Computer Science Department Technical Report
University of California
Los Angeles, CA 90024-1596

MINSTREL: A COMPUTER MODEL OF CREATIVITY AND STORYTELLING

S. R. Turner

December 1992
CSD-920057

MINSTREL: A Computer Model of Creativity and Storytelling

Scott R. Turner

December 1992

Technical Report UCLA-AI-92-04

For Jennifer

Table of Contents

ACKNOWLEDGEMENTS

In the long course of research that led up to this dissertation I received support from many people.

I owe much to the members of my dissertation committee. Most importantly to my advisor, Prof. Michael Dyer, whose own work provided a goal toward which I could aspire. Profs. Charles Taylor and David Jefferson provided me with the opportunity to do research outside my dissertation focus, which proved to be valuable as well as interesting. And Prof. Gerald Estrin provided me with encouragement, support and invaluable advice on the presentation and structure of my research, for which I will always be grateful.

My fellow graduate students at UCLA provided moral support and technical assistance. Particular thanks must go to John "The Big Man" Reeves, who not only taught me a great deal about critical thinking, but also helped implement the tools package upon which MINSTREL is written. Others who provided intellectual stimulation and friendship include Maria Pozzo, Michael Pazzani, Seth Goldman, Sergio Alvarado, Jack Hodges, Ron Sumida and Trent Lange. Special mention must also be made of the "Goombahs" who provided sporadic intellectual stimulation and taught me Skittgubbe: Mark Bannilower, Mark Sybert, Rick Gillespie, Matthew Merzbacher, and Adger Williams.

I also owe a debt to The Aerospace Corporation, which provided me with financial support and a stimulating work environment, without which I would surely have never finished this dissertation. Deserving of special thanks are my managers, Patricia Mangan and Ervin Frazier, who have been notably understanding and supportive. This research was also supported in part by the Hewlett-Packard Corporation.

Finally and most importantly, I must give thanks to the members of my family, who have provided me with love and encouragement for many years. To my parents, who raised me to cherish knowledge and to have the courage to seek after what I wanted. To my grandmother, who occupies a special place in my life. And most specially to Jennifer, who gave me a reason to finish and love beyond anything I have ever earned.

ABSTRACT OF THE DISSERTATION

MINSTREL: A Computer Model of Creativity and Storytelling

by

Scott R. Turner
Doctor of Philosophy in Computer Science
University of California, Los Angeles, 1992
Professor Michael Dyer, Chair

Telling a story is a difficult task that requires a variety of knowledge and cognitive processes: knowledge about themes, writing techniques, the story world, and presentation; processes such as planning, problem-solving, recall and creativity. This dissertation presents a model of the storytelling process which incorporates theories of creativity, memory and author-level planning. This model has been implemented in a computer program called MINSTREL which tells short, theme-based stories about King Arthur and his Knights of the Round Table.

MINSTREL's creativity process is based upon creativity heuristics called Transform-Recall-Adapt Methods (TRAMs). Each TRAM integrates a simple problem transformation which searches the problem-space for new knowledge to apply to a problem with a corresponding adaptation which can adapt any discovered knowledge to the original problem. By using TRAMs to augment the problem-solving process, MINSTREL is able to invent useful new problem solutions. By incorporating creativity into the recall process, MINSTREL makes creativity available to any cognitive process, and permits the use of multiple TRAMs to make creative "leaps".

MINSTREL's storytelling process is based upon a model of author-level problem-solving. In addition to a process model of author-level problem-solving, MINSTREL implements four important classes of author-level goals and plans: (1) Thematic, (2) Dramatic, (3) Consistency, and (4) Presentation. MINSTREL uses these goals and plans to tell a number of short stories in the King Arthur domain.

MINSTREL is a computational model of the cognitive processes of storytelling and creativity. MINSTREL (1) describes a process model of storytelling, (2) identifies important storytelling goals and plans, (3) identifies fundamental storytelling processes, (4) describes a process model of creativity, (5) explains how a problem-solver can find and adapt old knowledge to create new solutions, (6) identifies useful creativity heuristics, (7) explains creative "leaps", (8) explains the relation of creativity to problem-solving, (9) describes the relationship between memory and creativity, and (10) integrates creativity into a larger cognitive model.

CHAPTER 1
Storytelling and Creativity

1.1 Introduction

During my senior year in college, I was browsing the stacks in the research library and quite by accident came across a small but intriguing book called *The Morphology of the Folktale* [Propp 1968]. The author, Vladmir Propp, had studied common Russian folktales and distilled the form[1] of those tales into cryptic equations:

$$S \rightarrow ABC \uparrow DEFG \frac{HJIK \downarrow Pr - Rs^{0}L}{LMJNK \downarrow Pr - Rs} QExTUW *$$

Using letters to represent story elements such as "One of the members of a family absents himself from home", each equation captured a common pattern Propp had found in the folktales he studied. Altogether the equations in Propp's book formed a definitive description of the form of Russian folktales.

As a computer scientist, I found this fascinating. Propp had reduced the folktale to a grammar – a set of well-defined rules. Grammars are used throughout computer science to formalize structure and to understand well-structured input. Compilers, for example, use grammars to translate computer programs in languages like FORTRAN – which are easily understood by humans – into the binary "ones and zeros" understood by computers. In theory, Propp's grammar could be programmed into a computer and used to recognize folktales – provided someone first translated each folktale into Propp's notation.

But what was more intriguing to me was the notion of running Propp's grammar "backwards". Propp's grammar was intended as a tool for recognizing and understanding the underlying forms of folktales. But, I reasoned, the same grammar could be used in reverse to *create* folktales. By starting with an initial rule and then randomly choosing the next rule to apply, Propp's grammar could be used to "grow" a story from seed to completion. The random choices would ensure that the story created would likely be different from any actual folk tale, while the rules would ensure that the resulting story had the form of a folk tale. And programming a computer to do this would be trivial. A few hours in the Computer Lab and I would have a computer program that could tell stories!

Or so I thought.

I did eventually write a computer program that tells stories. But it took years, not hours, and in the end, Vladmir Propp's intriguing little grammar was nowhere to be seen.

1. Morphology is the study of form and structure.

This dissertation is the story of the program I wrote and what I learned in the process. It looks at the myriad problems an author faces when he sits down to write a story, and presents the processes, techniques, and knowledge needed to address these problems. Like all authors, I hope you find my story both interesting and enlightening.

1.2 The Storytelling Problem

It is surprisingly difficult to tell a story.

Even young children can *understand* stories. By four or five, children understand most aspects of folktales like the ones Propp studied. Indeed, the primary use of folktales is to teach the young principles they'll need as adults.

But *telling* a story is a different matter. It seems an easy enough task. Surely an adult should be able to easily create what a child can easily understand. But more than a few educated, intelligent adults have learned differently when put on the spot by their children. Most can manage little more than an embarassing hodge-podge of stereotypical cliches, inevitably starting "Once upon a time..." They sputter out a trite beginning and are soon lost. And if grown adults find it difficult to tell a story, imagine how much more difficult it must be to build a computer program to tell stories.

Why is storytelling so difficult?

Storytelling appears simple because at a surface level, stories *are* simple. As Propp showed, the form of a story can be captured by a simple, easily understood formalism. But there is more to a story than form. Underlying the form is the story content - the meaning of the story. And it is here that the difficulties arise.

Because in this case, form does not reflect function. Underlying the form of a story is a complex web of author goals, reader expectations and cultural knowledge. Just as an elegant mathematical proof does not reveal the knowledge and effort that went into its making, neither does the form of a story reflect the difficult process of its creation.

Authors craft stories to achieve a wide variety of complex and often competing goals. To understand why storytelling is so difficult, we must understand what an author is trying to achieve. To build a computer program to tell stories, we must understand and model the processes an author uses to achieve his goals. Both of these are difficult tasks.

The next few sections of this chapter illuminate some of the problems an author faces in telling a story. In the second part of this chapter we'll take a quick look at how these problems can be solved by a computer program. But for now we seek only to raise some of the intriguing problems a storyteller faces.

1.2.1 Why Form Alone is Insufficient

Around 1958, Roger Price and Leonard Stern came out with a party game called *Mad Libs*® [Price and Stern 1958] that became an instant classic. Each Mad Lib was a story with key words missing:

A Fable

> Once upon a time there was a very curious girl who was always sticking her nose into everybody's _____ (plural noun). She kept company with a/an _____ (adjective) man named Dave, who was always buying her _____ (adjective) presents...

The game is played by having people fill in the blanks knowing the proper type of word, but not the surrounding context. The result is often funny and occasionally hilarious.

Like Propp's work, each Mad Lib is a kind of grammar. It specifies the form of a story without specifying the exact content of the story. Mad Libs works as a party game because the final story has a legitimate form combined with an absurd meaning. That's a combination that is, at least in small doses, quite amusing.

But as a way to create stories, Mad Libs leave much to be desired. Mad Libs are amusing, but they aren't good stories. Propp's grammar, although more complex, has the same failing. It captures the form of a story but not the content of a story. And like Mad Libs, Propp's grammar can produce stories that have good form but absurd meanings. The fundamental failing of story grammars is that they capture form without meaning.

Of course, the form of a story is important. We can appreciate a well-crafted story, or admire a good turn of phrase. We also expect a story to have a certain form, and may classify it as a "bad" story if it does not. But most of our appreciation of stories comes from the content level. Storytelling is an act of communication between the author and his readers. It is what the story tells – the message – that matters most to both the author and the readers. What Mad Libs and Propp's grammar fail to capture is the message level of storytelling. Any storytelling system based solely on the surface features of stories – whether a complex system like Propp's or a simple system like Mad Libs – will inevitably fail to be successful. A story that doesn't mean anything is not a story, even if it has the proper form.

A storyteller must have an in-depth understanding of the stories he tells.

An author must understand the meaning of the stories he tells. One reason storytelling is difficult is because it requires the storyteller to understand the story he is telling at every level: the surface format, the message or point of the story, the actions of the characters, the events in the story world, the literary values in the story.

For human authors, this task isn't difficult. Humans spend the first twenty years of their lives learning about the world, about how people act, and about ways to understand the world. They are skilled and experienced at using this knowledge, whether to manage their day-to-day life or to understand a story.

But for a computer program this represents a tremendous barrier. To a certain extent a story is a model of a tiny world, peopled with story characters, natural forces, locations, and inanimate objects. To understand these things and their interrelationships requires a tremendous amount of knowledge that humans take for granted. Consider, for example, the simple sentence:

> When Galahad saw the dragon charge, he drew his sword and jumped to the side.

To understand this sentence in depth requires an enormous amount of knowledge:

- What is a dragon?
- What is a knight?
- How do we know Galahad is a knight?
- What is Galahad wearing?
- What does "charging" mean in this context?
- What is the dragon trying to accomplish?
- Why?
- What is a sword?
- What does "drawing" mean in this context?
- Why does Galahad draw his sword?
- Why does Galahad jump to the side?
- What is Galahad trying to accomplish?
- Why?
- What will Galahad do next?
- What will the dragon do next?
- What is Galahad feeling?
- Is the dragon feeling?

Capturing and applying all this knowledge to the task of storytelling is one of the challenges of building a computer program that can tell stories. But this is necessary because a story is more than just a form; it is has in-depth meaning to both the author and the reader.

1.2.2 Purpose and Message

Meaning is important to storytelling because storytelling is a form of communication. The author of a story isn't simply stringing together words randomly, or even according to a grammar. Storytelling is a purposeful activity. The storyteller constructs his story to bear a message to his readers. What affect does that purpose have on storytelling?

Consider the following story:

Rainy Day

One day, Tom got up in the morning and saw that it was raining. He went downstairs and had breakfast. Then he sat by the window and read a book for a while. It was still raining. Later, Tom fixed himself a sandwich for lunch.

The End

Rainy Day isn't much of a story. The problem isn't that it lacks form (the sentences are all grammatical) or that it has an absurd meaning (it's quite understandable). It's just boring. It has a message, but the message isn't interesting. As a story *Rainy Day* is a failure because the message conveyed isn't worth the work required to extract it.

So what makes a message worth the effort? What makes a message interesting?

Certain topics are inherently interesting. Sex and danger - both of which appeal to primitive drives - arouse interest in almost any context. Novelty and new ideas are also interesting - mankind has retained curiousity as one legacy of his primate heritage. Useful information is also interesting. An article on how to reduce your tax bill is likely to interest you for this reason. Still other topics appeal only to some readers. Presumably the reader of this dissertation is interested in artificial intelligence. Or perhaps you are a member of my family, and your interests are aroused for other reasons. There are many ways a story can be interesting.

One of the difficulties in storytelling is in choosing an interesting message and finding an interesting way to convey that message to the reader. Sometimes life provides an interesting message. The author of *Adrift* [Callahan 1986] was lost at sea for seventy-six days without food or water. The story of his experience is interesting because he faced danger, invented novel solutions to his problems, and learned useful information about survival under the most difficult of circumstances. But when life doesn't provide an interesting story, the author must create his own, interesting message:

A storyteller must fashion his story to convey an interesting message.

Finding, formulating and conveying an interesting message is one of the reasons that storytelling is such a difficult task. To build a computer program that can tell stories, we must build a model of communication. The computer program must (1) know what an interesting message is, (2) be able to select a message to convey, and (3) be able to create a story that illustrates the message. Building a model of "messages" and designing the processes that can illustrate a message is one of the challenges of creating a program that can tell stories.

1.2.3 Creativity

In literature, as in all the arts, there is a premium placed on creativity. To be art, a work must be new and different in significant ways. No publisher would accept a copy of *Romeo and Juliet* with only the names changed. Even an author who tells consistently interesting stories inevitably loses popularity if his stories are all very similar. One of the reasons that storytelling is so difficult is that the author is challenged to be creative. It isn't enough to tell an interesting story; the author must also strive to make the story new and different.

A storyteller must be creative.

And yet creativity is complex and difficult topic. Even judging whether or not something is creative is problematical. Surely copying *Romeo and Juliet* with only the names changed is not creative. But what if an author copied *Romeo and Juliet* and changed the setting to, say, the west side of New York City? The musical *West Side Story* was a tremendous Broadway hit and award-winning movie. Was Leonard Berstein being creative when he wrote *West Side Story*? Or does Shakespeare deserve the credit for that success?

Clearly whether or not something is creative depends upon the number and quality of its differences from similar works. But how can we distinguish inspiration from plagiarization? How can we judge when something has enough differences from previous work to be creative? And how can we determine if the differences are significant? These are just some of the problems in determining whether something is creative.

And if judging creativity seems difficult, *being* creative seems almost impossible. The creative person performs an almost miraculous feat: the bringing forth of something new and novel, something never before seen. The average person cannot write a creative story, paint a creative painting, or invent a new device. The few people who are consistently and greatly inventive - William Shakespeare, Albert Einstein, Leonardo da Vinci, Thomas Edison - are revered as geniuses. And yet storytelling is an activity that demands creativity. No wonder then that it is so difficult to tell stories.

What is the source of creativity? The ancients believed that creativity was the work of a supernatural Muse who spoke into the artist's ear. While few artists today would profess to believe in a literal Muse, many do believe that creativity originates in processes that are beyond human comprehension. Is that true? Is creativity a mystery that can never be explained by man, nor expressed in language? Or does creativity have a natural explanation in the cognitive processes of the human mind? Can those processes be simulated in a computer program?

Many psychologists and cognitive scientists today believe that creativity is the result of cognitive processes that bring together pieces of old knowledge in new ways. But that is hardly a full explanation of creativity. To embody this model of creativity in a computer program requires addressing a myriad of difficult issues:

- How is knowledge organized?

- How is knowledge searched?
- How does the creator determine what knowledge to use?
- How is knowledge combined to form new knowledge?
- What are the different methods of combining knowledge?
- How does creativity interact with other cognitive processes?
- Are there different types of creativity?
- Is scientific discovery different from artistic creativity?
- Is imagination different from creativity?
- What is the role of problem-solving in creativity?
- How does the creator recognize something new?
- How does the creator guide his creativity?
- How does the creator evaluate his creation?

The requirement to be creative is one reason that storytelling is such a difficult task for humans; the requirement to understand and model creativity is one of the reasons that building a computer program to tell stories is so difficult.

1.2.4 Art and Language

The sciences distinguish between content and presentation. We can and do speak of research as "important but poorly presented" or "polished but lacking substance". In the arts, however, it is much more difficult to separate presentation from content. In the sciences, presentation is a secondary concern, needed only to communicate content. But in the arts, presentation is part of the content. Art demands good presentation in a way that science does not.

This is true in literature as in the other arts. An author is expected not only to present a meaningful, interesting, and creative story, but also a well-crafted story. At one level, this requires that the author use language well by following the rules of grammar and punctuation while accurately conveying his meaning. At another level, this requires that the author present his story beautifully, by using language in poetic ways and by using dramatic writing techniques such as characterization and foreshadowing. In addition to all the other goals he is trying to achieve, a storyteller must try to create an artistic story.

A storyteller must create stories that are aesthetically pleasing.

Of course, this further complicates the problem of creating a storytelling program. To tell a story that meets literary standards as well as standards of comprehension, interest and creativity, a storytelling program needs knowledge of both language and drama. To use language well, a computer storytelling program requires knowledge about words and what they mean, an understanding of grammar, and a model of how language is produced – how concepts are expressed in language. To produce a story that is artistically pleasing, a computer storytelling program requires knowledge about the structure and parts of a story, knowledge about dramatic writing techniques (including how they are applied and what they achieve) and a model of dramatic writing that cap-

tures how an author uses dramatic writing techniques to improve the artistic value of his stories.

1.3 MINSTREL: A Computer Model of Storytelling

Clearly, telling a story is very difficult. A good story must be understandable, interesting, creative and artistic. Alone each of these goals is difficult to achieve; together they are truly formidable. It is no wonder that most adults are not good storytellers.

And as we have seen, the task of building a computer program to tell stories is even more daunting. Building a computer storyteller requires capturing all the knowledge a human author uses to tell a story, building models of storytelling, creativity, interestingness and art, and executing the plans and processes an author uses to tell a story. If not outright impossible, this is at least an enormous job.

In this section, we will give a brief overview of MINSTREL, a computer program that tells stories about King Arthur and his Knights of the Round Table. As one might expect from the difficulty of the storytelling task, MINSTREL is a large and complex program representing many years of research. It isn't possible to give a detailed description of MINSTREL in just a few pages. Instead, we hope to give the reader a general overview of MINSTREL's design and structure, and a preliminary glimpse into the kinds of knowledge and processes MINSTREL uses to tell stories. Hopefully this prelude will entice the reader deeper into this dissertation to discover the inner workings of MINSTREL.

1.3.1 What is MINSTREL?

MINSTREL is a computer program written in Common Lisp that tells stories about the Knights of the Round Table. MINSTREL is about 17,000 lines of code, and is built upon a tools package called Rhapsody [Turner 1987] that is itself about 10,000 lines of code.

The stories MINSTREL tells are about one-half to one page in length. MINSTREL tells about ten stories of this length, and can create a large number of shorter story scenes. In addition to storytelling, MINSTREL has been used to invent mechanical devices and to solve planning problems.

1.3.2 A Story

When they first hear about MINSTREL, most people are curious to read a story that MINSTREL has written, so that they can judge for themselves whether or not MINSTREL is a competent author. Accordingly, we will present here a representative story so that you can judge for yourself MINSTREL's capabilities. This story and other stories throughout the dissertation are presented exactly as produced by MINSTREL, with the exception of the titles, which were added later by the author. Here then, is one of MINSTREL's stories:

The Vengeful Princess

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 moved 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.

MORAL: Deception is a weapon difficult to aim.

The reader will probably have a number of immediate first impressions about this story:

- The story has a "point".
- The story is understandable and consistent.
- The story is reasonably clever, particularly the deception and how it leads to grief.
- The use of English is not as polished as a human author.

Most people judge MINSTREL's stories to be competent but not brilliant, equivalent to the kinds of stories people expect from children 10-15 years of age[2]. However, the reader will soon realize that it is difficult to evaluate MINSTREL's performance solely by looking at the stories it creates. As important as *what* MINSTREL creates is *how* it creates. Certainly *The Vengeful Princess* is less impressive if it was produced from canned text hidden in MINSTREL's code. To understand MINSTREL we have to look deeper. We have to look at the processes and knowledge MINSTREL uses to create stories.

2. See Chapter 15 for a discussion of a survey in which subjects were asked to evaluate MINSTREL's output.

1.3.3 The Architecture of MINSTREL

MINSTREL is based upon a model of the author as a *problem-solver.*

We are not accustomed to think of art as problem-solving. The phrase "problem-solving" brings to mind scientific disciplines, logic and schoolwork. And yet there are few fundamental differences between creative domains such as storytelling, art and music and more mundane problem domains, such as science and day-to-day problem-solving. In art as in day-to-day life, people have goals, find or create plans to achieve those goals, apply the plans, evaluate the results and so on. To be sure, the arts are generally more creative than other problem domains. But anyone who has performed scientific research or jury-rigged a temporary fix to a household problem knows that creativity is both possible and necessary in science and day-to-day life. We are not used to thinking of artistic endeavors such as painting and music in terms of problem-solving, but at a general process level there is little to distinguish between creating and playing a musical piece and creating and writing a thank-you note.

In MINSTREL, storytelling is treated as any other type of problem-solving. The same architecture and processes are used to write stories, to solve problems in the story world (such as how a knight can kill a dragon), and to invent mechanical devices. It is a basic tenet of this research that the problem-solving process is invariant across problem domains, whether they be mundane or artistic, creative or non-creative:

The problem-solving process is invariant across problem domains.

How then does MINSTREL solve problems?

MINSTREL is a type of *case-based reasoner* [Slade 1991][Reisbeck and Schank 1989]. The fundamental principle of case-based reasoning is that reasoners remember past problem-solving situations ("cases") and use that knowledge to solve current problems. A case-based reasoner solves a problem by recalling a similar past problem and then applying the solution from the past problem to the current problem. One of the major values of the case-based reasoning paradigm is that it explains how reasoners can learn from experience: they remember what happens and reuse that knowledge when appropriate.

MINSTREL's case-based problem-solving process is illustrated in Figure 1.1. Problem-solving proceeds in three steps: (1) The current problem is used to recall similar past problems, (2) The past solution is adapted to the current problem, and (3) The recalled solution is then applied to the current problem. Because there is nothing specific in this model to the storytelling problem domain, this same process can be used to solve both author-level storytelling problems and character-level planning problems, as well as problems in mechanical device invention.

MINSTREL's model of problem-solving is discussed in more detail in Chapter 3. However, one major shortcoming of this model should already be obvious. Case-based reasoners solve problems by re-using past solutions. A case-based storyteller would tell the same story over and over

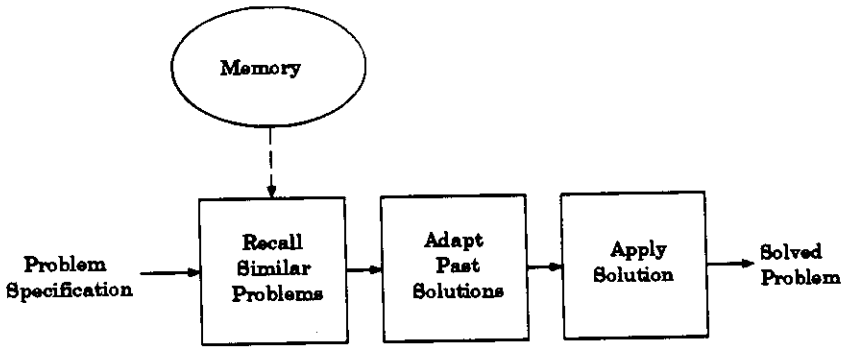


Figure 1.1 Case-Based Problem-Solving

again - repeatedly solving the storytelling problem by reusing an old solution. Such a storyteller would be the opposite of creative. What do we need to add to a model of case-based reasoning to support creativity?

1.3.4 The Challenge of Creativity

The challenge of creativity is to find and use old knowledge in new ways to form a novel and useful solution to a problem. What processes and knowledge do we need to be able to create *new* problem solutions from *old* knowledge? How can we extend the case-based model of problem-solving to include creativity? To begin answering these questions, let's look at an example of creativity. The following story concerns a UCLA engineering professor:

> One night, the professor and his wife were out late at a party. Returning home along a lonely road, the car they were driving slowly lost its electrical power. Looking under the hood, the professor discovered that the fan belt had broken.
>
> After a perfunctory search in the trunk for a spare fan belt, the professor returned to the car and asked his wife to remove her pantyhose. The professor tied a series of knots in the pantyhose, and then tied the pantyhose into a loop somewhat smaller than the fan belt. He forced the loop over the crankcase pulley and the alternator pulley. The elasticity of the pantyhose kept it on the pulleys; the knots provide traction so that the improvised belt wouldn't spin uselessly.
>
> After giving the battery a few minutes recovery, the professor started the car and successfully made it home.

In this example, the professor brings together things he already knows - requirements for a fan belt, available materials, the properties of pantyhose, and knowledge of knots - to create a new and useful solution to the "fan-belt" problem. From everything the professor knew, he somehow picked out the knowledge that when combined in a new and unique way would result in a solu-

tion to his problem.

This process of finding and combining knowledge seems paradoxical. On the one hand, if the professor just randomly grabs at old knowledge to try to create a new solution, he has an impossibly hard task in applying the knowledge he finds. Suppose the professor had recalled what he knew about grading papers, eradicating garden pests, and the rules of ping-pong and tried to apply that knowledge to the fan-belt problem. Surely he would fail. On the other hand, if the professor uses only knowledge that he knows how to apply to the current problem, he is unlikely to create a new solution. The professor's knowledge of normal car repair procedures and the tools and parts used can be easily applied to the fan-belt problem. But it is unlikely to result in a new solution.

This is a classic "Catch-22" situation. The creator must somehow search everything he knows to find knowledge that can be adapted to create a new solution. But without knowing what knowledge can be adapted, how can the creator recognize what knowledge is useful? Conversely, once the knowledge has been found, the creator must somehow adapt it to the current problem. But without knowing how the knowledge is related to the current problem, how can the creator adapt it? The challenge of creativity is to define a creative process which (1) Will find and recognize useful knowledge and (2) Know how to adapt that knowledge to create a new solution.

The solution is to *integrate* the search and adapt processes of creativity. By integrating the search and adapt processes, search can be guided by adaptation knowledge and adaptation guided by search knowledge. The search process finds only knowledge that it knows how to adapt; the adaptation process knows what adaptations to apply because it knows how the knowledge was found.

Creativity is an integrated process of search and adaptation.

In MINSTREL, the search and adaptation processes of creativity are integrated in heuristics called Transform-Recall-Adapt Methods, or TRAMs. Each TRAM bundles a search method with a corresponding adaptation. "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 memory. "Adapt" takes the recalled problem solutions and adapts them to the original problem. We'll illustrate how this works by looking at an example: TRAM:Cross-Domain-Solution.

TRAM:Cross-Domain-Solution is based on the idea that there are often enough similarities between two problem domains that solutions from one domain can be applied in the other. For example, military tactics can be applied to business problems – as evidenced by the popularity of *The Book of Five Rings* [Miyamoto 1982] with both Japanese and American businessmen. TRAM:Cross-Domain-Solution suggests that a creative person can take advantage of this by translating a problem into a new domain ("Transform"), solving the problem in that domain ("Recall"), and then translating the solution back into the original domain ("Adapt").

MINSTREL uses TRAM:Cross-Domain-Solution when telling a story called *The Mistaken Knight*. In the course of telling this story, MINSTREL has to create a scene in which a knight accidently meets a princess. As it happens, MINSTREL doesn't know any way in which a knight can accidently meet a princess, so it must use creativity to solve this problem. To do this, MINSTREL uses TRAM:Cross-Domain-Solution.

The first step of the creative process is to search for new knowledge to apply to the current problem. TRAMs do this by transforming the current problem into a new problem, and finding the problem solutions that have been previously used for the transformed problem. These problem solutions are "new" in the sense that they've never before been applied to the current problem. By transforming the current problem into a different problem, TRAMs search new areas of knowledge.

The "Transform" part of TRAM:Cross-Domain-Solution translates the original problem into a new problem domain. In this case, TRAM:Cross-Domain-Solution transforms the current problem ("A knight accidently meets a princess") from the domain of medieval stories into the modern day-to-day domain ("A businessman accidently meets a person"). TRAM:Cross-Domain-Solution does this by translating the original problem element by element, trying to find correspondences between the two domains. In this case, TRAM:Cross-Domain-Solution translates a knight into a businessman, leaves "accidental meeting" unchanged, and translates a princess into a person.

The second step of the creative process is to use the transformed problem as an index for recall, to see if the creator has ever encountered a similar problem before. In this case, the transformed problem recalls this story:

Walking The Dog

> John was sitting at home one evening when his dog began scratching at the door and whining for a walk. 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 wanted a walk.

The final step of the creative process is to adapt the recalled solution to the orginal problem. Because TRAM:Cross-Domain-Solution knows that the recalled solution was found by translating the original problem into a new problem domain, adaptation is easy. TRAM:Cross-Domain-Solution needs only translate the recalled story back into the original domain, creating this 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 picking berries.
```

The resulting scene is new and original, illustrating how TRAM:Cross-Domain-Solution can create a new solution by adapting knowledge from another problem domain.

MINSTREL's TRAMs integrate the search and adapt processes of creativity. This ensures that:

(1) Any knowledge found will be useful. The creative process involves only search methods for which the proper adaptations are *already* known. Therefore the creator will only find knowledge that he knows he can apply to the current problem.

(2) The creator will know how to apply the knowledge found. Because the creator knows how he found a piece of knowledge (i.e., what "Transform" led to the knowledge) he also knows how to adapt that knowledge. There is no need for brute-force adaptation; the search defines the adaptation.

Thus MINSTREL's TRAMs provide a model of creativity which resolves creativity's seeming paradoxes. This short example has only scratched the surface of the creativity problem. MINSTREL's model of creativity is discussed further in Chapters 3, 4, and 15 of this dissertation. Those chapters address a variety of other issues in creativity, including:

- How TRAMs incorporated into the case-based model of problem-solving.
- How multiple TRAMs can be used simultaneously to increase the power of creativity.
- How a problem-solver can recognize a creative solution.
- How creativity can be integrated into a model of episodic memory to create an *imaginative memory*.
- Descriptions and discussions of the TRAMs MINSTREL uses.
- Descriptions of experiments with the MINSTREL model of creativity.

1.3.5 Author Goals in Storytelling

In the previous section we saw in general how MINSTREL solves problems: by using a case-based problem-solver augmented with creativity heuristics. Let's return now to the particular problem of storytelling and look at the kinds of problems an author has to solve when he tells a story.

Earlier we identified four concerns of an author. An author wants to make his story (1) interesting, (2) understandable, (3) artistic, and (4) creative. We have already seen how MINSTREL achieves the last goal. How are the other three goals achieved?

MINSTREL has four classes of author-level goals corresponding to the major concerns of an author. They are:

- Thematic Goals
- Consistency Goals

- Drama Goals
- Presentation Goals

Thematic goals are concerned with the selection and development of a story theme. The story theme is the point or moral of the story. These goals assure that the story MINSTREL tells will have an interesting message. Consistency goals focus on creating a story that is plausible and believable. These goals assure that the stories MINSTREL creates will be understandable and consistent. Drama goals are concerned with the use of dramatic writing techniques to improve the artistic quality of a story. These goals represent MINSTREL's desire to tell a story that is aesthetically pleasing. Finally, presentation goals are concerned with how a story is presented to the reader, and represent MINSTREL's desire to tell the story in a pleasing and effective way.

These four classes of goals combine to create a complete story. In the following four sections we'll look briefly at each class of goals, what it contributes to a completed story, and how MINSTREL achieves each type of goal.

1.3.5.1 Theme in Storytelling

The theme of a story is the point, moral or general truth that the story illustrates. Some stories, like fables, have an easily identifiable theme. Other stories, such as novels, may have several themes. In general, we assume that an author tells a story in order to present some point to the reader, or to illustrate some truth. We call this the theme of the story.

Themes help make stories interesting. Partly this is curiosity: if the reader realizes that a story has a purpose or point, he is curious to discover what it is. Partly it is self-interest. If the theme represents useful knowledge, such as a general truth about how the world works, the reader is interested in learning the theme so that he can use it to improve his life. Themes are thus one way an author can give his stories purpose and make them interesting.

MINSTREL's stories have very specific themes called Planning Advice Themes, or PATs. Each PAT represents a stereotypical planning situation and advice about how to handle the situation. Each PAT is a morsel of planning advice, and can often be summarized by an adage.

For example, the theme of *The Vengeful Princess* is summarized by the adage "Deception is a weapon difficult to aim." This theme is called PAT:Juliet, and is based upon one of the themes in *Romeo and Juliet*. In the last act of *Romeo and Juliet*, Juliet takes a potion that makes her appear to be dead. Her intention is to deceive her family, but she deceives Romeo instead. Romeo, stricken with grief at the apparent loss of Juliet, kills himself. PAT:Juliet captures the mistake that Juliet made, and advice for planners considering a similar plan: "Be careful with deception plans because you may fool someone unintended."

MINSTREL's primary goal when storytelling is to tell a story that illustrates a particular Planning Advice Theme. By making this its primary storytelling goal, MINSTREL ensures that the stories it tells will (1) have a point or purpose and (2) be interesting.

MINSTREL's plan for illustrating a theme involves creating story events that form a specific example of the planning advice the theme represents. This can be seen in *The Vengeful Princess*, where MINSTREL has created story events to illustrate the theme "Be careful with deception plans because you may fool someone unintended":

```
Jennifer wanted to appear to be a dragon [to fool Darlene].
Jennifer drank a magic potion.  Jennifer transformed into a
dragon...

Grunfeld was near the woods.  Grunfeld fought a dragon.  The
dragon died.  The dragon was Jennifer.  Jennifer wanted to
live...
```

These events illustrate the story theme by showing how Jennifer's deception plan leads to grief when Galahad is fooled instead of Darlene. And by including an example of the theme in the story, MINSTREL makes the story purposeful and interesting.

MINSTREL's use of themes is discussed in more detail in Chapter 6. Among the issues addressed are:

- The structure and representation of Planning Advice Themes.
- How a theme is selected for storytelling.
- How the events that illustrate a theme are created.
- How new story themes can be invented.
- How story themes function as advice.
- Descriptions of the story themes that MINSTREL knows.

1.3.5.2 Consistency in Storytelling

As the reader may have noticed, the story events that illustrate the theme are only a small part of the final story. *The Vengeful Princess* is about thirty sentences long, but only four or five of those sentences concern the story theme. Even though illustrating a theme is MINSTREL's primary purpose in storytelling, a good story requires that MINSTREL be concerned with many other goals. One of these is story consistency.

A good story must be consistent at many levels. The author must have an in-depth understanding of both story form and of the world in which the story takes place. The characters and the world should act consistently and predictably. MINSTREL tries to create stories in which the characters act as rational and intelligent planners, in which the characters show the proper emotional reactions to events in their lives, and in which the story world is consistent and plausible. These

concerns are apparent in several places in *The Vengeful Princess*.

To illustrate the story theme, MINSTREL creates a scene in which Galahad kills a dragon (which turns out to be Jennifer). This story scene satisfies the thematic requirement that Jennifer's deception leads to grief. But in other ways the scene is inconsistent. There is no explanation of why Galahad killed the dragon, or how he reacts to the discovery that the dragon is really Jennifer.

To correct these problems, MINSTREL adds new story scenes. First MINSTREL adds scenes to explain why Galahad kills the dragon:

```
    Grunfeld wanted to impress the king.  Grunfeld wanted to
move towards the woods so that he could fight a dragon.  Grun-
feld moved towards the woods.  Grunfeld was near the woods.
Grunfeld fought a dragon...
```

MINSTREL uses knowledge about knights, what their typical goals are, and how they achieve those goals to create story scenes which explain why Galahad fights a dragon.

Similarly, MINSTREL notices another consistency problem in the scene in which Galahad fights the dragon: Galahad has no emotional reaction to the discovery that the dragon is actually a princess. MINSTREL uses a model of emotions to decide that Grunfeld would be grief-stricken to unintentionally cause the death of a princess:

```
The dragon died.  The dragon was Jennifer...  Grunfeld was
filled with grief.
```

By detecting and correcting story inconsistencies such as these, MINSTREL tells stories that are plausible and understandable. Story consistency is discussed in more detail in Chapter 9. Among the issues addressed are:

- How story consistencies arise.
- The types of planning inconsistencies MINSTREL can detect and correct.
- The types of story world inconsistencies MINSTREL can detect and correct.
- How emotions are modelled in MINSTREL.
- How emotional inconsistencies are detected and corrected.

1.3.5.3 Art and Drama in Storytelling

Another major concern of a storyteller is to create a story that has aesthetic appeal. By writing a story with literary values, an author helps make his story interesting and increases the emotional and intellectual impact of his story.

Human authors have many literary writing techniques: pacing, characterization, dialogue, suspense, foreshadowing, description, and many others. A look at the creative writing section of any bookstore will reveal that there are as many writing techniques as there are authors to expound them. Every human author develops a combination of literary writing goals and techniques that create a particular writing style.

It would be impossible to model all these techniques and their myriad combinations in MINSTREL. Instead, MINSTREL implements a few representative techniques: suspense, tragedy, characterization, and foreshadowing. This particular combination of techniques represents, if you will, MINSTREL's writing style.

One of the techniques that MINSTREL uses in creating *The Vengeful Princess* is tragedy. Tragedy is a literary form that evokes feelings of pity and regret in the reader, and often involves a character suffering a downfall because of a tragic flaw. MINSTREL has the ability to recognize when a story has the potential to be tragic, and has techniques it uses to emphasize the tragedy. The seed of a tragic situation occurs in *The Vengeful Princess* when Galahad accidently kills Jennifer. The tragedy here is that Jennifer's temper leads her to seek revenge for a trivial reason, and eventually results in her own death.

MINSTREL has several plans for accentuating a tragic situation. One of these plans increases the impact of a tragic situation by making the tragedy occur at the hands of a loved one. This plan was used in the telling of *The Vengeful Princess* by making Galahad and Jennifer lovers:

```
    Once upon a time there was a Lady of the Court named Jen-
nifer.  Jennifer loved a knight named Grunfeld.  Grunfeld loved
Jennifer...

...Grunfeld fought a dragon.  The dragon died.  The dragon was
Jennifer...
```

The fact that Jennifer's tragic flaw leads to her death at the hands of someone who loves her intensifies the tragic aspect of this story.

By applying dramatic writing techniques like tragedy, MINSTREL creates stories that have literary value and are aesthetically pleasing. MINSTREL's dramatic writing goals are discussed in more detail in Chapter 8. The writing techniques described are:

- Suspense

- Tragedy
- Characterization
- Foreshadowing

1.3.5.4 Presenting the Story to the Reader

The final task of an author is to present his story to his readers. He must express the story in language, in a manner that is both pleasing and understandable. A good author will also fashion his story presentation to reflect and accentuate the purposes of his story.

To express a story in English, MINSTREL must accomplish several tasks. First, MINSTREL must select an order in which to relate the events of the story. In many cases, this ordering can be guided by the temporal ordering of story events. In other case, such as foreshadowing, MINSTREL must make specific decisions about the order in which to present story scenes. Second, MINSTREL must generate the story events in English. This requires selecting words to express concepts, building up grammatical sentences, paragraphing, and many other tasks. Finally, MINSTREL may also create new story scenes in order to improve the presentation of the story.

In *The Vengeful Princess*, MINSTREL creates a story scene to introduce the reader to the main character of the story:

```
Once upon a time there was a Lady of the Court named Jen-
nifer...
```

and story scenes to resolve character fates:

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

These scenes are created to improve the story presentation. The introductory scene is created to ease the reader's transition into the story by providing an immediate identification of the main character and the story genre. The final scenes are created to resolve the reader's expectations about the fates of the story characters and create a sense of closure.

MINSTREL achieves a number of difficult tasks in presentation. The problems of presentation and how MINSTREL solves them are discussed in more detail in Chapter 10. Among the issues discussed are:

- How introduction scenes are created.
- How denouement scenes are created.
- How story events are ordered, both implicitly and explicitly.

- How paragraphing is used to reflect thematic structure.
- How natural language is generated from concepts.

1.4 A Reader's Guide

The remainder of this dissertation is divided into four sections.

The first section, Chapters 3 and 4, is concerned with MINSTREL's model of creativity. Chapter 3 describes MINSTREL's model of creativity, how creativity is used to augment case-based reasoning, and gives examples of MINSTREL's creativity in planning and storytelling. Chapter 3 also discusses a variety of related issues, including errors in creativity and learning. Chapter 4 is a dictionary of all the creativity heuristics that MINSTREL uses during storytelling.

The second section, Chapters 5 through 10, is concerned with MINSTREL as a storyteller. Chapter 5 describes MINSTREL's model of storytelling: what goals authors have, how these goals are achieved, and what role creativity plays in storytelling. Chapters 6 through 10 discuss the four major classes of author goals: thematic goals, instantiation goals, dramatic writing goals, and presentation goals.

The third section consists of a single chapter, Chapter 11. This chapter presents an extended example of how MINSTREL tells a story. A detailed trace of MINSTREL creating a story is analyzed to reveal the exact sequence of author goals and plans that leads to a finished story.

The final section, Chapters 12 through 16, is an evaluation of MINSTREL. In Chapters 12, 13, and 14, MINSTREL is compared to previous work in psychology and artificial intelligence. MINSTREL is evaluated as a cognitive model of creativity, and compared to previous AI models of storytelling and creativity. Chapter 15 presents a number of experiments and studies that were performed to determine the scope and robustness of MINSTREL. Chapter 16 contains conclusions and some thoughts on future work.

In addition to these sections, there are several appendices and a background chapter. The background chapter, Chapter 2, gives a brief description of MINSTREL's conceptual representation and MINSTREL's model of episodic memory for readers who may be unfamiliar with these ideas. The Appendices provide some additional information that will be of interest only to some readers.

CHAPTER 2
Background

2.1 Introduction

The research presented in this dissertation is based upon a rich heritage of previous work in artificial intelligence and cognitive science. The theories of creativity and storytelling presented in this dissertation build upon previous work in conceptual representation, models of human memory, and natural language understanding. For the most part, background information is reviewed as necessary for the reader's understanding throughout the dissertation. However, some topics are so fundamental to the research presented here that they merit separate explanation.

Accordingly, this chapter provides a simple introduction to the topics of (1) schema-based representation of concepts, and (2) the context-plus-index model of episodic memory. It is not intended to be an exhaustive discussion of these topics, a detailed critique of their strengths and weaknesses, or a philosophical argument. It is intended solely as an aid to the reader who lacks background in these areas. The advanced reader who is already familiar with these topics can safely skip this chapter, or refer back to it when encountering an unfamiliar concept or notation.

2.2 Schema-Based Representation of Concepts

When a human author writes a story, he does so by manipulating concepts or "thoughts" in his mind, building up story events and eventually expressing a complete story in language. Some of the concepts he manipulates are long-term, such as memories of stories he has previously read. Some are short-term, such as fleeting thoughts about how to develop the story he is writing. Some of these concepts make up the story he is telling, and will eventually be expressed in language so that they can be shared with other people. No one knows what physical reality concepts have in the human brain, but it is useful to speak of concepts, what they represent, and how they are manipulated.

Like a human author, MINSTREL creates stories by building and manipulating concepts. But unlike a human author, we know exactly how these concepts are represented in MINSTREL's "brain". We can describe them exactly, say what they represent, and define how they are manipulated.

In MINSTREL, concepts are represented using a *schema-based representation.* Schemas (or frames) have a long history in the AI community and have been used for a variety of purposes. The schema-based representation presented here is based upon previous work from many different researchers, including [Minsky 1975][Schank 1977][Schank 1982][Dyer 1983] and [Kolodner 1984].

The purpose of this section is to describe what is meant by a schema and to define precisely the schemas MINSTREL uses and what they represent. However, most of MINSTREL's representation is intuitive, and the reader should not need to study this material exhaustively. It should be

sufficient to read quickly through this material to get a general idea of MINSTREL's representation, and then refer back to this section if something later proves puzzling.

2.2.1 What is a Schema?

A schema represents a specific piece of knowledge, such as "John has the goal to satisfy his hunger." Each schema consists of two parts: (1) a schema type and (2) slot fillers. The *schema type* represents a generalized concept, such as "a goal", "an action", or "a belief". The *slot fillers* specify the generalized schema type to create a piece of specific information. Each schema type has a limited, specific set of slots. For example, the goal schema might have slots for "Actor" and "Type of Goal".

Thus "John has the goal to satisfy his hunger" can be represented as a schema of type goal, with the slot fillers "Actor is John" and "Type is Satisfy-Hunger". This is shown graphically in Figure 2.1. The schema type (goal) provides a specific, limited context in which to interpret the slot fillers of the schema. The slot fillers provide the information necessary to fully specify the goal being represented. In this case, the schema type identifies the concept being represented as a goal, and the slot fillers represent whose goal it is (John) and what type of goal it is (Satisfy-Hunger).

Figure 2.1 Example Goal Schema

Note that the schema shown in Figure 2.1 is labelled "&GOAL.14". It will be the convention throughout this dissertation to label specific instances of schemas with a name consisting of the general schema type ("GOAL") and an instance identifier ("14"). In addition, all symbols in MINSTREL that represent conceptual objects have an ampersand prepended to the name to make them easy to distinguish from symbols used as temporary variables, Lisp functions, and so on.

In addition to the graphical notation shown in Figure 2.1, we will sometimes also use a list notation for schemas:

```
(GOAL &GOAL.14
   :ACTOR   'JOHN
   :TYPE    &SATISFY-HUNGER)
```

This notation is based upon the underlying Lisp structure of schema instances. In the underlying Lisp, schema slot names are implemented as keywords, and so have a colon prepended to their names. Further details about Rhapsody ([Turner and Reeves 1987]), the tool kit upon which MINSTREL was built, can be found in Appendix A.

One final important feature of MINSTREL's schema representation is that it is *canonical*. This simply means that each concept is reduced to the simplest possible representation, that each concept has a unique representation, and that similar concepts have similar representations. There are numerous reasons why this is useful, but the most important is that it simplifies the structure of the cognitive model. Cognitive processes do not need to interpret the concepts (schemas) they manipulate; the schemas are already in a base, unambiguous form.

2.2.2 Slots Vs. Links

Slots provide a mechanism for creating a specific concept from a general concept. However, we also require a way to represent relationships between specific concepts. For example, we'd like to be able to represent that "John eats the ice cream" is the plan John has for achieving the goal "John wants to satisfy his hunger". Making the plan a slot filler on the goal schema would be inappropriate, because the plan does not further specify the goal. Rather, we need a mechanism for representing relationships between concepts.

In MINSTREL, relationships between concepts are represented by named links between schemas. Two concepts which are in specific relationship to each other are connected by a link which represents the relationship. For example, the schema representing "John wants to satisfy his hunger" can be connected via a "plan" link to the schema representing "John eats the ice cream". The link then represents the planning relationship that exists between the two concepts.

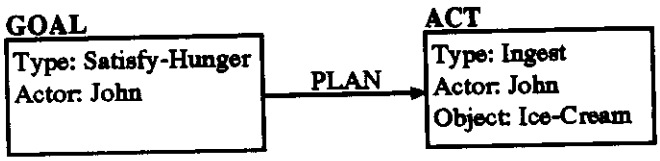


Figure 2.2 Graphical Representation of Links

In graphic notation, links between schemas are shown as labelled, directional arcs (see Figure 2.2). In list notation, links are distinguished from slots by having an initial ampersand instead of an initial colon. In addition, a "<==>" is printed to indicate a link:

