A CASE-BASED MODEL OF CREATIVITY

SCOTT R. TURNER
Department of Computer Science
University of California, Los Angeles
Los Angeles CA 90024 USA

Abstract

Creativity – creating new solutions to problems – is an integral part of the problem-solving process. This paper presents a cognitive model of creativity in which a case-based problem-solver is augmented with (1) a creative drive and (2) a set of creativity heuristics. New solutions are discovered by solving a slightly different problem and adapting that solution to the original problem. By repeating this process, a creative problem-solver can discover new solutions that are novel, useful and very different from known solutions. This model has been implemented in a computer program called MINSTREL. MINSTREL has been used for planning and problem-solving, to tell stories, and to invent mechanical devices.

1 Introduction

Creativity is an important element of human cognition. We all invent on a daily basis: we fix cars using spare change and bailing wire, invent jokes based on the latest domestic crisis, and make up bed-time stories for our children. The ability to invent original, useful solutions to problems is a fundamental process of human thought. To understand human cognition, we must understand the processes of creativity: the goals that drive people to create and the mechanisms they use to invent novel and useful solutions to their problems.

This paper presents a model of creative reasoning as an extension to case-based problem-solving. When problem-solving fails, the creative reasoner invents new solutions by combining old knowledge in new ways. This model has been implemented in a computer program called MINSTREL, which creates stories about King Arthur and his Knights of the Round Table. The creative reasoning portion of MINSTREL has also been applied to tasks such as mechanical invention.

2 Case-Based Problem-Solving

In case-based reasoning problems are solved by recalling similar past problem situations and applying the solutions from those situations to the current problem ([Slade 1992]). The problem-solver *recalls* a past problem similar to the current problem, *adapts* the solution from the old problem to the current problem, and then *assesses* the results.

Because it solves problems by looking for similar past problems, a case-based problem-solver cannot solve a problem which is not very similar to a previous problem. When a new problem is encountered, the problem-solver has no way to invent or discover a new solution, and problem-solving fails. So although case-based problem-solving is efficient at solving familiar problems, it cannot solve new problems.

3 Creativity

What needs to be added to case-based problem-solving to make it capable of creativity? Consider this example of creativity concerning the seven-year old niece of the author:

> One day, while visiting her grandparents, Janelle was seated alone at the dining room table, having milk and cookies. Reaching for the cookies, she accidently spilled her milk on the table. She decided to clean up the mess herself.
>
> Janelle went into the kitchen, but there were no towels or paper towels available. She stood for a moment in the center of the kitchen thinking, and then she went out the back door.
>
> Janelle returned a few minutes later carrying a kitten. The neighbor's cat had given birth to a litter about a month ago, and she had been over to play with the kittens the previous day. Janelle brought the kitten into the dining room, where he happily lapped up the spilled milk.

This story illustrates three important features of the creative process:

(1) *Creativity is driven by the failure of problem-solving* [Weisberg 1986]. Janelle could not use the plan she knew for cleaning up a spill because no towel was

available. In this case, problem-solving failed because a plan precondition could not be achieved. But problem-solving can fail for a number of reasons. The problem-solver may not be able to recall a similar past situation, or a solution may be rejected by a domain assessment, as when an engineer rejects a solution as too dangerous.

(2) *Creative solutions are both original and useful* [Wallas, 1926; Koestler, 1964]. Janelle's solution is original because it has significant differences from all other solutions Janelle knows, and useful because it solves her problem.

(3) *Old knowledge is used to create new solutions* [Koestler, 1964; Weisberg 1986]. Janelle knew that kittens like milk and that she could find a kitten next door. The creative problem-solver must be able to discover appropriate old knowledge and apply it in new ways to the current problem.

To be creative, the case-based problem-solver must be extended to find and combine useful old knowledge to create new solutions when problem-solving fails.

4 Discovering New Solutions

As we have seen, problem-solving finds solutions by recalling past problems similar to the current problem. To discover new solutions, the problem-solver must (1) recall a past problem that is *different* from the current problem and (2) adapt the solutions from the recalled problem to the current problem.

