'd say aaabaaa is halfway decent and aaabccc is pretty good.

Figure 10. ***aaabaaa*** versus ***aaabccc*** before encountering the snag.

The answer aaabaaa to the problem "eqe –> qeq, abbba –> ?" is similar to the answer aaabccc to the problem "eqe –> qeq, abbbc –> ?", since both rely on seeing two strings (eqe and abbba in one case and eqe and abbbc in the other) as going in the same direction, and on viewing one of the strings in terms of letters and the other in terms of numbers. However, in the former case, there is no compelling reason to view one of the strings in terms of letters and the other in terms of numbers, unlike in the latter case with eqe and abbbc, where viewing one of the strings in terms of letters and the other in terms of numbers avoids a snag that would otherwise arise from the fact that no letter–category swap is possible between the letter a, the bbb group, and the letter c in abbbc. All in all, I'd say aaabccc is the better answer, since it involves no unjustified ideas.

Figure 11. ***aaabaaa*** versus ***aaabccc*** after encountering the snag.

5 DISCUSSION

A number of researchers have developed cognitive models that exhibit some of the flavor of Metacat and Copycat—particularly in their focus on emergent processing as a consequence of many nondeterministic micro-actions occurring in parallel, and the idea of spreading activation among nodes of a semantic network in response to context-dependent pressures. The DUAL cognitive architecture, developed by Kokinov and incorporated into the AMBR model of human reasoning by Kokinov and Petrov, is one such model [10, 11]. In addition, much work has been done on other issues that figure prominently in Metacat, such as analogy-making and reminding (see, for example, [6, 19, 7, 1]).

Of particular interest are case-based reasoning (CBR) approaches to modeling creativity and analogy [14, 12, 13]. Metacat touches on many of the fundamental issues underlying research in CBR. For instance, answer descriptions stored in Metacat's memory can be likened to cases in CBR, in the sense that they form a corpus of experience on which the program can draw when faced with new situations. When Metacat finds a new answer, its stored experiences may cause it to be reminded of similar answers it has seen in the past, in a way that is reminiscent of the retrieval of previously-stored cases from memory in CBR according to their degree of similarity to the current situation. The retrieved answer can then be compared to the current answer on the basis of the thematic information stored with it. This is roughly akin to comparing two cases in CBR in order to see how the cases are similar (*i.e.*, which aspects of the stored case can be applied directly, without modification, to the current situation), and how the cases differ (*i.e.*, which aspects must be adapted to fit the new situation).

However, there are important differences between CBR and Metacat. First of all, even though Metacat is concerned with solving analogy problems, it is not intended to model problem-solving *per se*. Rather, its focus is on modeling the way in which context-sensitive concepts allow analogies between different situations to be perceived in a natural and psychologically plausible manner. It is more concerned with analogical *perception* in general, than with analogical *reasoning* employed specifically as a tool for solving problems. Furthermore, the emphasis in much CBR work has often been on systems that learn to solve problems in a progressively faster and more efficient manner, whereas in Metacat the notion of learning to perceive analogies with ever increasing efficiency and speed is irrelevant. This point is less applicable, however, to some of the more recent CBR-based approaches to modeling creativity (see, for example, [2, 4]).

As was mentioned earlier, Metacat is actually closer to work on

derivational analogy than to ordinary case-based approaches that store only a final problem solution. In contrast to derivational analogy and CBR, however, Metacat (like Copycat) is deeply concerned with the nature and representation of *concepts*. One of the prime objectives of this research is to explore how adaptable, context-sensitive concepts can give rise to understanding by enabling analogies between apparently dissimilar situations to be perceived. Metacat's concepts, to be sure, come nowhere close to exhibiting the full power and fluidity of human concepts. Nevertheless, there is a sense in which they *are* genuinely meaningful entities—not just empty static symbols that get shunted around by the program. A concept or theme—take *successor*, for example—responds to the situation at hand in a continuous, context-dependent way, reflecting the degree of perceived relevance of the idea of successorship in the Workspace at any given moment. A wide range of superficially dissimilar situations, represented abstractly as letter-strings, can in principle activate it—strings such as ***abc***, ***ijk***, ***pqrst***, ***iijjkk***, ***mrrjjj***, ***xxsssbbbb***, ***axypqr***, and ***aababcabcd***. Under the right circumstances, all of these strings can be interpreted by the program as examples of successor-groups. Given the program's ability to flexibly recognize a wide range of instances of the same concept, some of them quite abstract, it seems fair to say that Metacat's concepts have at least some small degree of meaningfulness, or genuine semantics, within the confines of its tiny, idealized world.

Work on Metacat is aimed at deepening Copycat's understanding of its answers by incorporating mechanisms for self-monitoring, memory, and reminding into the program. Many important ideas from case-based reasoning are relevant to this aim, such as the storing of past experiences as a repertoire of cases in memory, and the recall of stored cases by new, similar situations. Unfortunately, case-based reasoning research concentrates on these issues at the expense of understanding the nature of concepts and how they interact with perception. We worry that CBR's ultimate success—at least as a cognitive model—will be limited on account of its avoidance of this very difficult but critically important question. In contrast, Metacat can be seen as an attempt to broaden and enrich these ideas by focusing on them within a framework of fluid, context-sensitive concepts that are grounded in the program's microworld.