```
(GOAL &GOAL.14
   :ACTOR   'JOHN
   :TYPE    &SATISFY-HUNGER
   &PLAN  <==>  &ACT.27)
```

The primary purpose of distinguishing links from slots is to clarify the meaning of MINSTREL's representations. Not only is the meaning more obvious to a human examining MINSTREL's representations, but distinguishing relationships from specifiers also makes it obvious that the two types of knowledge must be manipulated in different ways.

2.2.3 MINSTREL

MINSTREL tells stories in the domain of King Arthur and his Knights of the Round Table. MINSTREL therefore has schemas for representing both (1) author-level concepts, such as the goals of the author and the story structure, and (2) character-level concepts, such as the actions, goals and beliefs of the characters in the story. This section describes the schemas MINSTREL uses for both these levels of representation.

2.2.3.1 Author-Level Concepts

As an author, MINSTREL has goals and plans to achieve those goals. MINSTREL also has an author-level representation of the story it is telling, which includes structures such as the theme of the story, the various parts of the story (introduction scenes, the body, denouement scenes).

MINSTREL's author-level goals are represented as goal schemas. Figure 2.3 shows the structure of MINSTREL's goal schema.

Goal Schema

Slots	*Meaning*
Type	Identifies the particular sub-type of goal.
Actor	Identifies the actor who has the goal. For author-level goals, this slot is filled with a symbol MINSTREL uses to refer to itself.
Object	The story scene or concept to which the goal is applied.
Priority	A number from 0-100 indicating the importance of the goal.

Figure 2.3 Goal Schema

As Figure 2.3 shows, MINSTREL's author-level goal schemas have a slot called "Type". Many of MINSTREL's schemas have "Type" slots. The Type slot is used to further specify a sub-type of the concept represented by the schema. In this case, the "Type" slot is used to identify the particular kind of goal represented. Figure 2.4 shows MINSTREL's specific author-level goals.

The Object slot in MINSTREL's author-level goals points to the representation of the story scene the author-level goal is being applied to. For example, an author-level goal to "build suspense in the scene in which Princess Jennifer is attacked by the dragon" is represented by a goal schema with the "Type" slot filled by "Add-Suspense-to-Scene" and the "Object" slot filled by the representation for the scene in which Princess Jennifer is attacked by a dragon. This is shown graphically in Figure 2.5. (&ACT.38 is another schema that represents the attack on Princess Jennifer by the dragon.) The plans to achieve MINSTREL's author-level goals are represented by a special structure called an Author-Level Plan, or ALP. Although MINSTREL's author-level

Sub-Type	*Meaning*
Tell-Story	Tell a story about the theme or reminding in the Object slot.
Check-Story-For-Suspense	Look over a story to see if there is any place where suspense might be heightened.
Check-Scene-For-Suspense	Check a scene to see if it might have its suspense heightened.
Add-Suspense-to-Scene	Try to heighten the suspense in a story scene.
Check-Story-For-Tragedy	Look over a story to see if there is any place where tragedy might be heightened.
Check-Scene-For-Tragedy	Check a scene to see if it might have its tragedy heightened.
Add-Tragedy-to-Scene	Try to heighten the tragedy in a story scene.
Check-Story-For-Foreshadowing	Look over a story to see if there is any place where foreshadowing might be added.
Check-Scene-For-Foreshadowing	Check a scene to see if it is suitable for foreshadowing.
Add-Foreshadowing-to-Scene	Foreshadow a scene.
Check-Story-For-Characterization	Look over a story to see if there are any characters that need further characterization.
Add-Characterization	Add additional characterization to a story.
Check-New-Scene	Check a new scene for various problems.
Check-Consistency	Check a scene for consistency problems.
Check-Preconds	Check an action to be sure that all the preconditions have been achieved.
Make-Goal-Consistent	Make a character goal consistent.
Check-Affects	Look over the story to make sure that characters have the proper emotional reactions to events.
Add-Story-Intros	Add introductory scenes to the story.
Add-Denouements	Add denouement scenes to the story.
Connect	Create the story scenes necessary to connect the Decision and Consequence parts of a story theme.
Instantiate	Fill in a story scene.

Figure 2.4 Sub-Types for Author-Level Goal Schemas

&GOAL.57

Actor:	&MINSTREL
Type:	&Add-Suspense-to-Scene
Object:	&ACT.38

Figure 2.5 Example Author-Level Goal Schema

plans could theoretically be represented as schemas, it is more convenient and efficient to represent them as small pieces of compiled LISP code. The representation of MINSTREL's author-level plans is discussed in more detail in Chapter 5.

In addition to author-level goals and plans, MINSTREL also has an author-level representation of the story it is telling. This representation is distinct from the character-level representation of the events of the story. The author-level representation of a story captures the author's knowledge of the story *as a story*. For MINSTREL, the author-level representation of the story includes a representation the theme of the story, and a representation of the structure of the story in terms of introductory scenes, the body of the story, and denouement scenes. Notice that this representation of the story, unlike the character-level representation, does not explicitly appear in the final story. Rather, this captures private knowledge the author has about the story he is telling.

Figure 2.6 shows the structure of MINSTREL's story schemas and planning advice theme (PAT) schemas. These are presented here primarily to alert the reader to some of MINSTREL's author-level representation. The structure of story schemas should be apparent. Themes, which are more complicated, are discussed in detail in Chapter 6.

Story Schema

Slots	*Meaning*
Introduction-Scenes	Introductory scenes which appear at the beginning of the story.
Body	The main body of the story, i.e., scenes that illustrate the theme of the story.
Denouement-Scenes	Denouement scenes which appear at the end of the story.

Theme Schema

Slots	*Meaning*
Value	Whether this theme is positive or negative advice.
Decision	The decision the planner of the theme makes.
Consequence	The consequence of the Decision.
Connection	How the Consequence is connected to the Decision.
Planner	The planner who made the Decision.
Active Goals	The planner's active goals at the time of the Decision.
Current Goal	The planner's current goal at the time of the Decision.
Current Plan	The planner's current plan at the time of the Decision.
World Facts	Any important facts about the world at the time of the Decision.

Figure 2.6 Author-Level Story Representations

2.2.3.2 Character-Level Concepts

To create a story, MINSTREL must also represent and manipulate concepts at the character level. These include character goals, plans and actions, character emotions, and character beliefs.

MINSTREL's character-level representation of goals is the same as the representation for author-level goals, as shown in Figure 2.1. It is necessary, however, to have a new set of goal sub-types to represent the kinds of goals that characters in the King Arthur domain have. MINSTREL's goal types are based upon a taxonomy of goals presented in [Schank and Abelson 1977]. This taxonomy is shown in Figure 2.7.

Goal Type	*Description*	*Examples*
Satisfaction Goal (S-Goal)	Recurring bodily desires	S-Hunger, S-Sleep.
Delta Goal (D-Goal)	Desired State Change	D-Location, D-Control
Enjoyment Goal (E-Goal)	Pleasurable activities	E-Travel, E-Entertainment
Achievement Goal (A-Goal)	Long-term attainment of social status	A-Skill, A-Status.
Preservation Goal (P-Goal)	Activated when status is threatened	P-Health, P-Possessions.
Crisis Goals (C-Goal)	Active preservation goals	C-Health, C-Possessions

Figure 2.7 Schank and Abelson's Goal Taxonomy [Schank and Abelson 1977]

Schank and Abelson's goal taxonomy was intended to represent a variety of common human goals while providing a structure in which to interpret those goals. Goal taxonomies based on Schank and Abelson's taxonomy have been used in a variety of projects (e.g., [Dyer 1983], [Kolodner 1984], [Mueller 1989], [Reisbeck and Schank 1989], [Reeves 1991]).

The Schank and Abelson taxonomy of goals is adequate to represent most of the goals which arise for characters in the King Arthur domain. However, one type of goal which cannot be represented using the taxonomy shown in Figure 2.7 is a *meta-goal*. Meta-goals are goals about other goals, such as the goal to do a favor for someone (literally, the goal to achieve another person's goal). These goals are useful for representing character interactions such as favors, deception and revenge. To represent these types of goals MINSTREL adds a new category to the taxonomy shown in Figure 2.7: Meta-Goals.

Figure 2.8 shows the specific goals used in MINSTREL and the categories they fall into. These goals are not intended to be an exhaustive listing of possible character goals in the King Arthur domain. They are simply the goals that arose during the development of MINSTREL.

Figure 2.9 shows an example of a character level goal. The schema in this figure represents

Class	*Goal Type*	*Description*
S-Goal	S-Hunger	Satisfy one's hunger.
D-Goal	D-Control	Control something.
	D-Loc	Change location.
	Destroy	Destroy some object or person.
	Scare	Cause fear in someone.
A-Goal	A-Status	Achieve status in society.
	A-Love	Find romantic love.
	A-Affection	Find brotherly affection.
C-Goal	C-Health	Health crisis.
Meta Goals	Favor	Do a favor for someone.
	Anti-Favor	Cause the failure of someone's goals.
	Retract	Retract a goal from being active.
	Deception	Cause someone to be deceived about something.

Figure 2.8 MINSTREL Character-Level Goal Types

"Lancelot wants to impress the King". (In actuality, both Lancelot and the King would be represented by human schemas with names like "&HUMAN.47". To make MINSTREL's representation more amenable to human understanding, meaningful names have been used in place of anonymous schemas wherever it does not introduce ambiguity or confusion.)

&GOAL.43

Actor:	&Lancelot
Type:	&A-STATUS
To:	&King-Arthur

Figure 2.9 Example Character-Level Goal Schema

To represent character-level actions, MINSTREL uses an act schema with slots as shown in Figure 2.10.

Like goal schemas, act schemas have a Type slot which identifies the precise type of action represented. MINSTREL's action types are based upon Conceptual Dependency (CD) theory ([Schank 1973][Schank 1975]). Conceptual Dependency is a representation for human actions in terms of a small set of primitive action types, as shown in Figure 2.11. Conceptual dependency can be used to represent a wide variety of common actions by using appropriate slot fillers. For instance, the act "Arthur hits Lancelot with a sword" could be represented by the act schema shown in Figure 2.12. By filling the slots of the act schema appropriately, "Arthur hits Lancelot with a sword" can be represented as "Arthur physically transfers a sword from somewhere to Lancelot." More complicated actions are represented by combinations and sequences of primitives. For example, a sword fight can be represented by a sequence of sword movements, character movements, bleeding (EXPELing blood) and so on. However, there is often no need for such detailed representation. It is more convenient to abstract a complicated sequence of events into a stereotype, and manipulate and reason about the stereotype, ignoring the underlying details. For

Act Schema

Slots	*Meaning*
Type	Identifies the particular sub-type of action.
Actor	Identifies the actor who performs the action.
Object	The thing or person being acted upon.
To	The direction the action is taking.
From	The origin of the action.
At	Where the action occurs.
Status	Whether the actor succeeded in performing the action.

Figure 2.10 Act Schema

Type	*Description*
ATRANS	Abstract transfer of possession.
ATTEND	Paying attention to something.
GRASP	Take physical possession of an object.
EXPEL	To expel objects from the body.
INGEST	To ingest something into the body.
MBUILD	Thought processes which create conceptualizations.
MTRANS	Transfer of mental information.
MOVE	Movement of a body part.
PROPEL	Application of physical force.
PTRANS	Transfer of physical location.
SPEAK	Vocalization.

Figure 2.11 Conceptual Dependency

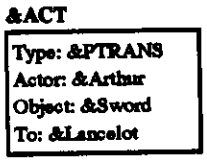


Figure 2.12 Example CD Representation

example, in the King Arthur domain it is sufficient in many cases to reason about a "sword-fight", without worrying about the details of swinging swords and character movements. Of course, if it is necessary to reason at a more detailed level, the composite stereotype can be expanded into the underlying primitives.

MINSTREL has two action stereotypes: M-Fight and M-Heal. M-Fight represents a fight between two characters; M-Heal represents one character healing another. (MINSTREL's implementation of composite stereotype schemas is based upon memory organization packets (MOPs) as used in [Schank 1982] and [Dyer 1983], hence the "M-" naming scheme.) M-Fight and M-Heal are used just as any action primitive. Figure 2.13 shows how M-Fight is used to represent Lancelot fighting Arthur using a sword.

&GOAL.21

Actor:	&Lancelot
Type:	&M-FIGHT
To:	&Richard
Object:	&Sword

Figure 2.13 Example Use of &M-FIGHT

Another type of concept which MINSTREL must represent in the stories it tells is a state of the world. A state is simply a representation of some fact that is true, such as "Lancelot is in the woods", "Jennifer loves Richard", or "Guinevere is dead." Figure 2.14 shows the slots of MINSTREL's state schema.

State Schema

Slots	*Meaning*
Type	Identifies the particular sub-type of state.
Object	The thing or person which is the subject of the state.
Value	The value of the state.
To	The direction of the state.

Figure 2.14 State Schema

Like goal and act schemas, state schemas have a "Type" slot that identifies a particular sub-type of state. The list of state sub-types and their meanings is shown in Figure 2.15.

Figure 2.16 shows how the state schema is used to represent the states "Lancelot is in the woods", "Jennifer loves Richard", and "Guinevere is dead." As this illustrates, the values and meanings of the slot fillers for state schemas vary depending upon the sub-type of state. Any meanings that are difficult to understand will be explained as they are introduced. (In particular, MINSTREL's representation for emotional states is discussed in detail in Chapter 9.6.) Another type of concept which MINSTREL must represent is a character belief. Character beliefs represent a state that a character believes is true, and the evidence he has to support (or undermine) that belief. Figure 2.17 shows the slots of MINSTREL's belief schema. MINSTREL uses belief

Type	*Description*
HEALTH	Health of something
POSSESS	Who possesses something
LOC	Location of something
EXIST	Existence of something
RAISE-GOAL	Actor raising a goal
AFFECT	An emotion (see 9.6)
KISS	Being kissed
SIBLINGS	Being siblings
KNOW	Knowing something
TEMPER	Character disposition
DATE	Identifies a time.
ROLE-CHANGE	Changing roles
BURIED	Being buried

Figure 2.15 State Sub-Types

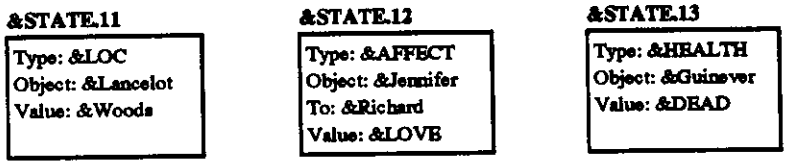


Figure 2.16 Example State Schemas

Belief Schema

Slots	*Meaning*
Type	Identifies the particular sub-type of state.
Object	The thing or person which is the subject of the state.
Value	The value of the state.
To	The direction of the state.

Figure 2.17 Belief Schema

schemas to represent deductive (a deduction from evidence about something that is true) and predictive (a prediction from evidence of something that is likely to happen) beliefs. How MINSTREL represents these kinds of beliefs and how MINSTREL represents evidence is discussed in detail in Chapter 7.5.3.

Finally, MINSTREL also has schemas to represent humans, monsters, animals and physical objects - all the objects in the King Arthur domain. The schemas for these objects and their slots

are shown in Figure 2.18.

Human

Slots	*Meaning*
Type	Identifies the character role in the King Arthur domain.
Name	The person's name.
Sex	The person's sex (&MALE, &FEMALE).

Monster

Slots	*Meaning*
Type	Identifies the type of monster (&DRAGON, &TROLL, &WYVERN).

Animal

Slots	*Meaning*
Type	Identifies the type of animal (&HORSE, &DOG).

Physical Object

Slots	*Meaning*
Type	Identifies the type of object (&BERRIES, &SWORD, &POTION).
Attributes	Additional attributes of an object (&MAGIC).

Figure 2.18 Various Object Schemas

As with all of MINSTREL's representation schemas, each type of object schema has a subtype. For humans, the subtype represents the role the character plays in the society of the King Arthur domain. The roles MINSTREL knows about are shown in Figure 2.19.

Type	*Description*
&KING	The King.
&KNIGHT	A knight of the Round Table.
&PRINCESS	A noble woman of the Court.
&HERMIT	A solitary religious figure, man or woman.
&PEASANT	A lowly serf.

Figure 2.19 Character Roles in King Arthur Domain

MINSTREL's use of roles captures some simple characterizations in the King Arthur domain. These roles, along with the name and sex of character, have proven sufficient for the stories that MINSTREL tells. In a more advanced storytelling system, or in a domain other than King Arthur, the storyteller would need a more in-depth representation of characters, which might include such things as a physical description, personality characteristics, egocentric goals, and so on.

2.2.3.3 Links

Links between schemas represent relationships. Examples of relationships that are typically represented by links are causal relationships: a state was caused by an act, an act is a plan to achieve a goal, and so on.

The table in Figure 2.20 illustrates the links MINSTREL uses when representing a story in the King Arthur domain.

		To Goal	Act	State
From	Goal	Subgoal Subsumes	Plan	
	Act			Intends Unintended Precond
	State	Motivates Achieves Thwarts		Supersedes Reaction

Figure 2.20 Character-Level Links

Each of these links has a "backlink" which represents the inverse relationship. For example, the backlink of "Motivates" is called "Motivated-By" and points from a goal back to the state that motivated the goal. Backlinks are maintained automatically by Rhapsody, the tool package which underlies MINSTREL.

The names of MINSTREL's links do not have any intrinsic meaning, but they were chosen to convey their usage to a human reader. Figure 2.21 describes in more detail the meaning and use of each link. These links are based on those in [Dyer 1983].

Name	From	To	Meaning	Example
Subgoal	Goal	Goal	Points from a goal to one of its sub-goals.	A sub-goal of killing a dragon is to be co-located with the dragon.
Subsumes	Goal	Goal	Points from a goal to another goal (usually of another character) that the goal subsumes.	Galahad's goal to do a favor for Jennifer subsumes Jennifer's goal of escaping the dragon.
Plan	Goal	Act	Points from a goal to one or more actions which make up a plan to achieve that goal.	Galahad's plan for killing a dragon is to fight it with a sword.
Intends	Act	State	Points from an act to a new state of the world that the act was intended to cause.	Galahad's fight with the dragon intends the dragon's death.
Unintended	Act	State	Points from an act to a new state of the world that the act caused unintentionally.	Galahad's fight with the dragon has the unintended result of Galahad's injury.
Precond	Act	State	Points from an act to a state of the world that enables the action.	Being co-located with the dragon is a precondition of fighting the dragon.
Motivates	State	Goal	Points from a state to a goal motivated by the state.	Jennifer being co-located with the dragon motivates her goal to be elsewhere.
Achieves	State	Goal	Points from a state to a goal achieved by the state.	The death of the dragon achieves Galahad's goal of doing a favor for Jennifer.
Thwarts	State	Goal	Points from a state to a goal thwarted by the state.	Lancelot possessing the magic sword thwarts Galahad's goal of possessing the magic sword.
Supersedes	State	State	Points from a state to a previous state that it supersedes.	Being located in the woods supersedes Galahad's earlier state of being located in the castle.
Reaction	State	State	Points from a state to another state triggered by the first state	Jennifer being dead makes Galahad feel sad.

Figure 2.21 Character-Level Links

2.3 Episodic Memory

MINSTREL is founded on the case-based model of problem-solving. In case-based problem-solving, the problem-solver finds solutions to the problems it faces by recalling similar past problems and using the solutions associated with those past problems. A fundamental element of the case-based paradigm is thus episodic memory - the autobiographical record of the events and experiences that make up a person's personal history ([Tulving 1972]). How are past cases stored? How are they recalled?

MINSTREL's episodic memory is based upon the "context plus index" model of episodic memory ([Reiser 1983][Kolodner 1984][Reiser and Black 1985][Reiser 1986]). In this model, episodes are organized according to their general contextual framework and their specific differences from other episodes in the same framework. In this section we will describe the content plus index model in more detail and briefly discuss MINSTREL's implementation of the model.

2.4 The Context-Plus-Index Model of Episodic Memory

One model of episodic memory is the "context-plus-index model" ([Reiser 1983], [Reiser, Black & Abelson 1985], [Reiser 1986]). This model explains recall as a two-step process.

In the first step, a *context* for recall is identified. The context is a generalized knowledge structure used to interpret a particular activity or episode. "Taking an Airplane Trip" is an example of a context. Knowledge about taking an airplane includes the kinds of preparations needed (packing a suitcase, making flight plans), events that might occur on the trip (eating a meal, losing luggage), typical objects (suitcases, airplanes, toothbrushes) and other general information necessary to successfully negotiate the trip. Since this knowledge is necessary to understand and encode trips as they are experienced, memories about trips becomes associated with the general "Taking an Airplane Trip" knowledge structure (the context).

Contexts are typically about activities rather than more abstract concepts like emotions or goal successes [Reiser 1986]. For this reason, if a subject is asked to recall an abstract concept (i.e., "Tell me about a time when you were sad"), the subject must find an appropriate context by reasoning about activities which might fit the cue (i.e., sadness is associated with funerals). Context-finding is illustrated in this example from [Reiser 1986]:

E: Think of a time when you felt impatient.

S: Felt impatient. Those are always tough ones to answer. Um, I'm trying to think of times I felt impatient...um...um...impatient always seems to mean you're waiting in line for something or waiting for something to happen... I can think of times when I felt frustrated waiting but not really impatient, and I think there's a difference. I remember waiting for someone who didn't show up for 4 hours, and it wasn't really that I was impatient, I was just frustrated with the fact that I didn't know whether this person was going to show up or not...

E: Was this waiting in line?

S: No, this was waiting to meet someone in front of a museum in Hartford...

Figure 2.22 Context-Finding Example

In this protocol, the subject is given a non-context specific cue. He then searches for likely contexts, using his knowledge about being impatient. First he tries the "Waiting In Line" context, and when that fails, tries "Waiting to Meet Someone", and eventually finds a memory that fits the criterion.

Once a context has been found, the second step in recall is to specify a set of features which will distinguish the target memory from all the other memories which share the same context. "Trips to a foreign land" is an example of a feature that will distinguish some of the episodes associated with the "Taking an Airplane Trip" context. To recall a particular episode, a set of features must be found that distinguish that episode from other episodes with the same context. Often the initial cue or set of features doesn't discriminate the target memory from other, similar memories, so new features must be generated in attempt to find a distinguishing feature. This process is called elaboration ([Norman & Bobrow 1979], [Williams & Hollan 1981], [Kolodner 1983]). An example of elaboration is shown in this protocol, also taken from [Reiser 1986]:

E: Think of a time you felt cold at an exam.

S: Cold at an exam...Wow, I keep think how most of my exam rooms have been really well heated! I remember being really hot in exams but never really cold, which I guess is pretty lucky. 'Cause most...I mean, at Christmas it's usually been really well heated.

E: What's going through your mind now...what can you think of?

S: Um, I was trying to start thinking through the exams I've taken at Christmas. Um, going through the different classes but they're not really coming...I think maybe my Bio exam freshman year was cold 'cause I had to walk all the way up to Science Hill to go take it, but the room was heated.

Figure 2.23 Elaboration Protocol

In this protocol, the examiner's question cues the "Taking an Exam" context. To fulfill the question, the subject begins hunting for features to add to this context which might locate an episode where the subject felt cold at an exam. The "exam at Christmas time" feature is added because it is likely to produce memories fitting the original question, since the subject knows that it is usually cold at Christmas time. When that feature alone proves insufficient, the subject begins walking through his class schedule, apparently beginning with his freshman year, and recalls an episode which almost fits the requirements.

The main features of the context-plus-index model of recall are (a) the use of a context, typically an activity, as a starting point for recall and (b) the use of features to find individual experiences related to the selected context. [Kolodner 1983] and [Schank 1982] extended the context-plus-index model by proposing specific mechanisms for organization and memory search that meet these behavioral characteristics.

2.5 The Kolodner Model of Episodic Memory

[Kolodner 1983] presented a process model for the context-plus-index model of memory. This model was used in a computer program called CYRUS which read news stories about Cyrus Vance and built an autobiographical memory of happenings in Cyrus Vance's life.

In the Kolodner model of episodic memory, episodes are organized in a difference tree. The episodes are stored at the leaves of this tree. When two episodes share some features a generalized episode, called an Episodic Memory Organization Packet (or E-MOP) is created, and the two episodes are then indexed underneath the new E-MOP.

E-MOPs and episodes are organized in a tree according to feature differences. Each link between E-MOPs represents a feature/value pair. At the leaves of the episodic memory tree are the individual episodes, also indexed by feature/value pairs. Figure 2.24 is an example of CYRUS's episodic memory organization after reading two stories about diplomatic meetings (EV1 and EV2).

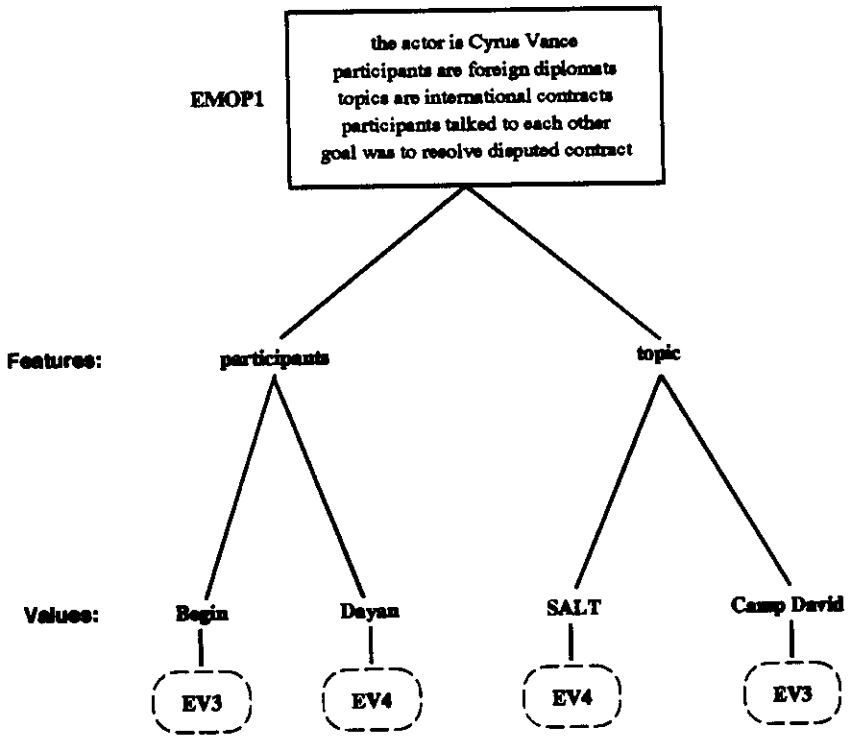


Figure 2.24 CYRUS Example

The top node of this organization is E-MOP1. E-MOP1 is a generalized episode of the "Diplomatic Meetings" context. This E-MOP represents the common features of the two episodes EV1 and EV2. Since EV1 and EV2 share no features, E-MOP1 simply represents the context, "Diplomatic Meetings".

Organized underneath E-MOP1 according to their differences are the two episodes EV1 and EV2. The links underneath E-MOP1 represent the differences between EV1/EV2 and E-MOP1. For example, in episode EV1, the "participants" feature has the value "Begin", so it is indexed underneath that feature/value pair.

Figure 2.25 shows a more complicated example, where E-MOPs and episodes are indexed at several levels within the episodic memory tree.

In Figure 2.25, three new episodes have been added to Cyrus's memory (EV1, EV2 and EV5). Since these new episodes shared some features with EV3 and EV4, new E-MOPs have been created which represent the common features of the episodes indexed below the E-MOPS (E-MOP2, E-MOP3 and E-MOP4). Some episodes are multiply indexed at various levels in the tree. EV3, for instance, is indexed under the "topic is Jerusalem" and under "participant is Begin and topic is Jerusalem".

2.5.1 Recall of Episodes

In the Kolodner model, episodes are recalled from memory by specifying a set of *target features.* These features are then used to search the episodic memory tree.

For example, if episodic memory were as shown in Figure 2.25, then the target feature "participant is Gromyko" would result in the recall of EV2. Starting at E-MOP1, the target feature would be used to traverse the tree downward until an episode is found. In this case, the feature "participant is Gromyko" leads directly to EV2. If the target features were "participant is Begin and topic is Jerusalem", then the search would traverse two levels of the tree before finding EV3.

In some cases, the target features may not be sufficient to recall an episode. For example, the target feature "participant is Thatcher" leads to no recall because there is no episode with that feature. Recall may also be stymied when the target features lead only to an E-MOP. For example, the target feature "participant is Begin" leads only to E-MOP2. To recall an episode, new features must be generated in hopes of continuing the traversal of the memory tree. This is called elaboration, and is discussed more fully in [Kolodner 1983].

2.5.2 The Distinction Criterion

The Kolodner model organizes memory according to the Distinction Criterion: *Episodes are indexed by features which distinguish them from other episodes.* A feature (or set of features) which does not distinguish a single episode will recall only a generalization (i.e., an E-MOP). In List of Figures, for example, the feature "participant is Begin" will not recall a particular episode because it is not sufficient to distinguish a single episode.

The Distinction Criterion has three important features:

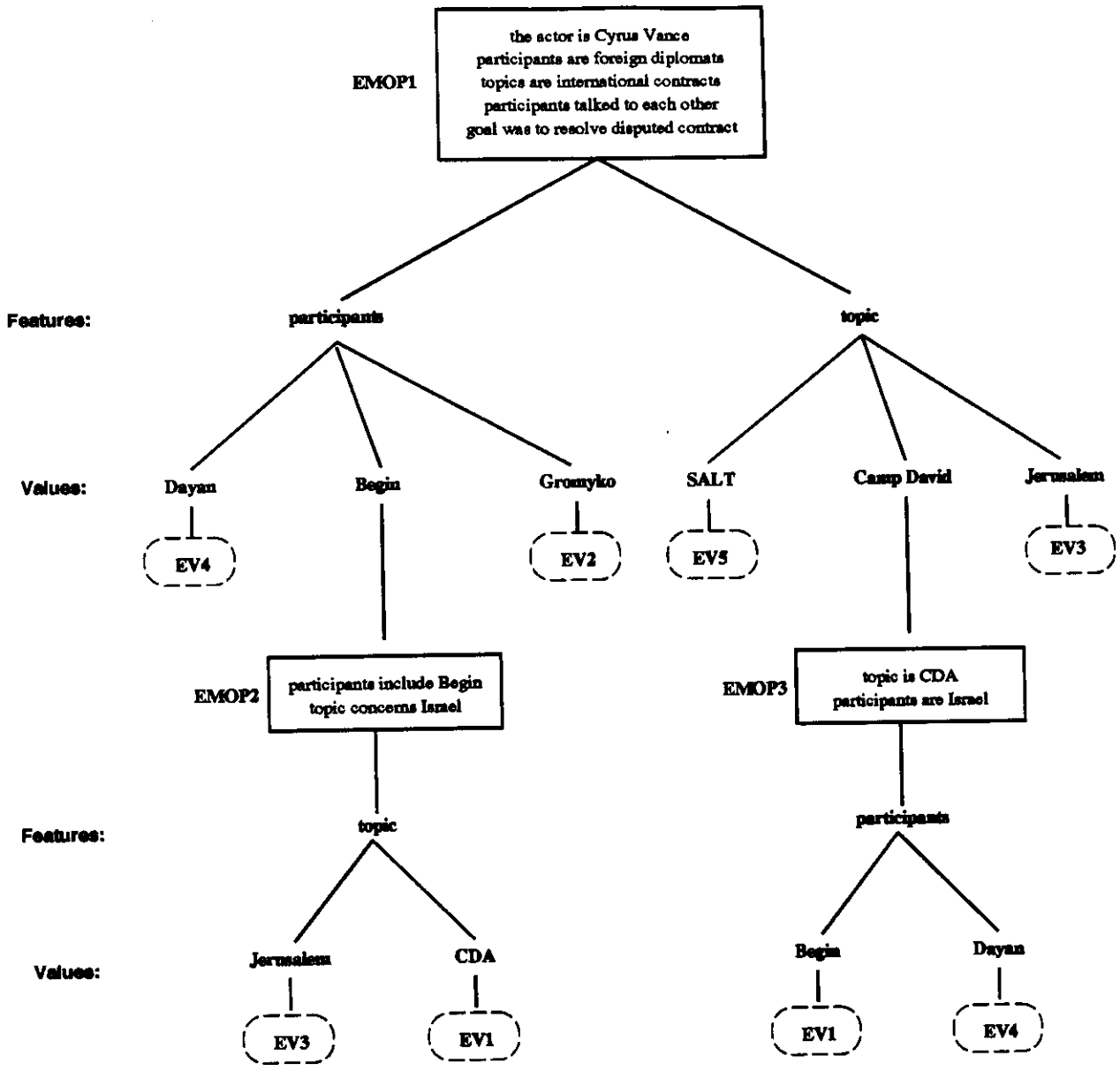


Figure 2.25 Complex CYRUS Example

1. The Distinction Criterion explains a behavioral feature of episodic memory. Humans cannot use features for recall unless the features index only a small number of episodes [Reiser 1986]. The Distinction Criterion organizes episodic memory to reflect this behavior.

2. The Distinction Criterion explains how semantic memory can be created from episodic memories. A feature which indexes a number of episodes represents general or semantic knowledge about the context, and this information is captured in a semantic knowledge

structure (an E-Mop) by generalizing the episodic memories indexed by the feature.

3. The Distinction Criterion explains how a content-independent mechanism can be used to organize episodes. The Distinction criterion discovers distinguishing features of episodes by their frequency in indexing. This process is independent of the content of the episodes being indexed, and explains how a simple but powerful mechanism can build an organizational structure for many different types of concepts.

2.6 The MINSTREL Model of Episodic Memory

The MINSTREL model of episodic memory is based upon the Kolodner model. The MINSTREL model differs from the Kolodner model in two ways.

First, MINSTREL's episodic memory adds a second organizing principle, called the "Utility Criterion". The utility criterion organizes memory according to the usefulness of the indices, and provides a mechanism for feature selection.

Second, MINSTREL does not implement all the features of the Kolodner model. Since MINSTREL is not intended as a model of learning or case-based explanation, MINSTREL's episodic memory does not build E-Mops as new episodes are indexed. MINSTREL also automates the elaboration strategies that were heuristic-based in the Kolodner model. Neither of these differences changes the overall behavior of episodic memory.

2.6.1 Utility Criterion

One aspect of episodic memory which the Kolodner model and the Distinction Criterion do not address is the issue of *feature selection.* Contexts such as "Taking an Airplane Trip", "Diplomatic Meetings" and "Jousting" have many possible features. There are numerous sensory inputs, stray thoughts, odd smells and so on, all which *could* be features. If the Distinction Criterion was the only organizing principle of episodic memory, we would expect that *any* of these features might be remembered and used as an index, regardless of whether or not that feature had any relevance to the context.

However, this is not the case. Irrelevant features do not become part of a context. Nor are they used to organize episodic memory. In the case of the "Taking an Airplane Trip" context, people typically do not remember the color of the carpeting in the airplane, the number of channels on the in-flight audio, or the billboards viewed on the way to the airport, even though these features might well distinguish one "Taking an Airplane Trip" episode from another.

Instead, people remember features that are relevant or useful to a particular context. Thus, the "airline flown", "number of layovers" and "amount of luggage" features are remembered because they bear important roles in the planning and executing of an airplane trip. On the other hand, "color of the airplane carpeting" has no bearing on the planning and execution of an air-

plane trip, and so is neither remembered nor used as an organizational index.

There is, therefore, be a second organizing principle of episodic memory which eliminates indices which distinguish episodes but which are not relevant to the context. This principle is the Utility Criterion:

Indices are selected according to their utility in the current context.

2.6.2 A Process Model of the Utility Criterion

As shown earlier, the Distinction Criterion acts during the *indexing* of episodic memories. As new episodes are added to memory, features which no longer distinguish episodes are eliminated as organizational indices. In contrast to this, the Utility Criterion acts as a frequency-based selector during recall:

The utility of a feature can be measured by how often it is specified as a target feature for recall.

A feature which is often specified as a target for recall will have a high utility value. A feature which is rarely or never used for recall will have a low utility value. Thus, the utility of a feature is determined by the processes that use episodic memory.

This measure of utility can then be used to organize episodic memory. Features which have a low utility value can be eliminated as organizational indices in the episodic memory tree.

As an example, suppose an author who knows little or nothing about medieval warfare decides to write a story about King Arthur. To learn about medieval hand-to-hand combat, he watches films about King Arthur and attends the Renaissance Fair where he views some jousts. Initially, the author organizes these episodes according to many features, because he does not yet know which features are useful in his problem domain - writing King Arthur stories. Figure 2.26 shows the (simplified) organization of the author's memory after he has seen three jousts. His episodic memory contains three episodes (EV1, EV2, and EV3) organized according to the type of weapon used, whether or not the knight rode a horse, and what color the knight's armor was. (Maybe he has the mistaken impression that the "good" knight always wears white and always wins the joust.)

To this point, episodic memory has been organized by the Distinction Criterion because only indexing has occurred. Now the author begins to use his memory for recall. He writes several short stories about King Arthur. In creating, planning and reasoning about fights, the author is often using the "weapon" and "horsed?" features, because they are relevant to determining the outcome of the fight. He never uses the "color of the armor" index in reasoning about fights because it has no relevance in that domain.

As the author continues to use his episodic memory to create and reason about fight scenes, the

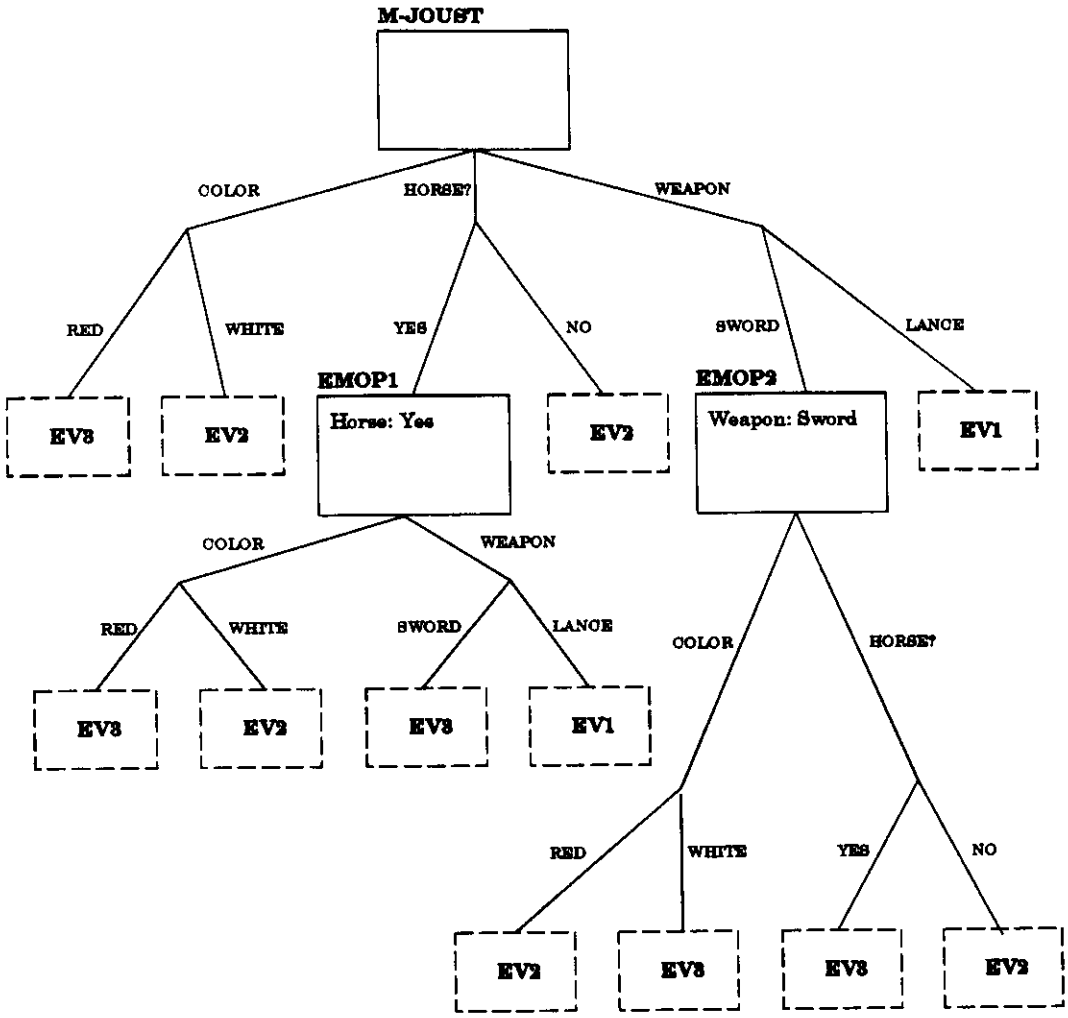


Figure 2.26 New Author Example

Utility Criterion reinforces the "weapon" and "horsed?" features and weakens the "color of the armor" feature. Eventually the Utility Criterion eliminates the "color of the armor" feature as an organizational index. The resulting episodic memory is shown in Figure 2.27.

The feature "color of the armor" has been eliminated as an organizational index because it is rarely used as an index for recall. The use of episodic memory has determined that "color of the armor" isn't relevant to the "Joust" context *as used by the author.*

The determination of the utility of features is dependent upon the recall processes. Episodic memory adjusts its organization to the tasks for which it is used. If the author had used his memory for another task, the Utility Criterion might have selected different features as relevant. For

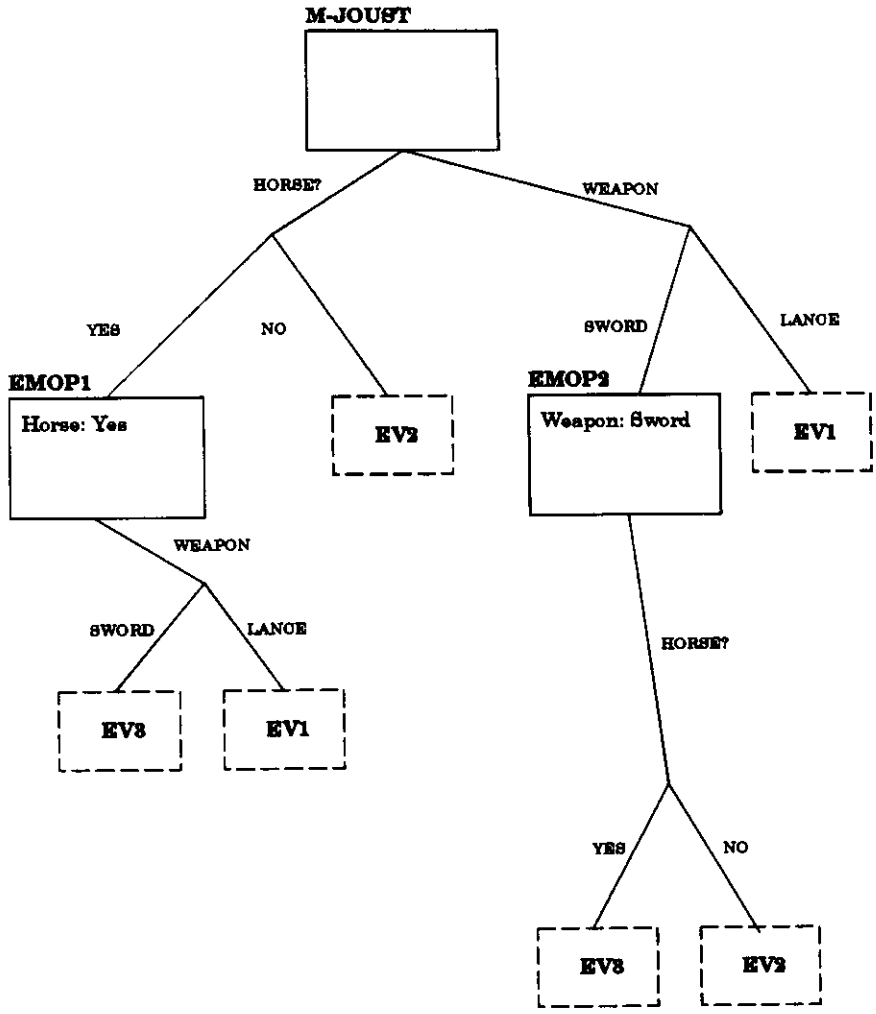


Figure 2.27 Revised Author Memory

example, in social situations the color of a knight's armor is more relevant, because it can indicate his social standing.

The Utility Criterion has three important features:

1. The Utility Criterion explains a behavioral feature of episodic memory. Humans cannot recall memories based on irrelevant features. The Utility Criterion organizes episodic memory to reflect this behavior.

2. The Utility Criterion explains how feature selection occurs. For any experience, there are a large number of potential features. This set of features can be reduced to a small set of relevant features by focussing on the features used during recall.

3. The Utility Criterion explains how a content-independent mechanism can be used to select features and organize episodic memory. The Utility Criterion discovers relevant features of episodes by their frequency in recall. This process is independent of the content of the episodes being recalled, and explains how a simple but powerful mechanism can build an organizational structure for many different types of concepts, since it relies on the knowledge of other processes (the recall processes) to guide the organization of episodic memory.

2.7 Summary of Memory Model

Episodic memory is organized according to two criterion: the Distinction Criterion and the Utility Criterion. The Distinction criterion selects indices that *distinguish* between episodes. The Utility Criterion selects indices that are *relevant* to the context. Together the Distinction and Utility Criteria create an organization of episodic memory that emphasizes features with are *useful* and *distinctive*.

The importance of this organization of episodic memory will become apparent during the discussion of creativity in the next chapter. Creative solutions are both *useful* and *novel* - features corresponding to the organization of episodic memory. An episodic memory that is organized according to the Distinction and Utility Criteria can therefore be used as the basis for both judging creativity and being creative.

CHAPTER 3
A Model of Creativity

3.1 Introduction

Creativity - the bringing forth of an original product of the human mind - is the pinnacle of human cognition. Artists, writers and scientists are treated with intellectual awe, and the word itself brings to mind the roll of men whose discoveries changed their social and scientific cultures: William Shakespeare, Albert Einstein, Leonardo da Vinci, Thomas Edison.

And yet creativity has a mundane side as well. From small child to adult, we are all creative in every aspect of our lives. Faced with a problem we have never before encountered, we combine past knowledge in new ways to create a solution. We fix our cars using hangers and baling wire, invent jokes based on the latest domestic crisis, and make up bedtime stories for our sons and daughters. Far from being the sole province of extraordinary thinkers, the ability to create new solutions to problems is one of the cornerstones of human problem-solving.

To understand human cognition, it is essential that we understand the processes of creativity: the goals that drive people to create and the mechanisms they use to create. And because creativity spans the intellectual spectrum from the highest peaks to everyday cognition, it is likely that studying creativity will provide special insights into human thought.

Understanding creativity also has practical value to computer scientists. Current computer programs have little adaptability. Faced with new situations, they act incorrectly or fail completely. Implementing the fundamental processes of creativity in computer programs has the potential to greatly increase the scope and power of the modern computer.

This chapter presents a cognitive model of creativity that addresses these issues. A case-based problem-solver is augmented with evaluations that act as a creative drive, and a set of creativity heuristics called Transform-Recall-Adapt Methods (TRAMs). TRAMs create new solutions to problems by transforming problems into slightly different problems, and adapting any solutions found to the original problem. Repeated application of TRAMs enables the problem-solver to make elaborate problem transformations by small steps, invent new solutions substantially different from old ones, and greatly extend the range of problems that can be solved.

This model has been implemented in a computer program called MINSTREL. MINSTREL tells stories about King Arthur and his Knights of the Round Table. MINSTREL uses creative problem-solving to flesh out story scenes, create new story themes, and to invent new actions, plans and goals for characters. Unlike previous work in storytelling [Meehan 1976][Lebowitz 1985]. MINSTREL focuses on the creative aspect of writing. Consequently, the creative problem-solving portion of MINSTREL can be applied independently to other domains and problem tasks.

3.2 Creativity and Problem-Solving

The goal of this research is to develop a model of creativity and answer the question: What are the processes of creativity common to many problem-solving domains? The first step to answering this question is to understand what it means to call the solution to a problem "creative."

According to psychologists, people recognize a solution as creative if it (1) has significant novelty and (2) is useful [Weisberg 1986][Koestler 1964][Kris 1953][Wallas 1926].

We all recognize that creative solutions must be original. They must be new and different from old solutions. But the differences must also be *significant*. If an artist were to paint the Mona Lisa in a red dress instead of a blue one, the resulting painting would not be considered creative, despite its differences from the original. Significant novelty distinguishes creative solutions from ones which are only adaptations of old solutions.

Usefulness is the second characteristic of a creative solution. We expect problem-solvers to be capable: they must develop solutions which solve their problems. Replacing a flat tire with an air raft is novel but not creative, because it doesn't effectively solve the original problem. Creativity must be purposeful and directed. Creative solutions must have bearing and utility on the problems to which they are applied.

Creative solutions are, therefore, both useful and significantly novel. How are these features reflected in the creative process? Consider the following anecdote concerning the seven year old niece of the author:

> One day, while visiting her grandparents, Janelle was seated alone at the dining room table, drinking milk and eating cookies. Reaching for the cookies, she accidently spilled her milk on the table. Since Janelle had been recently reprimanded for making a mess, she decided to clean up the spill 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.
>
> She returned a few minutes later carrying a kitten. The neighbor's cat had given birth to a litter about a month ago, and Janelle 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.

Most people find Janelle's solution to her problem creative. The use of a kitten as an agent and the substitution of "consumption of milk" for "removal of milk" are significant differences from Janelle's known solution to the "spilled milk problem," and the success of the new solution shows its usefulness. Janelle's example illustrates three important principles of creativity:

(1) The failure of problem-solving motivated Janelle to be creative. When Janelle could not find

a towel, her old plans for cleaning up a spill became unusable, forcing her to invent a new solution. The need for creativity arises from the failure of problem-solving [Weisberg 1986].

Creativity is driven by the failure of problem-solving.

(2) Creativity is an integral part of the problem-solving process. Janelle's solution to the spilled milk problem arises as part of her problem-solving process, and even the creative aspects of her solution - using a kitten as an agent and substituting consumption of the milk for removal of the milk - are broken down into sub-tasks in the manner of a classical problem-solving strategy (divide and conquer). Although some theorists (i.e., [Wallace 1926][Koestler 1964]) have suggested that creativity is a process fundamentally different from problem-solving, there is ample evidence to indicate that creativity is an outgrowth or extension of problem-solving [Weisberg 1986].

Creativity is an extension of problem-solving.

(3) New solutions are formed by combining old knowledge in new ways. Creative solutions do not spring forth newborn from the head of Zeus; they make use of what the problem-solver already knows. Janelle knew that the neighbors had kittens, that kittens like milk, and that the goal "consumption of milk" could subsume the goal of "removal of milk", and she combined this information into a new solution by using her general knowledge about agents. The significant novelty of creative solutions arises from the problem-solvers application of knowledge in a new way to the problem [Koestler 1964][Weisberg 1986].

New solutions are created by using old knowledge in new ways.

The model of creativity presented in this chapter shows how these three principles are incorporated into a model of case-based problem-solving.

3.3 Case-Based Models of Problem-Solving

MINSTREL's model of creativity is built on a case-based model of problem-solving [Hammond 1988][Kolodner 1987][Schank 1987].

Case-based reasoning systems are driven by an episodic memory of past cases rather than a base of inference rules or knowledge-intensive heuristics. Given a problem, a case-based problem solver (1) *recalls* a past problem with the same features and its associated solution, (2) *adapts* the past solution to the current problem, and then (3) *assesses* the result. To drive home from work, a case-based problem solver recalls a previous time he drove home and the route he used, modifies the route in light of changing road conditions (perhaps a particular street is closed for repair) and then assesses that route according to his general knowledge about driving.

This model is illustrated in Figure 3.1. Problem specifications are input at the left side, where they are used to recall similar past problems. The solutions from these past problems are then adapted to the current problem. Finally, the adapted solutions are assessed to determine if they

are useful and meet other domain-specific considerations (i.e., a mechanical engineer might want to create efficient solutions). (A more complete review of case-based reasoning can be found in [Slade 1991]).

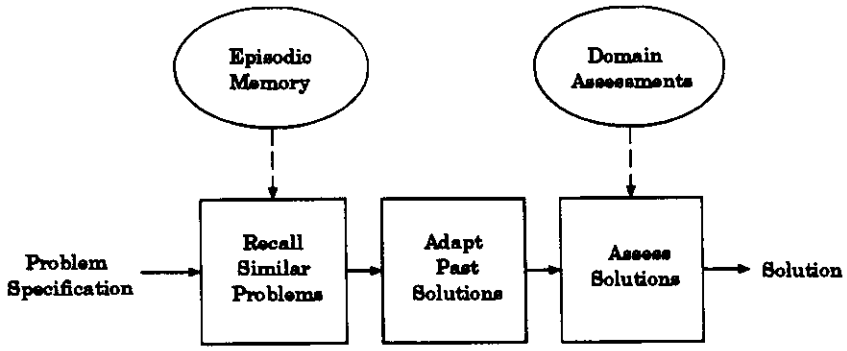


Figure 3.1 A Case-Based Model of Problem-Solving

Case-based problem-solving has several benefits. First, case-based problem-solving is similar to the reasoning done by human experts [Riesbeck 1989]. Expertise depends upon experience, and case-based reasoning captures this connection.

Second, case-based problem-solving explains how a problem-solver can use new experiences to extend his knowledge. Planning and storytelling systems which capture problem-solving knowledge as static rules (i.e., [Warren 1978][Meehan 1976][Lebowitz 1985]) require additional mechanisms to explain how that knowledge can be extended.

A final advantage of case-based problem-solving is efficiency at solving routine problems. By recalling a past problem which shares all the features of the current problem, the problem-solver is assured (1) that the solution from the recalled problem will apply to the current problem, (2) that adapting the recalled solution to the current problem will be simple, (3) that little or no assessment of the adapted solution will be necessary and (4) that the recalled solution is very likely to work. Re-using past solutions under identical circumstances requires little effort and results in a high degree of success.

However, case-based problem-solving has an obvious shortcoming: it fails when faced with a new problem. If a similar past problem cannot be recalled, the case-based problem-solver has no way to discover, build, or create a new solution. Using old solutions over and over again is acceptable and efficient in many domains. It would be tiresome, for instance, to create a new route home from work each day. But when no old solution is known, a problem-solver must be able to create a new solution.

Furthermore, case-based problem-solving has difficulty explaining domains such as art and literature. In these domains, problem-solvers often reject solutions simply because they are old or repetitive.

To capture the creative process, case-based problem-solving must be extended to include (1) a creative drive and (2) processes which can combine old knowledge in new ways to create problem solutions.

3.4 Failure-Driven Creativity

People are conservative problem solvers. They expend the effort to create new solutions when familiar, known solutions fail [Weisberg 1986]. Janelle created a new solution to the spilled milk problem only because the lack of a towel prevented her from applying an old plan.

Old solutions may fail for a number of reasons. They may not solve the problem, or they may solve the problem inefficiently. Or they may be rejected simply because they are old, as in literature and art, where a premium is placed on new ideas. Whatever the cause, it is the failure of old solutions that drives the problem-solver to create a new solution. A model of creative problem-solving needs a mechanism for detecting planning failures and using that to drive the creative process.

Case-based problem-solving can fail at each of the three steps shown in Figure 3.1: (1) if a past problem situation similar to the current problem situation cannot be recalled, (2) if a recalled solution cannot be adapted to the current problem, or (3) if the adapted solution fails a domain assessment. In the MINSTREL model of creativity, a failure at any of these steps causes MINSTREL to attempt to create a new solution to the problem. Like Janelle, MINSTREL is creative when it encounters a problem for which its past solutions fail.

Because MINSTREL is a model of storytelling as well as creativity, MINSTREL also implements an artistic drive to create. Unlike their counterparts in traditional problem-solving domains such as engineering, artists sometimes reject solutions even when problem-solving is successful. Artists create for the sake of creation; they find repetition of solutions boring and unacceptable. (In fact, all human problem-solvers get bored with repetitive solutions, but particular emphasis is placed on this motivation in the arts.) MINSTREL implements the artistic drive to create as a *boredom assessement.*

The boredom assessment operates during the Assess step of problem-solving (see Figure 3.1). The boredom assessment examines proposed problem solutions to determine if they've been used too many times previously, i.e., have become boring. To determine this, MINSTREL uses episodic memory.

Episodic memory is the autobiographical record of the events and experiences that make up a person's personal history [Tulving 1972][Cohen 1989]. MINSTREL uses a model of episodic memory based on work by Schank [Schank 1982] and Kolodner [Kolodner 1984], and elaborated and tested by Reiser, Black and Abelson [Reiser 1983][Reiser, Black & Abelson 1985][Reiser 1986], who term it the "context plus index" model. In this model, episodes are organized according to their distinctive differences. Two episodes with significant differences fall into different memory categories and will not be recalled together. Episodes with no significant differences fall into the same memory category and are recalled as a group.

To determine if a solution has become boring, MINSTREL indexes the solution in episodic memory and counts how many times similar solutions have been used. If the solution has been used more than a small number[1] of times, it is judged boring and rejected.

To illustrate this process, suppose that MINSTREL is building a story scene in which a knight's life is endangered. MINSTREL's episodic memory contains scenes about King Arthur and his Knights. Some of these scenes are initially seeded into MINSTREL's memory, as if MINSTREL had read several short stories about King Arthur. Others are from stories MINSTREL invented during earlier storytelling sessions. To solve the "build a scene in which a knight is endangered" problem, MINSTREL recalls a similar scene from a previous story. In the recalled scene, a knight fights a dragon. After adapting the recalled scene to the constraints of the current story, MINSTREL compares the new scene to episodic memory. Since similar scenes have only been used once before, the new scene passes the boredom assessment. The new scene is added to the current story and to episodic memory.

Now episodic memory contains two similar scenes in which a knight fights a dragon. The next time MINSTREL creates a scene in which a knight fights a dragon (perhaps again to solve the "build a scene in which a knight is endangered" problem), the boredom assessment will reject the scene. "Knights fighting dragons" has become boring, and MINSTREL will be driven to create a new way in which a knight can be endangered, *even though problem-solving succeeded in finding a useful solution to the original problem.*

As MINSTREL tells stories and indexes them in memory, MINSTREL's storytelling behavior changes. MINSTREL may tell stories about knights fighting dragons for a while, but these soon become boring and MINSTREL moves on to other topics. Thus, the boredom assessment models the artistic drive to create. MINSTREL, like any human faced with a repetitive task, becomes bored with the "known" solutions, and is driven to create new solutions. And because of its creativity heuristics, MINSTREL can invent new solutions when it becomes bored with old ones.

MINSTREL's boredom heuristic is simple. It ignores the type of problem being solved and other constraints that might realisticly affect an artist's determination of whether a concept or solution should be reused. A storyteller, for example, will not want to create new solutions for every character action in a story. How a knight gets from place to place is probably unimportant, and any known solution will do, no matter how frequently it has been used in the past. And in other domains there will be similar considerations.

For storytelling, MINSTREL only applies the boredom heuristic when creating the story events that illustrate the plot of the story. The plot, which is the sequence of story events that illustrate the theme or most important point of the story, is the most important element of the story, and so MINSTREL strives to be creative when building the plot. Other elements of the story, such as events that were added to keep the story causally consistent, are of less importance, and so MINSTREL accepts non-creative solutions for these problems.

1. Currently, MINSTREL considers a solution boring if it has been used twice previously. The threshold can be set to other values by the user.

For any particular artistic domain, then, there will be other considerations in determining whether a concept is "boring". But the basic consideration will remain the novelty of the concept: how often that concept or a similar[2] one has been used in the past. The basic process behind MINSTREL's boredom heuristic - using episodic memory to determine the novelty of an idea - supports this basic need and has the flexibility to be augmented into a more complete and realistic boredom assessment.

3.5 The Challenges of Creativity

By what processes are new solutions created? As Janelle's example illustrated, people invent new solutions by (1) recalling knowledge not part of the known solutions to the problem and (2) adapting that knowledge to create a solution [Weisberg 1986]. Four factors make this a difficult task.

First, the obvious source of knowledge about the current problem - similar past problems - has been exhausted. The creative process begins when problem-solving fails; the solutions indexed under the current problem have already been tried and rejected. To discover a new solution, the problem-solver must find knowledge *not* indexed under the current problem. The difficulty is knowing where in the space of episodic memories to seek knowledge. Knowledge grabbed willy-nilly from memory will be unlikely to apply to the current problem.

New solutions require knowledge not indexed under the current problem.

Second, the problem-solver may not have any incorrect or partial solutions on which to base his problem-solving efforts. If the current problem is sufficiently different from past problems, then the problem-solver may draw a complete blank. He may be unable to recall any past problem similar to the current problem. In this case, the problem-solver cannot base his creative efforts on an old, incorrect solution, because he does not have one. He can, however, base his creativity on the problem description, because that will always be available. In general, the problem-solver must find new knowledge by searching the *problem space*. The problem-solver must change the current problem into some new problem, in hopes of finding useful knowledge in the recalled solutions to the new problem. This casts the problem of creativity in a new light: the first step to creativity is changing the problem, not the solution.

Creativity involves recasting the problem.

Third, the complexity of adapting the discovered knowledge may overwhelm the creative effort. If Janelle had attempted to adapt her plan for the "getting to school" problem to the "spilled milk" problem, the difficulty of the adaptation task would be insurmountable. Trying to solve a difficult problem by substituting an impossible problem is a poor strategy. Somehow the creative process must limit the complexity of the adaptation process.

---

2. MINSTREL uses episodic memory to judge the similarity of solutions: two solutions which fall into the same category in episodic memory are considered similar. Chapter 2 discusses the organization and use of episodic memory in more detail.

Adaptation must avoid too much complexity.

Finally, the creative process must be capable of discovering solutions substantially different from the original solution. There is a need to limit the creative process to simplify the search for new knowledge and the adaptation problem, but there is a conflicting need to find the knowledge needed to created an original solution. Some mechanism must exist that will enable to the creative problem-solver to find necessary knowledge, even when it is conceptually distant from the original problem.

Creativity must be capable of "leaps" to new problem solutions.

The challenge of creativity research is to find a cognitive mechanism that is not crippled by these requirements. The creative process must be able to (1) find useful knowledge by searching the problem space, (2) limit the adaptation task, and (3) discover solutions substantially different from the original solution.

3.6 MINSTREL's Creativity Heuristics

The MINSTREL model of creativity is based on creativity heuristics that associate problem transformations with specific solution adaptations. To search the problem space for possible new solutions, MINSTREL begins at the original problem and applies small transformations that create new, slightly different problems. If one of these new problems can be solved, the associated solution adaptation can be used to create a new solution to the original problem.

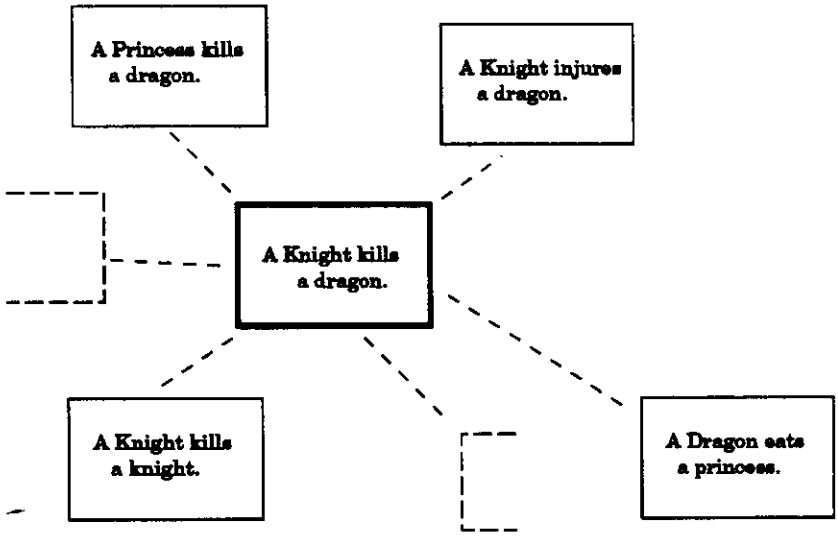


Figure 3.2 Search Space of Slightly Different Problems

Suppose, for example, that MINSTREL is trying to invent a method for a knight to kill a dragon. Figure 3.2 illustrates the search space for this problem. At the center is the original problem, and

around it are similar problems. To discover a new solution to the original problem, MINSTREL applies a problem transformation, jumping from the original problems to one of the nearby problems. MINSTREL now tries to solve this new problem. If MINSTREL is successful (by recalling a solution from episodic memory), it then tries to adapt the recalled solution to the original problem, jumping back to the center of the search space. The result is a new solution to the original problem.

The process of "Transform-Recall-Adapt" is the heart of MINSTREL's model of creative problem-solving. In the Transform step, MINSTREL takes a problem description and changes it into a slightly different problem. In the Recall step, MINSTREL takes the new problem description and tries to recall similar past situations from episodic memory, and the solutions used in those situations. In the Adapt step, MINSTREL takes the recalled solutions and adapts them to the original problem. The Transform step *searches* the problem space, the Recall step *finds* new knowledge, and the Adapt step *applies* that knowledge to the current problem.

MINSTREL implements this process in creativity heuristics called Transform-Recall-Adapt Methods, or TRAMs. Each TRAM consists of a transformation to be applied to a problem and a specific adaptation that will apply recalled solutions to the original problem. Figure 3.3 shows an example of one of MINSTREL's TRAMS, TRAM:Cross-Domain-Solution[3]. This TRAM suggests that a new solution to a problem can be found by (1) translating the problem into a new domain [Transform], (2) solving the problem in that domain [Recall], and (3) translating the solution back into the original domain [Adapt].

TRAM:Cross-Domain-Solution

Transform Strategy

Find a new problem domain with similar actions and actors, and map the original problem into this new domain. Retain the mapping for use in the Adapt step.

Adapt Strategy

Map any discovered solutions back to the original domain by reversing the mapping used in the Adapt step.

Figure 3.3 TRAM:Cross-Domain-Solution

---

3. A more specific description of TRAM:Cross-Domain-Solution is given in Chapter 4.

MINSTREL uses TRAM:Cross-Domain-Solution to create the following story scene (from "Richard and Lancelot"):

> 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-Solution transforms this specification by mapping it into the modern domain. Elements of the original problem specification are mapped to corresponding elements in the new domain. "A knight" becomes "a businessman" and "a princess" becomes "a friend". The new problem specification is "a businessman accidently meets a friend" 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.

This story is adapted to the original problem by mapping the story back into the King Arthur domain, creating the scene in which Lancelot's horse leads him to Andrea. The resulting scene is creative - novel and useful - because TRAM:Cross-Domain enabled the problem-solver to (1) discover knowledge previously unconnected to the original problem and (2) apply that knowledge to create a new solution.

3.7 The Scope of Creativity Heuristics

TRAMs find new solutions to problems by making *small* changes in the problem description and corresponding small adaptations to any discovered solutions. In light of the definition of creativity as a solution with a "substantial" difference from past solutions, this may seem counter-intuitive. Why not use creativity heuristics that make large changes to the problem descriptions? There are several reasons, which are founded in our current understanding of human cognition.

First, MINSTREL is an integrated model of problem-solving and creativity, in which creativity is an extension of problem-solving. In most problem-solving situations, there is no need for powerful creativity. Most problems are solved using standard solutions or past solutions that have been only slightly adapted. It is only in rare cases that a problem-solver must invent a solution substantially different from a past solution.

Consequently we expect creativity heuristics to be very efficient at discovering small adaptations while still capable of larger adaptations. By using heuristics that make small adaptations, MINSTREL is efficient at the types of simple problem-solving and creativity that make up the bulk of

problem-solving situations. (The question of whether MINSTREL is capable of larger adaptations is discussed below.) But there are other reasons to use small adaptations as a basis for creativity.

One advantage of creativity heuristics that make only small changes to a problem description is that they are more likely to find useful knowledge. Slightly different problems are good sources of useful knowledge because they share many of the same constraints as the original problem, and their solutions are likely to have some applicability to the original problem. Examining slightly different problems constrains the search task to an area of the problem space that is localized and fertile.

A second advantage is that small adaptations are easier and more likely to be successful than large adaptations. Adapting a very different solution to a new problem (i.e., adapting the "getting to school" solution to the "spilled milk" problem) requires a great deal of knowledge and effort and is likely to fail no matter what the expenditure. Adapting a solution that has only small differences (i.e., adapting the "spilled juice" solution to the "spilled milk" problem) requires less knowledge and effort and is more likely to succeed.

So there are several reasons to use small adaptations. But are small adaptations capable of discovering more creative solutions?

Sometimes even simple heuristics are capable of discovering unique solutions. Consider the heuristic to "substitute an agent" which Janelle used to create a solution involving a kitten. Although the heuristic itself is simple and commonplace, in this case the solution it discovers is surprisingly creative.

But even if creativity heuristics taken singly are not capable of major discoveries, they can be when taken *in combination*. The strategy of small problem transformations tends to find solutions with small differences from the original problem. But if a single heuristic does not find a new solution, transformations can be repeatedly applied until substantially different solutions are discovered. And because the adaptation of the created solution is done in many small, simple steps, the process of adaptation remains simple.

Leaps in creativity result from combinations of small modifications.

Thus, MINSTREL's TRAMs provide a spectrum of creativity. They permit the problem-solver to quickly and efficiently solve the majority of problems that do not require creativity or require only simple adaptation of past solutions. At the same time, they provide the capability of inventing unique and substantially different problem solutions by using several heuristics in combination. And because each individual heuristic makes only small changes, the adaptation task remains manageable.

3.8 The MINSTREL Model of Creative Problem-Solving

Figure 3.4 illustrates how TRAMs are integrated into the case-based problem-solving model. The Recall and Adapt steps of the basic problem-solving model are augmented with a pool of TRAMs. During problem-solving, a TRAM is selected from this pool and applied to the original problem. If the TRAM succeeds in discovering a solution and adapting it to the original problem, then problem-solving succeeds. If problem-solving fails, then the current TRAM is discarded, another selected from the pool of available TRAMs, and the cycle repeats.

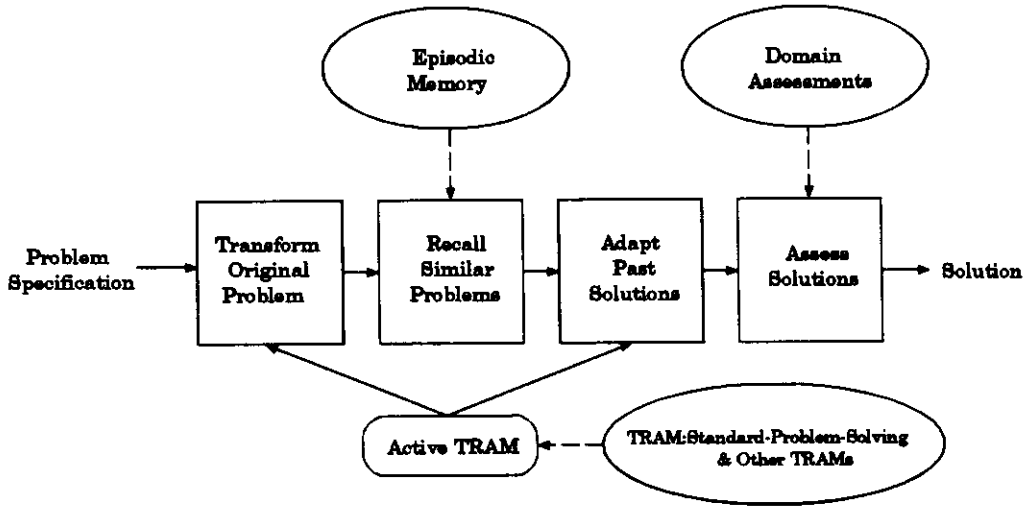


Figure 3.4 Model of Problem-Solving with TRAMs

3.8.1 Transform

The first step of the augmented problem-solving model is to transform the original problem. In this step, the Transform portion of one of MINSTREL's TRAMs is applied to the original problem description, changing it into a new problem. This step is a new addition to the model of case-based problem solving. It is this transformation that permits MINSTREL to search the problem space for new knowledge, a capability that normal problem-solving lacks.

Of course, a problem-solver needs to be creative only if problem-solving fails. So before applying a transformation to the original problem specification, the problem-solver should try to solve the original problem.

This is accomplished by a special creativity heuristic called "TRAM:Standard-Problem-Solving". TRAM:Standard-Problem-Solving is the null creativity heuristic: it applies no transformation to the problem description and applies no adaptation to any recalled solutions. So when TRAM:Standard-Problem-Solving is the Active TRAM, MINSTREL performs standard

case-based problem-solving. TRAM:Standard-Problem-Solving thus implements standard case-based problem-solving within the framework of creativity. TRAM:Standard-Problem-Solving is illustrated in Figure 3.5.

TRAM:Standard-Problem-Solving

Transform Strategy

Do not transform the problem description.

Adapt Strategy

Do not adapt the recalled solution. Apply as is to the current problem.

Figure 3.5 TRAM:Standard-Problem-Solving

TRAM:Standard-Problem-Solving is always the first TRAM selected from the pool of available TRAMs. This ensures that the first effort of MINSTREL's creative problem-solving will be to find a known solution. By selecting TRAM:Standard-Problem-Solving first from the pool of TRAMs, MINSTREL will always use an old, known solution if available and acceptable to the solution assessments.

3.8.2 TRAM Selection

When TRAM:Standard-Problem-Solving fails to find a solution, a new TRAM must be selected from the pool of available TRAMs to begin MINSTREL's effort to create a new solution.

The TRAMs in the pool of available TRAMs are organized according to the types of problems to which they apply. Some TRAMs are general, and can modify any problem description. Other TRAMs are specific to certain types of problem descriptions. To select a TRAM for the current problem, MINSTREL uses the problem specification to select the applicable TRAMs from the pool of available TRAMs.

MINSTREL indexes the pool of available TRAMs using episodic memory. Just as episodic memory contains past problems and their associated solutions, episodic memory also contains past problems and associated TRAMs. Finding the TRAMs that apply to a particular problem thus becomes a matter of recall. MINSTREL uses the current problem as an index to episodic memory, and finds similar past problems and their associated TRAMs. TRAMs which apply to many different types of problem are associated with very general problem descriptions; TRAMs which apply to specific types of problems are associated with more specific problem descriptions. By recalling all of the problem descriptions that match the current problem, MINSTREL gathers all the applicable TRAMs.

From the applicable TRAMs, one TRAM is selected randomly. The reason for using random

choice is straightforward. Creativity heuristics search the problem space for useful knowledge to apply to the current problem. But prior to actually performing that search, the problem-solver cannot know where the useful knowledge lies. In this sense, the applicable TRAMs are indistinguishable. Hence the problem-solver has no reason to prefer one heuristic over another, making random selection a reasonable algorithm.

Another possible algorithm for selecting between competing TRAMs is to use past creativity experience to guide TRAM selection. For example, the problem-solver might want to use TRAMs according to how frequently they've been successful in past problem-solving situations. Or, since that might lead to repetition, the problem-solver might want to try heuristics that haven't been frequently used. Or the problem-solver might want to try the most recently successful heuristic again, to reinforce the value of newly learned heuristics. All of these are interesting strategies.

Although MINSTREL does not currently use past creativity experience to guide TRAM selection, the use of episodic memory to organize TRAMs provides support for these types of strategies. As TRAMs are used and indexed into episodic memory, the problem-solver retains knowledge about the usage of creativity. However, what form a cognitively valid record of creativity would take remains an open question.

3.8.3 Recall

The second step in the MINSTREL model of creative problem-solving is recall of similar past problems. If similar past problems can be recalled, the associated solutions are passed on to the adaptation step. The index for recall is the problem specification. For this reason, episodic memory must be organized by past problems, and the recall process must be able to take a possibly incomplete problem specification and recall similar past problems.

MINSTREL's model of episodic memory is based on the "context plus index" model ([Schank 1982][Kolodner 1984][Reiser 1986]). Context plus index models of memory organizes a specific episode according to its general context (i.e., a plane trip) and the distinguishing features of the episode (i.e., taking Pan-Am, flying to Central America). In the MINSTREL model, problem types correspond to different contexts, and the specific features of each problem are used as indexing features. This permits MINSTREL to organize memory according to past problems, and to recall matching problems when given a problem specification.

MINSTREL's model of episodic memory is discussed in more detail in Chapter 2.

3.8.4 Adaptation

The third step of MINSTREL's creative process is adaptation. Past solutions to problems are passed to the adaptation step, where the Adapt portion of the controlling TRAM modifies them for use on the current problem.

In general, the problem of adapting knowledge - even useful knowledge - to a new problem is very difficult. Consider what a problem-solver must do to adapt the 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.

to the problem of "Lancelot meets Guinevere unexpectedly". The problem-solver must first fully understand this story, so that he can recognize that it is also an unexpected meeting. Then he must determine what its relationship is to original problem, to know how to apply this knowledge to the original problem. He must realize that it is in another problem domain, and determine what that domain is. Finally, he must build a mapping from this domain to the original problem domain and translate the story.

In MINSTREL, though, the problem of adaptation is greatly simplified by associating adaptations with specific problem transformations. Because each adaptation is applied only to problem solutions that arose from a particular problem transformation, there is no need to fully understand the problem solution and determine its relation to the original problem. The relation of the recalled solution to the original problem is fixed by the Transform portion of the TRAM. Instead, the adapt portion of each TRAM needs only do the final work of adaptation - to make the necessary changes to the recalled solution.

For example, in TRAM:Cross-Domain-Solution the Adapt step need only apply the cross domain mapping generated during the Transform step in reverse upon the problem solution. This translates the recalled solution back into the original problem domain, making it suitable for the original problem. The Adapt step does have to understand the recalled solution, or determine its relationship to the original problem.

3.8.5 Assessment

The fourth step of MINSTREL's creative process is assessment. The purpose of the assessment step is to evaluate a proposed solution in light of acceptance criteria specific to a particular domain. For example, a mechanical engineer might evaluate his designs in terms of their efficiency, what kinds of materials they use, and so on. Although the problem-solving process tries to create solutions that fit the original problem specification, it may inadvertantly fail (see the discussion

of "Creativity Errors" in this chapter), or there may be additional criteria that cannot be easily expressed in the problem specification. In these cases, domain assessments can be used to catch faulty solutions.

A special assessment for artistic problem domains is the boredom assessment. The boredom assessment rejects solutions which are too similar to previous solutions, and models the drive for originality in the arts. The boredom assessment is an example of an assessment that embodies a criteria which is difficult to express in the original problem specification - namely, that the solution be original.

When a proposed solution fails a domain assessment, problem-solving fails, the active TRAM and the proposed solution are discarded, and another TRAM is selected. The problem-solving process then repeats under the control of the new TRAM.

One shortcoming of this algorithm is that it discards the proposed solution, which may be almost entirely correct. Re-inventing the correct parts of that solution may take a great deal of effort. An alternative possibility is to "repair" proposed solutions which fail a domain assessment.

For example, if a domain assessment for mechanical invention notices that a device has a redundant component, it can repair that design by removing the redundant component. Another possibility is to use problem-solving recursively to repair a faulty solution. If a domain assessment in mechanical invention notices that a device lacks a power source, it can repair this problem by recursively using problem-solving to invent a power source.

MINSTREL is capable of both these types of repair. Domain assessments can directly manipulate the proposed solutions or call problem-solving to create a correction. Whether a proposed solution should be rejected or repaired depends both upon the cost and efficacy of the repair technique.

First, repair should be undertaken only when it provides a cost savings over rejecting the faulty solution and finding a new solution. Repairing a solution with major faults may be much more expensive and time-consuming than simply throwing out the faulty solution and finding an entirely new one. For a mechanical device lacking a power source, inventing a power source may be a very difficult problem that would take more time and effort than simply finding a different solution to the original problem. If this is the case, it would be better to reject the proposed solution rather than repair it.

Efficacy of repair can also be a factor. Consider the example given above in which a repair removes a redundant piece from a mechanical device. Is that always a correct and safe change to the design? Only insofar as the repair understands the device design. Perhaps the redundant pieces was added to the design by problem-solving in order to balance the weight of the device. In general, a repair heuristic will not be as knowledgeable and powerful as problem-solving, and so will sometimes make errors.

Deciding the likely costs of repair vs. rejection, or determining whether a repair will be efficient

without introducing more serious faults are difficult issues. Currently MINSTREL's boredom assessment uses rejection, while the two assessments used in mechanical invention use repair, but these are simply design choices that are not founded on any deep understanding of the issues of repair vs. rejection. For further discussion of the role of repair in planning, see [Hammond 1988]. For a comparison of MINSTREL's creativity to plan repair, see Chapter 13.

3.9 Summary of Creativity Model

Earlier we identified the three challenges of creativity as: (1) finding useful knowledge by searching the problem space, (2) limiting the adaptation task, and (3) discovering solutions substantially different from the original solution. The MINSTREL model of creativity answers these challenges by associating problem transformations with corresponding solution adapatations. Problem transformations permit MINSTREL to search the problem space, and because each problem transformation has an associated, specific solution adaptation, the complexity of adaptation is eliminated. Solutions with substantial differences from past solutions can be discovered by single creativity heuristics, or more likely, by several heuristics applied in succession.

3.10 Imaginative Memory

The central step of MINSTREL's creative problem-solving model is recalling a solution from episodic memory. But recall *itself* can be viewed as a problem. What happens if creative problem-solving is used to solve the "recall problem"?

To do this, the Recall step of creative problem-solving is modified to recursively call creative problem-solving. To prevent this from leading to endless recursion, TRAM:Standard-Problem-Solving is modified so that it will continue to use episodic memory for recall.

Now when a problem-solver needs to recall something, creative problem-solving is used with the problem specification "Find something in episodic memory that matches these features." TRAM:Standard-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:Standard-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 the Recall step of TRAM:Standard-Problem-Solving fails. If TRAM:Standard-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 the recursive call is TRAM:Standard-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.

Treating recall as problem-solving also enables the problem-solver to apply multiple TRAMs to a problem. Each time recall fails the recursive use of creative problem-solving will apply another TRAM to the recall features. In this way, a number of TRAMs can be successively applied to a problem. 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.

There are two advantages to imaginative memory.

First, it provides a simple and powerful mechanism for the repeated applications of creativity heuristics to a problem. Each time the Recall step of problem-solving fails, imaginative memory will recurse and apply a new creativity heuristic. If no useful knowledge can be found in the problem space near the original problem, repeated problem transformations will move the creative problem-solver into more distant areas of the problem space.

More importantly, imaginative memory implements creativity at a low cognitive level. By embedding creative problem-solving in the recall process, imaginative memory makes creativity transparently available to *any* cognitive mechanism that uses episodic memory for reasoning. By integrating creativity into the foundation of the cognitive process, imaginative memory increases the reasoning power of all cognitive mechanisms.

3.11 Integrated Model

Figure 3.6 illustrates how the boredom assessment, Transform-Recall-Adapt Methods and imaginative memory are integrated with case-based reasoning to form a complete model of creative problem-solving. The three steps of case-based reasoning (Recall, Adapt, Assess) have been augmented with a Transform step. An active TRAM controls the Transform and Adapt steps. Initially this is TRAM:Standard-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) except when controlled by TRAM:Standard-Problem-Solving.

The problem-solving cycle begins when a problem description enters the recall step. Initially TRAM:Standard-Problem-Solving is in control. The original problem description is used to recall similar problem-solving situations from episodic memory. If recall succeeds, the recalled situations are passed to the Adapt step. Under TRAM:Standard-Problem-Solving, no adaptation is needed because the recalled solutions are very similar to the original problem, so the recalled solutions are passed along to the Assess step. In the Assess step, all active assessments are ap-

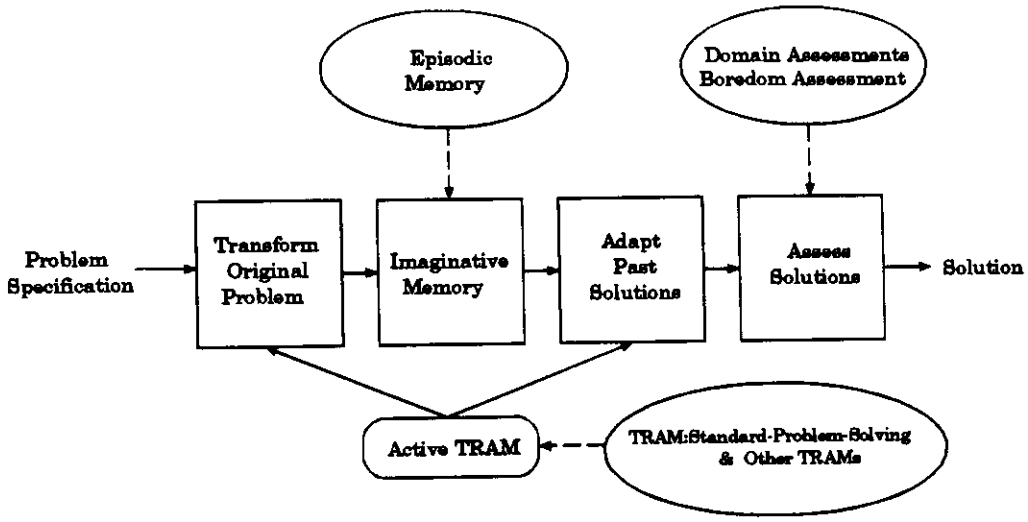


Figure 3.6 MINSTREL's Process Model of Creativity

plied to the recalled solutions, and if a solution passes all the assessments, it is output as a solution to the original problem. This is the normal problem-solving cycle.

If TRAM:Standard-Problem-Solving fails, either because no solutions were recalled or because the recalled solutions failed some assessment, TRAM:Standard-Problem-Solving is discarded and a new TRAM selected. 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 creative problem-solving repeats.

Figure 3.7 illustrates the recursive use of creativity problem-solving. When recall from episodic memory fails, imaginative memory resolves to a recursive use of problem-solving. This continues until recall from episodic memory succeeds (or a processing limit is reached). The solution from each level is passed back to the previous level, where it is adapted, assessed and passed up again. Eventually the solution reaches the top level, at which point it has been adapted to the original problem.

At each step, adapted solutions are assessed by domain assessments and, if appropriate, the boredom assessment. 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.

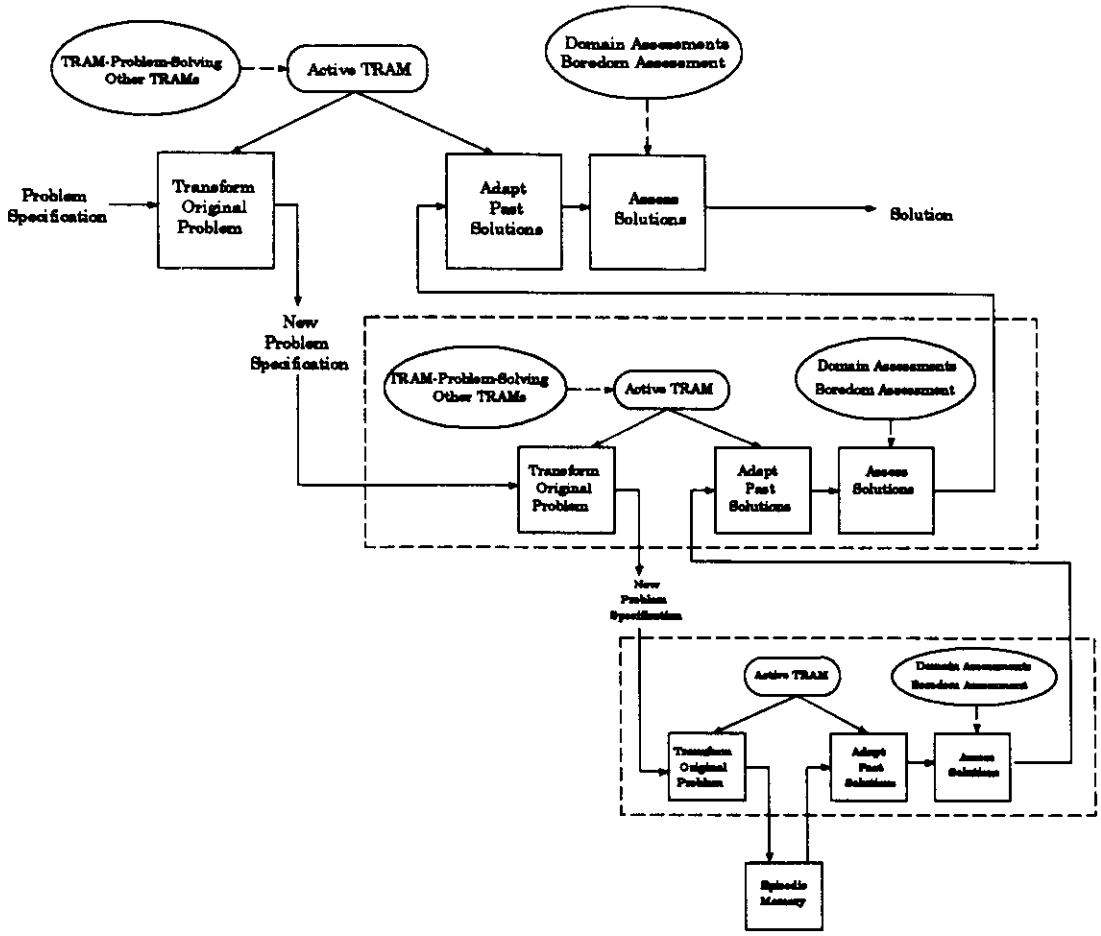


Figure 3.7 Recursive Creativity

3.12 Examples of Creativity

The model of creative problem-solving presented in this chapter has been implemented in a computer program called MINSTREL. The next two sections show how MINSTREL uses creative problem-solving to discover new solutions for a simple planning problem in the domain of King Arthur, and to create a scene for a story. These examples should give the reader a better understanding of the TRAM model of creativity.

3.12.1 Suicide Example

In this example, MINSTREL is trying to discover a way for a knight to commit suicide. Initially, MINSTREL knows nothing about suicide, but does know about killing dragons and drinking a potion to become ill. Using this knowledge and creative problem-solving, MINSTREL discovers three methods of suicide and invents[4] the notion of "poison".

3.12.1.1 Representation

MINSTREL uses a schema-based representation language called RHAPSODY [Turner 1985]. Goals, actions, and states of the world are represented as schemas; instances of these schemas make up the episodes in MINSTREL's memory and the elements of the stories MINSTREL tells. Each schema has named slots which contain schema information. Goal schemas, for example, have slots for the type of the goal and the actor of the goal. Schemas can also have named links to other schemas. Goal schemas typically have links to plans and to the states that achieve the goals. Schema names begin with an ampersand (&) and schema instances are given either descriptive names (such as &KNIGHT-FIGHT) or generated names based on the schema type (such as &GOAL.112). For a more complete discussion of MINSTREL's representation, see Chapter 2.

3.12.1.2 The Problem

Figure 3.8 illustrates MINSTREL's representation of the suicide problem. &HUMAN.12 is an instance of the human schema which represents the knight. The type slot of a human instance indicates the character's major role in the King Arthur world, and illustrates how MINSTREL uses schema slots to instantiate particular schema instances. The knight has a goal (&GOAL.11) which will be achieved by the knight being dead (&STATE.8). The plan for this goal (&ACT.4) is currently uninstantiated. MINSTREL's goal in this example is to instantiate &ACT.4 as an action or series of actions that will achieve the knight's goal of committing suicide.

3.12.1.3 Initial State of Episodic Memory

All of MINSTREL's knowledge of the King Arthur domain is contained in episodic memory. MINSTREL's creativity heuristics have general knowledge about goals, plans and states of the world, but specific knowledge about the goals, plans and actions of characters in the King Arthur domain is deduced from the contents of episodic memory.

At the beginning of this example, MINSTREL knows nothing about how a knight might kill himself. Initially, MINSTREL's episodic memory contains only these two episodes:

4. We use the term "invent" to indicate a concept that is new to MINSTREL, if not necessarily new to the reader.

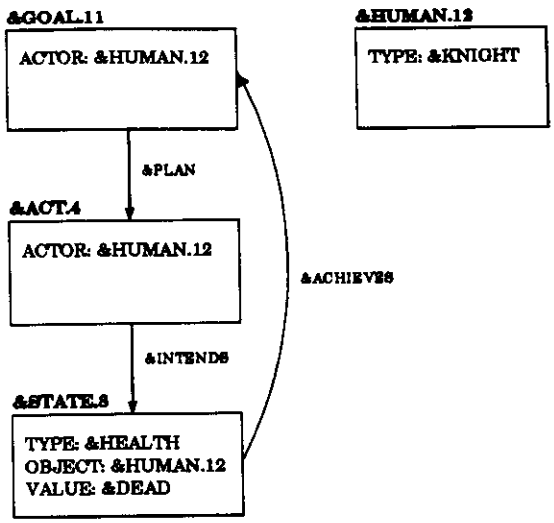


Figure 3.8 Representation of Suicide

Knight Fight

A knight fights a troll with his sword, killing the troll and being injured in the process.

The Princess and the Potion

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

Figure 3.9 shows the schema representation of "The Princess and the Potion". &ACT.14 represents the action of Lady Andrea quaffing a potion. &STATE.17 represents the intentional outcome of that action - Lady Andrea becoming ill.

3.12.1.4 Trace

Figure 3.10 shows a trace of MINSTREL inventing three different methods of suicide. In this example, MINSTREL has been configured to exhaustively invent solutions to the suicide problem and present them in English as created. Normally MINSTREL generates copious debugging and tracing output. To spare the reader, the trace shown in Figure 3.10 has been edited to improve readability. Uninteresting portions of the trace and reasoning dead-ends have been deleted. These deletions have been marked in the trace with "[...]". Except for this editing, the trace appears exactly as generated by MINSTREL. The level of indentation of the trace reflects the level of recursive problem-solving. MINSTREL begins the example shown in Figure 3.10 by generating the initial problem specification in English: "A knight named John did something. John died." This is an English description of the schema representation shown in Figure 3.6. MINSTREL's English descriptions of schemas are produced by a phrasal generator [Reeves

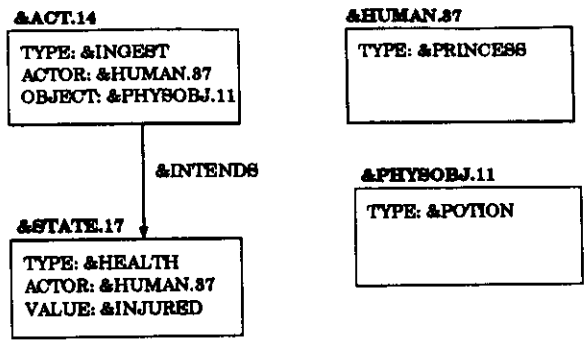


Figure 3.9 Representation of "The Princess and the Potion"

1989][Zernik 1987].

The problem specification is followed by a trace of the problem-solving cycle. As each new TRAM is applied, the name of the TRAM (i.e., TRAM:GENERALIZE-CONSTRAINT) is printed out. When TRAM:Standard-Problem-Solving is used to attempt recall from episodic memory, a message is printed out showing the recall index and what was recalled. The very first part of the trace shows TRAM:EXAGGERATE-SCALED-VALUE being applied to this problem and failing when nothing (i.e., NIL) is recalled from episodic memory.

When a solution is discovered, MINSTREL prints a message to that effect and generates an English language description of the solution.

3.12.1.5 TRAM:GENERALIZE-CONSTRAINT

The first TRAM which succeeds in discovering a solution to the suicide problem is TRAM:Generalize-Constraint. This TRAM 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. Figure 3.11 shows an informal outline of this heuristic.

In the suicide problem, the constraints available for generalization are the schema slot fillers of the problem specification (Figure 3.6): (1) the actor is a knight, (2) the object of the state is the actor, (3) the type of the state is health, and (4) the value of the state is dead. TRAM:Generalize-Constraint suggests recalling scenes in which one of these constraints has been generalized.

In this example, MINSTREL generalizes constraint (2). The original problem specification is "a knight kills himself." TRAM:Generalize-Constraint generalizes this specification by removing the constraint that the knight kill himself and replacing it with the more general constraint that the knight kill *something*. This is indicated by the message "`Generalizing :OBJECT on &STATE.112.`" printed in the trace. The new problem specification is "a knight kills something." This generalization recalls the "Knight Fight" episode:

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

Problem-Solving Cycle: &ACT.105.
 Executing TRAM:EXAGGERATE-SCALED-VALUE.
   Recalling &ACT.105: NIL.
 ...TRAM fails.
[...]
 Executing TRAM:GENERALIZE-CONSTRAINT.
   Generalizing :OBJECT on &STATE.112.
   Recalling &ACT.118: &KNIGHT-FIGHT.
 TRAM succeeds: (&ACT.405).

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

 [...]
  Executing TRAM:SIMILAR-OUTCOMES-PARTIAL-CHANGE.
    Recalling &ACT.136: NIL.
     [TRAM Recursion: &ACT.136.]
        Executing TRAM:GENERALIZE-CONSTRAINT.
        Generalizing :ACTOR on &ACT.138.
          Recalling  &ACT.138: &PRINCESS-POTION.
        ...TRAM succeeds: (&ACT.447).
  ...TRAM succeeds: (&ACT.447).

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

 [...]
  Executing TRAM:INTENTION-SWITCH.
    Recalling &ACT.174: NIL.
     [TRAM Recursion: &ACT.174.]
        Executing TRAM:SIMILAR-OUTCOMES-PARTIAL-CHANGE.
          Recalling  &ACT.178: &KNIGHT-FIGHT.
        ...TRAM succeeds: (&ACT.588).
  ...TRAM succeeds: (&ACT.588).

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

Figure 3.10 Suicide Trace

TRAM:Generalize-Constraint

Transform

1. Select and generalize a feature (call it $generalized-feature) of the scene specification. Use this new scene specification as an index for imaginative recall.

Adapt

1. Adapt the recalled solution to the current problem by adding $generalized-feature back to the recalled scene.

Figure 3.11 Informal Outline of TRAM:Generalize-Constraint