This would appear to be an impossible task. On the one hand, if Janelle grabs randomly at old knowledge to create a new solution, she has an impossibly hard adaptation task. Suppose that Janelle had recalled knowledge about how plants grow, the rules of ping pong, and the route the school bus followed. Surely she would fail in adapting this knowledge to the "spilled milk" problem. On the other hand, if Janelle only uses knowledge that she knows how to apply to the current problem, she is unlikely to invent anything new and useful. Janelle's knowledge about towels and cleaning is easy to apply, but won't result in a new solution. The challenge of creativity is to resolve this paradox.

MINSTREL's solution to this Catch-22 is to *integrate* the search and adapt processes of creativity. MINSTREL's creativity heuristics are called Transform-Recall-Adapt Methods (TRAMs). Each TRAM has three parts. "Transform" takes a problem and changes it into a slightly different problem. "Recall" takes the new problem description and tries to recall similar past problems from episodic memory. "Adapt" takes the recalled problem solutions and adapts them to the original problem. And because a specific adaptation (Adapt) is bundled with each search method (Transform) the problem of adapting discovered knowledge is eliminated.

TRAM:Cross-Domain-Reminding is an example of a Transform-Recall-Adapt Method. TRAM:Cross-Domain-Reminding suggests that a new solution to a problem can be found by translating the problem into a new domain, solving the problem in that domain, and then translating the solution back into the original domain. TRAM:Cross-Domain-Reminding is illustrated in Figure 4.1.

TRAM:Cross-Domain-Reminding

Comment:	Try solving the problem in a different problem domain.
Test:	Is this an action?
Transform:	1. Find a domain which has mappings for all the components of this problem description. 2. Translate the problem description into the new domain using the mapping.
Adapt:	Apply the reverse mapping on the solution to translate it from the new domain to the original problem domain.

Figure 4.1 TRAM:Cross-Domain-Reminding

MINSTREL uses TRAM:Cross-Domain-Reminding to create the following story scene:

> One day while out riding, Lancelot's horse went into the woods. Lancelot could not control the horse. The horse took him deeper into the woods. The horse stopped. Lancelot saw Andrea, a Lady of the Court, who was out picking berries.

The original specification for this problem is "create a scene in which a knight accidently meets a princess." TRAM:Cross-Domain-Reminding transforms this specification by mapping it into the modern domain and recalls this story:

Walking The Dog

> John was sitting at home one evening when his dog began acting strange. The dog was scratching at the door and whining for a walk. Finally, John decided to take the dog for a walk. While they were out, John ran across his old friend Pete, whom he hadn't seen in many years. John realized that he would never have run into Pete if his dog hadn't acted so strangely.

This story is adapted back to the original problem by mapping the story back into the King Arthur domain, creating a scene in which Lancelot's horse leads him to

Andrea. The resulting scene is creative – novel and useful – because TRAM:Cross-Domain-Reminding enabled the problem-solver to discover a solution different from the known solutions.

TRAMs are added to case-based problem-solving by augmenting the problem-solving process with a pool of TRAMs. When standard problem-solving fails, a TRAM is selected[1] from this pool and applied to the current problem. If the TRAM succeeds in discovering a solution to the problem, problem-solving succeeds. If problem-solving fails, the current TRAM is discarded, another selected from the pool of available TRAMs, and the cycle repeated.

Standard case-based problem-solving itself can be implemented as a TRAM. TRAM:Problem-Solving is a special TRAM that is always selected first from the TRAM pool. TRAM:Problem-Solving passes the original problem unchanged to the recall step and does no adaptation of any recalled solutions. This implements standard problem-solving and (since TRAM:Problem-Solving is always tried first) assures that the problem-solver will use an old, known solution if it is available and acceptable to the solution assessments.

There are two major benefits to Transform-Recall-Adapt Methods:

First, TRAMs discover new solutions by looking at *slightly* different problems. A slightly different problem shares many of the same constraints as the original problem, and its solution is likely to have some applicability to the original problem. Looking at slightly different solutions also simplifies the adaptation process, since fewer changes will be needed to adapt the solution to the original problem.

Second, TRAMs eliminate the adaptation problem by bundling appropriate adaptations with each search method. Knowing the transformation that led to a solution, the problem-solver can apply a specific adapation to reverse the effects of the problem transformation.

There is, however, one drawback to this model of creativity. TRAMs invent new solutions by adapting *slightly* different knowledge. But how can a creative problem-solving ever invent a solution with large differences from past solutions? To answer that question, we must look at the role of memory in problem-solving.