6 CONCLUSION

The examples presented in this paper convey the flavor of Metacat's ability to "talk" about its answers in various ways. Clearly, much is going on beneath the surface here. The program's ability to recall previously-encountered answers, and to explain the similarities and differences between answers in a plausible fashion, relies on storing abstract representations of answers in long-term memory. Like Copycat's Slipnet, Metacat's Slipnet serves as the program's ultimate repository for its knowledge of concepts about the letter-string microworld. These concepts acquire their semantics solely through the ways in which they become activated in response to situations arising in this world. Accordingly, they represent the substrate on which the program's understanding of its answers is based. Therefore, it is important to emphasize the fact that answer descriptions are ultimately just organized patterns of Slipnet concepts, since they are composed of themes and rules (which in turn are composed of concepts). The English-language commentary generated by the program about analogies, although just a surface-level "gloss" in many ways, nevertheless rests on a deeper foundation of conceptual representation.

ACKNOWLEDGEMENTS

The author would like to thank the anonymous reviewers for their comments and suggestions.

REFERENCES

[1] *Advances in Connectionist and Neural Computation Theory, Volume 3: Analogy, Metaphor, and Reminding*, eds., John A. Barnden and Keith J. Holyoak, Ablex, Norwood, NJ, 1994.

[2] Carlos Bento and Amílcar Cardoso, eds. *Proceedings of the Workshop on Creative Systems: Approaches to Creativity in AI and Cognitive Science*, ICCBR 2001, Vancouver, Canada, 2001.

[3] Jaime Carbonell, 'Derivational analogy: A theory of reconstructive problem solving and expertise acquisition', in *Machine Learning: An Artificial Intelligence Approach, Volume 2*, eds., Ryszard Michalski, Jaime Carbonell, and Thomas Mitchell, 371–392, Morgan Kaufmann, San Francisco, (1986).

[4] Amílcar Cardoso and Geraint Wiggins, eds. *Proceedings of the AISB'02 Symposium on AI and Creativity in Arts and Science*, London, England, 2002.

[5] *Artificial Intelligence and Creativity: An Interdisciplinary Approach (Studies in Cognitive Systems, vol. 17)*, ed., Terry Dartnall, Kluwer Academic Publishers, 1994.

[6] Brian Falkenhainer, Kenneth D. Forbus, and Dedre Gentner, 'The structure-mapping engine', *Artificial Intelligence*, **41**(1), 1–63, (1990).

[7] Kenneth D. Forbus, Dedre Gentner, and Keith Law, 'MAC/FAC: A model of similarity-based retrieval', *Cognitive Science*, **19**, 141–205, (1995).

[8] Douglas R. Hofstadter, 'How could a copycat ever be creative?', in *Artificial Intelligence and Creativity: An Interdisciplinary Approach (Studies in Cognitive Systems, vol. 17)*, ed., Terry Dartnall, 405–424, Kluwer Academic Publishers, (1994).

[9] Douglas R. Hofstadter and FARG, *Fluid Concepts and Creative Analogies: Computer Models of the Fundamental Mechanisms of Thought*, Basic Books, New York, 1995. Co-authored with members of the Fluid Analogies Research Group.

[10] Boicho N. Kokinov, 'The context-sensitive cognitive architecture DUAL', in *Proceedings of the Sixteenth Annual Conference of the Cognitive Science Society*. Lawrence Erlbaum Associates, (1994).

[11] Boicho N. Kokinov and Alexander Petrov, 'Integrating memory and reasoning in analogy-making: The AMBR model', in *The Analogical Mind: Perspectives from Cognitive Science*, eds., Dedre Gentner, Keith Holyoak, and Boicho Kokinov, 59–124, MIT Press, Cambridge, MA, (2000).

[12] Janet Kolodner, *Case-Based Reasoning*, Morgan Kaufmann, San Francisco, 1993.

[13] Janet Kolodner, 'Understanding creativity: A case-based approach', in *Topics in Case-Based Reasoning, selected papers from the First European Workshop on Case-Based Reasoning*, eds., M. M. Richter, S. Wess, K. D. Althoff, and T. Maurer, Springer-Verlag, (1994).

[14] *Case-Based Reasoning: Experiences, Lessons, & Future Directions*, ed., David B. Leake, MIT Press/AAAI Press, Cambridge, MA, 1996.

[15] James B. Marshall, *Metacat: A Self-Watching Cognitive Architecture for Analogy-Making and High-Level Perception*, Ph.D. dissertation, Indiana University, Bloomington, IN, 1999. URL: http://www.cs.pomona.edu/marshall/metacat.pdf.

[16] Gary E. McGraw, Jr., *Letter Spirit (Part One): Emergent High-Level Perception of Letters Using Fluid Concepts*, Ph.D. dissertation, Indiana University, Bloomington, IN, 1995.

[17] Melanie Mitchell, *Analogy-making as Perception*, MIT Press/Bradford Books, Cambridge, MA, 1993.

[18] John A. Rehling, *Letter Spirit (Part Two): Modeling Creativity in a Visual Domain*, Ph.D. dissertation, Indiana University, Bloomington, 2001.

[19] P. Thagard, K. Holyoak, G. Nelson, and D. Gochfield, 'Analog retrieval by constraint satisfaction', *Artificial Intelligence*, **46**(3), 259–310, (1990).

[20] M. M. Veloso and J. G. Carbonell, 'Derivational analogy in PRODIGY: Automating case acquisition, storage and utilisation', *Machine Learning*, **10**, 249–278, (1993).