```
[...]
 Executing TRAM:GENERALIZE-CONSTRAINT.
   Generalizing :OBJECT on &STATE.112.
   Recalling &ACT.118: &KNIGHT-FIGHT.
   Adapting by replacing &MONSTER.15 with &HUMAN.12.
 TRAM succeeds: (&ACT.405).

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

In "Knight Fight", a knight kills a troll by hitting it with his sword. This episode is adapted to the original suicide problem by reversing the original transformation. The troll corresponds to the generalized constraint that a knight kills "something". To adapt this solution to the original problem, this more general constraint must be replaced with the original constraint - that the knight kill himself. Therefore the Adapt portion of TRAM:Generalize-Constraint replaces the troll with the knight, creating a scene in which a knight kills himself by hitting 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.

Three issues which must be addressed in TRAM:Generalize-Constraint are (1) how features are selected for generalization, (2) how the selected feature is generalized, and (3) how the selected feature is added back into the created scene.

To maximize the success of the recall step of TRAM:Generalize-Constraint, the feature chosen for generalization should be likely to lead to the recall of a scene. To achieve this, MINSTREL makes a broad generalization of each feature in the representation and attempts recall using the generalized episode. Every generalization that results in recall is added to a pool, and the problem constraint to be generalized is selected randomly from this pool.

MINSTREL uses two methods to generalize a feature. First, the feature can be completely re-

moved from the problem specification. This is the broadest possible generalization, and is used when selecting the candidate pool. And while this method provides a good, quick indication of whether generalizing a particular feature is useful, it is so broad that it often leads to the recall of scenes which are difficult to adapt to the original problem.

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 removing it altogether from the problem specification. The new problem specification - "A knight gives somebody something that makes him happy" - can recall *any* episode in memory in which a knight gives something. If this recalls a scene in which a knight makes a troll happy by giving him a hunk of raw meat, it leads to a scene in which the knight pleases 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. One such generalization is based on *class hierarchies.*

Classes group concepts that have many similarities. The "People" class groups a variety of characters - princesses, knights, kings, and hermits - that have many similar features. There can be many class hierarchies, and objects can belong to several classes. Knights, for example, are members of both the "People" class and the "Violent Characters" class. By generalizing problem features within classes, MINSTREL is more likely to find a useful reminding.

For this reason, MINSTREL's implementation of TRAM:Generalize-Constraint uses class generalizations. In the suicide example, the "knight" feature is generalized to "a Violent Character" This recalls the "Knight Fight" episode, in which a knight fights a troll in order to kill the troll, because trolls are also members of the "Violent Character" class. If this generalization had failed, MINSTREL would have generalized to the superclass ("Actors"), and if that generalization failed, TRAM:Generalize-Constraint would have failed. A portion of MINSTREL's class hierarchy is shown in Figure 3.12.

The final step in TRAM:Generalize-Constraint is to adapt the recalled episode to the original specification. This is achieved by replacing the generalized feature value with the original feature value throughout the recalled episode. In the suicide example, the troll in "Knight Fight" is replaced by the knight throughout the recalled episode, resulting in a scene in which a knight kills himself by fighting himself with his sword.

Figure 3.13 illustrates TRAM:Generalize-Constraint as implemented in the current version of MINSTREL.

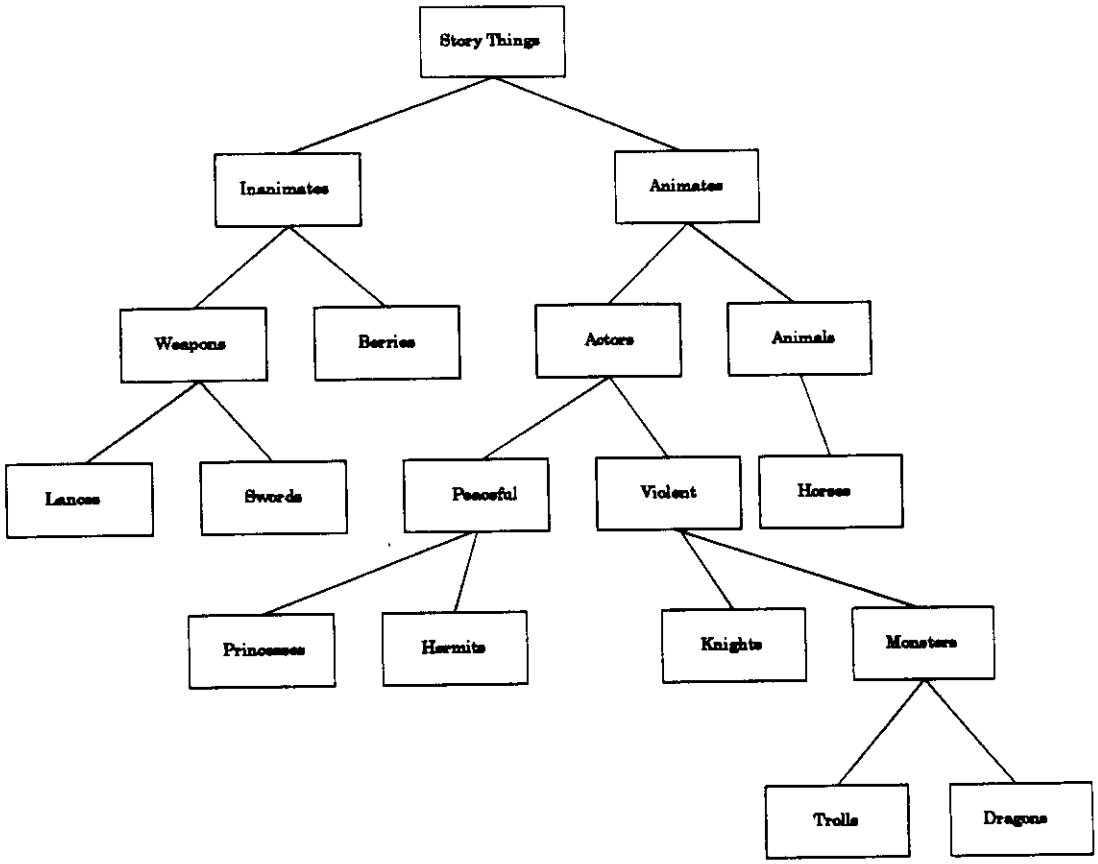


Figure 3.12 A Portion of MINSTREL's Class Hierarchy

3.12.1.6 TRAM:Similar-Outcomes

MINSTREL discovers a second method for committing suicide by using both TRAM:Generalize-Constraint and a new heuristic, TRAM:Similar-Outcomes-Partial-Change. TRAM:Similar-Outcomes-Partial-Change suggests that if an action results in a particular outcome, it might also result in other, similar outcomes. For example, if MINSTREL doesn't know anything about riding horses except that a knight once rode one to a castle, MINSTREL can use TRAM:Similar-Outcomes to guess that a knight might also ride a horse to some other destination.

In the suicide example, TRAM:Similar-Outcomes-Partial-Change recognizes that being killed is similar to being injured, and transforms the problem description from "a knight purposely kills himself" to "a knight purposely injures himself." If MINSTREL can recall an action in which a knight purposely injures himself, it will be adapted to the current problem by replacing the injury with death. In essence, TRAM:Similar-Outcomes-Partial-Change "guesses" that an action which is known to result in injury might also result in death:

TRAM:Generalize-Constraint

Transform Strategy

1. For each feature in the scene specification, eliminate the feature and attempt recall from episodic memory. If recall is successful, then add the feature to a pool of acceptable generalizations.

2. Randomly choose a feature from the pool of acceptable generalizations ($feature).

3. Create a generalization of $feature based on the class membership. Attempt to recall or create a scene based on that generalization. If successful, pass the recalled scene ($recall) to the adapt step.

4. Otherwise, create a generalization based on the superclass, and attempt recall.

5. Otherwise, create a generalization by eliminating the feature, and repeat, and attempt recall.

6. Otherwise, fail this feature and attempt another feature from the pool of possible generalizations.

Adapt Strategy

1. Build a correspondence between the original scene specification ($original) and $recall by matching all the features in $original with the same features in $recall.

2. Add to this correspondence a mapping from the selected feature ($feature) to the instantiation of that feature in $recall.

3. Copy features that are present in $recall but missing in $original from $recall to $original, translating through the correspondence.