1. Various strategies for selection (random or based on experience) are discussed in [Turner 1992].

5 Imaginative Memory

The central step of the Transform-Recall-Adapt Method is recalling a solution from episodic memory. But the process of recalling something from episodic memory can *itself* be viewed as a problem. What happens if creative problem-solving is used to implement recall?

To do this, the Recall step of each TRAM is modified to recursively call the creative problem-solving process. To prevent this from leading to endless recursion, TRAM:Problem-Solving is left unchanged, and continues to recall directly from episodic memory.

To recall something, the problem-solver uses creative problem-solving with the problem specification "Find something in episodic memory that matches these features." TRAM:Problem-Solving is the first TRAM used, and passes the recall features unchanged to episodic memory. If an episode that matches the recall features is found, problem-solving succeeds. Because TRAM:Problem-Solving is always the first TRAM used and continues to use episodic memory normally, recall behaves as expected when an episode exists that matches the recall features.

Something more interesting happens when recall fails. If TRAM:Problem-Solving cannot find an episode in memory that matches the recall features, problem-solving fails and a new TRAM is selected. This TRAM modifies the recall features and recursively calls the problem-solving process with the new recall features.

The first TRAM used on this recursive call is again TRAM:Problem-Solving. If the new features recall an episode, the episode is returned to the previous recursion of problem-solving, where it is adapted to the original problem by the previous TRAM, and recall succeeds. *But because the recalled episode was changed by the Adapt portion of the previous TRAM, it is no longer the episode that was found in memory.*

Recall has succeeded in a strange way: by recalling an episode that does not exist in episodic memory. Episodic memory has become imaginative. When an appropriate episode exists, it is recalled. When no appropriate episode exists, recall uses creativity heuristics to "imagine" an appropriate episode.

Imaginative memory is that it makes creativity transparently available to any cognitive process that uses memory. Imaginative memory also permits the problem-solver to apply multiple creativity heuristics to a problem. Each time recall fails, imaginative memory recurse and another TRAM is be applied to the recall features. Each TRAM changes the problem in only a small way, but the cumulative effect may be large, enabling the creative problem-solver to discover new solutions significantly different from known solutions.

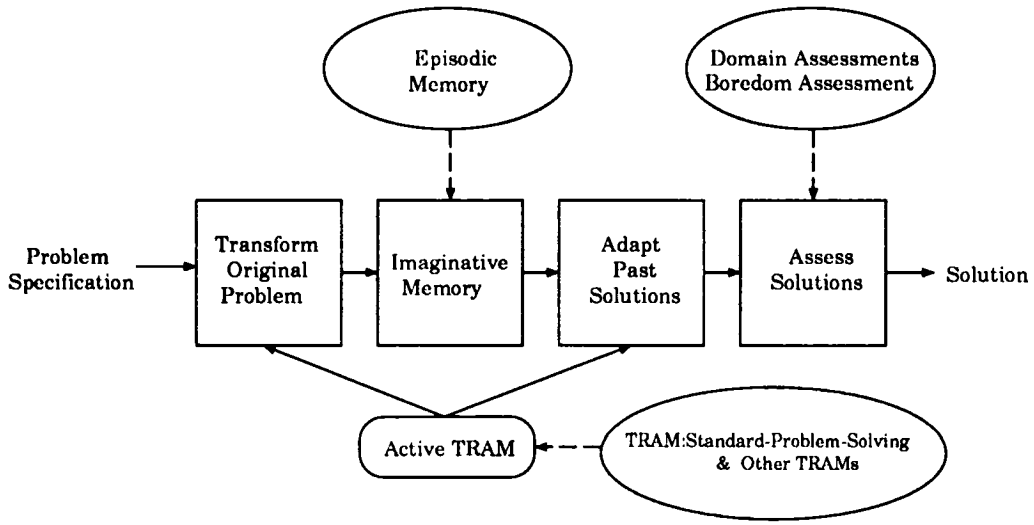

Figure 6.1 MINSTREL's Process Model of Creativity

6 A Model of Creativity

Figure 6.1 illustrates how Transform-Recall-Adapt Methods and imaginative memory are integrated into case-based problem-solving. The three central boxes represent the basic problem-solving processes of recalling a solution, adapting it to the current problem, and assessing the result. An active TRAM controls the recall and adapt steps. Initially this is TRAM:Problem-Solving, which is simply the strategy of recalling a similar past problem and using the solution from that problem. The assess step applies a pool of assessments to proposed solutions. In creative domains, this includes the boredom assessment, which rejects solutions that are too similar to past solutions. The recall step uses imaginative memory (a recursive call to problem-solving) instead of episodic memory.

The problem-solving cycle begins when a problem description enters the recall step. Initially TRAM:Problem-Solving is in control of the recall and adapt steps. The original problem description is used to recall previous similar problem-solving situations from episodic memory. If recall succeeds, the recalled situations are passed to the adapt step. Under TRAM:Problem-Solving, no adaptation is needed because the recalled solutions are similar to the original problem, so the recalled solutions are passed along to the assess step. In the assess step, all active assessments are applied to the recalled solutions, and if a solution passes all the assessments, it is output as a solution to the original problem. This is simply the normal problem-solving cycle.

If TRAM:Problem-Solving fails, either because no solutions were recalled or because the recalled solutions failed assessment, TRAM:Problem-Solving is discarded and a new TRAM selected as the active TRAM. The selection of a TRAM is based on the type of problem being solved and the TRAMs previously used.

The selected TRAM transforms the original problem specification, creating a new problem specification. The new problem specification is passed to imaginative memory. If recall succeeds, the recalled solutions are passed to the adapt step, where the active TRAM applies a specific adaptation which reverses the problem transformation. If recall fails, then imaginative memory recursively applies a second TRAM, and the problem-solving repeats at the inferior level.

Adapted solutions are passed to the assessment step, where they are assessed by domain assessments. If a solution passes all assessments, problem-solving succeeds. If all solutions fail, the active TRAM is discarded, a new TRAM selected, and the Recall-Adapt-Assess cycle repeats.

7 Creativity in MINSTREL

The primary creative task in storytelling is scene creation – inventing story events to achieve a particular storytelling goal. This section illustrates how MINSTREL uses the TRAM model to invent three scenes in which "a knight kills himself." At the beginning of this example, MINSTREL knows nothing about suicide. Using creative problem-solving, MINSTREL invents three novel scenes in which a knight kills himself: by purposely losing a fight with a dragon, by drinking poison, and by hitting himself with his sword.

MINSTREL's episodic memory intially contains only two story scenes:

Knight Fight

A knight fights a troll with his sword, killing the troll and accidently injuring himself.

The Princess and the Potion

A lady of the court drank a potion to make herself ill.

Using these scenes and three TRAMs (TRAM:Generalize-Constraint, TRAM:Similar-Outcomes and TRAM:Intention-Switch) MINSTREL invents three ways for a knight to commit suicide. Figure 7.1 shows an abbreviated trace of MINSTREL creating these three scenes (and generating English descriptions of the scenes).

```
==============================================
          MINSTREL Invention
==============================================
Initial specification is #{ACT.105}:
 (A KNIGHT NAMED JOHN DID SOMETHING *PERIOD*
  JOHN DIED *PERIOD*)

RAS Cycle: #{ACT.105}.
 Executing TRAM:GENERALIZE-CONSTRAINT.
 Generalizing :OBJECT on #{STATE.112}.
 Recalling #{ACT.113}: #{KNIGHT-FIGHT}.
[...]
 [TRAM Recursion: #{ACT.111}.]
    Executing TRAM:SIMILAR-OUTCOMES
    Recalling #{ACT.136}:       NIL.
    [TRAM Recursion: #{ACT.136}.]
       Executing TRAM:GENERALIZE-CONSTRAINT
       Recalling #{ACT.138}: #{PRINCESS-POTION}.
[...]
 [TRAM Recursion: #{ACT.112}.]
    Executing TRAM:INTENTION-SWITCH
    Recalling #{ACT.126}:       NIL.
    [TRAM Recursion: #{ACT.126}.]
       Executing TRAM:SIMILAR-OUTCOMES
       Recalling #{ACT.128}: #{KNIGHT-FIGHT}.
[...]
==============================================
          Invention Results
==============================================
Minstrel invented this story:
 (A KNIGHT NAMED JOHN FOUGHT HIMSELF BY MOVING
  HIS SWORD TO HIMSELF IN ORDER TO KILL HIMSELF
  *PERIOD* JOHN DIED *PERIOD*)

Minstrel invented this story:
 (A KNIGHT NAMED JOHN DRANK THE POTION IN ORDER
  TO KILL HIMSELF *PERIOD* JOHN DIED *PERIOD*)

Minstrel invented this story:
 (A KNIGHT NAMED JOHN FOUGHT A DRAGON BY MOVING
  HIS SWORD TO IT IN ORDER TO KILL HIMSELF
  *PERIOD* JOHN DIED *PERIOD*)
```