Figure 3.13 TRAM:Generalize-Constraint

```
[...]
 Executing TRAM:SIMILAR-OUTCOMES-PARTIAL-CHANGE.
   Transforming &DEATH to &WOUND.
   Recalling &ACT.136: NIL.
    [TRAM Recursion: &ACT.136.]
       Executing TRAM:GENERALIZE-CONSTRAINT.
       Generalizing :ACTOR on &ACT.138.
         Recalling: &PRINCESS-POTION.
       ...TRAM succeeds: (&ACT.447).
 ...TRAM succeeds: (&ACT.447).

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

However, the description "a knight purposely injures himself" does not recall either of the episodes in MINSTREL's episodic memory (as indicated by "`Recalling &ACT.136: NIL.`"). "Knight Fight" is not recalled because the knight does not intentionally injure himself; "Princess and the Potion" is not recalled because the actor is not a knight. Since recall fails, imaginative memory recurses and applies a new TRAM.

At this new level, MINSTREL applies TRAM:Generalize-Constraint to the description "a knight purposely injures himself" and generalizes the "knight" feature. "Knight" is generalized to "anyone" which results in the new problem specification "Someone does something to purposely injure themself." Note that this problem specification has been modified twice from the original specification, once by TRAM:Similar-Outcomes-Partial-Change and once by TRAM:Generalize-Constraint.

The new problem description recalls "The Princess and the Potion" in which a lady of the court drinks a potion to make herself ill. The Adapt portion of TRAM:Generalize-Constraint then adapts this scene by replacing "lady of the court" (the generalized constraint) with "a knight" (the original constraint). This results in a scene in which a knight makes himself ill by drinking a potion.

The adapted scene is returned to the previous problem-solving level, where TRAM:Similar-Outcomes-Partial-Change also adapts the scene, by replacing the illness with death. The adaptation reverses the transformation that turned "death" into "illness". 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 (vs. just making one ill).

(In fact, MINSTREL has invented the more narrow notion of "a potion which will kill a knight." To apply this to other animate beings, MINSTREL will have to use creativity again to generalize about the actor of this action. Although MINSTREL never needs the more general concept, it can make this generalization using TRAM:Generalize-Actor. For a description of this TRAM, see Chapter 4).

The main issue in TRAM:Similar-Outcomes is determining when two outcomes are interchangable. MINSTREL has two methods for deciding this question. These are implemented as separate TRAMs called TRAM:Similar-Outcomes-Partial-Change and TRAM:Similar-Outcomes-Implicit.

TRAM:Similar-Outcomes-Partial-Change reasons that if an action can result in a partial relative change of a state then the action can also result in a complete change of the state. TRAM:Similar-Outcomes-Partial-Change is shown in Figure 3.14. In this example, TRAM:Similar-Outcomes-Partial-Change reasons that something that makes someone ill (a partial negative change in health) might also kill them (a complete negative change in health).

Like many of MINSTREL's creativity heuristics, TRAM:Similar-Outcomes-Partial-Change is a

TRAM:Similar-Outcomes-Partial-Change

Transform Strategy

If the problem specification has an act which results in a partial relative change of a state in some direction, create a new specification in which the relative change is extended in the same direction.

Adapt Strategy

Replace the change of state in the recalled episode with a relative change of state copied from the original problem specification.

Figure 3.14 TRAM:Similar-Outcomes-Partial-Change

"common-sense" rule that captures reasoning that is often useful but not always correct. For example, extending a partial state change could be used to reason that because a man can lift a book in one hand he would also be able to lift an automobile in one hand. There are two things to be said about this type of error.

First, this type of error points out the value of simple, constrained TRAMs that make only small changes in problem descriptions. By making only small extensions to a problem-solver's knowledge, a TRAM is less likely to make an error in its extrapolation. In fact, MINSTREL's version of TRAM:Similar-Outcomes-Partial-Change only extends state changes one additional "step" in a known direction. If state of health is represented by the scale "Excellent Good Normal Ill Dead", and MINSTREL knows an action that changed a man's health from Normal to Ill, then TRAM:Similar-Outcomes-Partial-Change can only extrapolate that to an action that changes a man's health from Normal to Dead. TRAM:Similar-Outcomes-Partial-Change cannot extrapolate to an action that would take a man's health from Ill to Excellent, or even from Excellent to Dead. Similar limits can be applied to states which do not have discrete representations, although MINSTREL does not currently handle this. By using simple, constrained creativity heuristics, MINSTREL reduces the number of reasoning errors it makes.

Incremental imaginative steps reduces errors in reasoning.

Second, even with restricted TRAMs, MINSTREL can still make reasoning errors of this sort. But this is to be expected: A creative problem-solver *should* make errors. MINSTREL uses creativity to actively extend its knowledge. By making good use of what it already knows, MINSTREL can often make accurate guesses about what it doesn't know. But sometimes it will err. The challenge faced by a creative problem-solver is to find creativity heuristics that are productive without an inordinate number of reasoning errors. The issue of creativity errors is discussed in more detail in Chapter 14.

Imaginative reasoning will sometimes produce errors.

The second TRAM MINSTREL uses to determine when two outcomes are interchangable is TRAM:Similar-Outcomes-Implicit. TRAM:Similar-Outcomes-Implicit reasons that two outcomes are interchangable in *every* situation if it can recall *any* situation in which they are interchangable. For example, if MINSTREL can recall a scene in which a knight fought and killed a troll, and another scene in which a knight fought and killed a dragon, MINSTREL can use this knowledge to guess that trolls and dragons are generally interchangeable. TRAM:Generalize-Constraint used a class hierarchy - an explicit representation of object similarities - to substitute one feature for another. TRAM:Similar-Outcomes uses an implicit representation of similarities to substitute one outcome for another.

TRAM:Similar-Outcomes-Implicit

Transform Strategy

1. Create a new problem specification: An uninstantiated act that results in the state from the original problem specification. Use this to recall different acts which can cause the result from the original problem.

2. Use episodic memory to recall other possible results of the actions collected in (1). Build a pool of these alternate results.

3. Create a new problem specification in which the act from the original problem specification results in a randomly-selected alternate result. Use this new specification for recall.

Adapt Strategy

1. Replace the alternate result of the recalled episode with the result copied from the original problem specification.

Figure 3.15 TRAM:Similar-Outcomes-Implicit

Like TRAM:Similar-Outcomes-Partial-Change, TRAM:Similar-Outcomes-Implicit is a heuristics that can sometimes err. Again, this is a direct consequence of its function as an extrapolator of knowledge, and is to be expected in a creative problem-solver.

3.12.1.7 TRAM:Intention-Switch

MINSTREL's final plan for suicide is discovered using TRAM:Intention-Switch. This heuristic suggests that if the effect of an action was intentional it might just as well have been unintentional. TRAM:Intention-Switch is illustrated in Figure 3.16.

In the suicide example, TRAM:Intention-Switch transforms the original specification from "a knight purposely kills himself" to a "knight accidently kills himself." Recall on this new speci-

TRAM:Intention-Switch

Transform Strategy

If an action in the problem specification intends a result, create a new problem specification in which the same action unintentionally achieves the result.

Adapt Strategy

Replace the unintentional result of the recalled episode with a similar intentional result.

Figure 3.16 TRAM:Intention-Switch

fication (&ACT.174) fails, because MINSTREL's episodic memory does not contain any episodes in which a knight accidently kills himself.:

```
[...]
 Executing TRAM:INTENTION-SWITCH.
   Recalling &ACT.174: NIL.
    [TRAM Recursion: &ACT.174.]
       Executing TRAM:SIMILAR-OUTCOMES-PARTIAL-CHANGE.
         Recalling  &ACT.178: &KNIGHT-FIGHT.
       ...TRAM succeeds: (&ACT.588).
 ...TRAM succeeds: (&ACT.588).

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

Problem-solving is used recursively, and TRAM:Similar-Outcomes-Partial-Change modifies the current problem description "a knight accidently kills himself" by changing "kills himself" into something similar: "injures himself". The new problem description is "a knight "accidently injures himself." This recalls "Knight Fight," in which a knight is injured while killing a troll.

Both TRAM:Similar-Outcomes-Partial-Change and TRAM:Intention-Switch adapt this recalled scene. TRAM:Similar-Outcomes-Partial-Change replaces "injures himself" with "kills himself", and TRAM:Intention-Switch replaces "accidently" with "purposely", resulting in a scene in which a knight commits suicide by intentionally losing a fight with a troll.

Three things are interesting about this particular invention. First, although this particular TRAM is very simple, it results in a very novel and interesting plan - a knight purposely losing a fight to a dangerous opponent. This demonstrates that simple, limited TRAMs applicable to a wide variety of problems still have the power to invent new solutions to problems.

Second, it is interesting to note that MINSTREL has invented two different methods of suicide

from the same episodic memory. Using TRAM:Generalize-Constraint, MINSTREL transformed the "Knight Fight" episode into a plan in which a knight fights himself. Using TRAM:Intention-Switch and TRAM:Similar-Outcomes-Partial-Change, MINSTREL transforms the same episode into a plan in which a knight purposely loses a fight to a dangerous opponent. The ability to invent several solutions from a single episode shows the flexibility and power of MINSTREL's creativity process.

Third, unlike TRAM:Similar-Outcomes-Implicit and TRAM:Similar-Outcomes-Partial-Change, TRAM:Intention-Switch will never make a creativity error. Any action which can be done intentionally can be done unintentionally, and vice versa. Unlike the Similar-Outcome TRAMs, which extrapolate the problem-solver's knowledge into new areas, TRAM:Intention-Switch re-directs the problem-solver into a little-used area of his knowledge. Intentionally doing things accidently (a seemingly self-contradictory idea) is a reasoning strategy that is seldom useful to a problem-solver, so episodic memory is unlikely to contain general plans of this sort. Instead, TRAM:Intention-Switch re-directs the problem-solver to this strategy for the restricted type of problems in which it might be useful.

3.12.1.8 Summary

The suicide example demonstrates two points about the MINSTREL model of creativity. First, it demonstrates that MINSTREL's creativity process has operational validity; MINSTREL *can* invent novel solutions to a problem. Second, it demonstrates the power of Transform-Recall-Adapt Methods. Using only two episodic memories and three TRAMs, MINSTREL invents three different methods of suicide and the notion of poison.

3.12.2 Storytelling Example

We will now look at how MINSTREL's creativity functions in the context of a larger task: storytelling.

To tell stories, MINSTREL must select a theme, instantiate the events of the theme (the plot), assure that the events of the story are consistent, achieve literary goals such as building suspense, and so on. Many of these tasks can be achieved without creativity. MINSTREL knows that knights ride hoses, and hence doesn't have to invent a way for knights to travel from place to place. But sometimes MINSTREL encounters a new problem in the course of storytelling, or becomes bored with a particular story development. In these cases, creative problem-solving is used to invent a solution.

This example presents a specific task MINSTREL encountered in telling *The Vengeful Princess*:[5]

The Vengeful Princess

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.

MORAL: Deception is a weapon difficult to aim.

The particular portion of *The Vengeful Princess* this example focuses on is the creation of Jennifer's reason for wanting revenge on Darlene:

... Jennifer wanted revenge on a lady of the court named Darlene because Darlene had the berries which she picked in the woods and Jennifer wanted to have the berries.

When telling this story, MINSTREL knew nothing about what kinds of goal conflicts might lead a lady of the court to want revenge on another lady. This example shows how MINSTREL invents a conflict over possession of berries as a reason for wanting revenge.

5. Titles for MINSTREL's stories were provided by the author. Throughout this dissertation, MINSTREL's stories are presented exactly as produced, except for typography.

3.12.2.1 The Problem

This example looks at how MINSTREL invents a reason for Jennifer to want revenge on Darlene. MINSTREL's representation of this problem is shown in Figure 3.17. Jennifer's goal of wanting revenge (&GOAL.1751) is motivated by a state of the world (&STATE.992) that achieves Darlene's goal (&GOAL.3029) at the expense of Jennifer's goal (&GOAL.2112). (One of the interesting storytelling aspects of this story is that MINSTREL knows that Jennifer wants revenge before it knows why Jennifer wants revenge. This is a consequence of how MINSTREL develops stories from the theme outward, and is discussed in more detail in Chapter 5.)

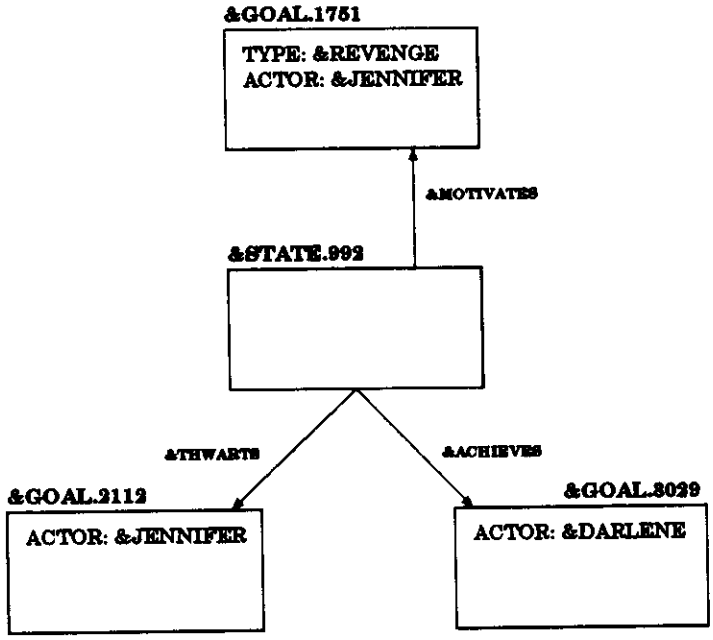


Figure 3.17 Initial Problem Specification

When this example begins, MINSTREL has three goals: (1) to instantiate the state that causes Jennifer to want revenge (&STATE.992), (2) to instantiate Darlene's goal achieved by this state (&GOAL.3029), and (3) to instantiate Jennifer's thwarted goal (&GOAL.2112).

3.12.2.2 Episodic Memory

For storytelling, MINSTREL's episodic memory contains ten story fragments from the King Arthur domain. For this example, the only relevant episode is "Picking Berries":

Picking Berries

A lady named Guinevere who wanted berries went to the woods and picked some.

3.12.2.3 Standard Problem-Solving During Storytelling

MINSTREL uses creative problem solving to instantiate Jennifer's thwarted goal (&GOAL.2112 in Figure 3.17). But before MINSTREL does this, it instantiates the state which thwarts this goal (&STATE.992 in Figure 3.17). To instantiate this state, MINSTREL uses standard case-based problem-solving.

TRAM:Standard-Problem-Solving is a TRAM that implements standard case-based problem-solving. Given a problem description, TRAM:Standard-Problem-Solving tries to recall from episodic memory an exactly similar problem. If it can, it uses the solution from that previous problem to solve the current problem. TRAM:Standard-Problem-Solving is illustrated in Figure 3.5.

In this case, the problem description is &STATE.992: "Something happens which fullfils a princess's goal". Without transformation, this recalls a similar episode from memory: the "Picking Berries" story fragment. This scene is used to fill in as much of &STATE.992 and surrounding schemas as possible. The scene development at this point is shown in Figure 3.18. Notice that this has also resulted in the addition of a new schema to the story, &ACT.1178. This new schema represents Darlene's action in picking the berries.

This examples illustrates how standard case-based problem-solving occurs in MINSTREL. TRAM:Standard-Problem-Solving recalls an episode from memory and applies it without modification to the current problem. What happens when TRAM:Standard-Problem-Solving cannot recall an appropriate episode?

3.12.2.4 Creative Problem-Solving During Storytelling

MINSTREL now tries to instantiate Jennifer's thwarted goal. To do this, MINSTREL must show how Darlene's possession of the berries could thwart one of Jennifer's goals. MINSTREL's author-level representation of this goal is &GOAL.3067, and Figure 3.19 shows a trace of MINSTREL achieving this goal. The first part of this trace shows MINSTREL recalling author-level plans for instantiating a story scene. (See Chapter 7 for further discussion of instantiation.) MINSTREL recalls 4 plans. The first, ALP:Dont-Instantiate, fails. The second, ALP:General-Instantiate, succeeds. ALP:General-Instantiate tries to instantiate a story scene by using creative problem-solving at the character level. The second portion of the trace (beginning with "TRAM Cycle: &GOAL.2112") shows creative problem-solving being used to instantiate the scene.

ALP:General-Instantiate passes the story scene to be instantiated to problem-solving. TRAM:Standard-Problem-Solving is tried but fails, because MINSTREL does not have any

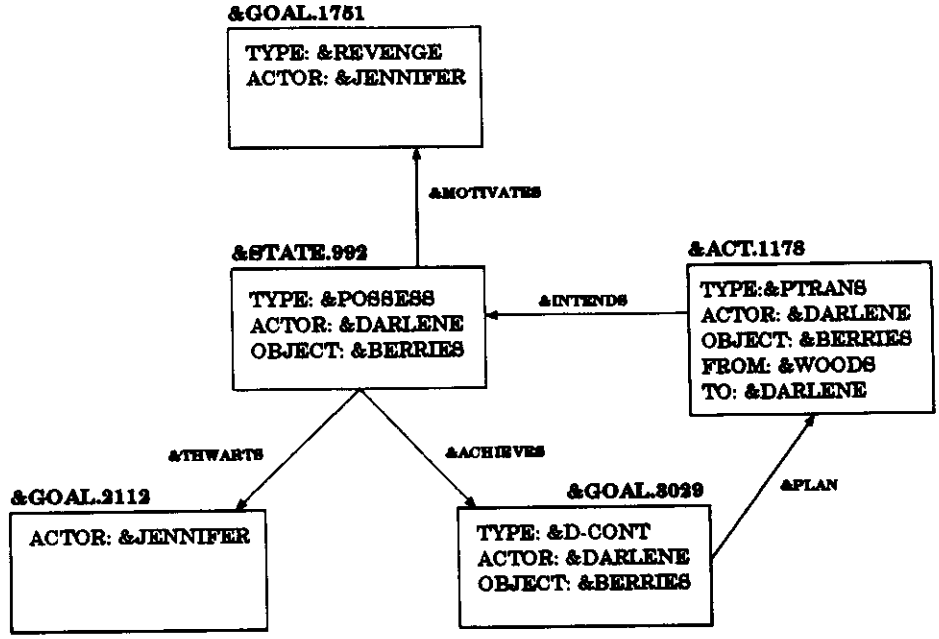


Figure 3.18 Intermediate Scene State

```
Author-level goal &INSTANTIATE applied to &GOAL.2112.
Recalling plans for &INSTANTIATE: 4 plans.
Trying author-level plan ALP:DONT-INSTANTIATE.
  ...plan fails.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
TRAM Cycle: &GOAL.2112.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    NIL.
   ...TRAM failed.
   Executing TRAM:OPPOSITE-STATE-ACHIEVES.
     Recalling:    (&GOAL-BERRIES).
   ...TRAM succeeds: (&GOAL.3155).
TRAM Cycle succeeds: (&GOAL.3155).
Found a reminding in ALP:GENERAL-INSTANTIATE:
Author-level planning succeeded.

(JENNIFER WANTED TO HAVE THE BERRIES *PERIOD*)
```

Figure 3.19 MINSTREL Trace of Story-Level Creativity

scenes in episodic memory in which a lady's goal is thwarted by someone else possessing some berries. (In fact, MINSTREL's memory does not contain *any* scenes in which a lady's goal is thwarted.)

TRAM:Standard-Problem-Solving fails, so it is discarded. The next TRAM used is TRAM:Opposite-State-Achieves. This TRAM suggests that the opposite of a state that achieves a goal will thwart the goal, and vice versa. If being healthy achieves the goal of protecting your health, then being dead will likely thwart the goal of protecting your health, and so on. So to invent a thwarted goal, you can recall an achieved goal and reverse it.

In this case, the opposite of the thwarting state (Darlene's possession of the berries) is Darlene not possessing the berries (i.e., someone else possessing the berries). TRAM:Opposite-State-Achieves changes the problem specification from "A princess's goal is thwarted by Darlene possessing some berries" to "A princess's goal is *achieved* by *someone* possessing some berries". If TRAM:Opposite-State-Achieves can recall something similar to this new specification, it can be used in the original problem by reversing the recalled scene from achievement to thwarting.

MINSTREL constructs this opposite state and tries to recall goals from episodic memory that are achieved by this new state. This recalls &GOAL-BERRIES, which is Guinevere's goal of wanting to possess berries from the "Picking Berries" episode: "Guinevere's goal of possessing berries is achieved by Guinevere possessing the berries".

The recalled episode can be adapted to the original problem by filling Jennifer and Darlene into the correct roles and "reversing" the outcome. This is achieved in three steps: 1) replacing the actor of the goal with Jennifer (i.e., "Jennifer's goal of possessing the berries is achieved by Guinevere possessing the berries"), (2) replacing the possessor of the berries with Darlene (i.e., "Jennifer's goal of possessing the berries is achieved by Darlene possessing the berries"), and (3) by replacing the achievement with thwarting (i.e., "Jennifer's goal of possessing the berries is thwarted by Darlene possessing the berries")

The adapted solution can now be used to fill in the original scene. The result is shown in Figure 3.20.

In English, the scene shown in Figure 3.20 is expressed:

> ... Jennifer wanted revenge on a lady of the court named Darlene because Darlene had the berries which she picked in the woods and Jennifer wanted to have the berries.

Notice what has happened during the creation of this thwarted goal. Prior to creating this goal, MINSTREL had no explicit knowledge about conflicts of possession, or of the idea that one person's possession of an object prevents another person from also possessing the object. By using a very simple story episode and a general creativity heuristic, MINSTREL was able to invent these concepts and apply them to a specific problem. And as these concepts are invented they are indexed into episodic memory, where they are available for future problem-solving, or as a basis

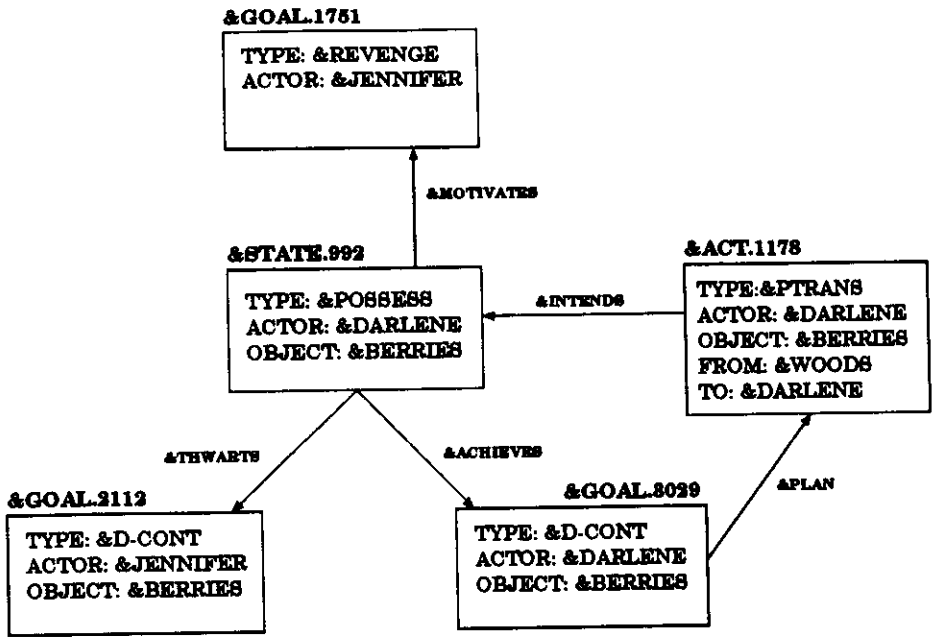


Figure 3.20 Finished Story Scene

for additional creativity. In this way, MINSTREL uses creativity to constantly expand its knowledge, and avoids having to re-invent the same concepts over and over.

3.12.2.5 Summary

This example demonstrates MINSTREL's use of problem-solving and creativity during story-telling. By using TRAM:Opposite-State-Achieves, MINSTREL solves a problem it cannot solve during standard problem solving. In solving this goal, MINSTREL discovers the idea of conflict over possession of an object. This is a concept not explicitly known to MINSTREL before its invention in *The Vengeful Princess*, and illustrates how MINSTREL's creativity can extend its knowledge.

3.13 Issues in Creativity

This chapter has no doubt raised in the readers mind a number of issues about creativity and MINSTREL's model of the creative process. This section addresses the more common questions about MINSTREL's creativity model.

3.13.1 Errors in Creativity

The previous section showed how MINSTREL used creativity to invent a scene in which Jennifer wants revenge on Darlene because Darlene has some berries that Jennifer wants. Readers of this scene sometimes complain that possession of berries is insufficient motive for revenge. MINSTREL's creative problem-solving has invented an incorrect solution, at least according to some readers. How did this happen, and what does it say about the creative process?

When MINSTREL begins telling *The Lady's Revenge*, it has only minimal knowledge about revenge. MINSTREL knows that revenge can be motivated by a thwarted goal, but it does not have any specific examples of revenge in the King Arthur domain in memory, and neither does it have any examples of thwarted goals in the King Arthur domain. Creativity must be used to extend MINSTREL's knowledge about both thwarted goals and revenge.

MINSTREL's creativity heuristics correctly extend its knowledge about possession of objects and thwarted goals. The scene in which Darlene takes the berries that Jennifer wants was created by TRAM:Opposite-State-Achieves from a scene that contained no thwarted goals. Initially MINSTREL knows that possessing an object can achieve a person's goal of controlling an object. After telling *The Lady's Revenge*, MINSTREL has discovered that possessing an object can also thwart another person's goal of controlling an object.

The same creativity heuristic is less successful in extending MINSTREL's knowledge about revenge. The thwarted possession of berries is invented as motivation for a revenge goal, but as most people recognize, that is probably insufficient motivation. Revenge is only plausible if the retribution is commensurate with the offense, or if the character seeking revenge is evil and likely to seek revenge out of porportion with the offense.

Creativity extends a problem-solver's knowledge. By making good use of what he already knows, a problem-solver can often make accurate guesses about what he doesn't know. But sometimes he'll be wrong. That is the nature of creativity. Unlike first-order predicate logic and standard problem-solving, creativity is a heuristic activity that trades infallibility for power. Creativity is able to discover many solutions that standard problem-solving cannot because it take risks. In this case, MINSTREL makes a reasonable guess about how the world operates - that possession of an object prevents possession by another, and that thwarting a possession goal is reason for revenge - but the guess is not completely correct.

Of course, this particular error could be easily corrected. MINSTREL could be given additional knowledge about revenge or possession of objects, and hence avoid this error. As this points out, creativity alone cannot extend a problem-solver's knowledge indefinitely or without error. The problem-solver must also be able to learn directly from outside experience. In a complete cognitive model, creativity must act in concert with non-creative learning to correctly extend a problem-solver's knowledge of the world.

But no matter how much knowledge a problem-solver has, creativity will extend beyond the borders of that knowledge, and errors of this type will continue to occur. The challenge is to find a

model of creativity that will limit errors while still providing the power to discover novel, useful ideas.

MINSTREL's model of creativity limits errors by using creativity heuristics that make specific, small problem transformations. By limiting the changes made to a problem specification, MINSTREL increases the likelihood that any solution it discovers will apply correctly to the original problem. The success of this strategy is apparent even in MINSTREL's failures. "Possessing berries" may a poor reason for revenge, but not an *unreasonable* one. The reader may judge it insufficient motivation, but he does at least understand the reasoning.

It is worth noting that in a complete cognitive system, errors in creativity would lead to learning experiences. If MINSTREL were able to learn from criticism, the reader's comments about the possession of berries being insufficient motivation for revenge would be an opportunity for MINSTREL to refine its knowledge about revenge and possession of objects, and avoid similar mistakes in the future. Creativity can thus be seen as a motivator and director of non-creative learning. At this time, however, MINSTREL has no ability to learn from criticism.

3.13.2 Learning in MINSTREL

Although MINSTREL does not have the ability to learn from criticism, it is able to do a simple sort of learning from creativity. When MINSTREL invents a new solution to a problem, the solution is indexed in episodic memory according to the problem it solves. So when MINSTREL discovers that a conflict over possession of berries is a reason for revenge, that new knowledge is stored in episodic memory, where it can be used in future problem-solving. This serves several purposes.

First, it permanently extends MINSTREL's knowledge. If MINSTREL did not remember its past inventions, it would always begin its creative problem-solving from the same base of knowledge, and would always explore the same area around the edges of its knowledge. But if a useful new solution is discovered and incorporated into MINSTREL's knowledge base, the new solution becomes an island from which MINSTREL can continue its creative exploration. By saving its creative successes, MINSTREL increases its ability to discover new solutions. The larger MINSTREL's episodic memory, the vaster its experience, the more powerful and effective its creativity.

Second, saving successful new solutions improves MINSTREL's efficiency as a problem-solver. Standard case-based problem-solving is very efficient, because it finds solutions which apply immediately and with a minimum of effort. Creativity is less efficient, because it must search the problem space and apply solution adaptations. By remembering past solutions, MINSTREL avoids having to re-invent them, and this improves MINSTREL's efficiency.

Both these benefits require MINSTREL to save *successful* solutions. Saving solutions which are wrong or even doubtful (like the "possession of berries" solution) can be counter-productive, because it will lead to MINSTREL repeating its past mistakes. It should be obvious, then, that crit-

icism is an important element to a creative problem-solver that learns from its creativity. Although MINSTREL does not currently address this issue, it is an area for future research.

Experiments which study MINSTREL's ability to learn from creativity and the effect that has on MINSTREL's problem-solving behavior are presented in Chapter 15.

3.13.3 MINSTREL's Efficiency

Saving invented solutions improves MINSTREL's efficiency in the long run because it allows standard problem-solving to solve problems that would otherwise require creativity. But how can we characterize the efficiency of creativity in the short run, ie., for a particular problem-solving effort?

To begin with, creativity does not become inefficient as episodic memory grows. Retrieval from episodic memory is porportional to the number of significant features in the problem description, not upon the number of episodes in memory.[6] Episodic memory is organized as a tree based on the values of significant features of the indexed episodes ([Schank 1982][Kolodner 1984][Reiser 1986]). Retrieval involves comparing the significant features of the recall description with the branches of this tree (i.e., traversing a multi-branched tree). Since there is one comparison for each feature of the recall description, the time efficiency of recall is characterized by the number of features.

In fact, creativity tends to become *more* efficient as episodic memory grows. As memory grows, the likelihood that a transformed problem description will recall a solution increases, and so the likelihood of creativity succeeding also increases. There is no easy way to characterize this trend, because creativity does not search memory in an orderly fashion, and memory does not grow in an orderly fashion. But in general, the more the knowledge captured in episodic memory, the more likely it is that creativity will discover a solution. Some studies of how MINSTREL's behavior changes as episodic memory changes are discussed in Chapter 15.

The efficiency of creativity depends primarily upon the number of creativity heuristics that are applicable to a particular problem. If a creative problem-solver has three heuristics that apply to a problem, and tries every combination of these heuristics without finding a solution, the problem-solver will try 24 different combinations. In the worst case (when no solution is found), creativity is O((n+1)!), where n is the number of applicable heuristics.

In practice, a creative problem-solver is likely to limit the amount of time and effort expended to find a solution. MINSTREL, for example, applies no more than three heuristics simultaneously, regardless of how many are applicable. Because combinations of heuristics search the problem space farther and farther from the original problem specification, they are correspondingly less likely to discover a solution (although if they do it is likely to be quite different from known solutions). Consequently, the problem-solver soon reaches a point of diminishing return and aban-

6. We assume a constant-time hashing function to traverse the memory tree.

dons the search for a solution.

Experience with MINSTREL also indicates that because MINSTREL's creativity heuristics are very specific only a few of the available heuristics apply to any given problem. An analysis of the problems solved by MINSTREL in telling stories based on four different story themes revealed that of MINSTREL's 24 creativity heuristics, on average only 1.4 TRAMs applied to any particular problem. This indicates that MINSTREL's strategy of specific problem transformations not only increases the likelihood of discovering a useful solution, it also limits the effort expended in searching for a solution. (For experiments and studies on MINSTREL's usage of TRAMs, see Chapter 15).

3.13.4 Randomness in MINSTREL

A common question when people first hear about MINSTREL is "How does it make up the stories? Does it just make random choices?" As should be apparent to the reader of this chapter, MINSTREL solves problems (including the problem of telling a story) by purposeful, directed use of the knowledge in episodic memory. MINSTREL does not, in general, make random selections when problem-solving or being creative.

The only use of random selection in MINSTREL occurs when MINSTREL must select between equally likely alternatives. When MINSTREL has no way to distinguish two or more choices, it selects randomly between the choices. This can occur in two situations.

First, episodic memory may recall two (or more) episodes which fulfill the recall criteria. Episodic memory is organized by the significant features of episodes. Two episodes with identical significant features will therefore fall into the same category in memory, and be recalled together. (If more than a small number of episodes fall into the same category, episodic memory builds and returns a generalization based on those episodes. For more on how episodic memory is organized, see Chapter 2.) When this occurs, MINSTREL selects and uses one of the episodes randomly.

Second, MINSTREL selects randomly from its available plans and creativity heuristics when there is more than one applicable plan or heuristic. For any particular author-level goal there may be several author-level plans available. Similarly, for any particular problem-solving situation, there may be several applicable creativity heuristics. In these situations, MINSTREL selects the plan or heuristic to apply randomly.

It is important to note that while MINSTREL does select randomly in these situations, MINSTREL does not make random *decisions*. When MINSTREL selects between two recalled episodes, it has already decided what it needs, and both episodes will fulfill those needs. Similarly, when MINSTREL selects an author-level plan randomly from those that apply to an author-level goal, it has already decided the goal. So MINSTREL's random selections are not random decisions, merely a way to distinguish otherwise indistinguishable outcomes.

CHAPTER 4
A Dictionary of TRAMs

4.1 Introduction

This chapter is a dictionary of MINSTREL's creativity heuristics. It describes in detail each of MINSTREL's twenty-four TRAMs.

The purpose of this chapter is not to provide a defining or canonical collection of creativity techniques. Indeed, in many places this chapter discusses the shortcomings of MINSTREL's creativity heuristics, and no researcher with any understanding of creativity would suggest that twenty-four heuristics could encompass all the myriad methods of creativity.

Rather, this chapter is intended to give the reader a more in-depth understanding of the particular heuristics that have proven useful for MINSTREL's problem domain. This will not only give the reader a better understanding of how MINSTREL operates, it will hopefully also serve to spark the reader's curiousity about the processes of creativity. The hope is that these heuristics will serve as signposts to the types of knowledge necessary to be creativity. Some of these signposts lean rather drunkenly, but that too is useful and interesting.

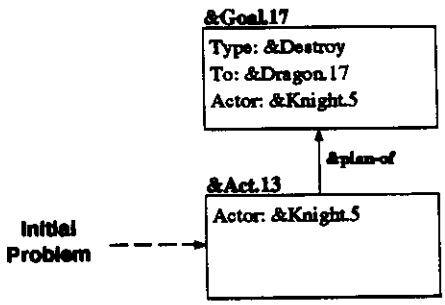
4.2 Organization and Format

In MINSTREL, problems are represented as partially instantiated networks of schemas. To represent the problem of finding a plan for a knight to kill a dragon, MINSTREL creates a schema to represent the knight's goal of making the dragon dead, and connects that via a &plan link to an act schema which represents the knight doing some unknown action. This schema represents the problem "What can a knight do to achieve the goal of making a dragon dead?" This is illustrated in Figure 4.1.

To solve this problem, MINSTREL must fill in the empty portions of the problem schemas. In this case, MINSTREL must fill in the type of action the knight is taking, what the action is directed towards, and so on.

MINSTREL's TRAMs are organized according to the problem schemas to which they apply. There are four classes of TRAMS: general, act, goal, and state TRAMs. General TRAMs apply to any kind of problem description; the others apply to particular schemas. By classifying TRAMs according to the schemas to which they apply, MINSTREL improves its efficiency in finding and applying TRAMs.

This chapter is divided into sections corresponding to the classes of TRAMs. General TRAMs are presented first, and then the TRAMs for each of the other classes.




```
A knight does something to destroy
a dragon.
```

Figure 4.1 Example Problem

4.3 General TRAMs

4.3.1 TRAM:Generalize-Role

TRAM:Generalize-Role suggests that a new solution to a problem can be found by changing the role of the actor in the problem to a similar role, solving the new problem, and then changing the role of the actor back to the original role.

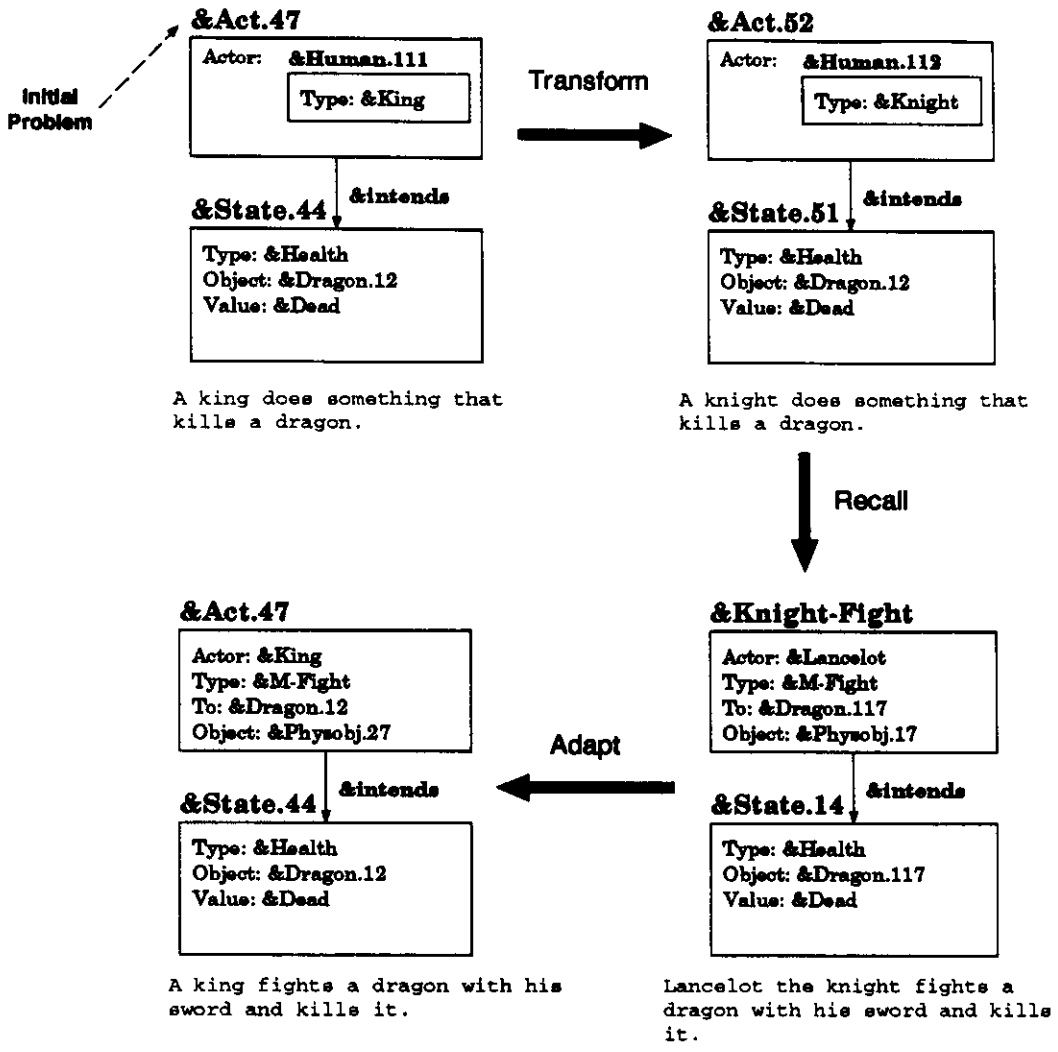
For example, to invent a way for a king to kill a dragon, TRAM:Generalize-Role suggests changing the king to a knight, finding a solution to the problem of a knight killing a dragon, and then adapting the discovered solution to the original problem by changing the knight back to a king. This example is illustrated in Figure 4.2.

Figure 4.2 is in two portions. The top portion illustrates, both graphically and textually, how TRAM:Generalize-Role transforms a problem specification into a new specification, and how it adapts a recalled solution to the new specification to fit the original problem. The bottom portion gives a brief algorithmic description of the heuristic. This format will be used throughout this chapter.

TRAM:Generalize-Role discovers new solutions by generalizing the actor of a problem description. This generalization must be done carefully, or MINSTREL will invent nonsensical solutions.

For example, suppose that MINSTREL was trying to create a scene in which a princess gained status with the king, and generalized the princess to a knight. Knights can gain status with the king by killing monsters. Consequently MINSTREL might invent a scene in which a princess gained status with the king by killing a dragon.

This mistake occurs because the generalization of a princess to a knight is too broad. To be c re-



TRAM:Generalize-Role

Comment:	Generalize the role of the actor.
Class:	General
Test:	Does the problem description have an actor?
Transform:	Change the role of the actor to a similar role.
Adapt:	Change the role of the actor in the solution to the original role.

Figure 4.2 TRAM:Generalize-Role

ative, generalizations must increase the scope of the problem without removing important constraints. In this case, the constraint that princesses don't use violence was removed when princess was generalized to knight. This resulted in a bad problem solution.

TRAM:Generalize-Role tries to preserve important constraints by generalizing within a class hierarchy of character roles. Classes group concepts that have many similarities. By generalizing the character role within similar classes, MINSTREL is more likely to find a solution that doesn't contradict any of the original problem constraints.

A portion of MINSTREL's class hierarchy was shown in Figure 3.12. By generalizing within role classes such as "Violent" and "Social", MINSTREL improves its chances of discovering a useful new solution.

Of course, even generalizing within a class hierarchy can lead to poor solutions. But any creative process will *necessarily* sometimes result in poor solutions. The goal of creativity is to reduce the occurence of poor solutions while retaining the power to create new solutions. By generalizing within class hierarchies, TRAM:Generalize-Role achieves this goal.

4.3.2 TRAM:Generalize-Actor

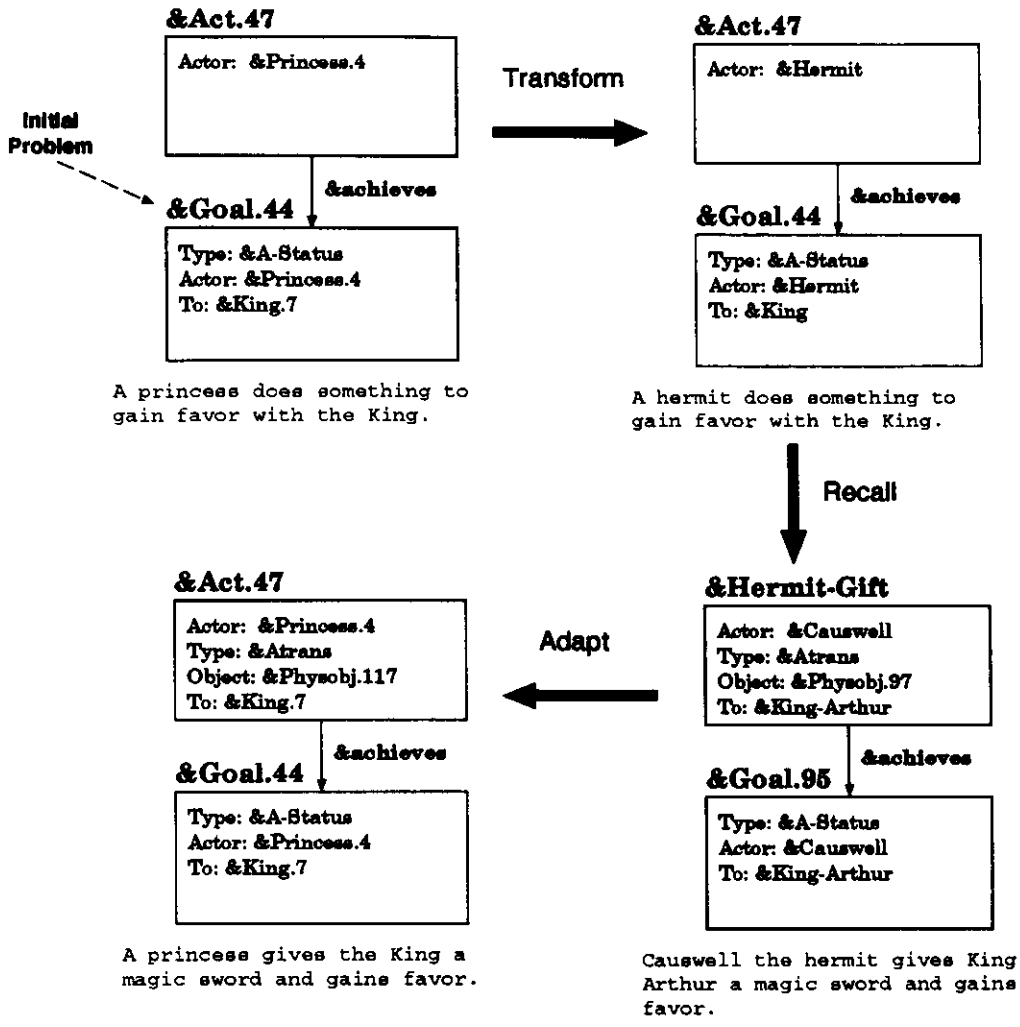
There are two difficulties with TRAM:Generalize-Role. First, it requires a pre-existing, static class hierarchy, which may not exist or which may categorize concepts along some irrelevant axis. Second, it begs the question of how such a hierarchy would be learned, or extended. A better solution would use the creator's experiences directly to guess at useful generalizations. TRAM:Generalize-Actor does this.

TRAM:Generalize-Actor uses episodic memory to suggest possible generalizations for the actor of an act, goal or belief schema. TRAM:Generalize-Actor begins by using episodic memory to find a typical action for that actor. For example, to generalize the princess in the "a princess does something to gain status with the king" problem, TRAM:Generalize-Actor tries to recall a typical action that a princess might make, such as picking berries in the woods.

When a typical action has been found, TRAM:Generalize-Actor tries to recall other actors who have also taken that action. In the case of the princess picking berries in the woods, TRAM:Generalize-Actor tries to recall other characters who have picked berries in the woods. This may recall a hermit, but is unlikely to recall a knight or the king.

If another actor can be found, that actor is used as a generalization in the original problem description. In this case, "hermit" would replace "princess" in the original problem description, resulting in the new problem "a hermit does something to gain status with the king". If this problem can be solved, the solution is adapted to the original problem by replacing the hermit with the princess. TRAM:Generalize-Actor is illustrated in Figure 4.3.

TRAM:Generalize-Actor performs a simple type of analogical reasoning. It discovers two con-



TRAM:Generalize-Actor

Comment:	Generalize the actor of a problem description.
Class:	General
Test:	Does the problem description have an actor?
Transform:	1. Find a typical action for the actor.
	2. Find another actor who has performed the same action.
	3. Substitute the new actor for the original actor.
Adapt:	Replace the new actor with the original actor.

Figure 4.3 TRAM:Generalize-Actor

cepts (actors) that are similar in one way (they have performed a common action) and guesses that they may be similar in other ways (sharing a solution to the original problem). Analogical reasoning is often mentioned as a source of creativity (e.g., [Koestler 1964]) and has been widely studied in cognitive science and artificial intelligence ([cites]).

The advantage of TRAM:Generalize-Actor over TRAM:Generalize-Role is that it does not require a static, pre-existing class hierarchy. By using episodic memory to discover appropriate generalizations, TRAM:Generalize-Actor avoids many of the difficulties of TRAM:Generalize-Role. Most importantly, TRAM:Generalize-Actor will become more powerful and useful as the scope of episodic memory increases.

4.3.3 TRAM:Limited-Recall

TRAM:Limited-Recall discovers new solutions by limiting the scope of the problem description.

In the King Arthur domain, problem descriptions are represented as goal, action and state schemas connected to one another by various types of links. For example, a problem description might consist of a knight performing an unspecified action that is connected by a "plan-of" link to the goal of being famous, and that goal might be connected by a "sub-goal" link to the goal to marry a princess. Together, these structures represent the problem "What action can a knight perform that will make him famous so that he can marry a princess?" This problem description is illustrated in Figure 4.4.

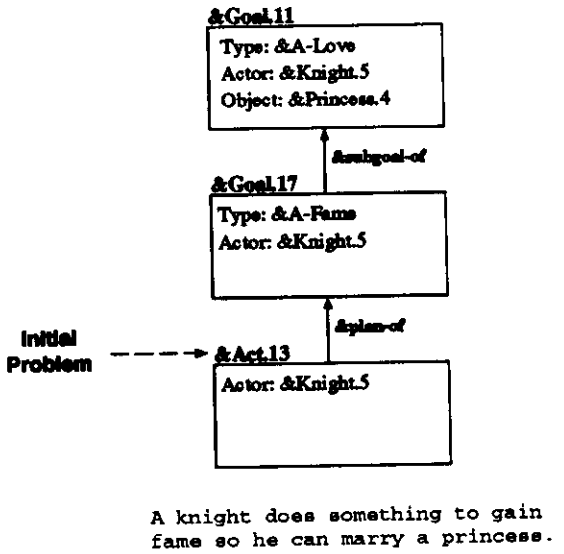


Figure 4.4 Example Problem Representation

During normal problem-solving, all of these schemas will be part of the problem description. To solve this problem, MINSTREL must recall a previous episode where a knight did something

that made him famous so that he could marry a princess.

TRAM:Limited-Recall suggests that a new solution can be found by limiting the problem description to only the most immediate constraints. These are defined as the original problem schema and all of its immediate neighbors. More distant schemas are removed from the problem description. In the case of the knight becoming famous shown above, TRAM:Limited-Recall removes the constraint that the action lead eventually to marrying the princess but leaves the constraint that the action lead to fame. Using TRAM:Limited-Recall, MINSTREL will find past solutions that led to fame, regardless of whether they were a part of plan to marry a princess. TRAM:Limited-Recall is illustrated in Figure 4.5.

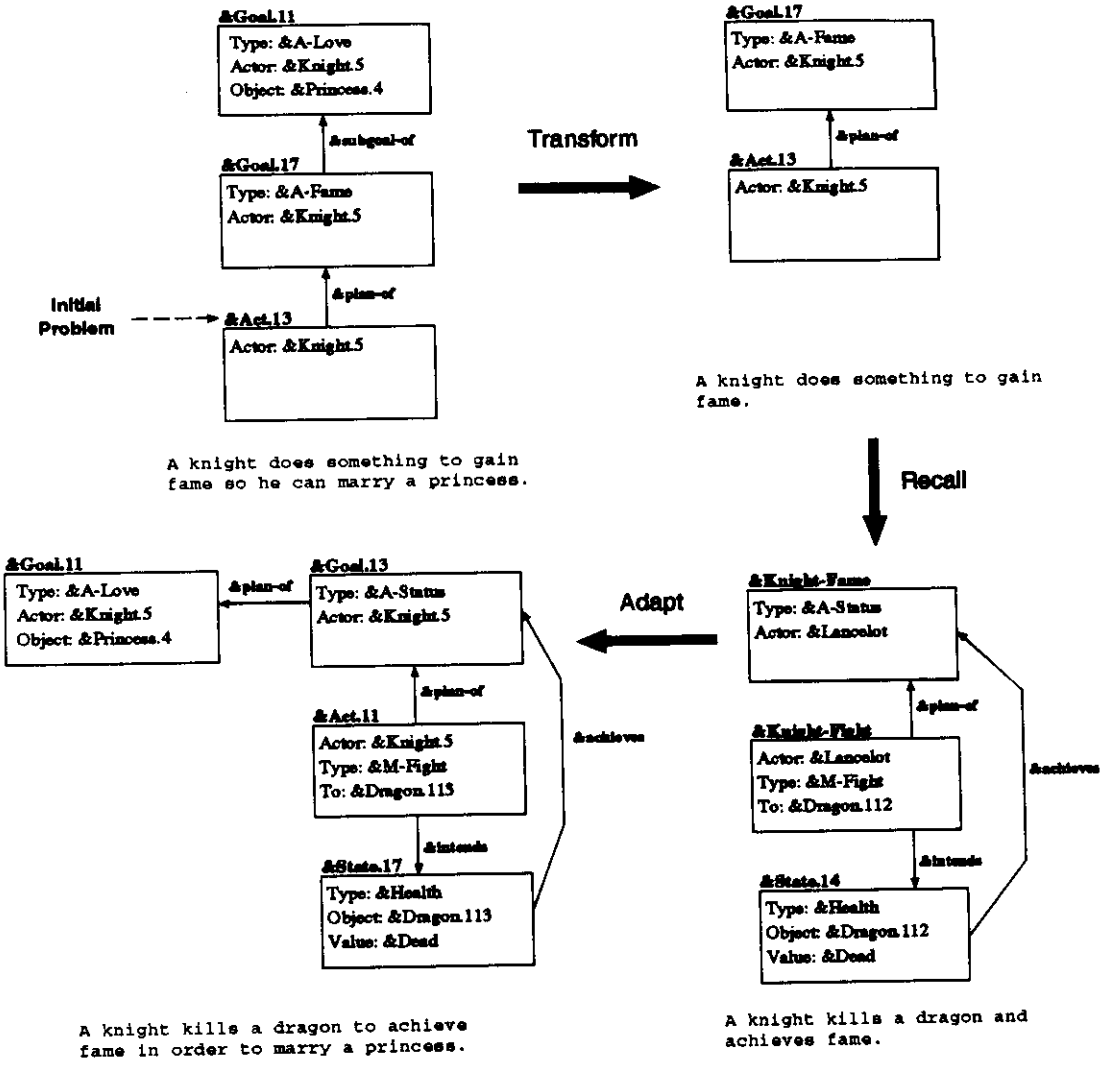
The primary difficulty with TRAM:Limited-Recall is that one of the removed schemas may be an important problem constraint. In general, however, the more distant the schema from the original problem, the less likely it is to represent an important constraint. An earlier failed version of this heuristic removed all schemas, including the immediate neighbors. This proved to be too sweeping a generalization, but the current implementation of TRAM:Limited-Recall seems to provide a good balance between power and possibility of failure.