Figure 7.1 Suicide Trace

7.1 TRAM:Generalize-Constraint

TRAM:Generalize-Constraint suggests that a new solution to a problem can be found by removing a solution constraint, solving the new problem, and then adding the constraint back to the new solution. For scene creation, the problem constraints are the features of the specification. TRAM:Generalize-Constraint is shown in Figure 7.2.

TRAM:Generalize-Constraint begins by using a broad generalization on each constraint. TRAM:Generalize-Constraint removes each constraint in turn and uses memory to see whether there exists any episode that matches the new problem specification. If an episode is recalled, MINSTREL marks the constraint as a feasible selection for generalization. The second step is to randomly select and generalize one of the feasible constraints using a class hierarchy. The first step finds possible generalizations; the second step creates a reasonable generalization.

In the suicide example, the possible constraints are: (1) the actor is a knight, (2) the result of the act is death, and (3) the dead character is the actor. Of these constraints, only removing (3) leads to a recollection, so TRAM:Generalize-Constraint selects (3) for generalization. This constraint is generalized using a class hierarchy of character roles. The actor is a knight, which is a type of "Violent Character", so TRAM:Generalize-Constraint generalizes this constraint to "the dead character is a violent character". The new, transformed problem is "A knight does something to kill a violent character." Note that after this transformation, the new problem situation no longer requires the knight to kill himself, only that he kills some violent character.

TRAM:Generalize-Constraint

Comment:	Remove a constraint; solve problem; replace constraint.
Test:	Is this an action?
Transform:	1. For each constraint (slot), make a broad generalization and use memory to determine if the the generalization is feasible. 2. Randomly select a feasible constraint, and use a a class hierarchy to find a semantically close generalization.
Adapt:	Replace the generalized constraint with the original value.

Figure 7.2 TRAM:Generalize-Constraint

This recalls the "Knight Fight" episode, in which a knight kills a troll by hitting it with a sword. This episode is then adapted back to the original problem by replacing the troll (the violent character) with the original constraint (knight), creating the scene "A knight kills himself by fighting himself with his sword." TRAM:Generalize-Constraint has used previous knowledge about how knights kill monsters to create a scene in which a knight kills himself.

The use of a class hierarchy prevents MINSTREL from making errors caused by using a too broad generalization. For example, suppose that MINSTREL is creating a scene in which "a knight gives a princess something that makes her happy" and chooses to generalize the "princess" feature by removal. This recalls a scene in which a knight makes a troll happy by giving him a hunk of raw meat. When this reminding is adapted for the original scene, the result has the knight pleasing the princess by giving her a hunk of raw meat!

This type of mistake occurs because there is little similarity between the original feature ("princess") and the instantiation of its generalization ("troll"). A better generalization would ensure some similarity between the original feature and the instantiations of the generalization. Since class heirarchies group concepts with similar features, it is very effective for this type of generalization.

7.2 TRAM:Similar-Outcomes

TRAM:Similar-Outcomes suggests that if an action results in a particular outcome, it might also result in similar outcomes. If riding a horse can carry a knight to the castle, then riding a horse might also carry a knight to the woods.

In the suicide example, TRAM:Similar-Outcomes recognizes that being injured is similar to being killed, and transforms the scene description to "a knight purposely injures himself." If MINSTREL can recall a scene which fits this description, it can be adapted to the current problem by guessing that an action which results in injury might also result in death. TRAM:Similar-Outcomes is shown in Figure 7.3.

TRAM:Similar-Outcomes

Comment:	Change a partial state change to a complete, or vice versa.
Test:	Is the problem a scaled state change?
Transform:	If the state change is partial in one direction along the state scale, change it to be complete in that direction. If it is a complete change in one direction, change it to be partial in that direction.
Adapt:	Change the outcome on the recalled solution to the original outcome.

Figure 7.3 TRAM:Similar-Outcomes

However, the description "a knight purposely injures himself" does not recall either of the episodes in MINSTREL's episodic memory. In "Knight Fight", the knight does not intentionally injure himself. In "Princess and the Potion", the actor is not a knight. Since recall fails, imaginative memory recurses and applies a new TRAM at the inferior level.

MINSTREL now applies TRAM:Generalize-Constraint to the description "a knight purposely injures himself" and generalizes the "knight" feature (to the superclass), creating a new scene description "a natural being purposely injures himself." This description recalls "The Princess and the Potion" in which a lady of the court drinks a potion to make herself ill. This scene is adapted by TRAM:Generalize-Constraint by replacing the "lady of the court" feature with "a knight," resulting in a scene in which a knight makes himself ill by drinking a potion.

This adapted scene is returned to the previous level and is adapted by TRAM:Similar-Outcomes by replacing the illness with death. This results in a scene in which a

knight kills himself by drinking a potion, filling the original description "a knight kills himself." (Note that in the course of inventing this scene, MINSTREL has also invented the idea of poison – a potion that kills.)

The main issue in TRAM:Similar-Outcomes is determining when two outcomes are interchangable. MINSTREL has two methods for deciding this question.

First, MINSTREL assumes that if an action can result in a partial relative change of a state (i.e., drinking a potion results in a partial negative change in health) then the action can also result in a complete relative change of the state (i.e., drinking a potion can make one's health completely negative).

Second, MINSTREL assumes that two outcomes are interchangable if it can recall other scenes in which they are interchangable. MINSTREL assumes that if two outcomes are interchangable in any scene then they are interchangable in every scene. This method is implemented in a second TRAM called TRAM:Similar-Outcomes-Implicit.

7.3 TRAM:Intention-Switch

TRAM:Intention-Switch suggests that if the effect of an action in a scene was intentional it might just as well have been unintentional, and vice versa. TRAM:Intention-Switch is illustrated in Figure 7.4.

TRAM:Intention-Switch

Comment:	Look for an unintentional outcome instead of an intentional one.
Test:	Is this an action with an intentional outcome?
Transform:	Change the intentional outcome to an unintentional outcome.
Adapt:	Change the unintentional outcome of the solution to an intentional outcome.

Figure 7.4 TRAM:Intention-Switch

In the suicide example, TRAM:Intention-Switch creates the new specification "a knight accidently kills himself." Recall on this new specification fails, because MINSTREL's episodic memory does not contain any episodes in which a knight accidently kills himself.

Problem-solving is used recursively, and TRAM:Similar-Outcomes creates the new scene description "a knight accidently injures himself." This recalls "Knight Fight," in which a knight is injured while killing a troll. Both TRAM:Similar-Outcomes and TRAM:Intention-Switch adapt this recalled scene, resulting in a scene in which a knight commits suicide by intentionally losing a fight with a troll.

Since intended and unintended outcomes are always interchangable, TRAM:Intention-Switch is simple and useful in a wide variety of situations.

8 Current Status of MINSTREL

The current version of MINSTREL is written in Austin Kyoto Common Lisp using the RHAPSODY representation package [Turner 1987] and runs on a Sun workstation. Independent of RHAPSODY, MINSTREL contains about 8,000 lines of code and representation. MINSTREL implements twenty-nine TRAMs.

MINSTREL is primarily a storytelling program. The current version of MINSTREL tells a number of stories based on four different story themes. The role of theme and author-level goals in storytelling is discussed in more detail in [Turner 1992]. MINSTREL implements twenty-eight author-level plans, and begins story telling with approximately ten story scenes in memory. "The Lady's Revenge" is typical of the stories that MINSTREL creates:

The Lady's Revenge[2]

Once upon a time there was a lady of the court named Jennifer. Jennifer loved a knight named Grunfeld. Grunfeld loved Jennifer.

Jennifer wanted revenge on a lady of the court named Darlene because she had the berries which she picked in the woods and Jennifer wanted to have the berries. Jennifer wanted to scare Darlene. Jennifer wanted a dragon to move towards Darlene so that Darlene believed it would eat her. Jennifer wanted to appear to be a dragon so that a dragon would move towards Darlene. Jennifer drank a magic potion. Jennifer transformed into a dragon. A dragon move towards Darlene. A dragon was near Darlene.