4.4 Act-Based TRAMs

4.4.1 TRAM:Recall-Act

TRAM:Recall-Act is a specialization of TRAM:Limited-Recall that applies specifically to actions. TRAM:Limited-Recall modifies the problem description by removing all distant problem constraints. TRAM:Recall-Act extends this by also removing all immediate constraints, except for those particularly important to actions: the goal of the action and its intended effect. Given an action, TRAM:Recall-Act finds a solution by removing all connections to other schemas except for "plan-of" connections to goals and "intends" connections to states. Figure 4.6 outlines TRAM:Recall-Act. TRAM:Limited-Recall showed how removing unimportant problem constraints could lead to the discovery of a new problem solution. TRAM:Recall-Act extends this by using specific knowledge about acts to remove all but the two most important constraints for an action: the goal it achieves and the state it intends. By focusing only on the most important problem constraints, TRAM:Recall-Act broadens the space of applicable past problems without creating an undue number of creativity errors.

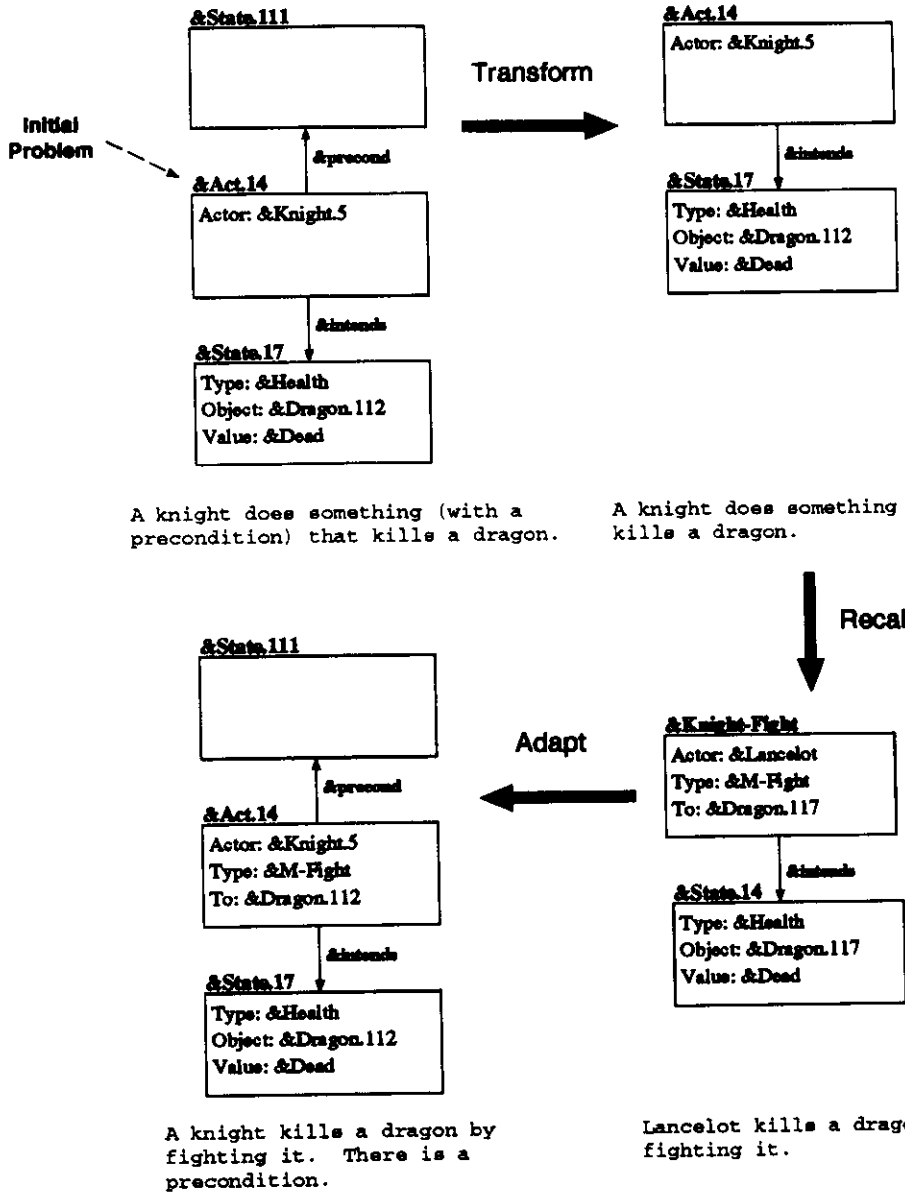
TRAM:Recall-Act illustrates how specialized knowledge about a domain can be used to increase the power and accuracy of creativity. Specific knowledge about acts is used to discover problem solutions that would not be found with only the more general (and more cautious) TRAM:Limited-Recall. This suggests that people may have a hierarchy of creativity heuristics, from general heuristics that apply weakly to a wide range of problems, down to specific heuristics that apply strongly to a single problem type.



TRAM:Limited-Recall

Comment:	**Find a reminding by removing distant connections.**
Class:	**General**
Test:	**Does problem description have any second-level connections to other knowledge structures?**
Transform:	**Remove all connections to second-level knowledge structures.**
Adapt:	**No adaptation.**

Figure 4.5 TRAM:Limited-Recall



TRAM:Recall-Act

Comment:	Remove all but the most important constraints.
Class:	Acts
Test:	Is the problem an act?
Transform:	Remove all connections to other schemas except "plan-of" connections to goals, and "intends" connections to states.
Adapt:	None necessary.

Figure 4.6 TRAM:Recall-Act

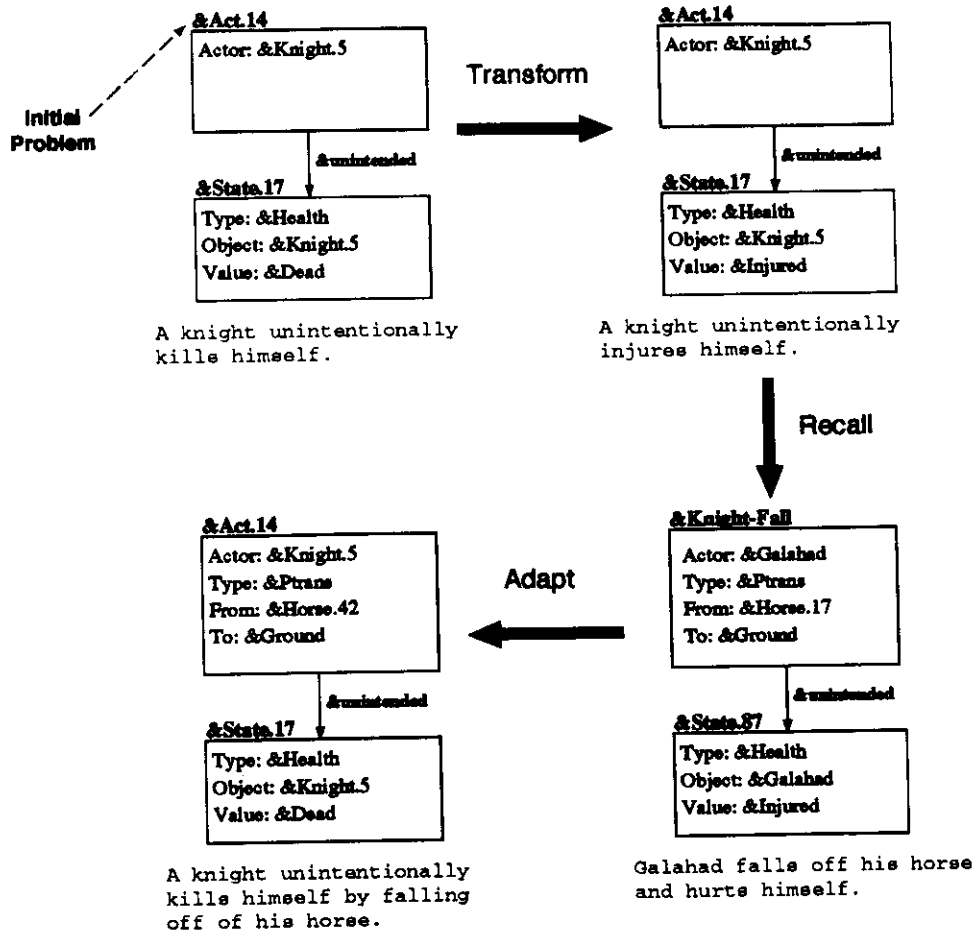
4.4.2 TRAM:Similar-Outcomes

TRAM:Similar-Outcomes suggests that if an action results in a particular outcome, it might also result in other, similar outcomes. If MINSTREL cannot recall an action that results in a desired outcome, TRAM:Similar-Outcomes suggests looking for actions that had outcomes similar to the desired outcome, and then using one of those actions to achieve the original outcome.

For example, suppose that MINSTREL is trying to invent a method for a knight to die, but does not know of any episodes in which a knight dies. TRAM:Similar-Outcomes suggests changing the outcome in the problem description to something similar. In this case, that results in changing the problem description from "a knight dies" to "a knight is injured". This recalls a scene in which a knight falls off his horse and breaks his arm. TRAM:Similar-Outcomes adapts this to the original problem by replacing the new outcome with the original outcome, resulting in a scene in which a knight falls off his horse and dies. This is illustrated in Figure 4.7.

The main issue in TRAM:Similar-Outcomes is how to find a state that is "similar" to the outcome in the original problem description. MINSTREL has two methods for achieving this task. These are implemented as separate TRAMS: TRAM:Similar-Outcomes-Partial-Change and TRAM:Similar-Outcomes-Implicit.

TRAM:Similiar-Outcomes-Partial-Change assumes that if an action can result in a partial relative change of a state then the action can also result in a complete change of the state, or vice versa. For example, something that makes someone ill (a partial negative change in health) might also kill them (a complete negative change in health). TRAM:Similar-Outcomes-Partial-Change allows a partial negative change to become a complete negative change, or vice versa, but not, for instance, a partial negative change to become a partial positive change. This limits creativity errors while maximizing the usefulness of this heuristic. This TRAM is illustrated in Figure 4.8. TRAM:Similar-Outcomes-Implicit assumes that two outcomes are interchangable in *every* situation if they are interchangeable in *any* situation. For example, if MINSTREL can recall a scene in which a knight fought and killed a troll, and another scene in which a knight fought and injured a troll, MINSTREL can use this knowledge to guess that injury and death are generally interchangeable, because they are interchangeable in the example situations. TRAM:Similar-Outcomes-Implicit uses the implicit representation of similarities in MINSTREL's episodic memory to substitute one outcome for another. In this way, TRAM:Similar-Outcomes-Implicit is very similar to TRAM:Generalize-Actor. TRAM:Similar-Outcomes-Implicit is illustrated in Figure 4.9.



TRAM:Similar-Outcomes	
Comment:	Exchange an outcome for a similar one.
Class:	Acts
Test:	Is the problem a state outcome?
Transform:	1. Find a similar state outcome. 2. Replace the original outcome with the similar one.
Adapt:	Replace the original outcome.

Figure 4.7 TRAM:Similar-Outcomes

4.4.3 TRAM:Act-of-a-Favor

TRAM:Act-of-a-Favor uses specific knowledge about favors to transforms acts that achieve a favor goal. TRAM:Act-of-a-Favor suggests that if you are trying to do a favor for someone by achieving one of their goals, then an action which achieves the subsumed goal will also achieve the favor goal.

TRAM:Similar-Outcomes-Partial-Change	
Comment:	Change a partial state change to a complete, or vice versa.
Class:	Acts
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 4.8 TRAM:Similar-Outcomes-Partial-Change

TRAM:Similar-Outcomes-Implicit	
Comment:	Exchange an outcome for a similar one.
Class:	Acts
Test:	Is the problem a state outcome?
Transform:	1. Generalize the type and value of the state's outcome. 2. Attempt recall from episodic memory. 3. Note the type and value of any recalled episodes. Assume that these are interchangeable with the original type and value. 4. Select one of the interchangeable type and value pairs and substitute it for the original type and value.
Adapt:	Replace the substitute type and value with the original type and value.

Figure 4.9 TRAM:Similar-Outcomes-Implicit

To better understand how TRAM:Act-of-a-Favor works, it is necessary to understand how MINSTREL represents favors. In MINSTREL, a favor is represented as a goal for one character to achieve the goal of another character. A graphical illustration of this is shown in Figure 4.10. In this figure, &GOAL.114 is a knight's goal of saving his life (&C-Health stands for "a critical health goal"). &GOAL.117 represents a hermit's goal of doing a favor for the knight, by helping the knight achieve his goal of saving his life. &ACT.47 is the hermit's plan to achieve his goal of doing the knight a favor, and &STATE.87 is the result of this action that will achieve the knight's goal of saving his life, and consequently the hermit's goal of doing the knight a favor. At this point, the knight's actions to achieve the favor have not been determined.

Now suppose that MINSTREL is trying to fill in &ACT.47 in Figure 4.10 ("The hermit does something that achieves his goal of doing a favor for the knight."). If MINSTREL has never previously encountered a situation in which a hermit did a favor for a knight, standard problem-solving will fail, because MINSTREL will not recall any previous similar problem situation. When this occurs, TRAM:Act-of-a-Favor suggests looking for an act that achieves the subsumed goal instead of an act that achieves the favor goal. In this example, MINSTREL looks for a memory that matches "A hermit does something that saves a knight's life."

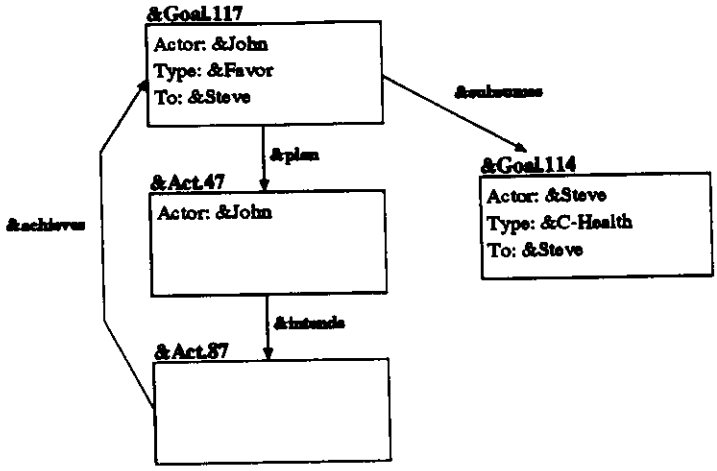


Figure 4.10 Representation of a Favor

This new problem description recalls a scene in which a hermit heals a knight in order to save the knight's life. TRAM:Act-of-a-Favor adapts this to the original problem by also making this act achieve the favor goal as well as the subsumed goal. TRAM:Act-of-a-Favor is shown in Figure 4.11.

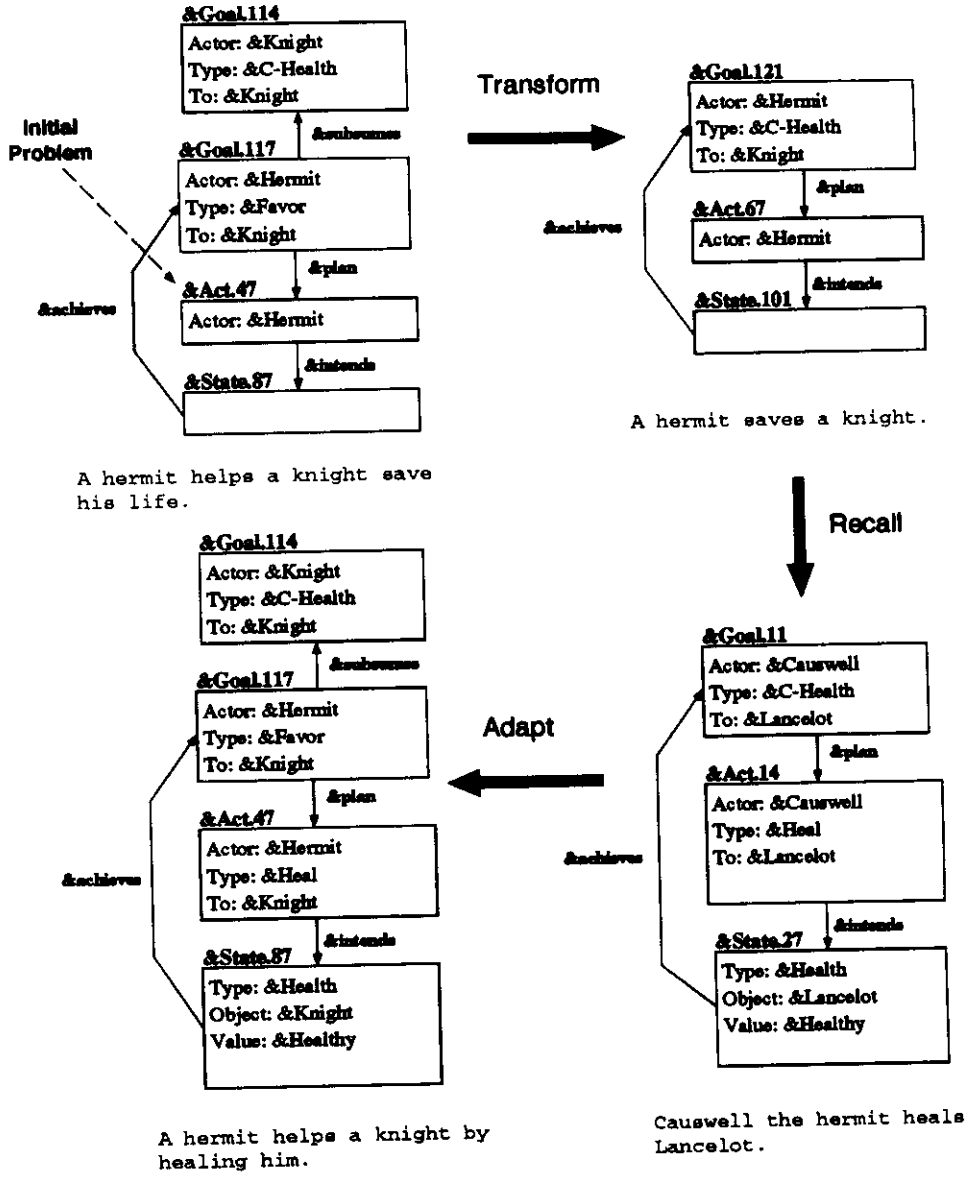
4.4.4 TRAM:Achieve-B-Motivated-P-Goal

TRAM:Act-of-a-Favor used knowledge about favors to transform problem specifications. TRAM:Achieve-B-Motivated-P-Goal uses knowledge about protection goals and beliefs to transform problems. Specifically, TRAM:Achieve-B-Motivated-P-Goal suggests that if you are trying to achieve a protection goal which is motivated by a belief, then it can be achieved by thwarting the mental event that supports the belief. Again, understanding this heuristic requires understanding MINSTREL's representation of beliefs and protection goals.

A belief schema represents something that a character believes is true about the world. There is no need to explicitly represent most things a character believes, because these beliefs are true or unimportant to the story. But sometimes a character's belief is central to the plot of a story, such as when Romeo mistakenly believes that Juliet is dead. In situations like these, MINSTREL explictly models character beliefs.

One type of belief MINSTREL models is a deductive belief[1]. A deductive belief represents the situation where a character perceives a state of the world and deduces that some action, state or goal has occurred. For example, a character witnessing a knight taking a horse might deduce that the knight has the goal of possessing a horse. The perceived state of the world is called the "evi-

1. MINSTREL also models predictive and expectation beliefs, but these beliefs are not pertinent to this discussion of TRAM:Achieve-B-Motivated-P-Goal.



TRAM:Act-of-a-Favor

Comment:	An action which achieves the subsumed goal also achieves the favor.
Class:	Acts
Test:	Is this an action that achieves a favor goal?
Transform:	1. Remove the favor goal from the problem specification. 2. Make the action achieve the subsumed goal.
Adapt:	Make the recalled action also achieve the favor goal.

Figure 4.11 TRAM:Act-of-a-Favor

dence" of the belief and the deduced action, state or goal the "mental event". An example of a belief is shown in Figure 4.12.

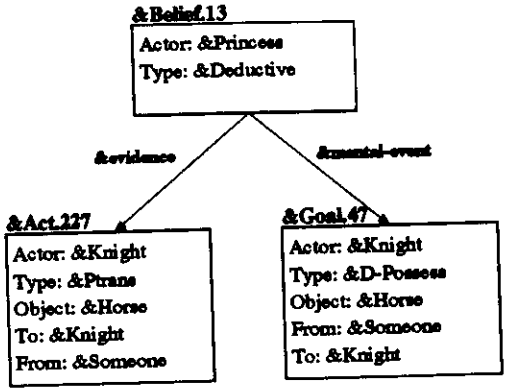


Figure 4.12 Example Deductive Belief

In Figure 4.12, &BELIEF.13 is the schema that represents a princess's deductive belief. &ACT.227 represents a knight taking possession of a horse. &ACT.227 is the evidence of &BELIEF.13. &GOAL.47 represents the knight's goal of gaining possession of a horse. &GOAL.47 is the mental-event of &BELIEF.13. Note that &GOAL.47 may or may not be true; it is a deduction that the princess has made based on seeing the knight take the horse. The knight may in fact have an entirely different motive for taking the horse. (He might, for example, be taking it to be shoed.)

The knight's goal of possessing the horse is called a "Delta" or "D" goal. It is one of a class of goals that involve the change of a state of the world to a mutually exclusive state (hence "Delta" for change). In this case, the change is in who possesses the horse. Another class of goals are "Protection" or "P" goals. These goals represent a character's desire to protect and maintain a state of the world when the character recognizes a situation that might result in a change in that state. For example, if a character is confronted with a dragon, he immediately has a goal to protect his good health, because he recognizes that the dragon may do something to hurt his health. This particular protection goal is called "P-Health".

TRAM:Achieve-B-Motivated-P-Goal applies to protection goals that are motivated by deductive beliefs. TRAM:Achieve-B-Motivated-P-Goal suggests finding a solution to the protection goal by finding something that thwarts the mental-event of the belief. In simpler terms, you can protect yourself from something you think might happen by preventing it from happening.

For example, TRAM:Achieve-B-Motivated-P-Goal would apply when a character sees a dragon, deduces it *may* eat him, and consequently has the goal to protect his life. In this case, TRAM:Achieve-B-Motivated-P-Goal suggests solving the P-Health goal by finding something that thwarts the mental event of the belief, which is the act of the dragon eating the character. This could lead MINSTREL to create a scene in which the character fed the dragon to prevent it

from eating him, or a scene in which the character ran away from the dragon, or any action at all that would thwart the dragon eating the character. TRAM:Achieve-B-Motivated-P-Goal is illustrated in Figure 4.13.

4.4.5 TRAM:Recall-wo-Precond

Preconditions are states that must be true before an action can be taken. For example, being located near someone is a precondition to speaking with them. During normal planning situations, the preconditions of an action cannot be determined until the action is determined, because the preconditions depend on the action. In the King Arthur domain, having a weapon is a precondition to fighting a monster, but not to picking flowers. So normally preconditions are determined and solved after the action they condition.

In storytelling, however, just the opposite can occur. The storyteller may know the preconditions before the actions, because he may be inventing the scene to purposely include a precondition. For example, one way MINSTREL knows to create tragedy is to kill a character before the character can do something important. One way to do this is to have the character killed achieving a precondition to the important act. In this situation, MINSTREL knows that there will be some precondition to the act, and that a result of attempting to achieve that precondition will be the death of the character, even before it knows what the act will be.

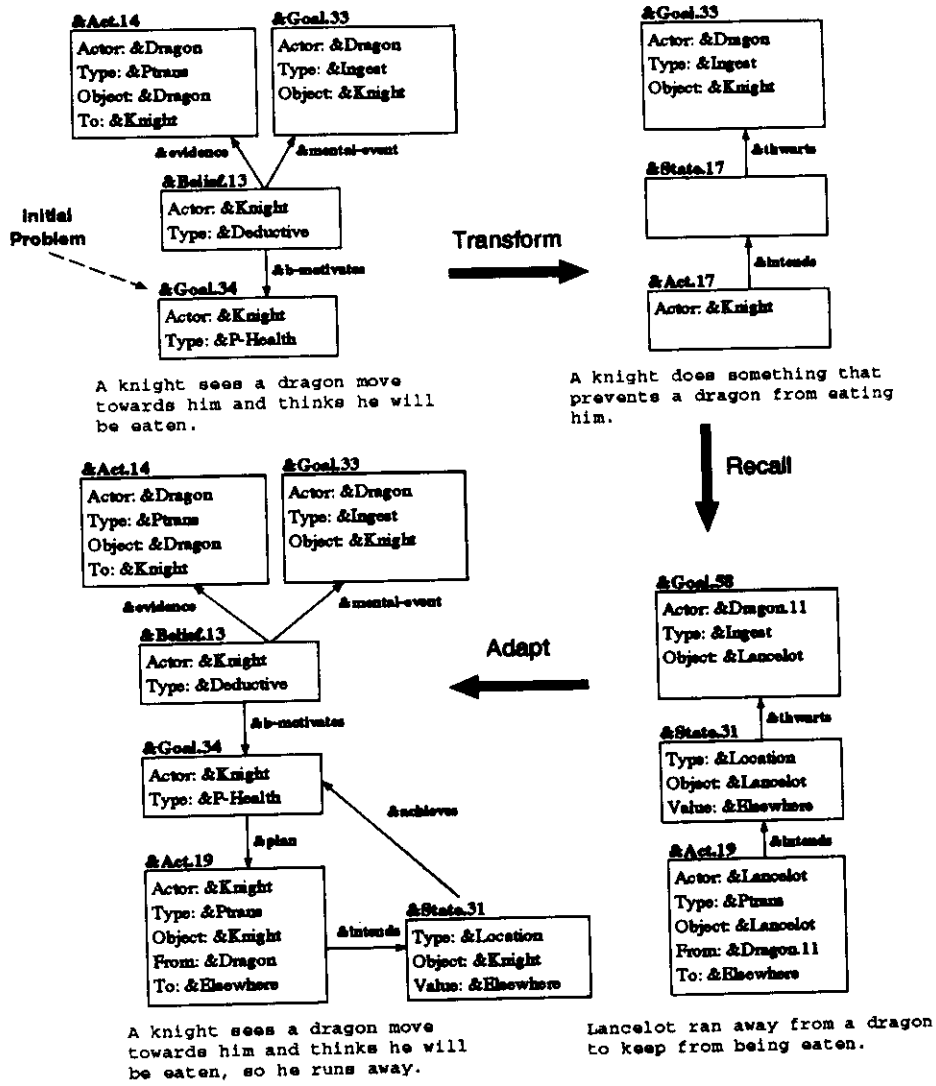
This situation – knowing a precondition before the action it conditions – is an example of an unusual planning situation that can arise during storytelling. A planner that learned all of its creativity heuristics solving day-to-day problems would never develop creativity heuristics to apply to this situation, because it occurs only in a specialized domain. This illustrates that creative planners may need problem domain specific creativity heuristics, and suggests that part of the effort required to learn a new problem domain may be in developing creativity heuristics for that domain.

TRAM:Recall-wo-Precond is a creativity heuristic for this particular situation. It suggests that uninstantiated preconditions of actions can be ignored when trying to create an action. Because preconditions are dependent upon the actions they condition, they cannot be instantiated until the action is instantiated. So they may be safely ignored while the action is being determined. Figure 4.14 shows a graphical representation of TRAM:Recall-wo-Precond. &ACT.117 is an action, and &STATE.214 is an unknown precondition. TRAM:Recall-wo-Precond suggests ignoring the unknown precondition while creating the action.

4.4.6 TRAM:Exists-As-Precond

Like TRAM:Recall-wo-Precond, TRAM:Exist-As-Precond applies to uninstantiated actions with preconditions. TRAM:Exist-As-Precond applies when the storyteller is trying to create a scene in which the existence of an actor is a precondition to an uninstantiated action.

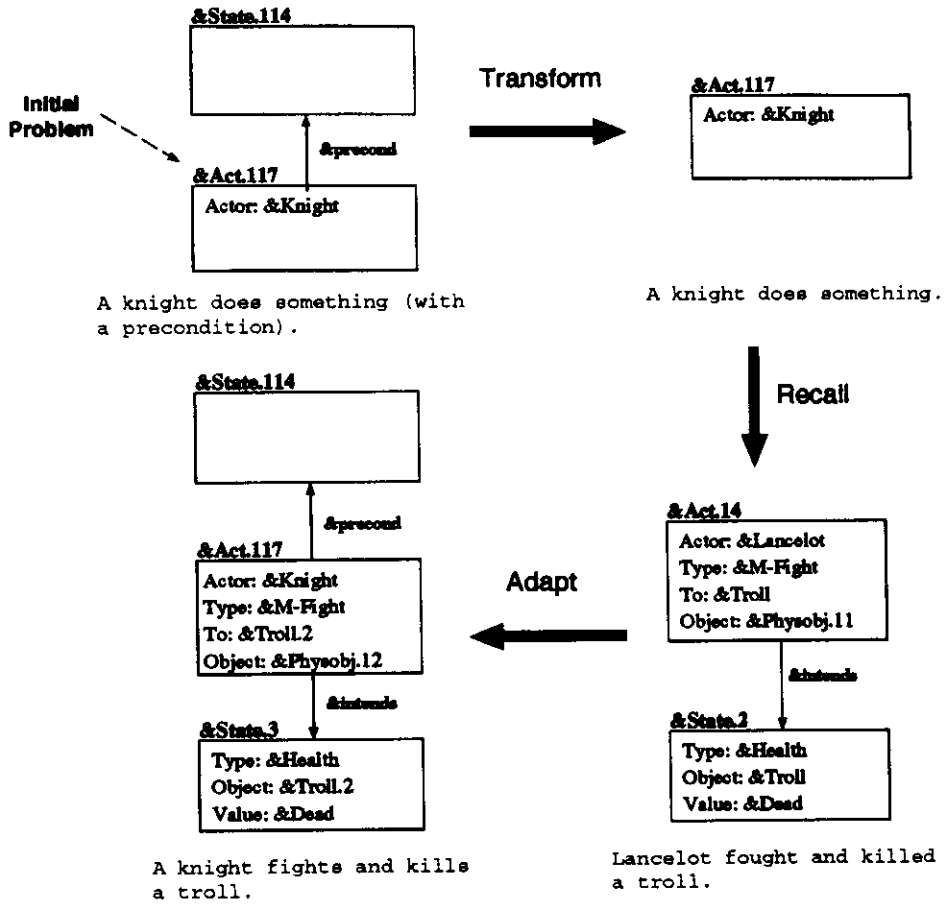
TRAM:Exist-As-Precond suggests that the existence of a character is a precondition to any ac-



TRAM:Achieve-B-Motivated-P-Goal

Comment:	Achieve a p-goal by thwarting the motivating belief.
Class:	Acts
Test:	Is this an action that achieves a p-goal motivated by a deductive belief?
Transform:	1. Create a new action which intends a state which thwarts the mental-event of the belief. 2. Use this as the new problem description.
Adapt:	Use the solution as an action that achieves the p-goal.

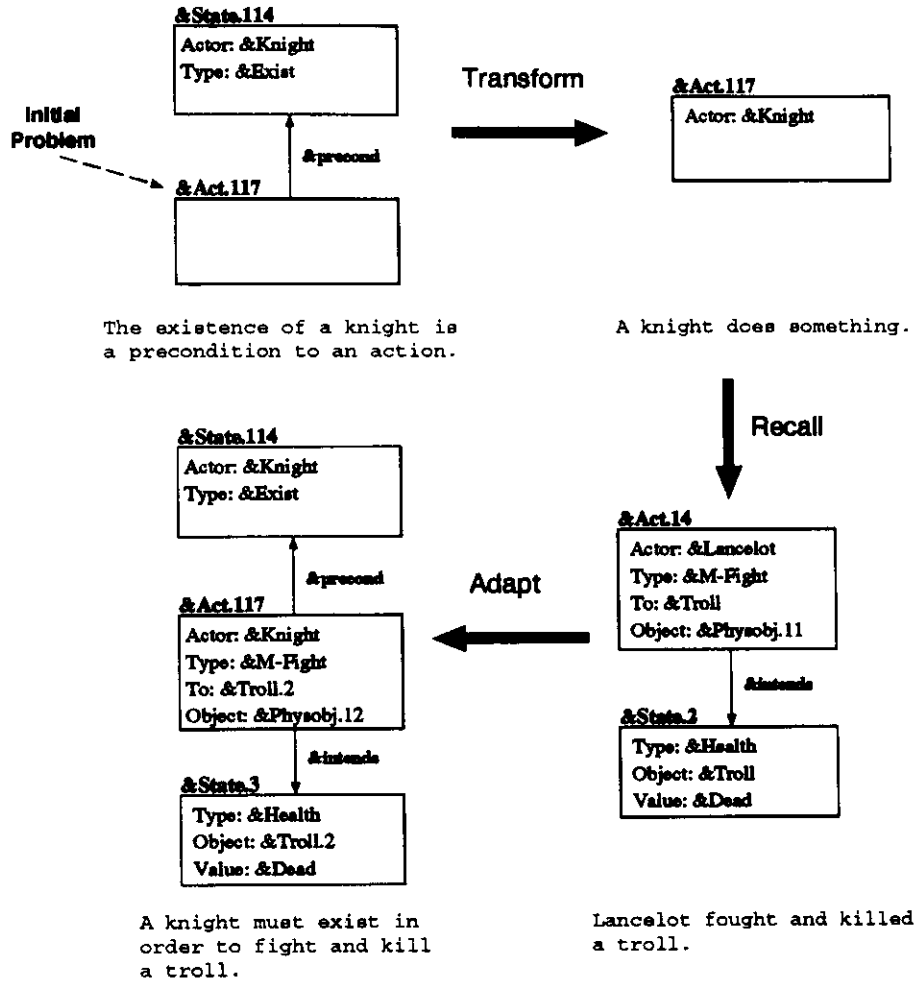
Figure 4.13 TRAM:Achieve-B-Motivated-P-Goal



TRAM:Recall-wo-Precond	
Comment:	Ignore the precondition of an action while solving it.
Class:	Acts
Test:	Is this an action with an unknown precondition?
Transform:	Remove the action's preconditions.
Adapt:	No adaptation necessary.

Figure 4.14 TRAM:Recall-wo-Precond

tion that character takes. So to create a scene which requires the existence of a character, make the scene involve an action by that character. TRAM:Exist-As-Precond is illustrated in Figure 4.15.



TRAM:Exist-As-Precond

Comment:	Existence of actor is precondition to an action.
Class:	Acts
Test:	Is a precondition of this action the existence of a character?
Transform:	1. Make the character the actor of this action. 2. Remove the precondition.
Adapt:	None needed.

Figure 4.15 TRAM:Exist-As-Precond

4.4.7 TRAM:Intention-Switch

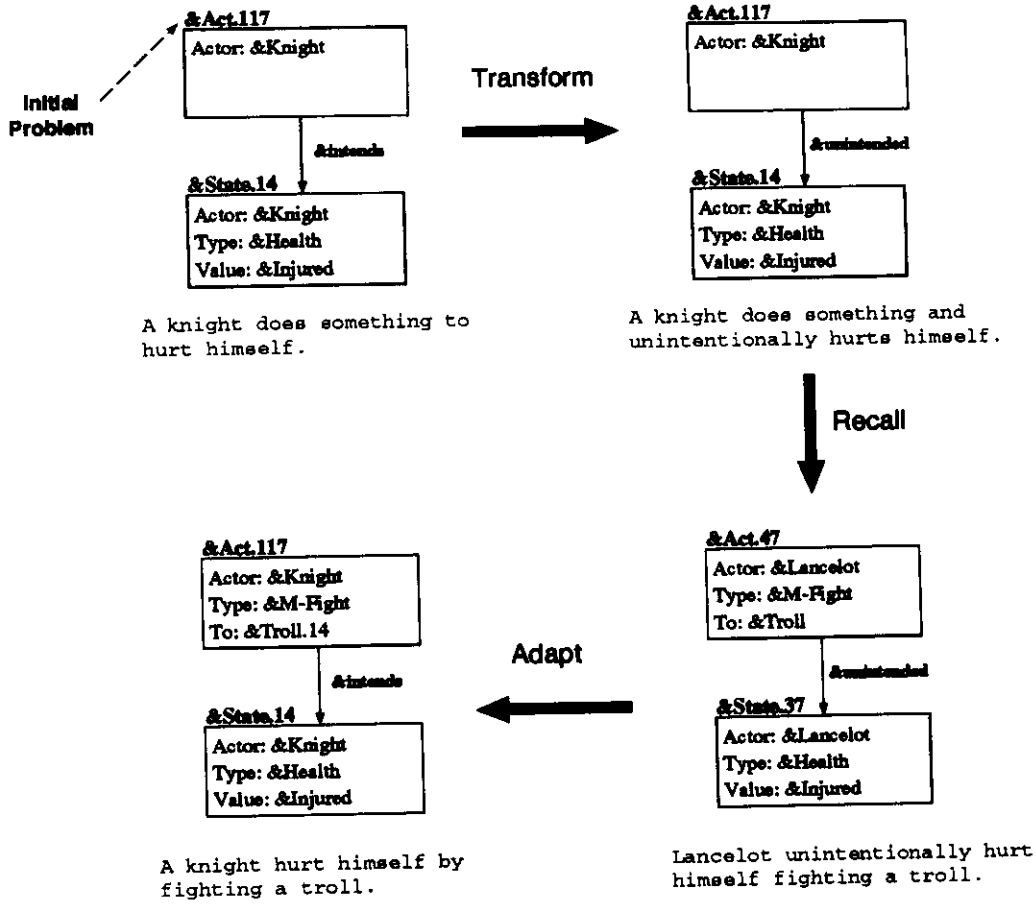
TRAM:Intention-Switch is a simple heuristic which applies to actions which have intentional effects. It suggests that if the effect of an action was intentional it might just as well have been unintentional.

TRAM:Intention-Switch is a creativity heuristic often used in day-to-day living as well as in the storytelling domain. For example, if you are doing your laundry and find yourself short a quarter, you may think to look in places where you've unintentionally found change before - under the cushions of your couch, on the floor of the back seat of your car, and so on.

Since intended and unintended outcomes are always interchangable, TRAM:Intention-Switch is simple to implement. Figure 4.16 shows how TRAM:Intention-Switch is implemented in MINSTREL.

4.4.8 TRAM:Create-Failure

TRAM:Create-Failure is a heuristic which suggests that a failure looks much like a success, except that any effects of a successful action should not be present on a failed action. If the problem is to create a scene in which a character fails to use a car, then this heuristic suggests looking for a scene in which a character uses a car successfully, and then removing any effects of that action, i.e., removing the change of location that resulted from using the car. Figure 4.17 illustrates TRAM:Create-Failure. TRAM:Create-Failure is limited because it does not capture the knowledge that a failed action may have detrimental effects as well as a lack of successful effects. If TRAM:Create-Failure is used to create a scene where a knight fights a dragon unsuccessfully, the scene created will not have the knight killing the dragon, but neither will it have the dragon killing the knight. A more complex and robust version of this heuristic would use a recursive call to problem-solving to try to discover some detrimental effects that might result from a failed action.



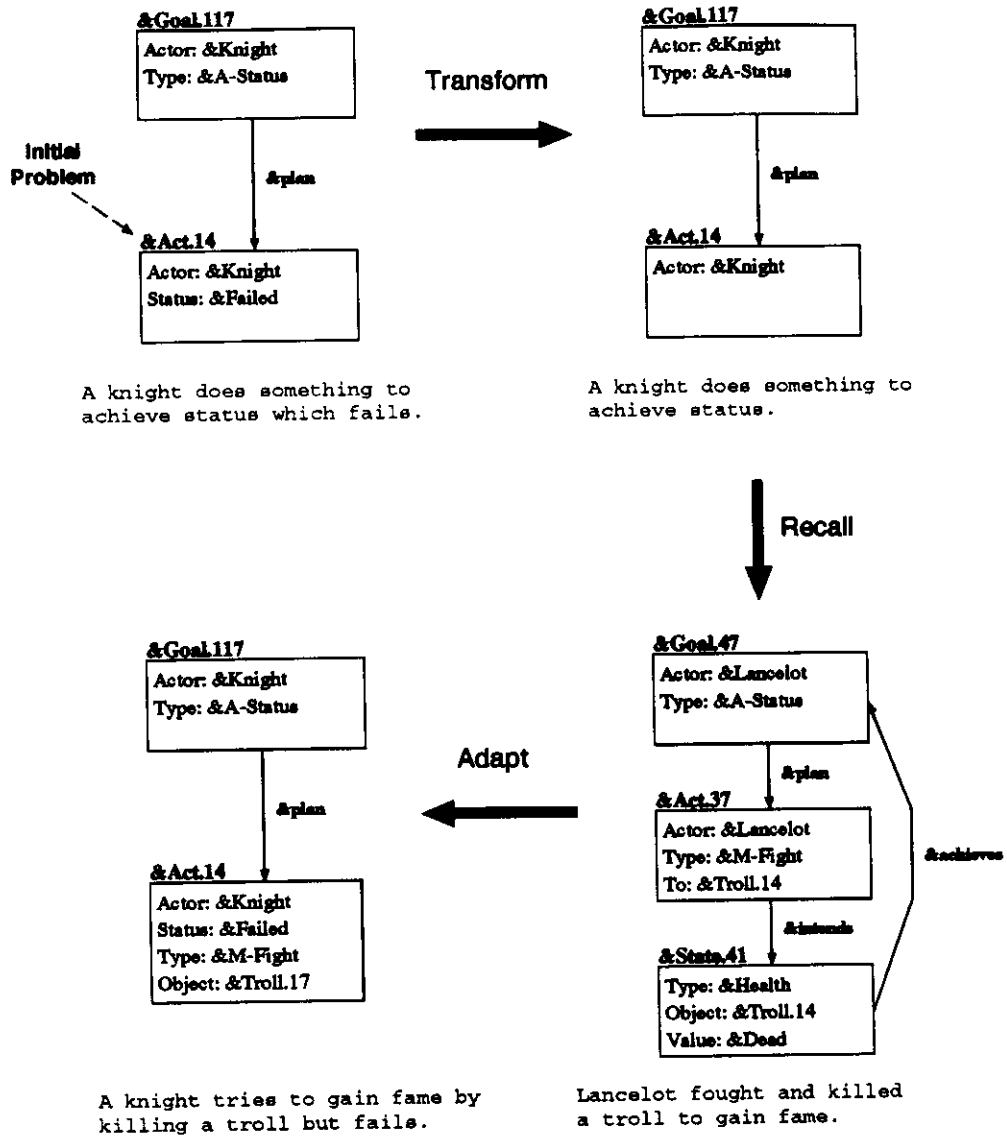
TRAM:Intention-Switch

Comment:	Look for an unintentional outcome instead of an intentional one.
Class:	Acts
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 4.16 TRAM:Intention-Switch

4.4.9 TRAM:Plan-Of-A-Different-Actor

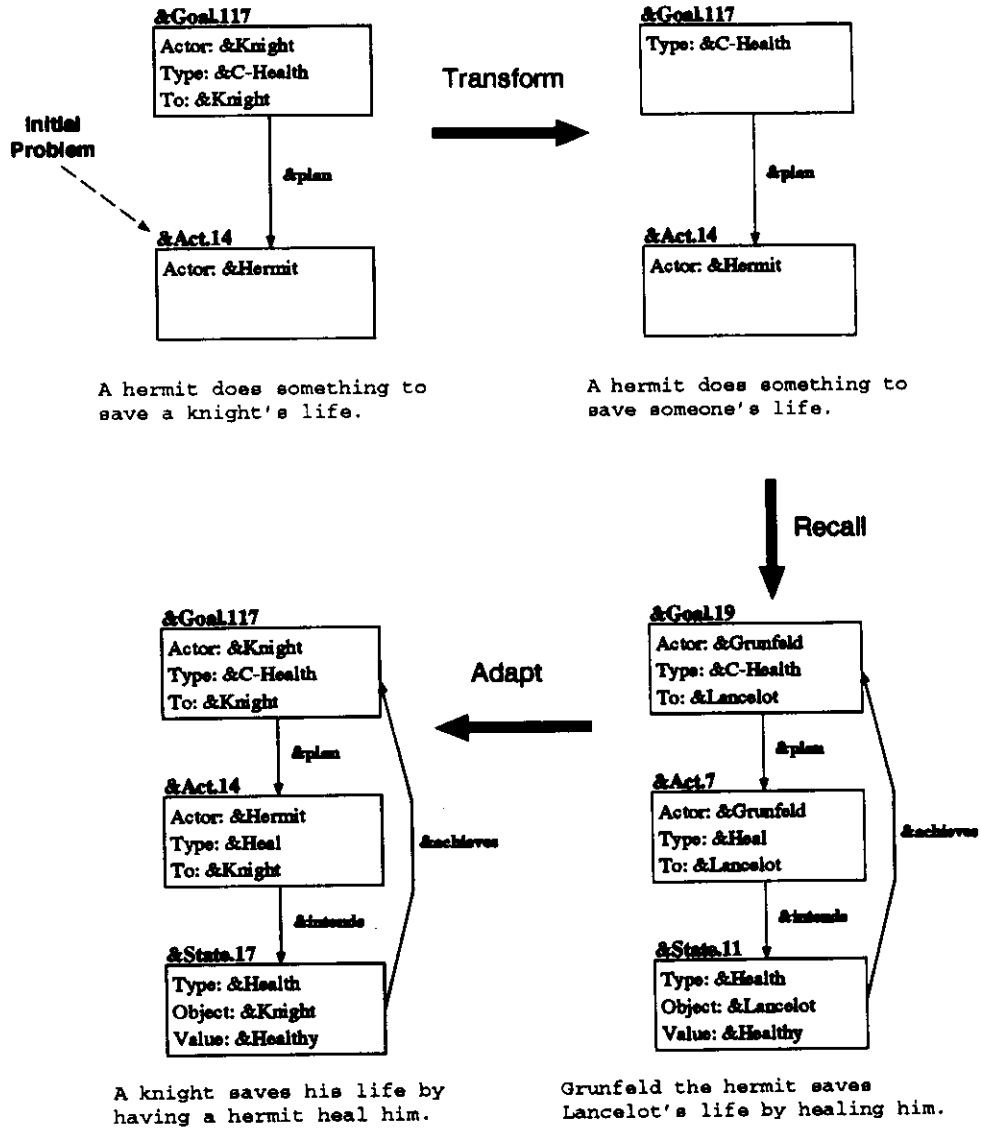
TRAM:Plan-Of-A-Different-Actor suggests that if the actor of the goal of an action is different from the actor of the action (as is the case when a character does a favor for another character) then the actor of the goal can be ignored while creating the action. In simpler terms, it doesn't matter for whom you perform an action, only that it achieves its goal. Figure 4.18 illustrates the problem transformation and outlines this TRAM. This TRAM permits MINSTREL to instantiate scenes in which one character does a favor for another character. Suppose that MINSTREL is



TRAM:Create-Failure

Comment:	A failure is a success without effects.
Class:	Acts
Test:	Is this a failed action?
Transform:	Change the failed action to a succesful action.
Adapt:	Remove any effects of the succesful action and mark it as a failed action.

Figure 4.17 TRAM:Create-Failure



TRAM:Plan-Of-A-Different-Actor

Comment:	Ignore for whom you are doing this action.
Class:	Acts
Test:	Is the actor of the goal different than the actor of the action?
Transform:	Replace the actor of the goal with the actor of the act.
Adapt:	Replace the original actor of the goal.

Figure 4.18 TRAM:Plan-Of-A-Different-Actor

trying to create a scene in which a knight does something to save a hermit from a dragon. This recalls nothing, because MINSTREL has never before read or created such a scene. But when TRAM:Plan-Of-A-Different-Actor is used, MINSTREL recalls a scene in which a knight killed a dragon to save *himself* from the dragon. When adapted to the original problem, this results in a scene in which the knight kills the dragon to save the hermit.

4.4.10 TRAM:Use-Magic

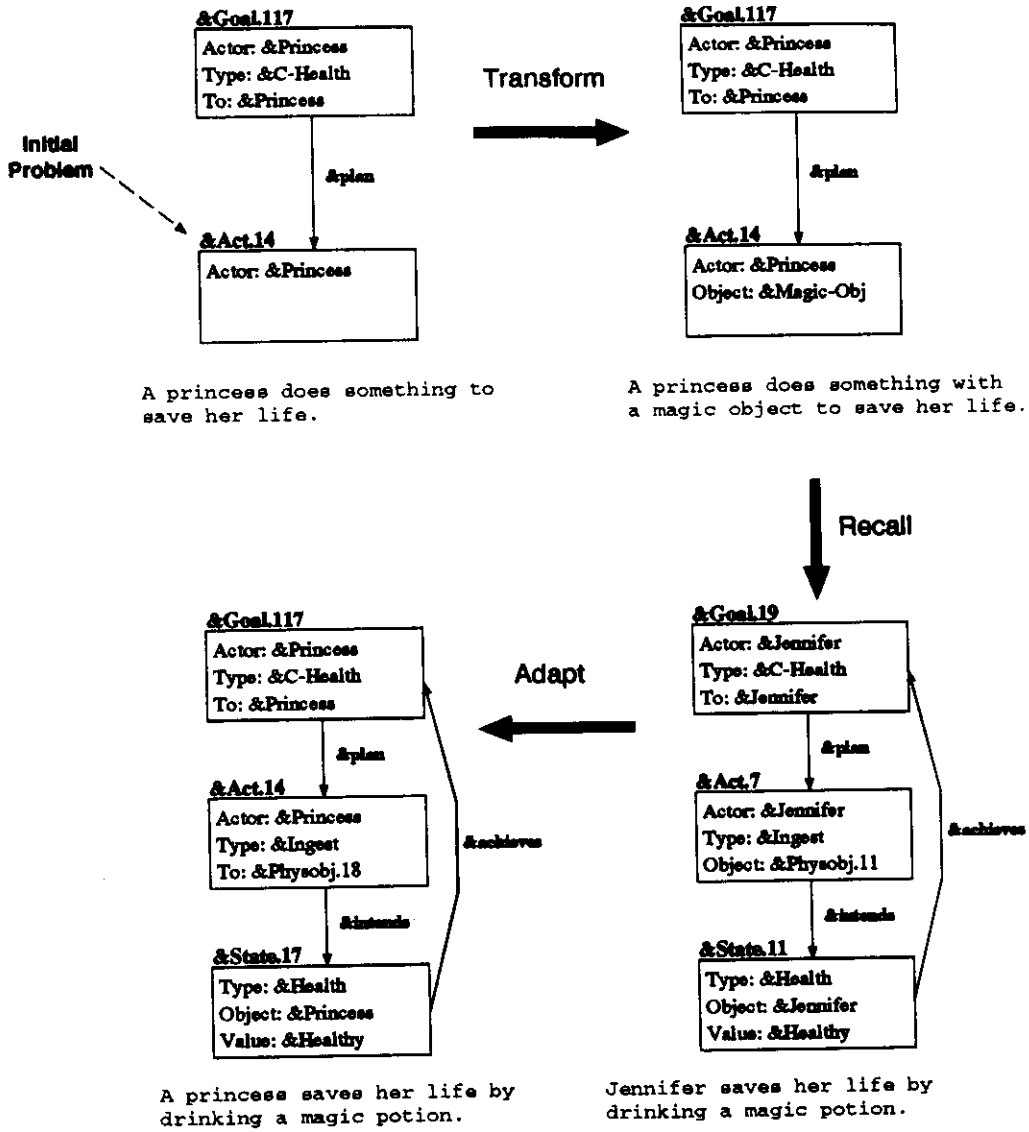
TRAM:Use-Magic is a creativity heuristic which applies specifically to the King Arthur domain. This heuristic suggests that an action might be achieved by using a magical object of some sort. To prevent MINSTREL from solving all story problems with magical items, this heuristic requires that the action being achieved have a goal, and that the goal is of a type previously solved using magic. Figure 4.19 shows the problem transformation used and outlines this TRAM.