Grunfeld wanted to impress the king. Grunfeld wanted to move towards the woods so that he could fight a dragon. Grunfeld moved towards the woods. Grunfeld was near the woods. Grunfeld fought a dragon. The dragon died. The dragon was Jennifer. Jennifer wanted to live. Jennifer tried to drink a magic potion but failed. Grunfeld was filled with grief.

Jennifer was buried in the woods. Grunfeld became a hermit.

2. Except for typography, this story appears exactly as produced by MINSTREL.

MORAL: Deception is a weapon difficult to aim.

Unlike previous work in storytelling (TALESPIN [Meehan, 1976], UNIVERSE [Lebowitz, 1985] and AUTHOR [Dehn, 1989]), MINSTREL focuses on the creative aspect of writing. Consequently, the creative problem-solving portion of Minstrel can be applied independently to other invention tasks. In addition to storytelling and the suicide example, MINSTREL's creative problem-solver has been applied to the mechanical invention domain, where MINSTREL uses TRAMs and an episodic memory of simple mechanical devices to invent heavy-duty staplers.

9 Previous Work

The most interesting previous work in artificial creativity is AM [Lenat 1976]. AM was a computer program that used a large pool of heuristics to discover "new" mathematical concepts. Starting with some basic tenets of set theory, AM discovered numbers (the lengths of certain sets), addition, subtraction, multiplication, prime numbers and eventually, the unique factorization rule.

One of the interesting differences between AM and MINSTREL is in the focus of their creativity. MINSTREL is a purposeful creator: it begins creation with a problem that it is trying to solve, and ends with a proposed new solution. AM, on the other hand, started with a group of base concepts and elaborated those in many directions. The purpose of AM was not to solve a particular problem, but to find interesting problems in a large problem space. More detailed comparison of MINSTREL and AM may reveal interesting similarities and differences between purposeful creativity and explorative creativity.

10 Conclusions

The challenge of creativity is to develop a cognitive model which explains how a creative problem-solver can (1) find old knowledge that can be useful applied to a new problem, and (2) adapt that knowledge to create a working solution. MINSTREL answers this challenge by integrating the search and adapt portions of the creative process in heuristics called TRAMs. This model has been implemented and extensively tested in a computer program that tells stories, solves problems, and invents mechanical devices.

References

Dehn, Natalie, *Computer Story-Writing: The Role of Reconstructive and Dynamic Memory*, Yale Technical Report 792 (Ph.D. Dissertation), Yale University, Dept. of Computer Science, 1989.

Koestler, A. *The act of creation.* MacMillan, New York, 1964.

Lebowitz, Michael, *Story Telling and Generalization*, Proceedings of the Seventh Annual Conference of the Cognitive Science Society, Irvine, California, 1985, pp. 100-109.

Lenat, Douglas B., *AM: An Artificial Intelligence Approach to Discovery in Mathematics as Heuristic Search*, Stanford AI Lab, Memo AIM-286 (Ph.D. Dissertation), Stanford University, Dept. of Computer Science, 1976.

Meehan, James R., *The Metanovel: Writing Stories by Computer*, Technical Report #74 (Ph.D. Dissertation), Yale University, Dept. of Computer Science, 1976.

Reiser, B.J. (1986). "Knowledge-directed retrieval of autobiographical memories." In Kolodner, J., & Riesbeck, C.K. (Eds.) *Experience, Memory and Reasoning.* Hillsdale, New Jersey: Lawrence Erlbaum Associates.

Schank, Roger C., *Dynamic Memory*, Cambridge University Press, Cambridge, 1982.

Slade, S., "Case-Based Reasoning: A Research Paradigm", *AI Magazine*, Spring, 1991.

Tulving, E., "Episodic and Semantic Memory." In E. Tulving & W. Donaldson (Eds.), *Organization of memory.* New York: Academic Press, 1972.

Turner, Scott and Reeves, John, *The RHAPSODY Manual* (Technical Note UCLA-AI-N-85-87). Artificial Intelligence Laboratory, Computer Science Department, University of California, Los Angeles, 1987.

Turner, Scott. *MINSTREL: A Model of Storytelling and Creativity*, Technical Note UCLA-AI-17-92 (Ph.D. Dissertation). Artificial Intelligence Laboratory, Computer Science Department, University of California, Los Angeles, Dec., 1992.

Weisberg, Robert W., *Creativity: Genius and Other Myths.* W.H. Freeman and Company, New York, 1986.