There are two interesting facets to this TRAM. First, it is an example of a creativity heuristic specialized to a particular problem domain. Using magic is a type of creative thinking that is applicable to the King Arthur domain but not generally applicable to other problem domains. Second, this TRAM has the potential to be too powerful. Magic could be used to solve any problem. To prevent this, TRAM:Use-Magic has to restrict the range of problems to which it applies. It does this by requiring that magical items be specialized - that they apply only to the types of goals that they've applied to in the past.

When MINSTREL begins storytelling, it knows only one scene involving a magical item: the scene in which Juliet uses a magic potion to appear to be dead. In this scene Juliet uses the potion to achieve a health goal (the goal of being dead). Consequently, MINSTREL only applies magical items to achieve health goals.

However, MINSTREL is not limited to strictly matching the previous goal (Juliet's goal of being dead). Other TRAMs (such as TRAM:Plan-of-a-Different-Actor) can be recursively applied to a problem description created by TRAM:Use-Magic to modify the goal of the action. In fact, MINSTREL creates a variety of scenes in which magic potions are used to achieve various health goals. MINSTREL even creates a scene in which a princess uses a magic potion to save her life - the opposite of the example scene in MINSTREL's memory.

Although TRAM:Use-Magic has built-in constraints to prevent its overuse, the best solution to the general problem of restricting creativity heuristics is to use domain assessments. It would be possible, for instance, to create a domain assessment for storytelling in the King Arthur domain which would reject solutions which use magical items except in special situations (such as the climactic scene of a story, or if the character is a wizard). This technique can be generalized to other domains. In the day-to-day problem-solving domain, one possible solution to any problem is to get an agent to solve the problem for you. Rather than try to limit the applicability of this solution, it is better to have various domain assessments that will reject this solution when it is too costly, when it is difficult to find an agent, and so on.



TRAM:Use-Magic

Comment:	Use a magical item to solve a goal.
Class:	Acts
Test:	Is this an action to achieve a goal?
Transform:	1. Remove all intended and unintended results of the action. 2. Do not remove or change the goal of the action. 3. Make the object of the action be a magical item.
Adapt:	No adaptation necessary.

Figure 4.19 TRAM:Use-Magic

4.4.11 TRAM:Cross-Domain-Reminding

In general, MINSTREL's TRAMs make small changes to a problem description. Small changes to the problem description search the problem space near the original problem, where the likelihood of finding useful, applicable knowledge is high. Large changes to the problem description are also possible, but they must be specific changes that will lead the problem-solver to another area of the problem space where the likelihood of finding applicable knowledge is high.

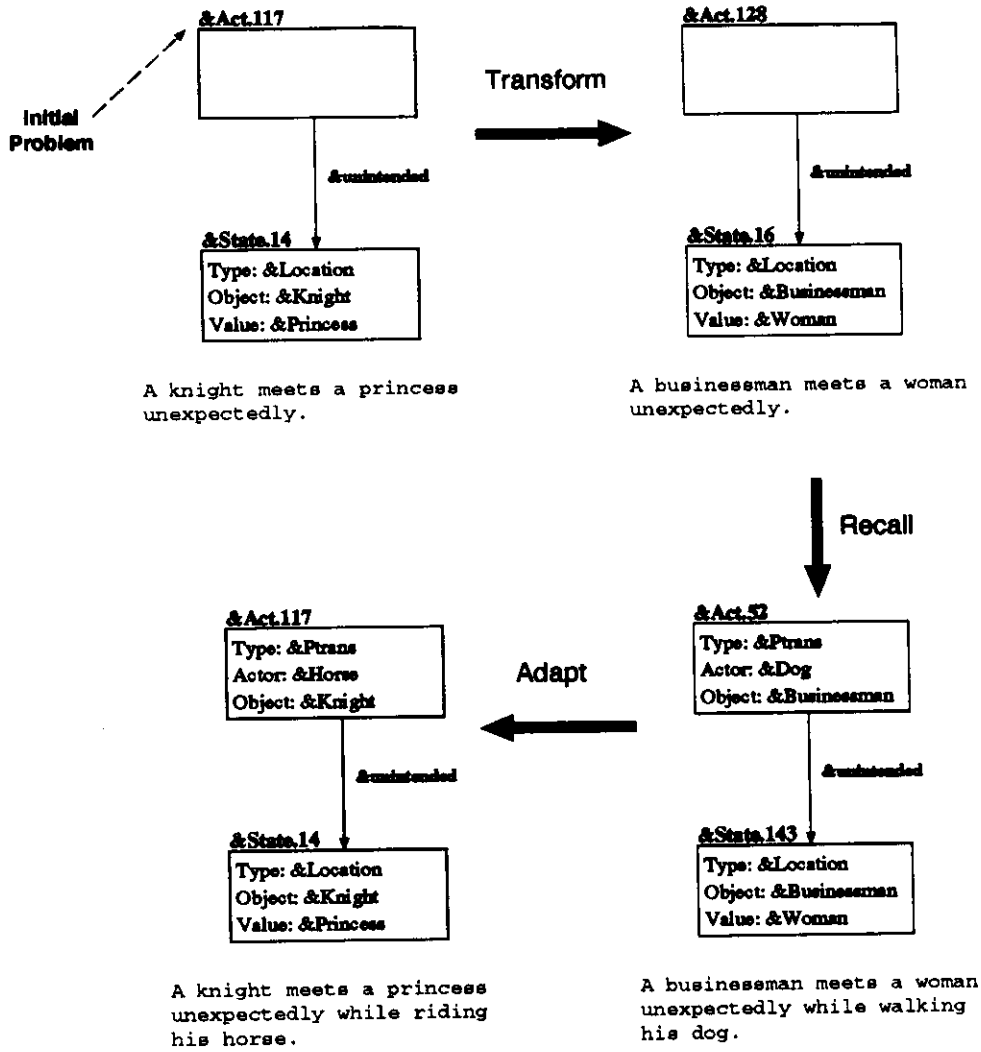
TRAM:Cross-Domain-Reminding is an example of a creativity heuristic that makes a large change to the problem description while maintaining the likelihood of finding a solution. TRAM:Cross-Domain-Reminding finds a problem domain with components analogical to the current problem domain, translates the current problem into the new domain, tries to solve the new problem in the new domain, and then translates any solution back into the original problem domain. There are two reasons this is likely to find a solution. First, the fact that the new domain has components analogical to the problem description suggests that the new domain will also have solutions analogical to the original domain. Second, the large transformation applied to the original problem is applied to all parts of the problem, so that there is no change in the structure of the problem description, only a translation of the contents of the structure to a new domain. The retention of a structural similarity improves the chance that any knowledge found in the new domain will apply to the original domain.

Figure 4.20 shows an example of TRAM:Cross-Domain-Reminding. In this example, MINSTREL is trying to create a scene in which a knight unexpectedly meets a princess. This is translated into the "modern day-to-day living" domain by mapping the knight to a businessman and the princess to a woman. This recalls an episode in which a businessman was out walking his dog when he ran into a friend. This episode is then translated back to the King Arthur domain, resulting in a story scene in which a knight meets a princess unexpectedly while out riding his horse.

There are a number of interesting issues in cross-domain reminding. Many of these are concerned with how analogies are recognized and learned, how they are indexed and recalled from memory, and how they are applied to particular problems. There is a large body of work in cognitive science and psychology concerning analogy. Some of the more important works are (cites).

Analogy has not been a research focus for MINSTREL. MINSTREL's implementation of TRAM:Cross-Domain-Reminding uses a simple list of problem domains and a similar list of paired concepts for mapping concepts between domains. Further, MINSTREL knows only one domain other than King Arthur: the domain of modern day-to-day living. To translate a problem description from the King Arthur domain to the modern domain, MINSTREL translates each element of the description according to the mapping list (i.e., knight -> businessman, etc.).

A more valid cognitive approach would use reminding and analogical reasoning to build and explore possible cross-domain solutions. But the focus of MINSTREL is on the creative results of analogy, not the precise mechanism involved, so for MINSTREL's purposes the more simple-



TRAM:Cross-Domain-Reminding

Comment:	Try solving the problem in a similar problem domain.
Class:	Acts
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 4.20 TRAM:Cross-Domain-Reminding

minded approach has sufficed.

4.5 Goal-Based TRAMs

4.5.1 TRAM:Favor-Goal

TRAM:Favor-Goal is similar to TRAM:Act-of-a-Favor. (For a description of TRAM:Act-of-a-Favor and an explanation of MINSTREL's representation of favors, see section 4.4.3.) TRAM:Act-of-a-Favor applied when the problem description was an act that achieved a favor goal. TRAM:Act-of-a-Favor instantiated the act of a favor by realizing that the act actually achieves the favor's subsumed goal. TRAM:Favor-Goal performs this same transformation upon the favor goal itself. To determine what actions are needed to achieve a favor, the problem-solver can look at the subsumed goal and determine what actions are needed to achieve that goal.

TRAM:Favor-Goal is illustrated in Figure 4.21. In this example, MINSTREL discovers a way for a hermit to do a favor for a knight by temporarily making the hermit the actor in the knight's &C-Health goal, i.e., by making the favor-doer the actor in the subsumed favor goal.

4.5.2 TRAM:Ignore-Subgoal

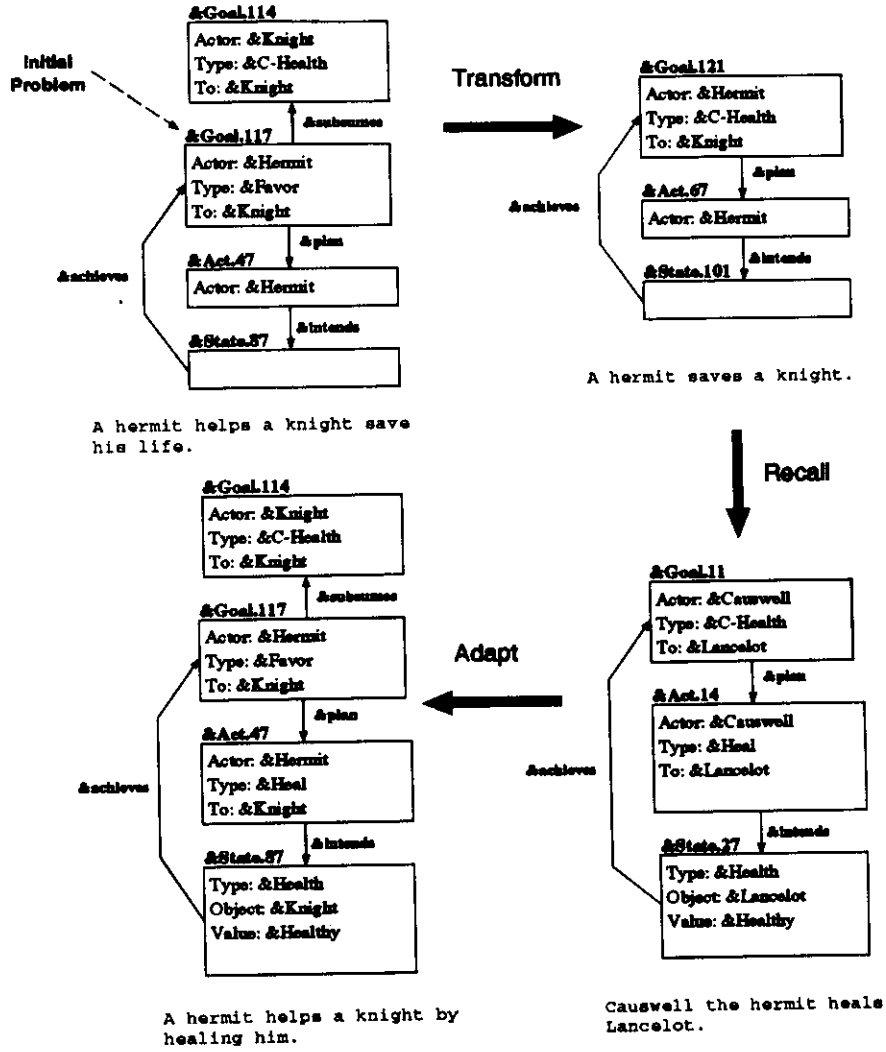
TRAM:Ignore-Subgoal suggests that when you are solving a goal, you can ignore the fact that it is a subgoal of another goal if you know what the purpose of this goal is. For example, if a subgoal of killing a dragon is travelling to where the dragon is, then the fact that it is a subgoal can be ignored; the solution found will achieve the goal regardless of its higher purposes. TRAM:Ignore-Subgoal is similar to TRAM:Limited-Recall, TRAM:Recall-Act and TRAM:Recall-wo-Precond in that it improves the chances of solving a problem by removing unnecessary knowledge that may be interfering with the recall of a solution.

TRAM:Ignore-Subgoal is illustrated in Figure 4.22.

4.5.3 TRAM:Opposite-State-Achieves

TRAM:Opposite-State-Achieves applies when MINSTREL is trying to instantiated a thwarted goal. It suggests that the opposite of a state that achieves a goal will thwart a goal. To create a state which thwarts a goal, TRAM:Opposite-State-Achieves suggests looking for a state which would achieve the goal, and then constructing the "opposite" of that state. MINSTREL currently knows how to construct opposites for scaled states such as health, and for exclusive states such as possession.

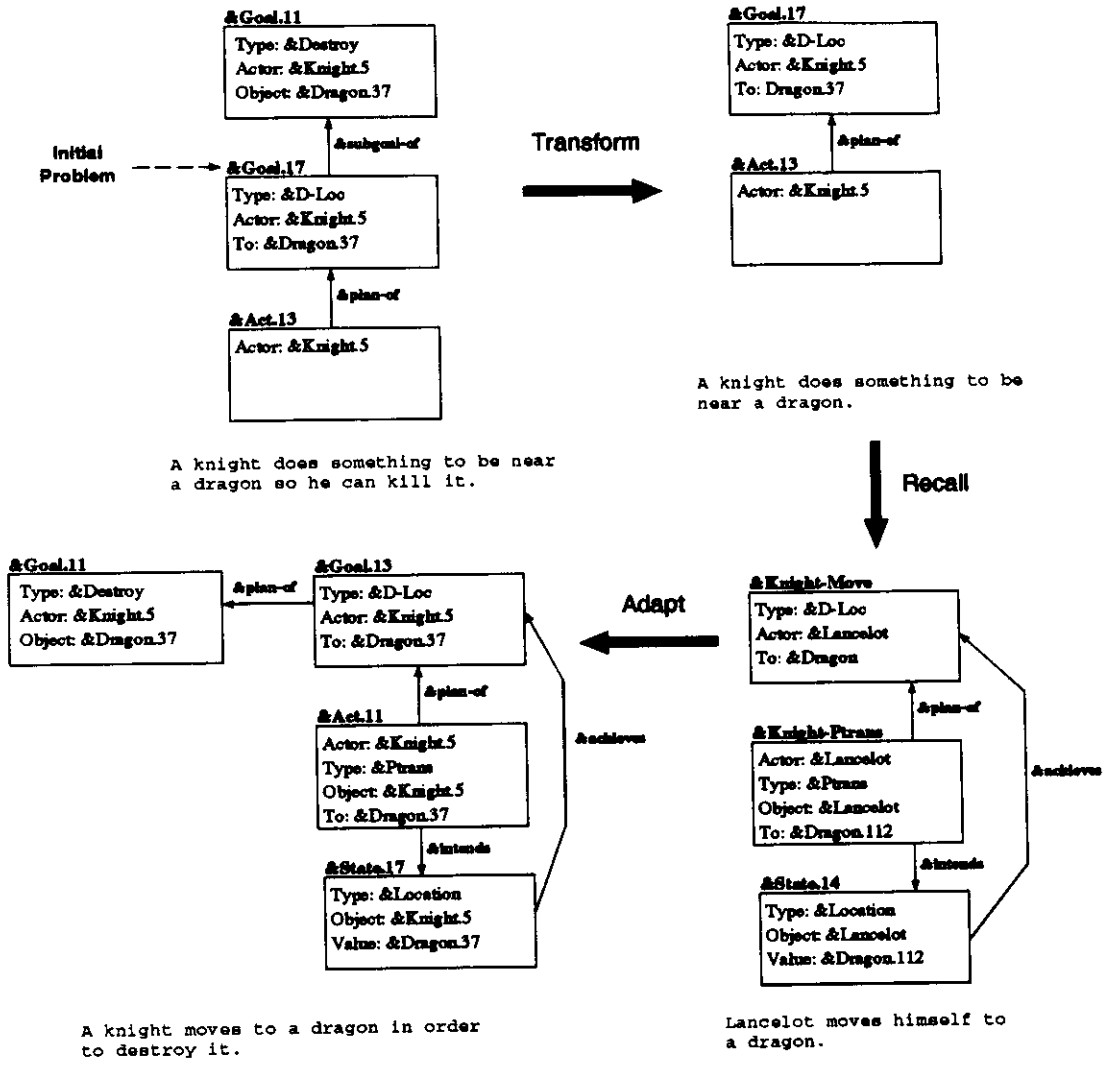
As an example of how TRAM:Opposite-State-Achieves works, suppose that MINSTREL is trying to create a state which thwarts a princess's goal of possessing some berries. TRAM:Opposite-State-Achieves suggests finding a state that would achieve this goal and then constructing the opposite of that state. MINSTREL recalls that the state of possessing the berries will achieve the princess's goal of having some berries. TRAM:Opposite-State-Achieves con-



TRAM:Favor-Goal

Comment:	To solve a favor, solve the subsumed goal.
Class:	Goals.
Test:	Is this a favor goal?
Transform:	1. Copy the subsumed goal of the favor and make this the new problem description. 2. Add to the new problem description an achieving action whose actor is the actor of the original favor goal.
Adapt:	1. Create a new favor goal that matches the original favor goal. 2. Remove from the solution the copy of the subsumed goal. 3. Attach the plan to achieve the subsumed goal to the original favor goal.

Figure 4.21 TRAM:Favor-Goal



TRAM:Ignore-Subgoal

Comment:	If a goal is a subgoal, and the goal is known, ignore the subgoal.
Class:	Goals.
Test:	Is this a goal an instantiated subgoal?
Transform:	Delete the superior goal of the original goal.
Adapt:	None necessary

Figure 4.22 TRAM:Ignore-Subgoal

structs the opposite of this state by noticing that it is an exclusive state and changing the possessor of the berries to an (unspecified) other character. This results in a scene in which a princess is thwarted in having some berries because someone else already has them.

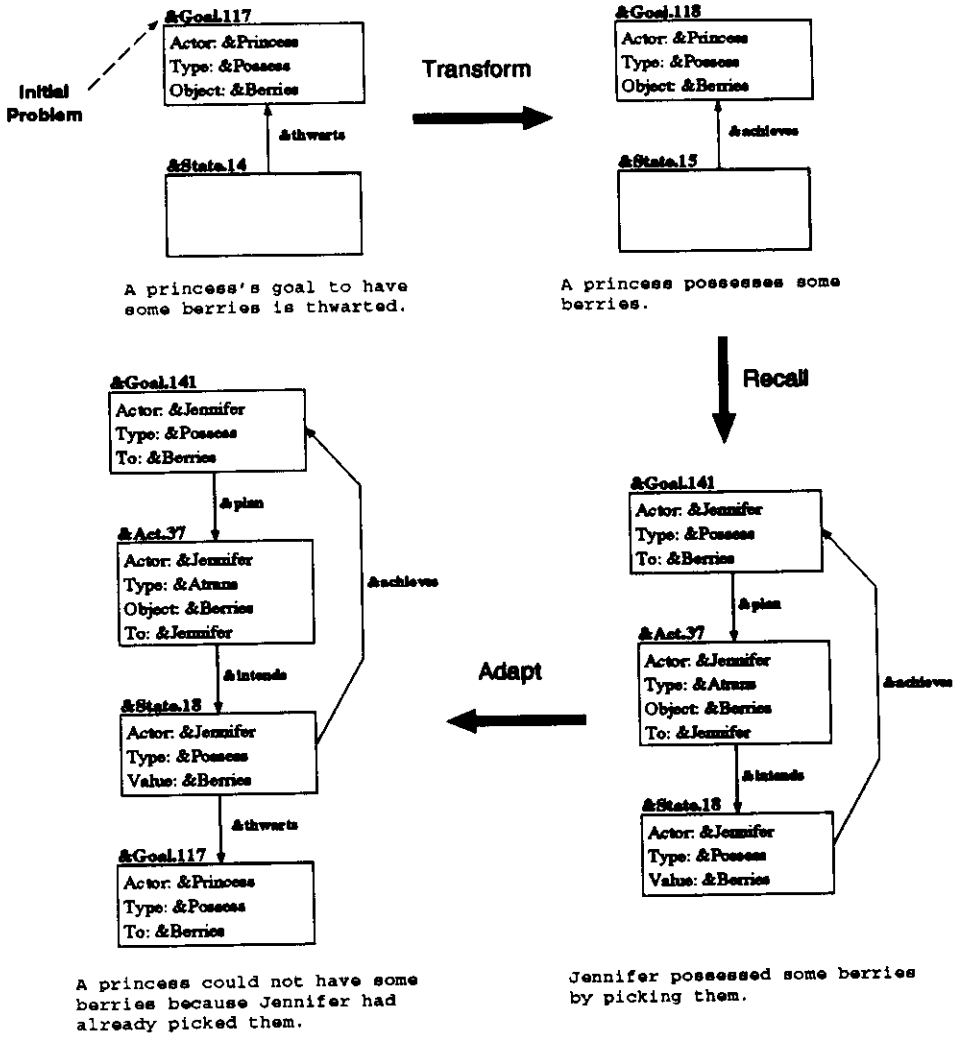
TRAM:Opposite-State-Achieves is illustrated in Figure 4.23.

This TRAM illustrates a new feature of the MINSTREL TRAM model. In TRAM:Opposite-State-Achieves, the adaptation step can fail if the recalled achieving state is neither a scaled state or an exclusive state. When this occurs, creative problem-solving fails, just as it would if the Transform or Recall step had failed.

4.5.4 TRAM:Ignore-Motivations

TRAM:Ignore-Motivations is another heuristic similar to TRAM:Ignore-Subgoal, TRAM:Limited-Recall, TRAM:Recall-Act and TRAM:Recall-wo-Precond. It removes from a problem specification unnecessary knowledge that may be preventing the problem-solver from discovering a solution. In this case, TRAM:Ignore-Motivations suggests that the motivation for a goal is not needed to solve the goal. For example, suppose that a knight decides to rescue a princess. The knight's plan to achieve this goal does not depend on why he wants to save the princess. Whether he saw the princess being carried off or he was told to save her by the king will not affect his planning.

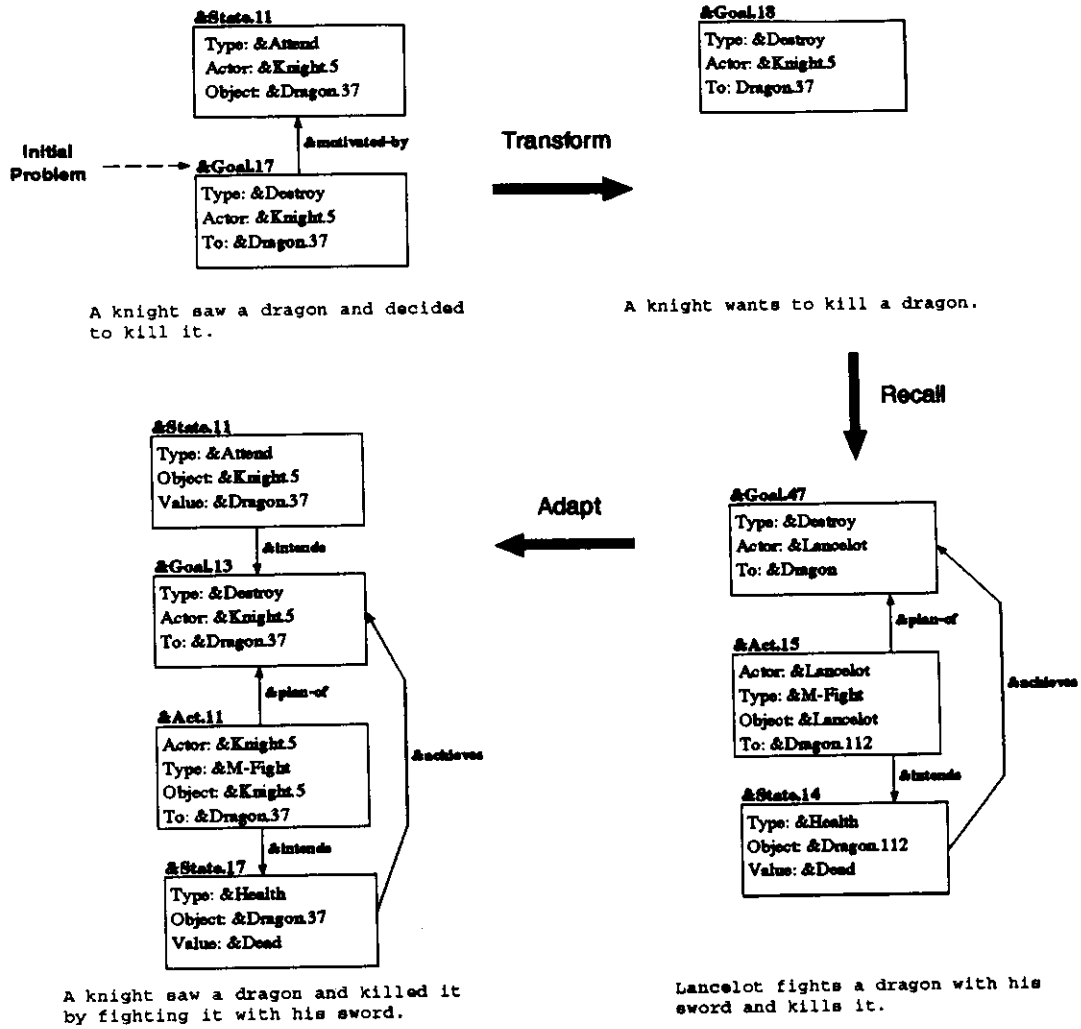
TRAM:Ignore-Motivations is illustrated in Figure 4.24.



TRAM:Opposite-State-Achieves

Comment:	To thwart a goal, achieve it and then reverse the achievement.
Class:	Goals.
Test:	Is this a thwarted goal?
Transform:	1. Change the thwarted goal to an achieved goal.
Adapt:	1. Locate the state that achieves the goal. 2. If the state is a scaled state, change the value to the opposite extreme. 3. If the state is an exclusive state, then change the value of the state to a different value of the same type. 4. Otherwise fail; don't know how to do a proper adaptation.

Figure 4.23 TRAM:Opposite-State-Achieves



TRAM:Ignore-Motivations

Comment:	Ignore the state that motivates a goal.
Class:	Goals.
Test:	Is this a motivated goal?
Transform:	Delete the motivating state.
Adapt:	No adaptation necessary.

Figure 4.24 TRAM:Ignore-Motivations

4.5.5 TRAM:Similar-Thwart-State

TRAM:Similar-Thwart-State is similar to TRAM:Similar-Outcomes (see section 4.4.2). TRAM:Similar-Outcomes suggests that if an action results in a particular outcome, it might also result in other, similar outcomes. TRAM:Similar-Thwart-State suggests that if a goal is thwarted by a particular state, it might also be thwarted by other, similar states. For example, if being injured thwarts a knight's goal of being healthy, then being dead might also thwart that goal.

TRAM:Similar-Thwart-State is illustrated in Figure 4.25. TRAM:Similar-Thwart-State uses the methods for determining similar states outlined in TRAM:Similar-Outcomes.

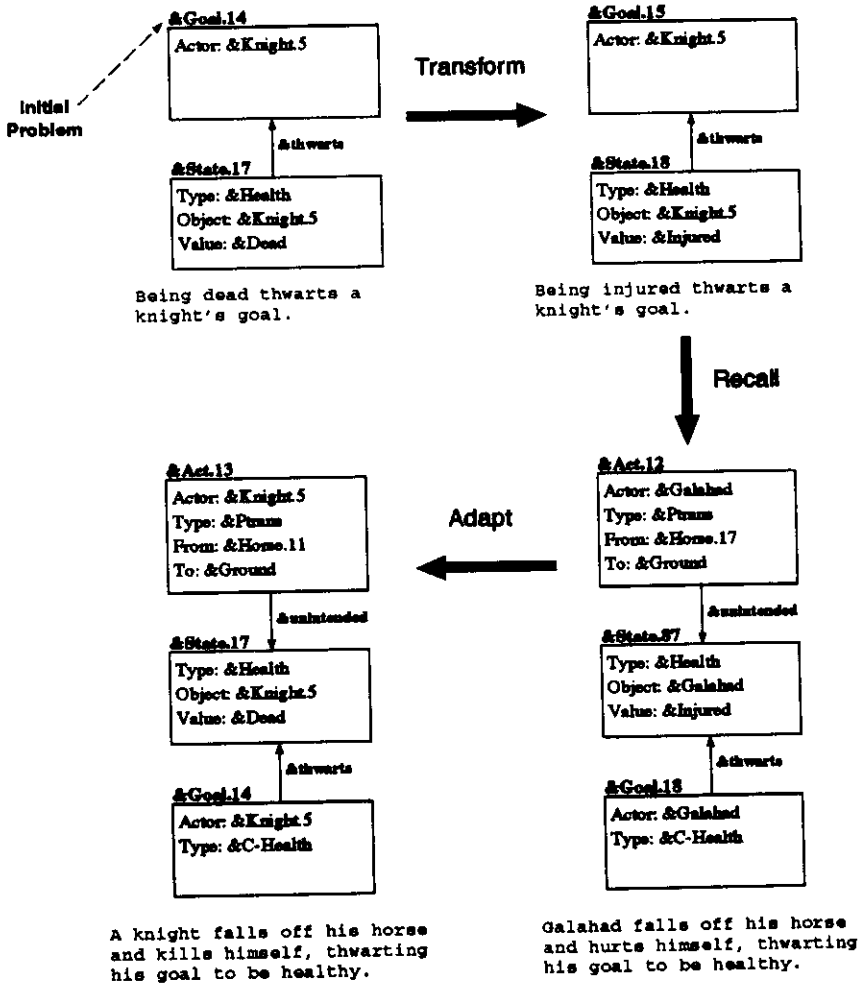


Figure 4.25 TRAM:Similar-Thwart-State Diagram

TRAM:Similar-Thwart-State

Comment:	A goal can be thwarted by similar states.
Class:	Goals.
Test:	Is the problem a thwarted goal?
Transform:	Change the thwarting state to a similar state:
	1. If the state is a scaled state:
	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.
	2. Otherwise:
	A. Generalize the type and value of the state's outcome. B. Attempt recall from episodic memory. C. Note the type and value of any recalled episodes. Assume that these are interchangeable with the original type and value. D. Select one of the interchangeable type and value pairs and substitute it for the original type and value.
Adapt:	Change the outcome on the recalled solution to the original outcome.

Figure 4.26 TRAM:Similar-Thwart-State Explanation

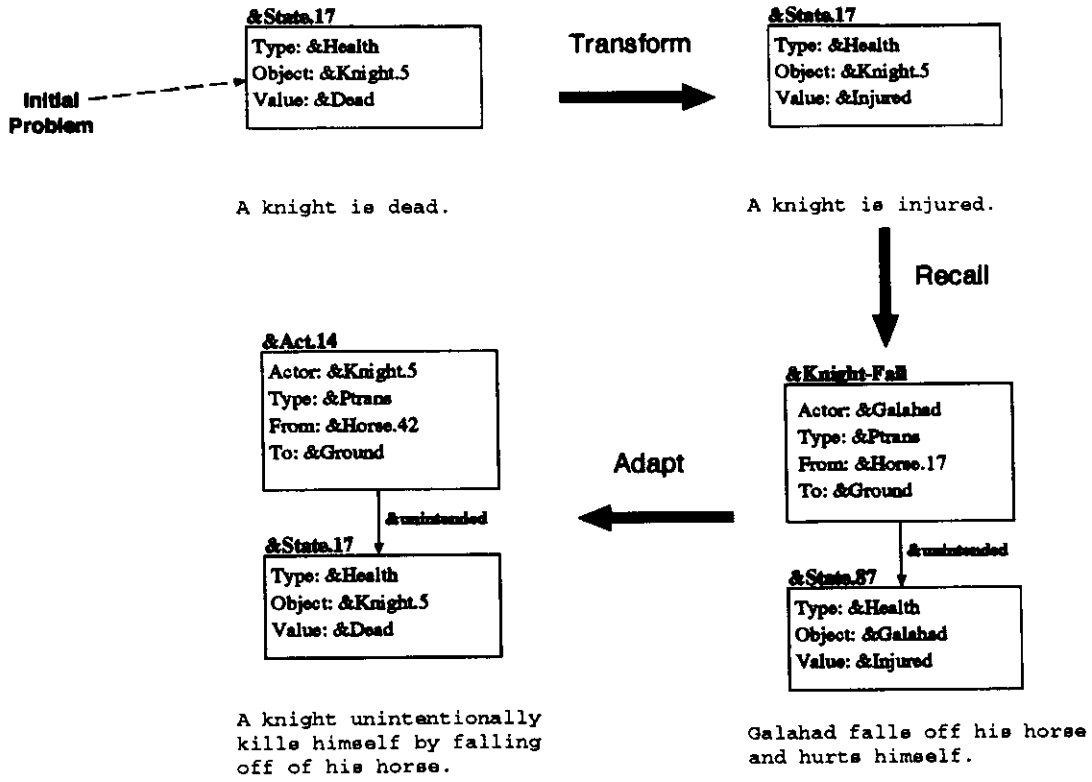
4.6 State-Based TRAMs

4.6.1 TRAM:Similar-States

TRAM:Similar-States is similar to TRAM:Similar-Outcomes and TRAM:Similar-Thwart-State. TRAM:Similar-Outcomes applied to acts that have an outcome. TRAM:Similar-Thwart-State applied to goals that have a thwarting state. TRAM:Similar-States applies to the states themselves. All three heuristics make use of the knowledge that similar states may be interchangeable.

TRAM:Similar-States suggests that if the problem description is a state, then to instantiate that state it may be useful to look at other similar states. For example, if MINSTREL is trying to determine what type of goal might be motivated when a knight gets injured, TRAM:Similar-States suggests looking at scenes in which a knight was killed, to see what goals were motivated in those cases.

TRAM:Similar-States is illustrated in Figure 4.27. TRAM:Similar-States uses the same methods for determining similar states outlined in TRAM:Similar-Outcomes.



TRAM:Similar-States

Comment:	Similar states have other similarities.
Class:	States.
Test:	Is the problem a state?
Transform:	1. If the state is a scaled state, and 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. 2. Otherwise: A. Generalize the type and value of the state's outcome. B. Attempt recall from episodic memory. C. Note the type and value of any recalled episodes. Assume that these are interchangeable with the original type and value. D. Select one of the interchangeable type and value pairs and substitute it for the original type and value.
Adapt:	Change the outcome on the recalled state to the original outcome.

Figure 4.27 TRAM:Similar-States

4.6.2 TRAM:Intention-Switch-2

TRAM:Intention-Switch-2 is a variant of TRAM:Intention-Switch. TRAM:Intention-Switch applies to actions which have intentional effects. It suggests that if the effect of an action was intentional it might just as well have been unintentional. TRAM:Intention-Switch-2 applies to the intentional effects of actions, and makes the same suggestion.

Figure 4.28 shows how TRAM:Intention-Switch-2 is implemented in MINSTREL. TRAM:Intention-Switch-2 is identical to TRAM:Intention-Switch except that it looks at the problem from the intentional state instead of the intending act.

4.6.3 TRAM:Generalize-Object

TRAM:Generalize-Object is a variant of TRAM:Generalize-Role. TRAM:Generalize-Roles applied to schemas with an "actor" slot. States do not have an "actor" slot, but characters often appear in the "object" slot of states. For example, when a princess possesses some berries, this is represented as a state schema with the princess in the "object" slot, a marker for possession in the "type" slot, and the berries in the "value" slot. This representation is shown in Figure 4.29.

TRAM:Generalize-Object generalizes the "object" slot of a state when it is a character, by generalizing the role of the character. For details on how this is accomplished, see Section N.M. TRAM:Generalize-Object is illustrated in Figure 4.29.

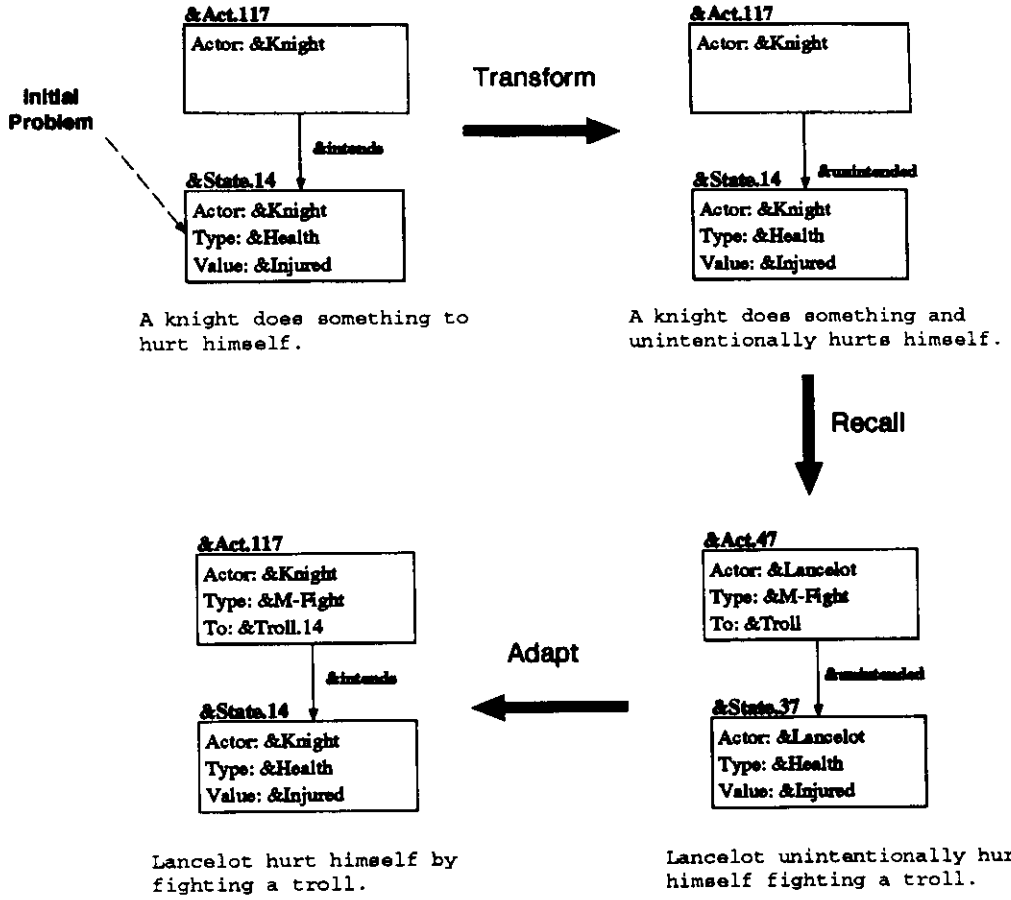
4.6.4 TRAM:Thwart-Via-Death

TRAM:Thwart-Via-Death is the first of two TRAMs which capture specific knowledge about how to thwart a goal. TRAM:Thwart-Via-Death suggests that one way to thwart a character's goal is to kill the character who has the goal. For example, if a knight wants to become king, one way to prevent that goal from being achieved is to kill the knight. TRAM:Thwart-Via-Death is shown in Figure 4.30.

TRAM:Thwart-Via-Death is an unusual TRAM because it captures an actual plan: thwarting a goal by killing the actor of the goal. Most of MINSTREL's TRAMs do not have such specific content. They modify the structure of problem descriptions, or add or remove knowledge to the descriptions, but they are not themselves plans. TRAM:Thwart-Via-Death, however, is very similar to an actual plan.

In MINSTREL, plans are normally episodes in memory that are recalled and applied to problems. TRAM:Thwart-Via-Death was an experiment to see if planning knowledge could be usefully encoded in the creativity mechanism. The reason for this was to explore how TRAMs might be learned.

In the context-plus-index model of episodic memory, generalizations are learned as individual episodes that share features accumulate in memory. When a number of similar episodes are in-



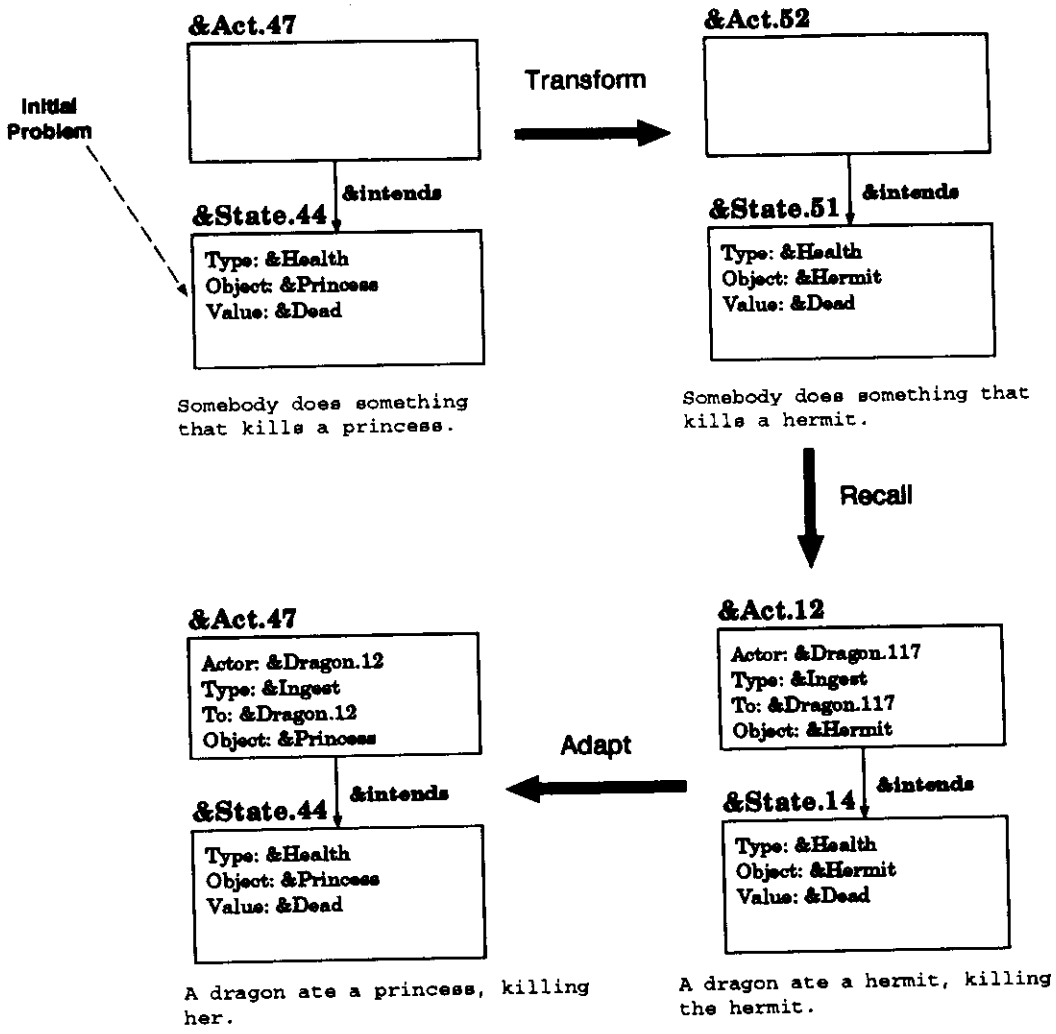
TRAM:Intention-Switch-2

Comment:	Switch an intentional outcome with an unintentional outcome.
Class:	States.
Test:	Is this a intentional state?
Transform:	Change the intentional state to an unintentional state.
Adapt:	Change the unintentional state of the solution to an intentional state.

Figure 4.28 TRAM:Intention-Switch-2

dexed into memory, a generalization of those episodes is created. Thus, after a person experiences (for instance) a number of fast food dining situations, he learns some general knowledge about eating in fast food restaurants.

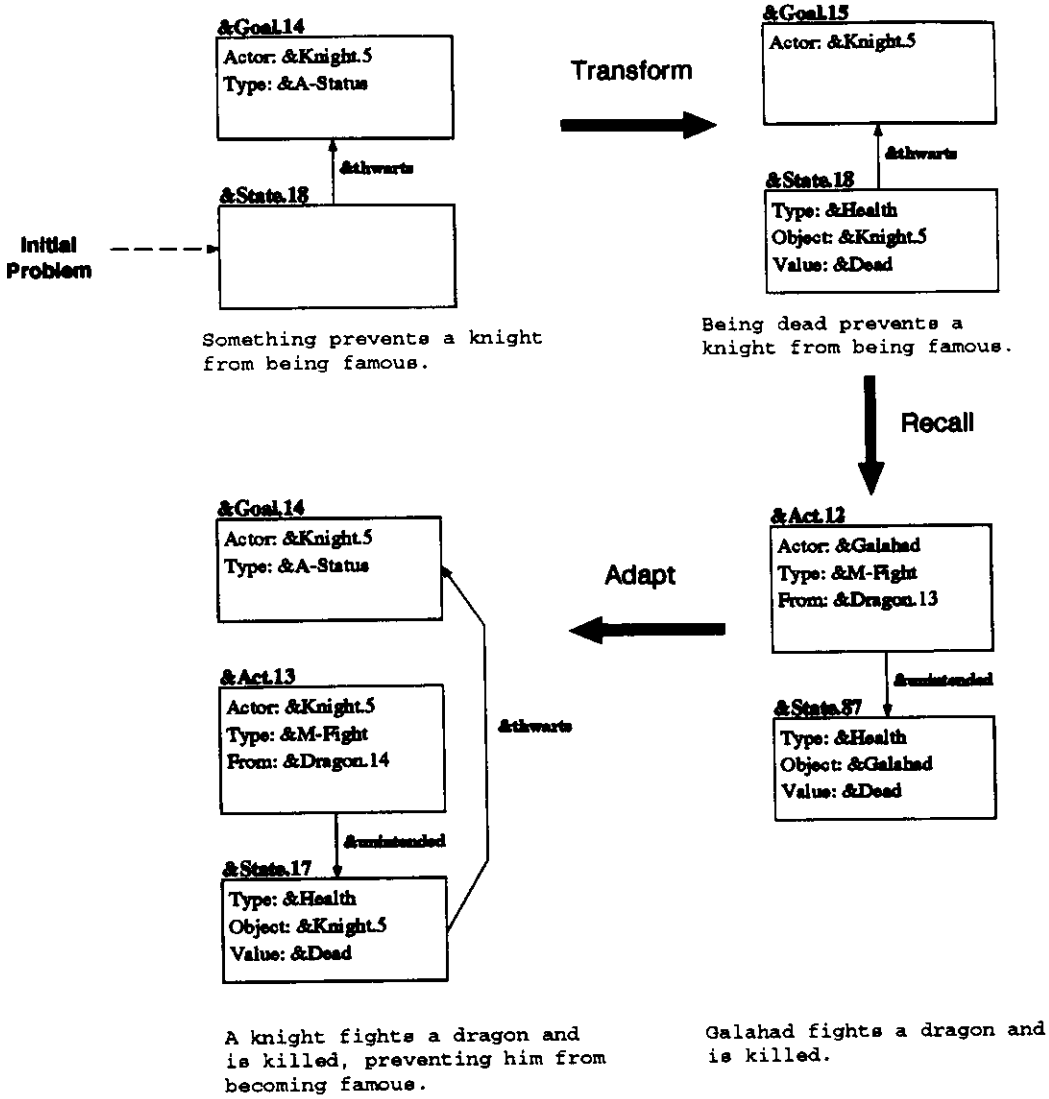
TRAMs could be learned in an analogous way. As a problem-solver learns creative solutions to problems they accumulate in memory, existing there as individual, specific TRAMs. When a number of similar TRAMs have accumulated, they would then be generalized into a more general and more powerful TRAM. For example, when a problem-solver reads *Romeo and Juliet* he may recognize the cleverness of thwarting a goal by being dead. He saves this in memory for fu-



TRAM:Generalize-Object

Comment:	Generalize the object slot of a state if a character.
Class:	States.
Test:	Is this a state with a character in the object slot?
Transform:	Change the role of the character in the object slot to a similar role.
Adapt:	Change the role of the character in the solution to the original role.

Figure 4.29 TRAM:Generalize-Object



TRAM:Thwart-Via-Death

Comment:	Thwart a goal by killing the actor of the goal.
Class:	States.
Test:	Is this a state which thwarts a goal?
Transform:	Instantiate the state as the death of the actor of the goal.
Adapt:	No adaptation.

Figure 4.30 TRAM:Thwart-Via-Death

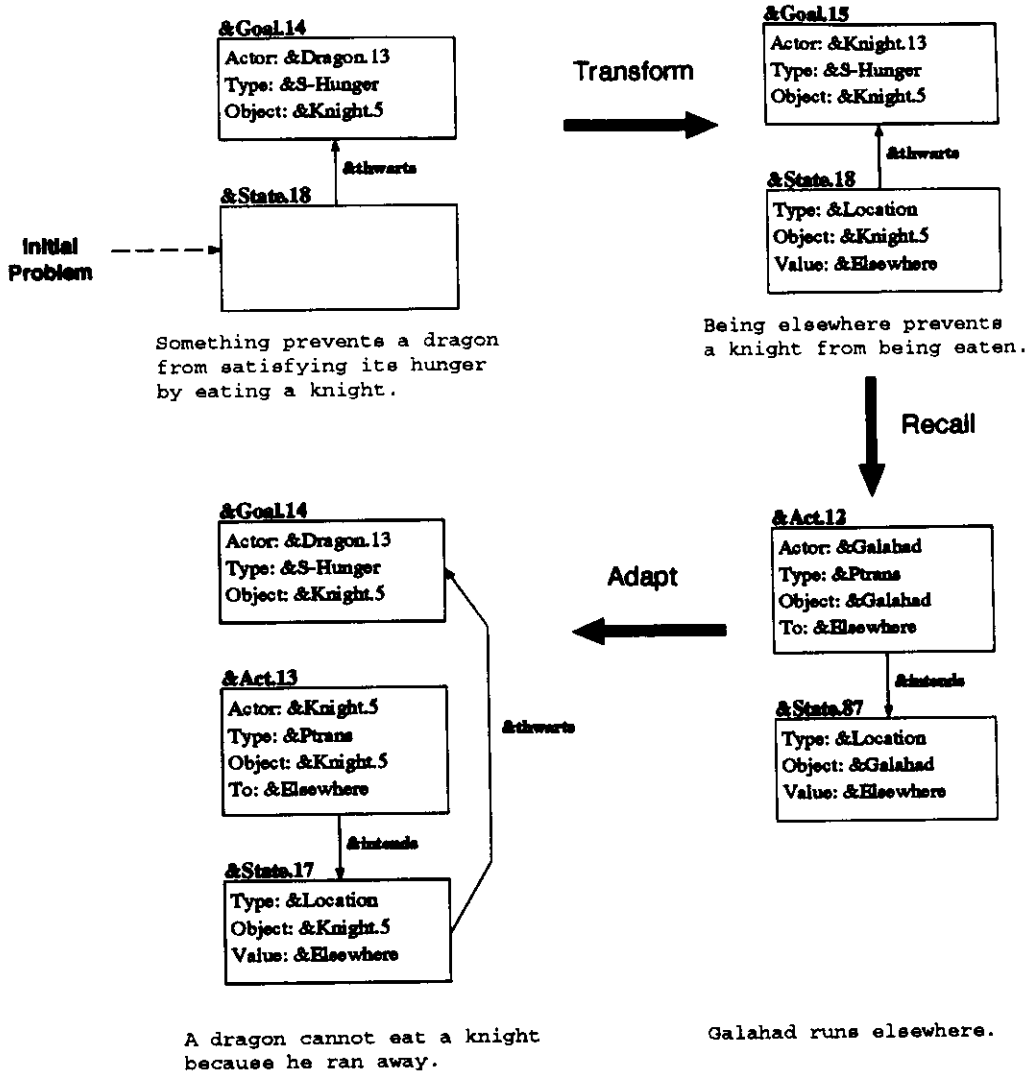
ture use. Eventually it accumulates with other similar strategies, such as taking away a key resource to prevent a plan from working. Eventually these are generalized into a TRAM about thwarting by removal of a key element.

TRAM:Thwart-Via-Death represents an early stage in this process. It is a TRAM which is still specific enough to contain planning. It has not yet been generalized to a more embracing principle.

TRAM:Thwart-Via-Death shows that it is possible to represent planning knowledge as creativity heuristics. Thus the first step of the generalization method of learning TRAMs is possible. Whether the generalization process as a whole is possible or desirable is an area for future research.

4.6.5 TRAM:Thwart-Via-Escape

TRAM:Thwart-Via-Escape suggests that a goal can be thwarted by running away, if the object of the goal is the actor who runs away. Like TRAM:Thwart-Via-Death, TRAM:Thwart-Via-Escape is another specific plan for thwarting a goal. It represents another facet of the general principle of thwarting by removal of a key element. In a creativity system with a learning component, the similarities of TRAM:Thwart-Via-Death and TRAM:Thwart-Via-Escape might be generalized into a more general TRAM. TRAM:Thwart-Via-Escape is shown in Figure 4.31.



TRAM:Thwart-Via-Escape

Comment:	Thwart a goal by escaping.
Class:	States.
Test:	Is this a state which thwarts a goal?
Transform:	Instantiate the state as the escape of the character in the object slot of the goal.
Adapt:	No adaptation.

Figure 4.31 TRAM:Thwart-Via-Escape

Part II: Storytelling

What do we need to know to build a computer that can write stories like a human author?

To begin with, we need to understand *why* authors write. We must understand what kinds of goals authors are trying to achieve when they are writing and the relationships between those goals.

We also need to understand something about *how* authors write. We must understand what kinds of plans authors have to achieve their writing goals, and what processes and techniques they use to accomplish those plans. Pursuit of this knowledge may lead us into the fundamental processes used in general cognition, such as memory and creativity.

This section looks at these two issues. MINSTREL has been given a set of author-level goals and author-level plans that enable MINSTREL to tell theme-based stories about King Arthur and his Knights of the Round Table; the following chapters describe those goals and plans and the processes by which they are achieved.

It is important to remember that this research is an initial step in understanding the storytelling process. MINSTREL has neither the depth nor flexibility of a human author. In some cases, MINSTREL has only one plan to achieve an author-level goal where a human author would have many and would be more capable than MINSTREL of inventing new ones. But the purpose of MINSTREL is not to exhaustively explain the authoring process, or even to catalog all the author-level plans for telling stories in the King Arthur domain.

Rather, MINSTREL is an exploration of the issues and problems in storytelling. This exploration has necessarily involved creating initial answers to many of the issues and problems encountered, for only by trying to answer questions can we truly understand their ramifications. What appears in this section, then, are the footprints of exploration, in the form of the author-level goals and plans MINSTREL uses to tell stories. But these footprints should not be mistaken for a finished road.

Like any exploration, the criterion for the evaluation of this research is simple: Does it take us to interesting places? MINSTREL suggests a variety of answers to the fundamental questions of why and how authors write; hopefully the reader will find the issues raised and the answers proposed interesting and thought-provoking.

CHAPTER 5
A Process Model of Storytelling

5.1 Why tell stories?

Authors create stories for a bewildering variety of reasons. A mother spins a fanciful bedtime tale to lull her young child asleep; a rabbi crafts an elegant anecdote to illustrate the generosity of God; a distressed young woman writes a novel to heal the grief she feels over losing her mother. And not only do authors write for many different reasons, they often pursue many goals at once in their writing. Shakespeare wrote works which both illuminate the human condition and delight the ear; Jonathon Swift wrote stories that were both entertaining adventures and biting social commentary on the England of his day. The goals and purposes of storytelling are as diverse and varied as human intellect itself.

But whatever the reasons, it is clear that human authors write *intentionally*. The stories authors create are carefully crafted to achieve particular goals. These goals and the ways they are achieved differ greatly from author to author, but every author has an explicit awareness of writing as a way to achieve some personal goals.

The importance of author goals in storytelling is best illustrated by an early model of computer storytelling called TALESPIN. TALESPIN was a computer program developed at Yale by James Meehan [Meehan 1976]. TALESPIN had knowledge about the likely goals and plans of a cast of simple woodland creatures. To tell a story, TALESPIN generated some likely goals for these creatures and then simulated their attempts to achieve those goals:

> *John Bear is somewhat hungry. John Bear wants to get some berries. John Bear wants to get near the blueberries. John Bear walks from a cave entrance to the bush by going through a pass through a valley through a meadow. John Bear takes the blueberries. John Bear eats the blueberries. The blueberries are gone. John Bear is not very hungry.*

As this example illustrates, TALESPIN often told stories that lacked purpose. The characters act in reasonable ways and the story world is consistent and detailed, but the stories have no point or reason. TALESPIN's stories don't read like *stories*.

The reason for this is simple. TALESPIN knows about the characters in its story world, about the kinds of things they can do and the kinds of goals they can have, but TALESPIN lacks any knowledge about itself as an author. TALESPIN does not know *why* it tells stories. At best TALESPIN has an implicit understanding of storytelling as "making characters do something to achieve likely goals". But because TALESPIN focuses on character-level goals rather than author-level goals, its characters have the purpose its stories lack.

Clearly storytelling is more than creating plausible accounts of how characters might achieve their goals. Authors are not purely "simulators" of reality; they have purpose and intention in

their writing. The events of a story are crafted to fulfill goals other than a mere slavish consistency with real life. To be cognitively plausible, and to create stories with purpose and direction, a model of storytelling must explicitly represent the author's goals and the process of achieving those goals.

Stories are the purposeful achievement of author goals.

Aside from its ability to be creative, MINSTREL's fundamental advancement over TALESPIN is an explicit author model. Like a human author, MINSTREL tells stories to achieve particular goals. As MINSTREL tells a story, it has an agenda of author-level goals it is trying to fulfill, such as illustrating a specific story theme, and building suspense in a particular part of the story. Because MINSTREL is a purposeful storyteller with knowledge of its goals as an author, it creates stories that are better organized, more purposeful, and more recognizable as "stories" than those created by TALESPIN.

5.2 Author Goals

If the uses of storytelling are as diverse as human intellect itself, then surely cataloguing the goals of authors is a hopeless task. How then can we learn about the authoring process?

One way to begin is by identifying and defining the goals that are necessary to tell stories of a particular type. The hope is that by carefully examining the authoring process for one particular type of writing, something will be discovered about the authoring process in general. So although we may not understand everything about *why* authors write, we will learn something about *how* they write. And this knowledge will serve as a basis for further research that will lead to a deeper and more general understanding of author-level goals.

This approach has led us to create MINSTREL, a computer program that tells short, theme-based stories about King Arthur and his Knights of the Round Table. Narrowing the range of storytelling to a specific style, a specific length, and a specific milieu makes the storytelling problem manageable and permits MINSTREL to focus on the process of storytelling rather than the diverse "whys" of storytelling.

Restricting MINSTREL to theme-based stories limits the types of author goals MINSTREL must solve. Selecting a single, specific primary author goal – to tell a story that illustrates a particular theme – greatly narrows the range of author goals. MINSTREL does not have to tell bedtime stories, satires, or any of the other myriad types of stories. At the same time, theme-based stories are complex and rich enough to address a variety of issues in storytelling, the way certain storytelling styles – such as the stories of very young children, or mathematical story problems – would not.

Restricting the length of the stories it tells to about one page allows MINSTREL to concentrate on stories in which immediate character actions predominate. Longer works such as novels often use character interactions, interplays of moods and emotions, digressions and complicated pre-

sentation techniques to effect their purposes. Limiting the length of MINSTREL's stories concentrates this research on how one particular tool – creating story events – can be used to achieve a variety of author-level goals.

Finally, restricting MINSTREL's storytelling to a specific milieu focuses this research on issues in storytelling rather than issues in understanding and representing knowledge about the world. The King Arthur milieu is relatively straightforward: knights love princesses and kill dragons, hermits live in the woods and heal people. Were MINSTREL to tell stories in a more complicated milieu, or in several different milieus, more effort would have had to be expended to give MINSTREL knowledge about those milieus. Although this might have led to some interesting results, the time and effort it would take to understand and represent knowledge about different milieus would have subtracted from the time available to develop a general model of storytelling and creativity. It was decided early in this research effort that the development of the general models of storytelling and creativity was of greater interest, and limiting MINSTREL to a single storytelling domain permitted a more in-depth development of this area of the storytelling model.

5.2.1 MINSTREL's Author-Level Goals

Limiting MINSTREL to telling short, theme-based stories about King Arthur revealed four important classes of author-level goals:

(1) Thematic Goals
(2) Drama Goals
(3) Consistency Goals
(4) Presentation Goals

Thematic goals are concerned with the selection and development of a story theme. Drama goals are concerned with the use of dramatic writing techniques to improve the artistic quality of a story. Consistency goals focus on creating a story that is plausible and believable. And presentation goals are concerned with how a story is presented to the reader.

To further explain these goals and illustrate how they combine to create a complete story, we will look at the role each class of goals play in one of MINSTREL's stories. The story we will use is called *Richard and Lancelot*. Except for typography, it is reproduced here exactly as written by MINSTREL:

Richard and Lancelot

```
It was the spring of 1089, and a knight named Lancelot returned
to Camelot from elsewhere.  Lancelot was hot tempered.  Once,
Lancelot lost a joust.  Because he was hot tempered, Lancelot
wanted to destroy his sword.  Lancelot struck his sword.  His
sword was destroyed.
```

One day, a lady of the court named Andrea wanted to have some berries. Andrea went to the woods. Andrea had some berries because Andrea picked some berries. Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot saw Andrea. Lancelot loved Andrea.

Some time later, Lancelot's horse moved Lancelot to the woods unintentionally, again causing him to be near Andrea. Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Lancelot hated Frederick. Because Lancelot was hot tempered, Lancelot wanted to kill Frederick. Lancelot went to Frederick. Lancelot fought with Frederick. Frederick was dead.

Andrea went to Frederick. Andrea told Lancelot that Andrea was siblings with Frederick. Lancelot believed that Andrea was siblings with Frederick. Lancelot wanted to take back that he wanted to kill Frederick, but he could not because Frederick was dead. Lancelot hated himself. Lancelot became a hermit. Frederick was buried in the woods. Andrea became a nun.

Moral: ''Done in haste is done forever.''

5.2.2 Thematic Goals

The theme of a story is the main point or purpose of the story. Because there are many possible reasons to tell a story there are many possible story themes. MINSTREL tells stories about a particular type of theme called a Planning Advice Theme, or PAT. Planning Advice Themes represent concise pieces of advice about planning, and they can often by summarized by adages, such as "A bird in the hand is worth two in the bush."

MINSTREL's author-level thematic goals are concerned with selecting and illustrating a story theme. *Richard and Lancelot* is based on a Planning Advice Theme called PAT:Hasty-Impulse-Regretted. This theme advises a planner to avoid making hasty decisions that cannot be retracted if they turn out to be incorrect. The events in a story that illustrate the theme are called the story *plot*. In *Richard and Lancelot*, the following scenes illustrate the theme of the story:

Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Fred-

```
erick.  Because Lancelot was hot tempered, Lancelot wanted to
kill Frederick.  Lancelot went to Frederick. Lancelot fought
with Frederick.  Frederick was dead.

Andrea told Lancelot that Andrea was siblings with Frederick.
Lancelot believed that Andrea was siblings with Frederick.
Lancelot wanted to take back that he wanted to kill Frederick,
but he could not because Frederick was dead.
```

These events form an example of the abstract advice represented in the story theme. By structuring the stories it tells around themes, MINSTREL assures that they will have the purpose that was missing from stories told by TALESPIN.

MINSTREL has two author-level thematic goals. The first goal is to select a theme for storytelling. The second is to create a sequence of story events that form an example of the selected theme.

Chapter 6 discusses MINSTREL's representation of story themes, MINSTREL's thematic goals and the plans MINSTREL uses to achieve those goals.

5.2.3 Drama Goals

Human authors use a wide variety of techniques to improve the craftmanship and literary quality of their stories. Foreshadowing, characterization, irony, suspense and tragedy are all examples of writing techniques that authors use to improve the quality and impact of their stories. Using these techniques is rarely the primary purpose of an author's storytelling. Instead, these are secondary writing goals that improve the artistic values of a story while supporting the theme of the story.

MINSTREL implements four drama goals: suspense, tragedy, foreshadowing and characterization. Two of these techniques are used in *Richard and Lancelot.*

Foreshadowing is used to increase the impact of the scene in which Lancelot (erroneously) discovers that Andrea loves another knight by echoing parts of that scene earlier in the story:

```
Lancelot's horse moved Lancelot to the woods.  This unexpected-
ly caused him to be near Andrea.  Because Lancelot was near An-
drea, Lancelot saw Andrea.  Lancelot loved Andrea.
[...]
Some time later, Lancelot's horse moved Lancelot to the woods
unintentionally, again causing him to be near Andrea.  Lancelot
knew that Andrea kissed with a knight named Frederick because
Lancelot saw that Andrea kissed with Frederick.
```

Lancelot's willful horse first causes him to unexpectedly meet and fall in love with Andrea, and then later causes him to unexpectedly see Andrea kissing Frederick and fall out of love with An-

drea. This juxtaposition and repetition of similar scene elements improves the impact of the story theme by echoing and strengthening the underlying pattern of the story.

Characterization is used to establish the hot temper of Lancelot, which contributes to his later hasty decision:

```
Lancelot was hot tempered.  Once, Lancelot lost a joust.  Be-
cause he was hot tempered, Lancelot wanted to destroy his
sword.  Lancelot struck his sword.  His sword was destroyed.
```

To develop the characterization of Lancelot as hot-tempered, MINSTREL creates a story scene which shows how his hot temper affects how he reacts to events. By establishing the personality of the main character early in the story, MINSTREL improves the plausibilty of later events and enhances the overall quality of the story.

Chapter 8 discusses MINSTREL's use of dramatic writing techniques.

5.2.4 Consistency Goals

Another concern for authors is to tell stories that are consistent and believable. Characters should act rationally and events should happen in accordance with the author's best understanding of how the world functions. Readers expect stories to reflect and agree with what they know about the world, and so the author must take care to maintain that plausibility, and to explain it when absent or different from common understanding.

Story inconsistencies normally arise as side-effects of other author-level goals. For example, when MINSTREL creates the story events needed to illustrate a story theme, it creates *only* the events necessary for the theme. This might include a scene in which a character dies. But unless the theme happens to also include scenes explaining how the character died, who killed him, and what emotional reactions all the characters in the story had to the character's death, the resulting story will be incomplete. The reader expects explanations of how and why things happen. The purpose of MINSTREL's consistency goals is to detect these types of situations and to correct them by adding explanatory story events.

MINSTREL implements a variety of author-level goals aimed at maintaining story consistency. One class of goals checks to see that characters are shown achieving all the steps of successful plans. A story in which characters achieve their goals without executing the necessary plans is implausible and unacceptable to most readers:

```
Lancelot was hot tempered, Lancelot wanted to kill Frederick.
Lancelot went to Frederick.  Lancelot fought with Frederick.  Frederick
was dead.
```

In *Richard and Lancelot*, Lancelot's murder of Frederick was created to illustrate the story theme.

After this scene is created, a consistency goal notices that a necessary precondition to fighting someone – being colocated with them – hasn't been fulfilled. Although Frederick's death is necessary to illustrate the story theme, an explanation of how Lancelot and Frederick came to be in the same place is not, and so MINSTREL's thematic goals did not create one. To make the story understandable, a story event is created that achieves the colocation precondition. Consistency goals "repair" the story by noticing and correcting inconsistencies left over from other author-level goals.

Another class of goals checks to be sure that characters are reacting properly to events in their world:

> Lancelot wanted to take back that he wanted to kill Frederick, but he could not because Frederick was dead. *Lancelot hated himself.*

People normally have emotional reactions to the events in their lives. They feel happy when they achieve important goals, sad when a major plan fails, anxious when they are worried about their self-preservation, and so on. In this example, Lancelot discovers that he has violated a major goal because of a character flaw, but has no emotional reaction to this event. A consistency goal notices this and creates a scene describing a plausible emotional reaction. This improves the consistency of the story and the believability of the character.

The range of MINSTREL's consistency goals and the plans MINSTREL uses to achieve them are discussed in Chapter 9.

5.2.5 Presentation Goals

Presentation goals concern how the story is communicated to the reader. The author of the story must decide the order in which events in the story are presented to the reader, which events must be fully described and which can be summarized or ommitted, and how each story event will be expressed in English.

Richard and Lancelot contains the following sequence of story scenes:

> Lancelot was hot tempered. Once, Lancelot lost a joust. Because he was hot tempered, Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed...
>
> Because Lancelot was hot tempered, Lancelot wanted to kill Frederick. Lancelot went to Frederick. Lancelot fought with Frederick. Frederick was dead...

The first scene is created by MINSTREL to illustrate the characterization of Lancelot as hot tempered. The second scene is part of the theme, and turns upon Lancelot's hot temper. MINSTREL's presentation goals must recognize the purposes of these two scenes and use that knowl-

edge to order them correctly.

MINSTREL's presentation goals are also concerned with selecting scenes to be in the story and with expressing story events in English. MINSTREL's presentation goals and plans are discussed in Chapter 10.

5.3 Author-Level Planning and Problem-Solving

The previous sections identified four classes of important author-level goals and showed how they combined to create a complete story. Now we shift our attention to the process of how those goals arise and are achieved.

As noted above, authors tell stories to achieve particular goals. The process of storytelling involves selecting a goal from the author's pool of goals, trying to find a plan to achieve that goal, and executing the plan, possibly adding new goals to the agenda or deleting old ones. This continues until the author is satisfied with the story, i.e., until the author has no more unsatisfied author-level goals.

MINSTREL models this using two processes. The *planning process* is concerned with the management of author-level goals. The planning process maintains the pool of author goals and when necessary, selects a goal to achieve. The *problem-solving process* is concerned with solving author-level goals. It takes a goal selected by the planning process and finds, evaluates and executes a plan to achieve that goal. These processes are illustrated in Figure 5.1.

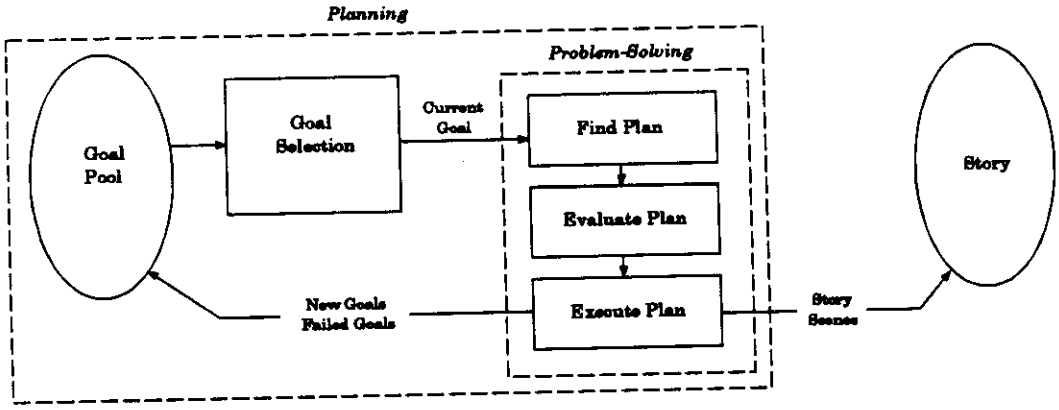


Figure 5.1 Author-Level Processes

5.3.1 Author-Level Planning

MINSTREL uses an agenda-based planning model, patterned on similar models in [Lenat 1976] and [Warren 1978]. As goals arise they are given a priority and placed on an agenda. Priorities are represented by integer numbers on a scale of 1 to 100. At each planning cycle, the highest priority goal is selected from the agenda and passed to the problem-solving process.

MINSTREL begins storytelling with an initial goal to "tell a story." This goal breaks down into subgoals including selecting a theme, illustrating a theme, applying drama goals, checking the story for consistency, and presenting the story to the reader. At each cycle, MINSTREL selects the author-level goal with the highest priority from the goal agenda and passes it to the problem-solving process. Problem-solving finds a plan for that goal and executes it.

Two important actions that MINSTREL's plans can take are to create and add new scenes to the story, and to create and add new author-level goals to the planning agenda. As new scenes are created, they are added to the current story. As new goals are created they are added to the goal agenda. Storytelling finishes when the goal agenda is empty.

One important feature of MINSTREL's planning model is that MINSTREL has the ability to re-queue failed goals. When all the plans found for achieving a goal fail, the failed goal is placed back on the agenda at a lower priority. This can be repeated until the priority drops to a cut-off level, below which the goal "falls off" the agenda and is ignored.

This facility permits MINSTREL to periodically retry goals that have failed, in hopes that the achievement of some intermediate goal will have changed the problem-solving situation enough to permit the failed goal to now succeed. The purpose of this is to permit MINSTREL's goals to interact with one another to synergistically find solutions that a straightforward, top-down pursuit of goals would miss.

In MINSTREL, synergy between author-level goals occurs when an author-level goal initially cannot be achieved because it lacks necessary information, and then that information is filled in as a side-effect of achieving a different author-level goal.

This situation occurs in telling the story *Richard and Lancelot* (see Section 5.2.1). At one point in telling this story, MINSTREL is creating a story scene in which Lancelot accidently encounters Andrea. The abstract description of this scene is:

> *Something happens which has an unintentional side-effect. This side-effect supersedes some previous state, and causes Lancelot to be near Andrea.*

When trying to create a scene with a complicated, multi-part description such as this, MINSTREL works piecemeal, creating the scene a piece at a time. In this case, MINSTREL begins by trying to create the previous state that was superseded by the unintentional side-effect. This fails, because, not yet knowing the side-effect, MINSTREL lacks the knowledge needed to decide what the side-effect might supersede. So the author-level goal to create the superseded state fails

and is re-queued at a lower priority.

Next MINSTREL tries to create the side-effect that causes Lancelot to be near Andrea. This is straightforward: the side-effect must be that Lancelot is in the same location as Andrea.

Now the goal to create the superseded state is attempted again. This time, there is sufficient knowledge to create the superseded state. The solution of the intervening author-level goal has filled in information necessary to create the superseded state. Since the state is superseded by Lancelot being with Andrea, the superseded state must be a previous location for Lancelot. MINSTREL invents a previous location (at the castle) and the goal which previously could not be solved is now solved.

When faced with a set of interdependent goals, it is often difficult for a problem-solver to determine which goal to work on first. Tracing the dependencies between goals can be a very difficult and time-consuming task. MINSTREL's ability to re-queue goals permits MINSTREL to avoid the problem of selecting a proper starting point from amongst a collection of interdependent author-level goals. Instead, MINSTREL begins with some likely goal. If the starting point selected turns out to be unsuccessful, it is re-queued and another tried until a fruitful starting point is found.

Thus MINSTREL's ability to re-queue goals not only permits it to solve goals that a straightforward, top-down approach would not be able to solve, but it also simplifies MINSTREL's planning model by eliminating the need for a mechanism to correctly select amongst competing author-level goals.

Although it does not occur in any of the stories that MINSTREL currently tells, there is a possibility in the MINSTREL model for a second type of synergy. This type of synergy occurs when MINSTREL *learns* some useful information in solving an intermediate goal that helps it solve a goal that originally failed. In the previous type of synergy, an intervening goal elaborated the problem description with a necessary piece of information. This change in the problem description permitted MINSTREL to solve a previously unsolvable goal. In this new type of synergy, an intervening goal invents a necessary piece of information, augmenting MINSTREL's knowledge. The problem description does not change, but what MINSTREL *knows* changes, permitting it to solve the previously unsolvable goal.

As discussed in Chapter 3, MINSTREL learns as it tells stories. As MINSTREL solves problems, it remembers those problems and their associated solutions in episodic memory. These solutions can then be used in the future, either directly to solve a similar problem, or as a starting point for creativity. It is possible, then, that MINSTREL might learn something in solving an intermediate goal that would enable it to solve some goal that was previously unsolvable.

An experiment which illustrates this type of synergy is discussed in Section 7 of Chapter 15. In this experiment, MINSTREL was asked to use creativity to invent methods for a knight to commit suicide. MINSTREL invents three solutions: hitting yourself with a sword, intentionally losing a fight to a monster, and drinking poison. These methods are remembered by indexing them

in episodic memory.

After this, MINSTREL is asked again to invent a method for a knight to commit suicide and invents a new method: having an agent poison you. This is possible because the learned solution "drinking poison" serves as a new starting point for creativity.

In a similar way, it is possible for the solution of an intermediate goal to be remembered and used as a basis for inventing a solution to a goal that was previously unsolvable. In this case, synergy occurs not because new knowledge has been added to the problem description, but because the problem-solving process has been improved by learning.

MINSTREL's planning model, and in particular, the role that re-queuing of goals plays in MINSTREL's storytelling are further discussed in Chapter 15.

5.3.2 Author-Level Problem-Solving

It's not unusual to view artists with something approaching awe. The process of making art seems very different from the kinds of prosaic tasks one tackles in day-to-day life. So few are successful at art, and what they produce is so different and interesting, that there is an automatic tendency to assume that art involves unique and special mental processes.

But although [Wallas 1926], [Koestler 1964] and others have argued that creative domains such as storytelling, art and music are somehow fundamentally different from more mundane problem domains, most psychological evidence suggests the opposite [Weisberg 1986]. People solve problems in creative domains in much the same way they solve problems in more traditional problem-solving domains. In art as in day-to-day life, people have goals, find or create plans to achieve those goals, apply the plans, evaluate the results and so on. We are not used to thinking of artistic endeavors such as painting and music in terms of problem-solving, but at a general process level there is little to distinguish between creating and playing a musical piece and creating and writing a thank-you note.

MINSTREL's author-level model of problem-solving is shown in Figure 5.2. Author-level goals are input at the left side, where they are used to recall similar past storytelling situations. The author-level plans used in these past situations are then adapted and applied to the current goal. Finally, the adapted plan is assessed to determine if it meets domain-specific considerations (i.e., if you are telling a realistic story you might reject a plan that would be acceptable for a fantasy).

Although this model is being applied to author-level problems, it is the same case-based model used for all problem-solving processes in MINSTREL. The portion of this model within the dotted lines is the same model described in Chapter 3. All that has been changed is the types of goals being solved.

In MINSTREL, a single problem-solving model is used for all problem domains, artistic and otherwise. MINSTREL's author-level storytelling goals are solved by the same process used to

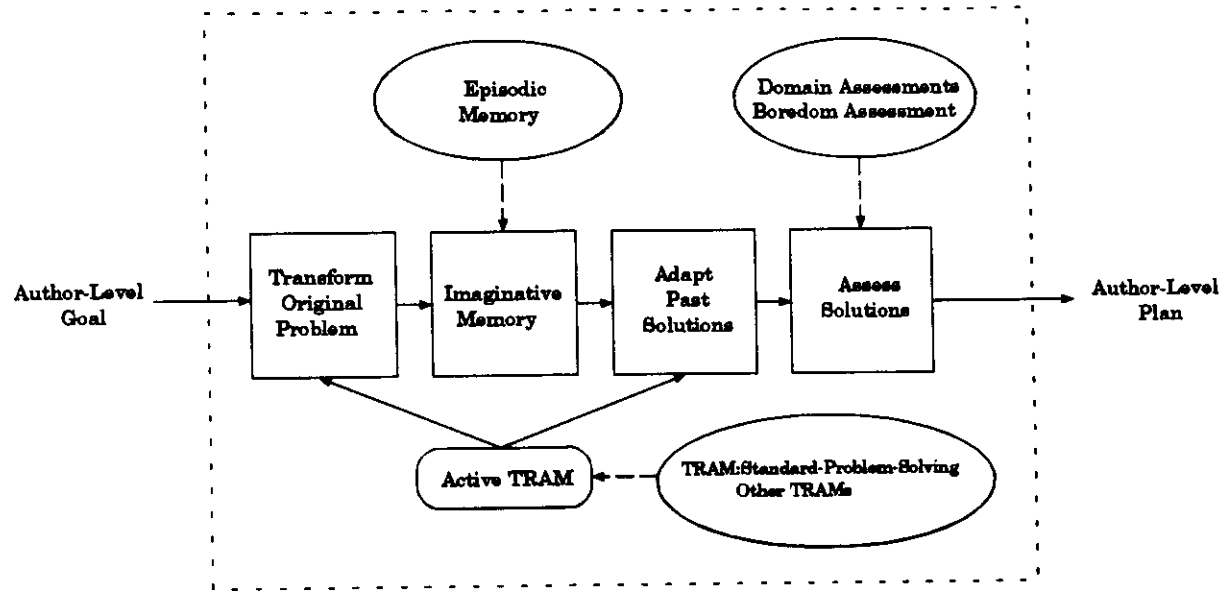


Figure 5.2 A Case-Based Model of Problem-Solving

solve character-level goals in the King Arthur domain and to invent devices in the mechanical invention domain. Uniformity of the problem-solving process is a fundamental tenet of this research.

The process of problem-solving is invariant across problem domains.

One interesting consequence of MINSTREL's uniform model of problem-solving is that the same creative process used to invent new story scenes and new solutions to problems in the King Arthur domain is also active in problem-solving at the author level. But before we can look at the role of creativity in solving author-level goals, it is necessary to digress momentarily to discuss the representation of author-level goals and plans.

5.3.2.1 Representation of Author-Level Goals And Plans

Character-level goals in MINSTREL's stories are represented using goal schemas. For example, Lancelot's goal to kill a dragon is represented as a &GOAL schema with appropriate values for the TYPE, ACTOR, and OBJECT slots. An example of a character-level goal is shown in Figure 5.3.

MINSTREL's author-level goals are also represented using goal schemas. Each of MINSTREL's goals in telling a story is represented using a &GOAL schema and appropriate values for the TYPE, ACTOR, and OBJECT slots. Figure 5.4 shows an example of an author-level goal to check the consistency in a particular story scene. Note that the actor of this goal is &MIN-

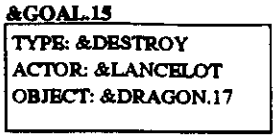


Figure 5.3 Example Character-Level Goal Schema

STREL, a symbol that MINSTREL uses to refer to itself, and that the object of this goal is an event in the story being told. This example author-level goal represents MINSTREL's desire to check the schema pointed to by the OBJECT slot (&State.99) for consistency.

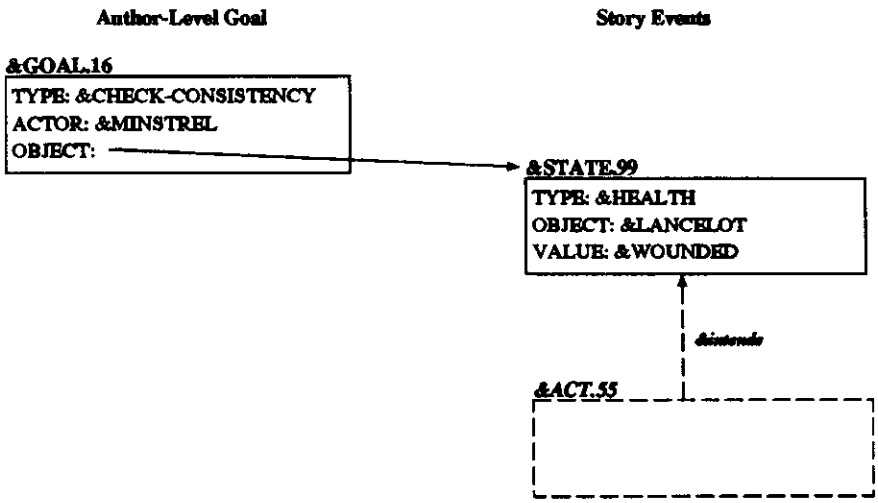


Figure 5.4 Example Author-Level Goal Schema

This consistent representation permits MINSTREL to treat character-level and author-level goals identically. The same processes can be used to store, recall and manipulate both types of goals.

Unlike the representation of goals, MINSTREL's representation of plans is *not* consistent across the character and author levels. Character-level plans in MINSTREL are represented as interconnected collections of goal, act and state schemas. Figure 5.5 shows the representation for a knight's plan to achieve the goal of destroying a dragon.

This plan is represented as a goal (destroy a dragon, &Goal.15), a plan to achieve that goal (fight the dragon, &Act.17) and the result of executing that plan (the dragon is dead, &State.7).

Although the same type of representation could be used for MINSTREL's author-level plans, it would be clumsy and time-consuming. Schemas for complicated computational actions such as looping, recursion and so on would have to be defined and an interpreter built to perform those actions. Fortunately, MINSTREL is built upon a representation for computation - Lisp. Rather

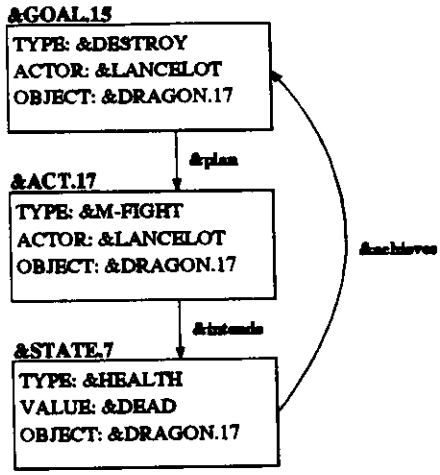


Figure 5.5 Example Character-Level Goal Schema

than re-invent the wheel, MINSTREL uses Lisp to represent its author-level plans and uses the Lisp interpreter to execute those plans.

Each of MINSTREL's author-level plans (ALPs) is a structured, independent block of Lisp code. Each ALP contains a test that determines when the plan is applicable and a body that executes the plan. Because Lisp code can be difficult to understand, the author-level plans presented in this text will be shown in a structured English format. An example of this format is shown in Figure 5.6. (Examples of MINSTREL's author-level plans in Lisp can be found in Appendix B.)

Name:	ALP:Make-Consistent-P-Health
Goal Type:	&Check-Consistency
Object:	*AL-OBJ*
Test:	*AL-OBJ* is a state schema that represents a character being wounded, and it is not motivating a goal by that character to protect his health.
Body:	1. Create a &P-Health (health protection) goal for the character being wounded in *AL-OBJ*. 2. Add the new goal to the story by creating a &THWARTS link from *AL-OBJ* to the new goal (i.e., being wounded thwarts a character's goal of protecting his health). 3. Create author-level goals to make sure the new goal is consistent and fully instantiated.

Figure 5.6 ALP:Make-Consistent-P-Health

The author plan shown in Figure 5.6 is one of the plans MINSTREL uses to check the consistency of a story. This plan assures that characters who are injured react properly by having a goal to

protect their health. This plan would apply to the author-level goal shown in Figure 5.4.

There are five components to each of MINSTREL's Author-Level Plans (ALPs). The Name identifies the plan. The Goal Type specifies the type of author-level goal to which the plan applies. The type of an author-level goal is the value of the TYPE slot of the &GOAL schema. The Object is a symbol by which the value of the OBJECT slot of the author-level goal can be referenced when this plan is applied. The Test determines whether this plan applies to a particular instance of a goal. Finally, the Body of the ALP is the series of actions required to execute the plan.

5.3.3 Creativity in Author-Level Problem-Solving

MINSTREL has a consistent representation for goals and a single model of problem-solving that is used to solve problems at both the character and author levels. This model of problem-solving includes creativity - the ability to invent new problem solutions when needed. Consequently, when MINSTREL cannot solve an author-level goal, creativity heuristics will be applied to try to invent a new plan for solving that goal, just as happens when MINSTREL cannot solve a character-level goal.

There are three steps to the creative problem-solving process:

(1) Transform the original problem specification.
(2) Recall a similar past problem situation.
(3) Adapt the associated plan to the original problem.

Because MINSTREL has a consistent representation for both author-level and character-level goals, and a single model of episodic memory, the first two steps of creative problem-solving are the same for both author-level and character-level problem-solving. The primary difference between character-level and author-level creativity lies in the third step: adapting the associated plan.

MINSTREL's character-level plans are represented by act and state schemas, and MINSTREL's creativity heuristics (TRAMs) know how to modify and adapt this representation. But MINSTREL's author-level plans are represented as structures of Lisp code, and MINSTREL's TRAMs do not know how to adapt Lisp code. MINSTREL's author-level plans are opaque and non-adaptable, and so MINSTREL *cannot* adapt author-level plans. This limits the creativity heuristics (TRAMs) that MINSTREL can apply when problem-solving at the author level.

In case-based problem-solving, plans to solve problems are found by recalling similar past problems. Creativity requires recalling plans from past problems different from the current problem and adapting them to the current problem. But because MINSTREL cannot adapt plans at the author-level (because they are opaque blocks of Lisp code) MINSTREL cannot apply any creativity heuristics which involve plan adaptation. However, there are a few types of creativity which do not require plan adaptation.

Consider, for example, a problem-solver finding his way home from a newly-built shopping mall. The problem-solver has never come home from this mall before, so standard problem-solving will not recall any ready-made plans. But if he recalls that the shopping mall stands on the site of a former restaurant he frequented, then he can recall a route for driving home from the restaurant and use it without change. The problem-solver has invented a solution to a problem by making use of an old solution *without adaptation.*

We call this type of creativity "non-adaptive creativity", because it hinges upon finding a solution that can be applied to a new problem without having to change the solution.

At the character-level, we have already seen some examples of creativity heuristics which do not require adaptation. One of these is TRAM:Recall-Act. TRAM:Recall-Act suggests that if you are trying to find an act that fits some particular set of constraints, then you can probably ignore all the constraints except the goal the act is intended to fulfill and the intended effects of the action. By ignoring the other constraints, TRAM:Recall-Act permits the problem-solver to recall solutions which he would not have otherwise recalled, because they would not fulfill the extraneous constraints. And because the extra problem constraints are extraneous, the recalled solution - even though it does not fill those constraints - does not need adaptation.

TRAM:Recall-Act illustrates the central feature of non-adaptive creativity: re-directing recall to a new area of memory where immediately useful plans are likely to be found. TRAM:Recall-Act finds new solutions to a problem by re-directing recall to solutions that lack the extraneous constraints.

So although MINSTREL cannot use all types of creativity at the author-level, it can use non-adaptive creativity. To understand MINSTREL's author-level non-adaptive creativity works, it is first necessary to understand how MINSTREL's author-level episodic memory is organized.

MINSTREL's author-level plans are indexed in episodic memory according to specific past goals they have solved, just as character-level plans are indexed according to past goals they have solved. So, for example, the author-level plan ALP:Make-Consistent-P-Health is indexed under a goal of type &Check-Consistency applied to a &State schema. A portion of MINSTREL's author-level episodic memory illustrating this organization is shown in Figure 5.7.

Episodic memory is organized as a tree of (feature,value) pairs. &Goal.16 is an author-level goal from a past instance of storytelling in which MINSTREL's goal to check the consistency of a particular state in a story (&State.99) was achieved by ALP:Make-Consistent-P-Health. This is organized in episodic memory by the features and values of &Goal.16, including the object of the goal, &State.99. If a new goal is encountered with similar features, ALP:Make-Consistent-P-Health will be recalled.

MINSTREL also has generalized author-level plans. These are plans that can apply to a number of different objects. Generalized author-level plans have a null Object and are indexed accordingly. Figure 5.7 shows the indexing for a generalized author-level plan called ALP:Default-Consistent. ALP:Default-Consistent is a simple plan that recognizes a schema as consistent if it

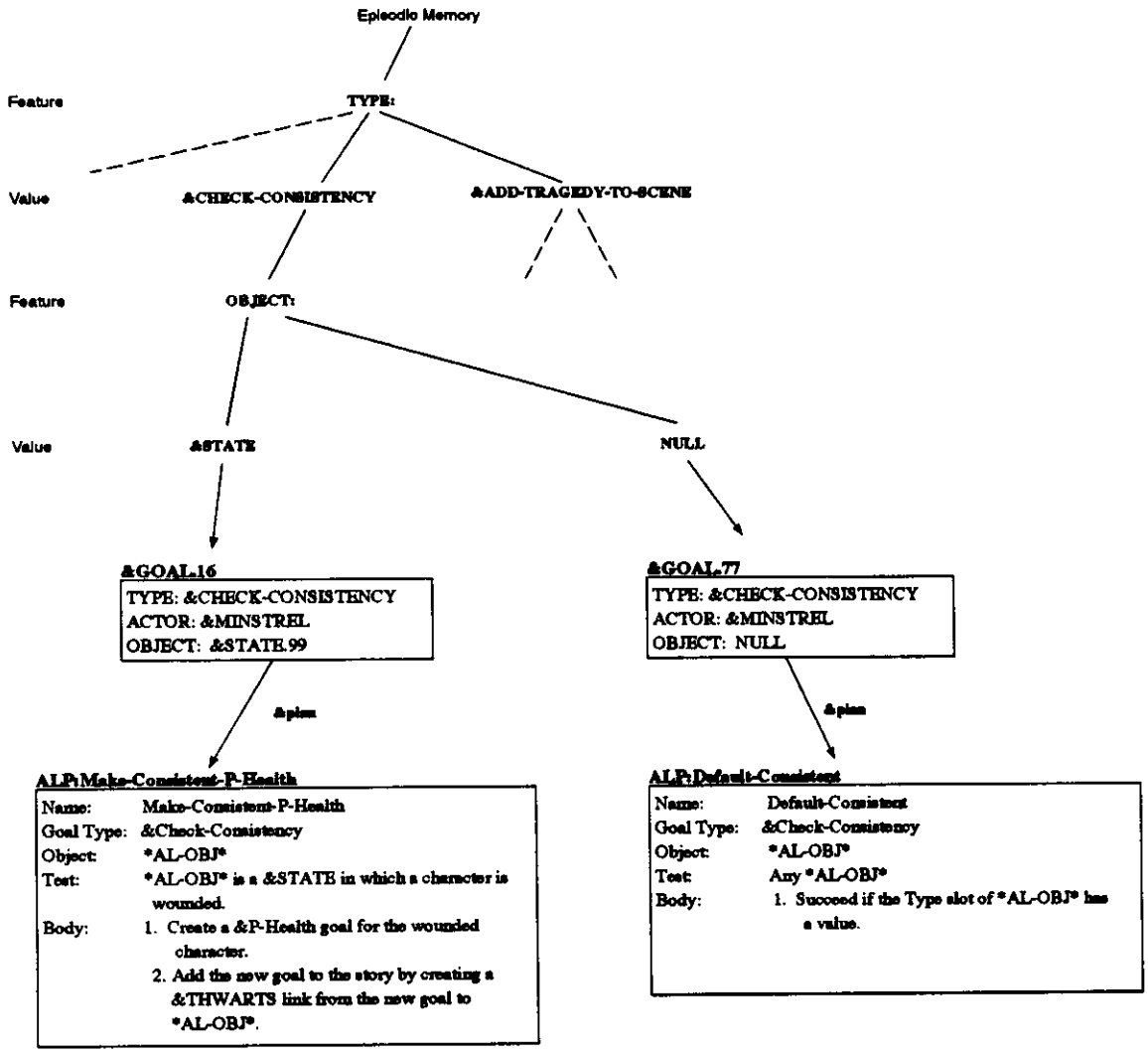


Figure 5.7 Example Organization of Author-Level Memory

has had its Type slot filled in.

When MINSTREL has a goal to the check the consistency of a state schema, the goal will be used as an index to memory and because its features and values match those of &GOAL.16, MINSTREL will find &GOAL.16 and its associated plan, ALP:Make-Consistent-P-Health. This is standard problem-solving - recalling a similar past problem and using the solution to that problem for the current problem.

Non-adaptive creativity comes into play when this fails. When an author-level goal does not recall a similar past goal and its associated solution, or if all the recalled solutions fail, then MINSTREL must look elsewhere in author-level episodic memory for an author-level plan that can be

applied to the current goal without adaptation. There is one place that such a plan can be found: a generalized author-level plan.

Generalized author-level plans will not be found by standard problem-solving because the Object slot of the current goal will not match the null slot under which the generalized plans are indexed. But precisely because these plans are generalized we know that they can be applied without adaptation to any goal of the proper type. What is needed is a creativity heuristic that will find these plans by looking in the appropriate place, i.e., a non-adaptive creativity heuristic.

MINSTREL's TRAM for achieving this is called TRAM:Generalized-AL-Plans. This TRAM finds generalized plans to apply to a specific author-level goal by eliminating the Object slot from the current goal and returning without adaptation whatever plans it finds. TRAM:Generalized-AL-Plans is shown in Figure 5.8.

TRAM:Generalized-AL-Plans

Transform Strategy

Remove the Object slot of the author-level goal.

Adapt Strategy

Do not adapt the recalled solution. Apply it as is to the current problem.

Figure 5.8 TRAM:Generalized-AL-Plans

When TRAM:Generalized-AL-Plans is applied to the author-level goal illustrated in Figure 5.4, it eliminates the Object slot and uses the transformed goal as an index to memory. This recalls ALP:Default-Consistent, which is then applied to achieve the original goal.

Because MINSTREL's TRAMs cannot adapt MINSTREL's author-level plans, creativity at the author-level is limited to non-adaptive creativity. Although non-adaptive creativity is not as powerful as other types of creativity, it does illustrate the invariance of MINSTREL's model of problem-solving and demonstrate that the same creative problem-solving process used to solve character-level goals can also solve author-level goals.

5.3.4 Achieving Author-Level Plans

Our examination of the authoring process began with the identification of author goals. For the particular type of stories that MINSTREL tells, we identified four important classes of author-level goals. Then we looked at the processes of planning and problem-solving: how author goals were managed, and how plans to solve author-level goals were found or created. The last step is to look at the contents of author-level plans: what they do to achieve author-level goals.

The following chapters will discuss in detail each of MINSTREL's author-level plans, identifying

what goal each plan applies to, and describing how the plan achieves that type of goal. But before we turn to the details of MINSTREL's author-level plans, we should like to examine this problem in more general terms. What does an author do to illustrate a theme, use a dramatic technique or correct an causal inconsistency?

Consider a hypothetical author writing a short story about Lancelot, with a goal to portray Lancelot as deceptive. He achieves this goal in two steps.

First, the author uses his knowledge of the goal he is trying to achieve and his knowledge of how stories are told to specify his goal as an abstract description of story events which, if part of the story, would achieve his goal. In this case, the author knows that "deception" is a character trait and that character traits are reflected in character actions. From this he realizes that his goal of portraying Lancelot as deceptive will be achieved if the story contained some scenes in which Lancelot used a deception plan. Note that the author hasn't yet achieved his goal. He has only further specified it as a particular, abstract description of story events which would achieve the goal.

The abstract specification the author arrives at will depend upon his knowledge of storytelling. From reading and writing stories, he will have built up a library of author-level plans that translate author-level goals into story specifications. Illustrating a character trait by including a story event demonstrating the trait might be a plan the author learned through conscious study, or by reading many stories that used this technique.

Next, the author tries to create story elements to fulfill this abstract specification. Using his knowledge of the genre of the story, the goals of the story, and the already-completed portions of the story, the author tries to invent scenes to fit the abstract specification. Thinking about one person deceiving another may remind the author of a time when a co-worker fooled him with a falsified memo from their boss. Being reminded of this scene, the author may decide to use it as the basis for the Lancelot story scene. But to make that reminding work in his story, the author must make some adaptations. Lancelot will have to take the co-worker's place, the memo will have to be replaced with something appropriate to the King Arthur milieu - perhaps a note from the king - and so on. The end result is a scene in which Lancelot fools Guinevere by forging a note from the King.

We call this process of taking an abstract specification and general story constraints and inventing scenes to fit the specification *instantiation.*

The claim of this research is that this two-step process of specification and instantiation is the fundamental process by which author-level goals are achieved. Not all author-level goals are achieved by this process: goals may be achieved by creating sub-goals, reordering events in the story, creating English language sentences, and so on. But the process of specification and instantiation is a pervasive and fundamental element of storytelling.

1. Define an abstract specification of a needed story element.

2. Create a specific story element to match the abstract specification.

5.3.5 The Role of Episodic Memory in Storytelling

How is instantiation achieved? In the example above, the author instantiated his abstract specification by a process of transform, recall and adapt, which the reader will recognize as creative case-based problem-solving. In fact, instantiating an abstract concept description can be viewed as a special form of creative case-based problem-solving. But instantiation differs from normal problem-solving in an important way. In normal problem-solving, the problem-solver uses a complete problem description to recall similar past problems, so that the problem-solver can use the associated plans. But in instantiation, the problem-solver uses an incomplete description to recall a complete description, without any interest in the associated plans.

The product of normal problem-solving is a plan for the current problem. Suppose, for example, that a knight finds himself threatened by a dragon and wants to save himself. The knight recalls a past situation in which he was threatened by a troll. He'd solved that by charging the troll on his horse, so he decides to apply that same plan to the dragon. By recalling a similar past problem-solving situation, the knight has found a plan to solve his current problem.

In instantiation, though, the product of problem-solving is the *recalled problem situation*, not the associated solution. Suppose, for example, that an author is creating a scene in which "Lancelot, a knight, is endangered." This abstract description is passed to case-based problem-solving and recalls a past situation in which a knight was threatened by a troll and consequently killed the troll by charging it on his horse. The author can now use the recalled problem situation to fill in or instantiate the current scene. The associated plan (charging the monster on horseback) may or may not be used, depending on the author's particular storytelling needs. In fact, the scene being instantiated could be a belief, an emotion, or some other type of story element that does not have a plan associated with it at all.

Because there is not necessarily an associated problem solution, and because instantiation may not make use of the problem solution even if it exists, there is no need to perform the second and third steps of problem-solving: adapting the recalled solution and assessing the result. Instantiation requires only the recall of a similar past scene, and not the rest of the problem-solving effort.

However, a problem arises when the author cannot recall a similar past scene. In this case, recall alone is not sufficient, because the author needs to create a story scene to fit his abstract scene description.

The solution is to use *imaginative memory*. Imaginative memory incorporates creativity into the recall process, permitting episodic memory to invent a memory to match a set of recall indices. If the recall indices are the features of an abstract scene specification, then imaginative memory will either recall or invent the specific story scene needed to instantiate the scene specification. Instantiation is thus simply the process of imagination – using creativity and the knowledge in episodic memory to imagine scenes to fit a particular criteria.

Instantiation is achieved through imaginative memory.

It's a common intuition to link imagination with storytelling. By explicitly representing instantiation as a fundamental process in storytelling and showing how instantiation can be achieved by imaginative memory, MINSTREL identifies and defines the link between imagination and storytelling. And by showing how creativity can be incorporated into the recall process, MINSTREL demonstrates the link between creating new problem solutions and the ability to imagine plausible new situations and ideas.

5.3.6 Planning With Many Constraints

One reason storytelling is a difficult task for humans and computers alike is that it requires the storyteller to solve a large number of interdependent goals simultaneously. In a successful story, the events of a story fulfill a range of goals. They illustrate the theme of the story, develop the literary value of the story, maintain the story consistency, and so on. But it is difficult for humans to solve planning situations that involve a number of simultaneous goals [Flower 1980]. Each added goal increases the complexity of the planning task, until it may be nearly impossible. "The act of writing is best described as the act of juggling a number of simultaneous constraints." ([Flower 1980], pp. 31).

One solution to planning with many constraints which is often applied to the storytelling problem is to partition the problem into semi-independent subproblems ([Flower 1980]). Rather than try to solve all the problems involved in creating a story at once, the author breaks the writing process down into a sequence of goals, i.e., theme, drama, consistency, presentation.

The technique of delaying constraints is often sounded in writing handbooks:

> ...begin by doing some "free writing". This simply means writing down whatever comes into your mind about a subject... this is not the time to think about spelling, punctuation, or the correct choice of words... What you'll produce is generally called the first or "rough" draft, and it will probably need some revisions and changes... [Tchudi 1984], pp. 21-24.

Delaying constraints or partitioning the storytelling problem into semi-independent subproblems permits the author to solve problems that would otherwise be too complex. But one consequence of this strategy is that solving one subproblem may violate another. For example, MINSTREL creates the following story scene in order to illustrate the theme "Done in haste is oft regretted":

```
One day, Lancelot wanted to kill Frederick.  Lancelot fought
with Frederick.  Frederick was dead.
```

This scene does serve to illustrate the theme (by showing Lancelot performing a hasty action he will later regret). But it is otherwise incomplete. It does not explain why Lancelot wants to kill Frederick, or how Lancelot came to be in the same location as Frederick. The goal to illustrate

the theme has been achieved, but other goals have not.

Human authors have the same difficulties when they partition complex problems. Problem constraints may be violated or even forgotten. An article on revising that appeared in *Writer's Digest* includes a "Tactical Analysis Checklist" of mistakes of this sort:

> • Look at the motives of your major story characters. Have you included sufficient information about their past lives to make it credible that they want what they want, feel what they feel, think as they think, act as they act, or have the skills they call on in your plot? ([Bickham 1992], pp. 28)

For both human and computer authors, breaking a complex problem with many constraints into simpler, semi-independent subproblems is a valuable writing strategy. But because a story is an integrated whole, no subproblem can be truly independent of the other subproblems. So the author must be prepared to detect and correct constraint violations during the writing process. In MINSTREL, this is achieved through opportunistic goals.

5.3.7 Active vs. Opportunistic Goals

At any time during the storytelling process, MINSTREL has a number of goals that it is actively trying to achieve. Each of these goals is present on the goal agenda and the active goal with the highest priority is in the process of being achieved. Initially, the active goals include the goals to (1) tell a story illustrating a particular theme, and (2) to present the story to the reader in English. These goals are achieved via author-level plans that may create additional active goals (i.e., subgoals).

In addition to the active goals, storytelling conditions can cause *opportunistic goals* to arise. An opportunistic goal is an author-level goal that becomes active whenever a specified storytelling situation arises. For example, MINSTREL has an opportunistic goal which arises whenever a story scene is created which contains inconsistencies. When an inconsistent story scene is created an opportunistic goal to correct the inconsistency is triggered. Similarly, MINSTREL has opportunistic author-level goals to achieve various dramatic writing purposes, such as building suspense. When a story scene is created which is suitable for suspense (such as a character's life being threatened), an opportunistic goal arises to consider building suspense in that scene.

Opportunistic goals serve two purposes in MINSTREL. First, opportunistic goals provide a mechanism for detecting and correcting constraint violations. As described above, MINSTREL reduces the difficulty of the storytelling problem by breaking it into a sequence of semi-independent sub-goals. Opportunistic goals provide a mechanism for detecting when the solution of one sub-goal violates the constraints of another sub-goal, as for example when a scene created to illustrate the theme of a story contains causal inconsistencies.

Second, opportunistic goals are used to represent secondary author goals. In general, we suppose that any author has a variety of goals when telling a story, some which are primary and some

which are secondary. These two categories represent a sharp, qualitative divide in author priority. The author's primary goals are those which the author considers essential to the telling of the current story. If the author cannot achieve his primary goals, then the storytelling process has failed. For MINSTREL, the primary goals are (1) to tell a story concerning a particular theme, (2) to make the story consistent and believable and (3) to present the story in English. If any of these goals fail, then the storytelling process itself has failed.

Secondary goals, on the other hand, represent goals which are desirable but not essential to the storytelling process. The author is pleased if he can achieve these goals, because it means that his story is better than it might otherwise be. But if the opportunities do not arise, the story is not a failure. MINSTREL's goals to use literary techniques are secondary goals. If they succeed, they add additional quality and complexity to the stories MINSTREL tells, but the story can succeed even if they fail. (Although it will likely have less literary value.)

Of course, how an author categorizes his goals varies from author to author, and from story to story. For a writer of mystery stories, building suspense is probably an essential goal in the writing process. But for the writer of romantic fantasies, it is an secondary goal, and for the writer of comedy it may be actively avoided. And even for a single author, goal priorities will vary from story to story.

In terms of the storytelling process, primary and secondary goals differ primarily in how they arise. Primary goals are self-directed, while secondary goals are reactive. In MINSTREL, primary goals arise directly from MINSTREL's initial storytelling goal. MINSTREL's initial goal ("tell a story") creates subgoals to tell a story about a particular theme, to tell a consistent and believable story, and to tell the story in English, i.e., all of MINSTREL's primary goals. The secondary goals, on the other hand, arise in reaction to the developing story. In MINSTREL, each secondary goal is represented by an opportunistic goal. If the opportunity arises to achieve a secondary goal, then the opportunistic goal triggers. Thus primary goals are guaranteed to arise and be attempted, while secondary goals may or may not arise, depending on the story development.

It is also interesting to note that author-level plans used to achieve a primary goal are generally not suitable for achieving that same goal opportunistically, and vice versa. For example, MINSTREL has an author-level plan to add suspense to a story scene by having a character attempt an escape from a dangerous situation. This plan cannot be used actively, because it depends upon the prior existence of a dangerous situation. In general, plans to achieve primary goals must be able to create the story events necessary to achieve the goal from scratch, while opportunistic goals can be achieved by plans which modify or augment existing story scenes.

5.4 Conclusions

MINSTREL has three major advancements over previous models of storytelling.

First, MINSTREL demonstrates the importance of an explicit author model in storytelling. The particular author-level goals and plans MINSTREL uses to tell stories are of greate interest, and

are fully discussed in the following chapters. But as important is MINSTREL's architecture as a storyteller with an explicit knowledge of its own goals.

Second, MINSTREL models storytelling as problem-solving. By this, MINSTREL clarifies the relationship between achievements in artistic domains such as storytelling and achievements in traditional problem-solving domains such as mechanical repair. MINSTREL demonstrates the fundamental similarities between artistic endeavors and traditional problem-solving, and MINSTREL is prima facie evidence that artistic ability can be explained in terms of problem-solving, and that no further or different cognitive process need be stipulated.

Third, MINSTREL explicitly models instantiation as a fundamental storytelling process. By recognizing the importance of instantiation, MINSTREL is able to clarify and define a process that has previously been unrecognized and unilluminated. And by implementing instantiation using imaginative memory, MINSTREL further defines the links between creativity, memory, and problem-solving.

CHAPTER 6
Thematic Goals in Storytelling

6.1 Introduction

When MINSTREL tells a story, its primary goal is to illustrate a particular story theme. MINSTREL's story themes are stereotypical planning situations, which can often be summarized by an adage such as "A bird in the hand is worth two in the bush" or "Deception serves the devil." The author-level goals concerning the selection and development of a story theme are called thematic goals. To understand the role of story themes in MINSTREL's storytelling process, there are five issues that must be addressed:

(1) What is a theme?
(2) How is a theme represented?
(3) What themes does MINSTREL know?
(4) What author-level thematic goals does MINSTREL have?
(5) What plans does MINSTREL have to achieve these goals?

6.2 What Is A Theme?

Webster's New Collegiate Dictionary defines a theme as "the subject or topic of discourse or of artistic expression"; Roget's Thesaurus lists synonyms for themes that include "point", "motif", "topic", "pattern" and "design". The theme of a story is the underyling concept or topic that organizes the story into a coherent whole; it is the theme that gives rise to a story's organization and structure.

But stories are told for a wide variety of purposes. People use stories as teaching devices, for amusement, and to illustrate human nature. Consequently themes are as varied as stories themselves.

Fortunately, it isn't necessary to have a precise definition or classification of story themes in order to investigate the processes and knowledge involved in storytelling. Instead, we can restrict our study to specific themes or classes of themes that we can define and classify, and look at how stories are told about those themes. The hope is that what we learn about a specific type of theme will give us insight and understanding about themes in general.

One type of story theme is characterized by the tales in "Aesop's Fables." Each fable presents a stereotypical planning situation and shows the consequence of a particular planning decision. An example of an Aesop's Fable, "The Fox and the Crow", is shown in Figure 6.1.

"The Fox and the Crow" illustrates the consequences of listening to false praise. The themes of Aesop's Fables are captured by simple adages or morals, such as "Pride goes before a fall." Because these themes give advice about how a planner should act, we call this type of story theme a

A crow who had stolen some meat,
Came down in a tree
Where a fox who wanted the treat
Could look up and see.

So the clever fox started to sing
The bird's praises. "Our choice,"
He said, "you would be for King,
If you had a good voice."

To show him, the crow dropped the meat
And raised a loud croaking,
While the fox ate the meat at his feet
So fast he was choking.

But he managed these words to the crow,
Who now felt rather small:
"Kings need lots of things, you know,
But good sense most of all!"

Figure 6.1 The Fox and the Crow [Rees 1966]

"Planning Advice Theme". Examples of other Planning Advice Themes are shown in Figure 6.2.

Adage	Meaning
A bird in the hand is worth two in the bush.	*Don't abandon an achieved goal to achieve another goal, because you may end up with neither.*
Pride goes before a fall.	*Don't use pride as a justification for an action.*
Honesty is the best policy.	*Don't use plans which call for lying.*
The early bird catches the worm.	*Initiate plans to achieve goals in a timely fashion.*

Figure 6.2 Example Planning Advice Themes

Planning Advice Themes are suitable for computer storytelling for several reasons. They are simple, easy to understand and recognize, and yet interesting enough to appear in a wide range of literature. MINSTREL uses Planning Advice Themes from Aesop's Fables, Hollywood movies, and *Romeo and Juliet*. By limiting MINSTREL to Planning Advice Themes, we are able to address many of the issues in telling theme-based stories without the need to create a general theory of themes.

6.3 Representing Planning Advice Themes

Story themes have been of interest to researchers in artificial intelligence for some time, and many different representations for story themes have been proposed. These include Thematic Organization Packets ([Schank 1982][Hammond 1990]), story points ([Wilensky 1982]), and plot units ([Lehnert 1982]). Although each of these representations captures some type of story abstraction, none of them looked specifically at planning advice themes, or sought to explicitly represent themes as planning advice.

One representation which did capture the planning advice aspect of themes was Thematic Abstraction Units, or TAUs ([Dyer 1983]). TAUs were used in program called BORIS that read and developed an in-depth understanding of narratives. BORIS used TAUs to guide its understanding of stories in which unexpected events occurred. An example TAU (TAU-POST-HOC) is shown in Figure 6.3.

TAU-POST-HOC

(1) x has preservation goal G active since enablement condition C is unsatisfied.

(2) x knows a plan P that will keep G from failing by satisfying C.

(3) x does not execute P and G fails. x attempts to recover from the failure of G by executing P. P fails since P is effective for C, but not in recovering from G's failure.

(4) In the future, x must execute P when G is active and C is not satisfied.

Adage: "Closing the barn door after the horse is gone."

Figure 6.3 TAU-POST-HOC

TAU-POST-HOC represents a specific type of planning situation, in which a planner tries to recover from a goal failure by executing a prevention plan after the goal failure. This is represented as a sequence of abstracted planning actions. When BORIS encountered a story scene that matched one of the actions in a TAU, it could use the TAU to predict possible future story events, or to understand past story events.

For example, upon reading the story fragment:

> The hired hand always wanted a raise, but the farmer would not grant it. Finally, the hired hand got an offer to work at a neighboring farm.

BORIS could use TAU-POST-HOC to expect a story event in which the farmer belatedly offers a raise, and this fails to retain the hired hand.

As TAU-POST-HOC illustrates, TAUs consist of an abstracted sequence of planning actions. This sequence of planning actions can be used to anticipate or expect future events, or used to

retroactively understand confusing past events. But although TAUs capture abstract planning situations, and can be used to index recovery or avoidance plans (as in (4) in Figure 6.3), they do not explicitly structure this knowledge as advice. TAU-POST-HOC does not identify what the planner's decision was, what part of the planning process the decision applied to, what the bad consequence of the decision was, in what contexts the decision is bad, and so on. To find and apply this knowledge, a planner must reason from the goal/plan structure of TAU-POST-HOC. In general, this may be difficult and inefficient.

For example, in TAU-POST-HOC, a reasoner looking for advice might find either of two bits of planning advice. One is captured by (4) in Figure 6.3, namely "One should execute a preventive plan when an enablement condition is unsatisfied", and the other is "One should not execute a preventive plan when the goal failure has already occurred." These two pieces of advice apply to different parts of the planning process (the first applies to goal activation, the second to plan selection), in different contexts, and so on. Extracting this knowledge from the goal/plan structure of TAU-POST-HOC, identifying its structure as advice, and resolving any ambiguities is a difficult task.

MINSTREL has addressed this problem by expanding TAUs to explicitly represent the advice aspect of planning advice themes. To the representation of TAUs as abstract goal/plan networks MINSTREL adds a structuring of the network as advice. Making the advice aspect of story themes explicit (1) identifies unambiguously the advice content of a story theme, (2) permits efficient use of this knowledge, and (3) more accurately defines a class of story themes.

What is needed in a representation of planning advice? Planning advice tells a planner what decision to make in a particular planning situation. Consequently, a representation of planning advice must capture three things: (1) the planning decision, (2) a value judgement about the planning decision, and (3) a justification for the value judgement.

The *planning decision* is a description of the planning decision to which the advice applies. It represents the decision the planner is making as well as the context in which the decision is being made.

The *value judgement* indicates whether the planning decision was good or bad.

The *justification* for the advice is the reason or explanation for the value judgement. If the judgement is positive, the reason explains why the planning decision is a good decision. If the judgement is negative, the reason explains why the planning decision is a bad one.

The two pieces of advice implicit in TAU-POST-HOC are:

(1)	*Planning Decision*	Planner chooses not to activate a goal to alleviate an unsatisfied enablement condition.
	Judgement	Negative
	Justification	The unsatisfied enablement condition will cause a goal failure.
(2)	*Planning Decision*	Planner selects a prevention plan P to recover from a goal failure.
	Judgement	Negative
	Justification	Prevention plans cannot be used to recover from goal failures.

MINSTREL represents themes as schemas called Planning Advice Themes (or PATs). Figure 6.4 lists the parts of a PAT and gives a simple textual example of a Planning Advice Theme, PAT:Violent-Plans. PAT:Violent-Plans advises a planner to avoid plans which involve violence, because the violence might backfire, i.e., "Live by the sword, die by the sword."

The first part of the PAT identifies the type and value of the planning advice:

Type identifies the decision point in the planning process to which the advice applies. MINSTREL currently knows about two decision points in the planning process: Plan Selection and Goal Activation. Plan Selection occurs when a planner selects a plan to achieve a goal. PATs about Plan Selection either encourage or discourage the use of particular plans in particular situations. Goal Activation occurs when a plan activates a goal in response to a change in the state of the world. PATs about Goal Activation either encourage or discourage activating particular goals in particular situations. Other decision points in the planning process include Goal Selection (which goal to attempt to achieve from a collection of goals) and Goal Abandon (when to quit pursuing a goal), but MINSTREL does not currently represent any themes based on these planning decision points.

PAT:Violent-Plan is a PAT of Type "Plan Selection" because it advises one not to select a plan that involves violence to achieve a goal. An example of a Goal Activation PAT is "A bird in the hand is worth two in the bush". This theme advises a planner not to activate a goal that could cause the thwarting of another, already achieved goal.

Value determines whether a PAT is positive or negative advice. Positive advice encourages a planner to do something; negative advice informs him of a planning failure to avoid. For PAT:Violent-Plans, the Value is Negative, because PAT:Violent-Plans is advising the planner to *avoid* selecting a plan that involves violence.

The second portion of the PAT represents the advice itself:

Decision is an abstract schema-based representation of the planning choice identified by Type.

Type of Advice	Explanation	PAT-Violent-Plan
Decision Point	The planning decision type this advice applies to.	*Plan Selection*
Value	Is this positive or negative advice?	*Negative*
Advice		
Decision	An abstract, stereotypical representation of a planning decision.	*Planner chooses a violent plan to achieve a goal.*
Consequence	An abstract, stereotypical representation of the consequence of the decision.	*The violent act backfires, and Planner is hurt.*
Connection	The causal connection between the decision and its consequence.	*The violent plan intends a violent action.*
Object	The goal, plan, or action to which the decision applies.	*The violent plan.*
Planner	Who the decision-maker is.	*"Planner"*
Context		
Active Goals	A description of the relevant goals in the planner's goal tree at the time he makes the decision.	*The goal Planner is trying to achieve.*
Current Goal	The planner's current goal.	*The goal Planner is trying to achieve.*
Current Plan	The planner's current plan.	*The violent plan.*
World Facts	Context in which to apply this advice.	*None*

Figure 6.4 Representation of Planning Advice Themes

A Plan Selection decision is represented by a goal and the plan selected to achieve that goal. The details of the goal and the plan represented depend upon the advice being represented. For PAT:Violent-Plans, the goal is any goal, and the plan is any violent plan. Thus, the Decision for PAT:Violent-Plans represents a planner's decision to use a violent plan to achieve a goal.

Consequence is an abstract, schema-based representation of the consequence of the Decision. The Consequence of a PAT represents the reason why Decision should be made (if the PAT is Positive advice) or avoided (if the PAT is Negative advice).

The Consequence of a Negative PAT (a theme which advises against some planning decision) generally involves a failed goal for the planner. In Positive PATs, the Consequence typically includes an achieved goal for the planner.

In PAT:Violent-Plan, the Consequence is a representation of the plan backfiring, causing the planner, rather than some other person, to be hurt. Since this PAT is negative advice, the Conse-

quence involves a goal failure (loss of health) for the planner. This bad result is the reason to avoid the planning decision represented in the Decision part of the PAT.

Connection identifies the causal link between the Decision and the Consequence. Typically, this is the chain of events that leads from the planning decision to the achieved or failed goal in the Consequence. In PAT:Violent-Plan, the connection between the Decision and the Consequence is that the violent plan intends a violent action (which subsequently backfires).

Object identifies the specific schema within the Decision to which the advice applies. This is necessary because the representation of the Decision may have many different plans, goals, actions and actors. In PAT:Violent-Plan, the Decision has only one representation of a Plan Selection, so identifying the planning decision to which the advice applies is trival. But in general, a PAT may have a complex network of goals and plans in the Decision, so Object is necessary to identify the specific decision point to which the advice applies.

Planner identifies the actor to which the advice applies, in a way similar to Object. The Decision, Consequence and Connection may involve many different actors. Planner identifies the actor to which the PAT applies.

The third portion of the PAT represents the context in which this advice applies. The context has two parts: knowledge about the world and knowledge about the planning process. Context knowledge is used to determine when advice is applicable. For example, there may be advice that applies only when the planner's life is in danger (knowledge about the world) or when the planner is trying to achieve two mutually incompatible goals (knowledge about the planning process).

Current Goal represents the planner's current goal. The current goal is the goal associated with the planning decision of this PAT.

Current Plan represents the plan, if any, associated with the planning decision of this PAT.

Active Goals represents the planner's active goals which affect whether or not this advice should apply. A planner is assumed to have a prioritized list of goals that he is trying to achieve; Active Goals is a list of goals that must be present in this goal tree for this advice to apply.

For PAT:Violent-Plan, the goal the planner is trying to achieve by using a violent plan is the only relevant goal. The current goal will always be in the Goal Tree part of a PAT. For some planning advice, other goals may also be relevant.

Consider, for example, the advice "Save yourself first." This adage advises a planner to achieve a goal of saving himself from some harm before attempting to achieve a goal to save someone else from harm. In this case, the relevant goals from the planner's goal tree are the goal to save himself and the goal to save another.

World Facts represent facts about the world which must be true in order to apply the advice rep-

resented by this PAT. As a simple example, the advice "Save yourself first" would apply only if it were true that the planner's life was in danger. In MINSTREL, facts about the world are represented by state schemas.

The primary advantage of MINSTREL's Planning Advice Themes is that they explicitly organize thematic goal/plan structures as advice. Previous representations of story themes (TOPS [Schank 1982][Hammond 1990], story points [Wilensky 1982], TAUs [Dyer 1982]) have concentrated on representing and categorizing the goal/plan structures that appear as story themes. To this representation PATs add a secondary structure as advice. PATs add indices into goal/plan representations of story themes that identify parts of the themes as parts of an advice structure.

The purpose of representing story themes as both goal/plan structures and as advice structures has been to permit MINSTREL to use the secondary organization of the theme as advice to guide the use of the theme in creating stories. This knowledge is used in a number of ways by MINSTREL. For example, because MINSTREL knows that the World Facts of a PAT are preconditions for applying the advice of the theme, it knows to present the story scenes that correspond to the World Facts *before* presenting the rest of the theme. Similarly, because it is easier to create the result of an action once the action is known, and MINSTREL knows from the PAT that the Consequence is a causal result of the Decision, MINSTREL knows to create the story scenes corresponding to the Decision before creating the story scenes corresponding to the Consequence.

A secondary reason to develop a combined representation of story themes and advice is to clarify, simplify and consolidate these two different types of representation. Advice is often given in the form of stories; and stories can often be seen as advice. Creating a unified representation for story themes and advice may help us better understand the underlying connections between these two topics. One question which MINSTREL's representation of themes as advice raises is: Are there common story themes which *cannot* be represented as advice? If not, this would suggest that story themes and advice share an underlying representation. If there are, it would advance our understanding of the types and forms of story themes. MINSTREL does not address these questions directly, but hopefully MINSTREL's representation of story themes as PATs will provide the basis for investigating these and similar questions in the future.

6.4 MINSTREL's Plan Advice Themes

In MINSTREL, Plan Advice Themes have been used to represent six different story themes: two themes from *Romeo and Juliet*, a theme from the Frank Capra movie *It's a Wonderful Life*, and three themes based on adages. Unlike PAT:Violent-Plans, these themes have been fully represented using a schema-based representation system. These themes are described in the following six sections.

6.4.1 PAT:Good-Deeds-Rewarded

PAT:Good-Deeds-Rewarded is based on a theme from the movie *It's a Wonderful Life*. *It's a Wonderful Life* stars Jimmy Stewart as George Bailey, a good, unselfish man who constantly turns away from his dreams in order to do good deeds for others. When it appears that the town tyrant is going to take the Bailey Savings and Loan away from George he despairs and wishes that he had never been born. An angel shows George what the lives of his friends and family would have been like if he had never been born and that experience restores his spirit. At the end of the film, the people he has helped all his life band together to help him save the bank. One of the central themes of this movie is that one should be kindly and help others when possible, because someday that kindness may be returned. PAT:Good-Deeds-Rewarded represents this theme.

PAT:Good-Deeds-Rewarded is a Goal Activation theme. This PAT advises a planner to activate the goal to help others under any circumstances (i.e., there is no restricting context). The Consequence that justifies this advice is that someday the favor will be returned.

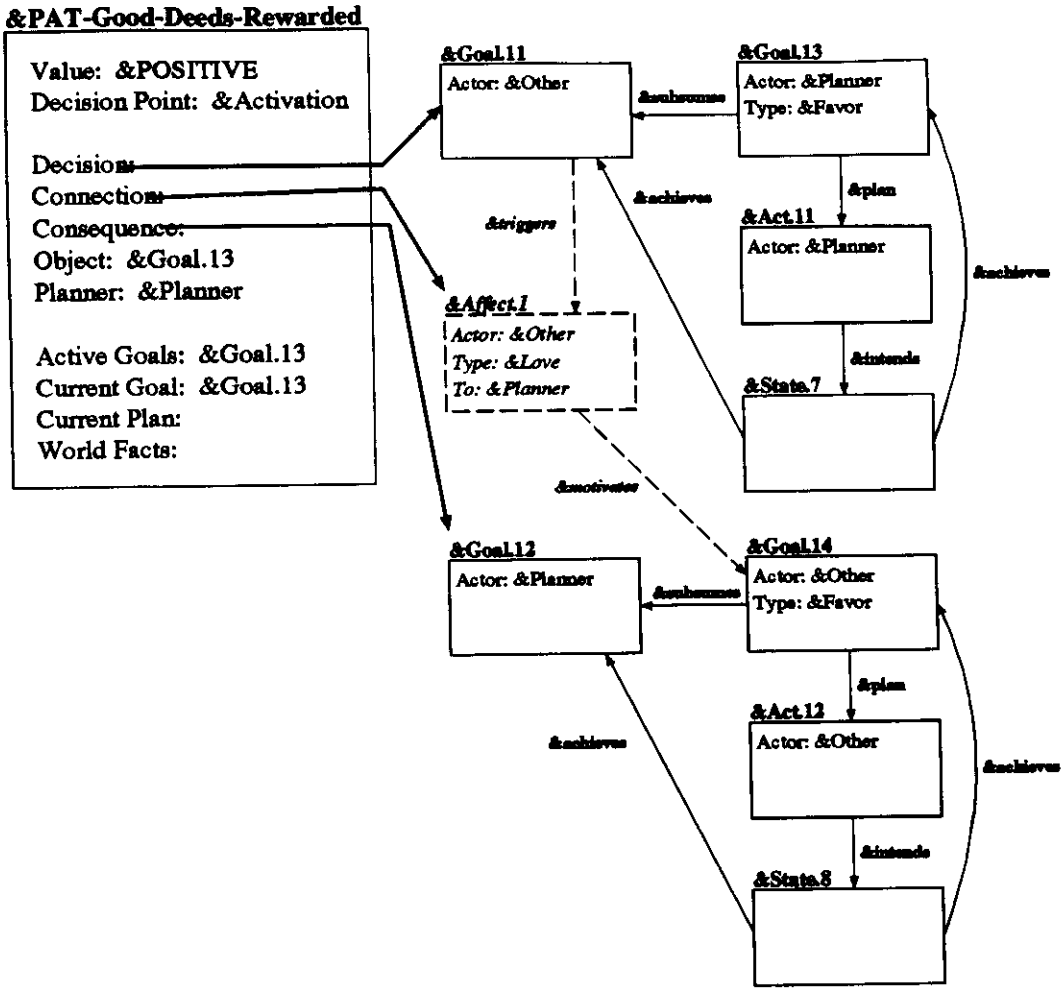
Figure 6.5 illustrates MINSTREL's representation of PAT:Good-Deeds-Rewarded. The Value of PAT:Good-Deeds-Rewarded is &POSITIVE, because it advises a planner to do something (activate a goal). The Decision Point is &Goal-Activation, because it applies to the point in the planning process in which a planner is deciding whether or not to activate a goal.

The Decision, Connection and Consequence portions of this PAT all point to abstract schema representations. These schemas involve two characters, &Planner and &Other. As indicated by the Planner slot of PAT:Good-Deeds-Rewarded, the character to whom this planning advice applies is &Planner. The actual goal the planner is advised to activate is indicated by the Object slot, which points to &Goal.13.

The schema representation pointed to by the Decision slot represents &Planner doing a favor for &Other. A favor happens when one character intentionally helps another character achieve a goal without achieving any goals of his own. (The help must be intentional to distinguish a favor from a happy accident; the help must not achieve one of the planner's own goals to distinguish from self-interest such as acting as a paid agent.) MINSTREL represents favors as a &FAVOR goal of the planner, which subsumes the other character's goal. In PAT:Good-Deeds-Rewarded, &Goal.13 represents the &Planner's goal to help &Other. &Act.11 represents &Planner's actions to achieve both the favor goal and the subsumed goal.

The schemas pointed to by the Connection slot (which are shown dotted) represent &Other's emotional response to &Planner's favor. &Other has a positive emotional response to &Planner (&Love) as a result of the favor. This emotional response later motivates &Other to return the favor. The Connection explains the causal events that lead from a planning decision (Decision) to eventual good or bad consequences of that decision (Consequence).

In PAT:Good-Deeds-Rewarded, Consequence points to another favor, this time with &Other acting to help &Planner.



Summaries	
Decision:	&Planner does a favor for &Other, helping him achieve a goal.
Connection:	&Other loves &Planner for his good deed.
Consequence:	&Other returns &Planner's good deed.

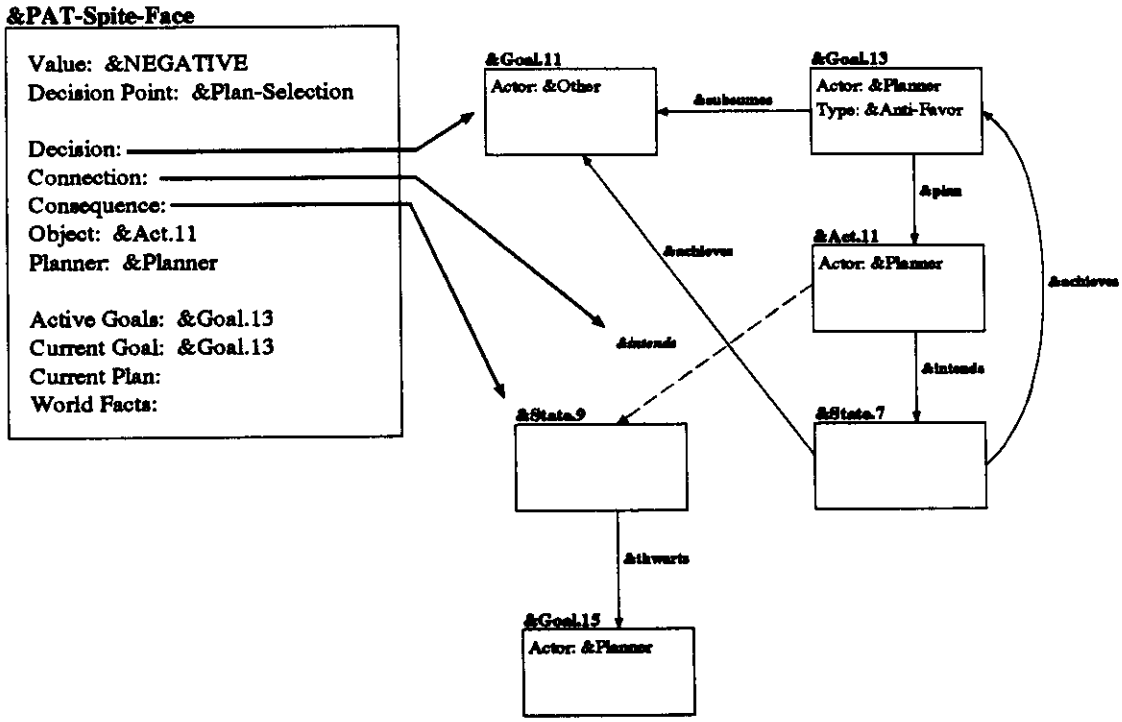
Figure 6.5 PAT:Good-Deeds-Rewarded

In English, PAT:Good-Deeds-Rewarded can be summarized as: A planner should do favors for others, because if he does, he'll make the other person love him, and that will later lead to the favor being returned.

6.4.2 PAT:Spite-Face

PAT:Spite-Face is based upon the adage "cutting off one's nose to spite one's face." PAT:Spite-Face is a Plan Selection theme which advises a planner not to cause a goal failure for another person by an action which will also cause a goal failure for the actor.

Figure 6.6 illustrates PAT:Spite-Face. The Decision portion of this PAT represents a planner (&Planner) taking an action (&Act.11) to spite another character (&Goal.13). Notice that MINSTREL represents a spiteful act as an "anti-favor" - one character intentionally doing something to cause another character's goal failure. The Consequence of this action is that the very act which spites the other character also spites the &Planner, by causing him a goal failure (&Goal.15).



Summaries	
Decision:	&Planner does an anti-favor for &Other, causing him a goal failure.
Connection:	There is another outcome of the anti-favor act.
Consequence:	The other outcome causes a goal failure for &Planner.

Figure 6.6 PAT:Spite-Face

Notice that the action by which the planner spites himself is intentional. That is, it is part of the definition of this theme that the planner knows and intends to hurt himself in order to hurt another. This distinguishes this theme from themes like "Revenge is a dish best eaten cold" which ad-

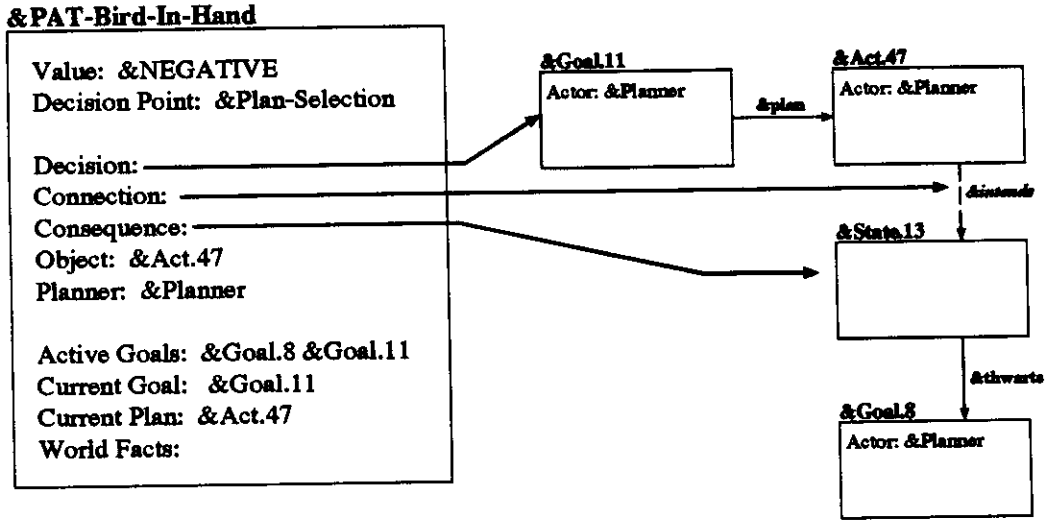
vise a planner to take care in executing anti-favors, to avoid *unintentionally* spiting oneself.

In English, PAT:Spite-Face can be summarized as "Don't do an action to harm another that will harm you as well."

6.4.3 PAT:Bird-In-Hand

PAT:Bird-In-Hand is based on the adage "A bird in the hand is worth two in the bush." This adage advises a planner to avoid using plans which will cause the failure of a previously-achieved goal, because if the plan fails, the planner will not only not achieve his goal, he will also suffer a goal failure of the previously-achieved goal.

PAT:Bird-In-Hand is a Plan Selection theme. Figure 6.7 illustrates PAT:Bird-In-Hand.



Summaries	
Decision:	&Planner choose an action to achieve a goal.
Connection:	That action thwarts &Planner's other goal.
Consequence:	The other goal is thwarted.

Figure 6.7 PAT:Bird-In-Hand

Unlike PAT:Good-Deeds-Rewarded and PAT:Spite-Face, PAT:Bird-In-Hand has a context which restricts its applicability. PAT:Bird-In-Hand is applicable only if the planner has a previously-achieved goal that can be violated by an action in the plan (i.e, if you don't have a bird in your hand, trying to catch the two in the bush is fine). This requirement is captured in the Active Goals slot of PAT:Bird-In-Hand, which indicates that both the current goal (&Goal.11) and a previously achieved goal that will be violated (&Goal.8) must be in the &Planner's goal tree.

The Decision portion of PAT:Bird-In-Hand represents a planner choosing a plan to achieve a goal. The Connection and Consequence show the plan intending a state which thwarts a previously-achieved goal, and which does not achieve the original goal.

6.4.4 PAT:Juliet

PAT:Juliet is inspired by Shakespeare's *Romeo and Juliet*. In the last act of *Romeo and Juliet*, Juliet takes a potion that makes her appear to be dead. Her intention is to deceive her family, but she deceives Romeo instead. Romeo, stricken with grief at the apparent loss of Juliet, kills himself. There are several planning errors in the last act of Romeo and Juliet. PAT:Juliet captures one of these: using a deception plan that fools an unintended person. PAT:Hasty-Impulse-Regretted, below, captures another.

PAT:Juliet is a Plan Selection PAT which advises a planner not to make use of deception plans, because they may fool someone unexpected and lead to a goal failure for the planner.

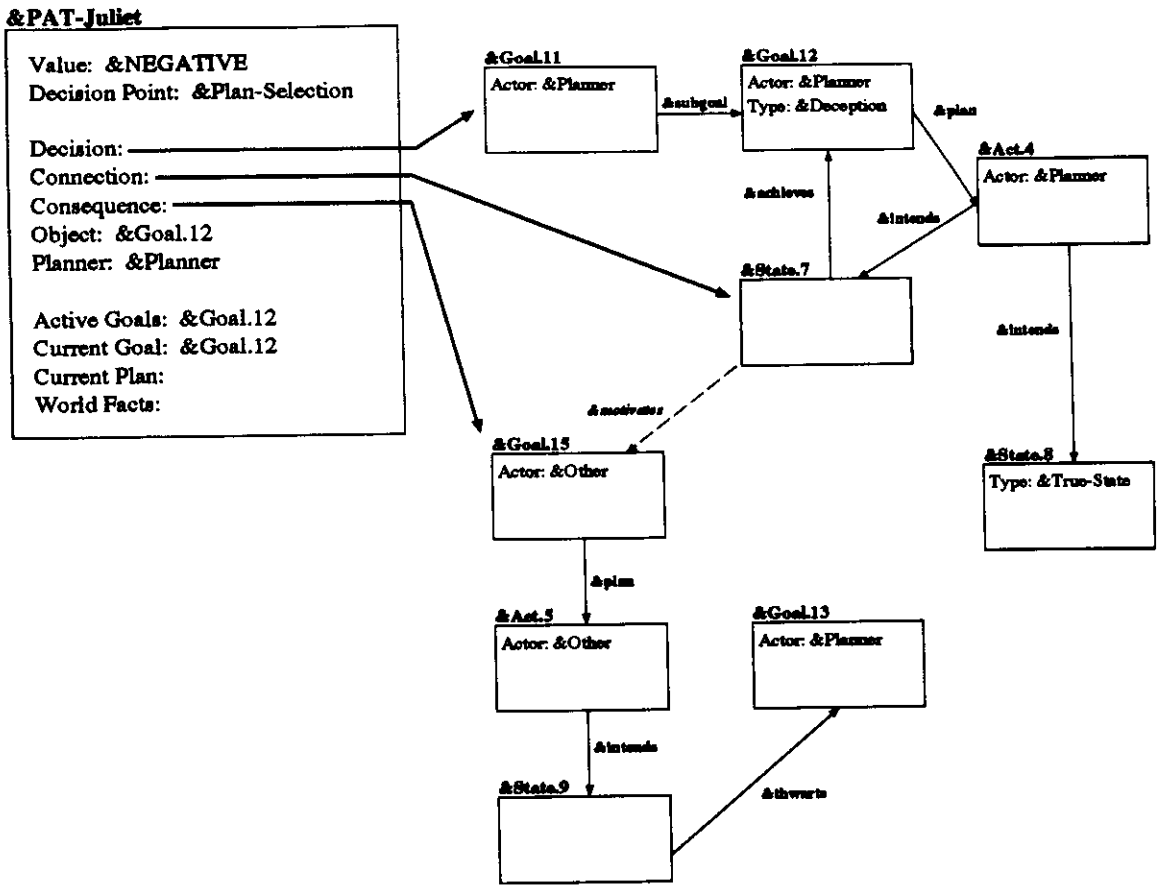
PAT:Juliet is a generalization of TAU-Avoid-Deceiving-Coplanner as presented in [Dyer 1982]. TAU-Avoid-Deceiving-Coplanner represents the situation in which a planner fails to inform a co-planner of a deception plan and a failure results. PAT:Juliet differs from TAU-Avoid-Deceiving-Coplanner in that the deceived party need not be a co-planner. TAU-Avoid-Deceiving-Coplanner advises that deception plans are dangerous only if one fails to inform one's co-conspirators of the plan; PAT:Juliet advises that a deception plan is bad because even fooling an uninvolved third party can lead to failure.

Figure 6.8 illustrates MINSTREL's representation of this theme.

In this figure, the Decision represents a planner's selection of a deception plan to achieve a goal. A deception is when an actor purposefully presents a false state to the world, i.e., makes it appear as if one thing is true when in fact it is not. MINSTREL represents a deception as an act that intends two different states of the world. One of these is explictly marked as the true state of the world. The other state is the deceptive state. In PAT:Juliet, &State.7 is the deceptive state and &State.8 the true state arising from the planner's goal to use deception (&Goal.12).

Also worth noting in the Decision portion of PAT:Juliet is that the planner's deception goal (&Goal.12) is the subgoal of another goal (&Goal.11). Goals can be achieved either directly by actions, as when an actor satisfies a hunger goal by eating, or indirectly by a subgoal or series of subgoals, as when an actor satisfies a hunger goal by having a goal to get money and having a goal to buy breakfast. In this case, &Goal.12 is a subgoal that achieves &Goal.11, so although the type of this PAT is Plan Selection, the object of the advice is actually a goal, because the "plan" to achieve &Goal.11 is a subgoal.

Figure 6.8 is an illustration of the abstract theme PAT:Juliet. In the actual story of *Romeo and Juliet*, the Decision portion of PAT:Juliet is instantiated by the story scenes in which Juliet decides to take a potion that makes her appear dead, as a plan for breaking her ties with her family.



Summaries	
Decision:	&Juliet choose a deception plan to achieve a goal.
Connection:	The deception also fools &Romeo.
Consequence:	&Romeo does something which thwarts one of Juliet's goals.

Figure 6.8 PAT:Juliet

In this case, Juliet is the planner who makes the error of using a deception plan.

The Consequence of PAT:Juliet is an action by another person that leads to a failed goal for the planner. The Connection between the two halves of the theme is the other person being deceived and consequently motivated by the deceptive state.

In *Romeo and Juliet*, the Consequence and Connection are instantiated by Romeo's goal to kill himself when he discovers Juliet dead. This leads to a goal failure for Juliet - namely, that she cannot spend her life with Romeo.

In English this PAT can be summarized "Don't use deception to achieve a goal, because the deception may fool an unintended party and lead to the failure of another goal."

6.4.5 PAT:Hasty-Impulse-Regretted

In the final act of Romeo and Juliet, when Romeo discovers Juliet's body, he is moved by his love of Juliet to kill himself. This is a tragic error, for if had waited but a moment longer he would have seen Juliet awaken.

PAT:Hasty-Impulse-Regretted captures Romeo's planning error: "Do not do in haste what you cannot undo." PAT:Hasty-Impulse-Regretted is a Plan Selection theme which advises a planner not to use irreversible plans to achieve goals motivated by a hasty deduction about the world, because the deduction might prove to be incorrect. PAT:Hasty-Impulse-Regretted is a complex theme that involves both beliefs and character traits. MINSTREL's representation of PAT:Hasty-Impulse-Regretted is shown in Figure 6.9.

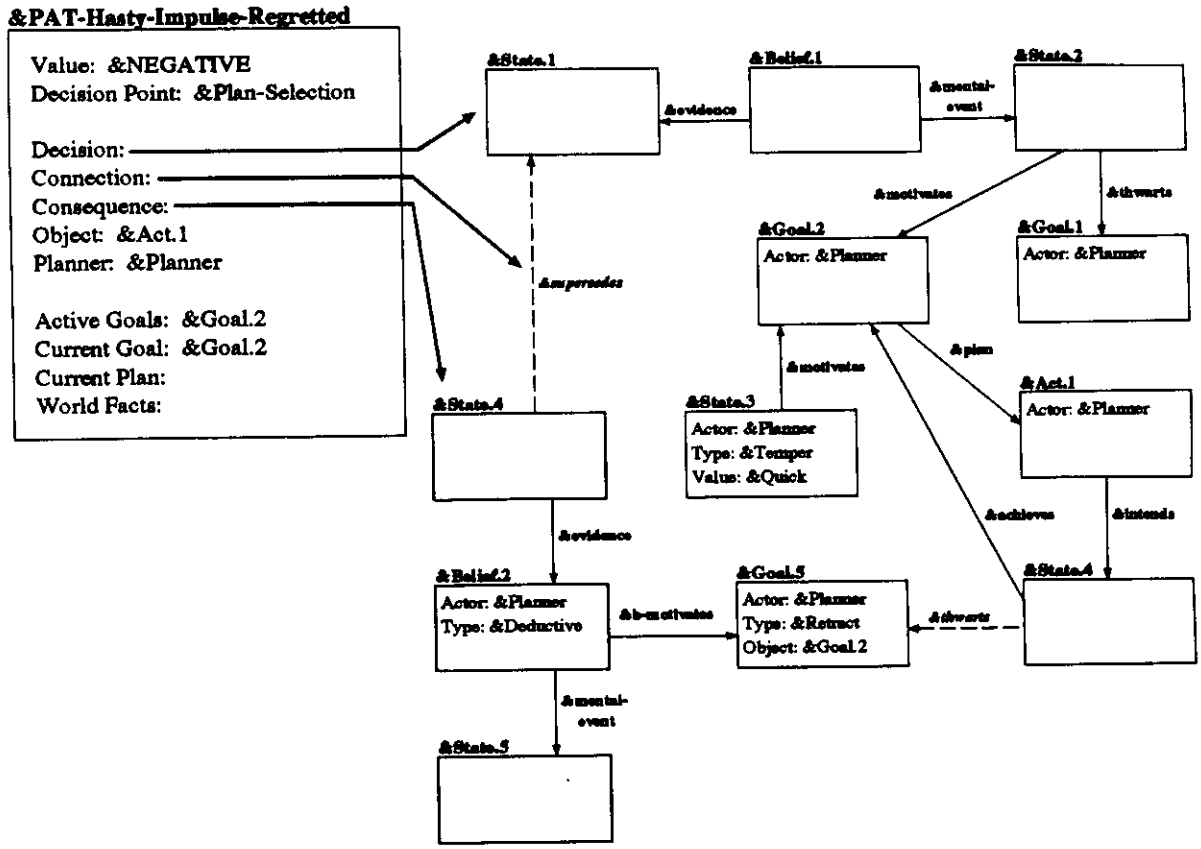
MINSTREL represents deductions about the world as a type of belief. In the Decision part of PAT:Hasty-Impulse-Regretted, &Belief.1 represents the planner's initial, hasty deduction about the world. The type of the belief is &Deductive, to indicate that it represents a deduction about the world. (MINSTREL also uses predictive beliefs, which represent character beliefs about things that may occur in the future, as when a character believes he will be eaten by a dragon.) The &Evidence link of a &Deductive belief points from the belief to the state of the world that is evidence for the belief. The evidence is the state of the world from which another state was deduced. The &Mental-Event link of a &Deductive belief points to the deduced state of the world.

In the actual story of *Romeo and Juliet*, the Decision portion of PAT:Hasty-Impulse-Regretted is instantiated by Romeo's deduction from Juliet's still and unresponsive form (the &Evidence) that she is dead (the &Mental-Event).

In the Decision part of PAT:Hasty-Impulse-Regretted, the &Mental-Event of the planner's belief (&State.2) thwarts one of the planner's goals (&Goal.1) and motivates another goal (&Goal.2). The motivated goal, which is also motivated by the planner's quick temper, is achieved by &Act.1 and &State.4.

The Consequence of PAT:Hasty-Impulse-Regretted involves a second deductive belief. New evidence (&State.4) leads the planner to a new deduction (&State.5) and motivates him to retract his previous, hasty goal (&Goal.2). But this is thwarted by the irreversible nature of the state that achieved that goal.

Interestingly enough, the Consequence of PAT:Hasty-Impulse-Regretted is not instantiated in *Romeo and Juliet*. In *Romeo and Juliet*, Romeo's hasty action is killing himself, and this prevents him from later realizing that his belief was in error. However, the reader of *Romeo and Juliet* can still recognize the theme by empathic reasoning. Placing himself in Romeo's shoes, the reader can reason "I made a mistake by killing myself, because Juliet is really still alive."



Summaries	
Decision:	&Romeo believes something (&Belief.1) that causes a goal failure for him (&Goal.1). This and his hasty disposition motivate him to do something irreversible (&Act.1).
Connection:	&Romeo learns something new (&State.4) that supersedes the evidence for his earlier belief (&Belief.1).
Consequence:	&Romeo now has a different belief, which motivates him to retract his earlier goal (&Goal.2) but he cannot, because his earlier action (&Act.1) is irreversible.

Figure 6.9 PAT:Hasty-Impulse-Regretted

This reliance on the reader to mentally extend the events of the story to understand the theme of the story is a literary technique MINSTREL is not currently capable of using. However, it is interesting to note that in this case the omitted portions of the theme correspond exactly to the Consequence of the theme, suggesting that the advice representation of story themes might be used as a basis for determining what parts of a theme could be omitted.

6.4.6 PAT:Pride-Fall

PAT:Pride-Fall is based upon the adage "Pride goes before a fall." This theme advises a planner to heed warnings, because if he is too proud to accept a warning it may bring him grief. Figure 6.10 illustrates PAT:Pride-Fall.

PAT:Pride-Fall is interesting because it makes use of the World Facts part of the PAT representation. In PAT:Pride-Fall, the World Facts represent someone (an &Advisor) warning the planner (&Planner) not to do a particular thing (&Act.2) because it will result in a thwarted goal. The warning is represented as a predictive belief (&Belief.1).

Predictive beliefs represent a prediction about what the consequences of an action will be. In a predictive belief schema, a "&Evidence" link points to an action, and the "&Mental-Event" link points to the predicted consequence of the action. So if a knight believed that killing a dragon would make him famous, his belief would be represented by a belief schema with a "&Evidence" link pointing to the representation of the knight killing a dragon, and an "&Mental-Event" link pointing to the representation of the knight becoming famous.

In this case, the action is &Act.2 (the &Planner doing something), and the predicted event has an unintended side-effect that causes a goal failure for the planner. In other words, the advisor is predicting that if the planner takes some action, something will happen (&State.1) which will result in a goal failure for the planner.

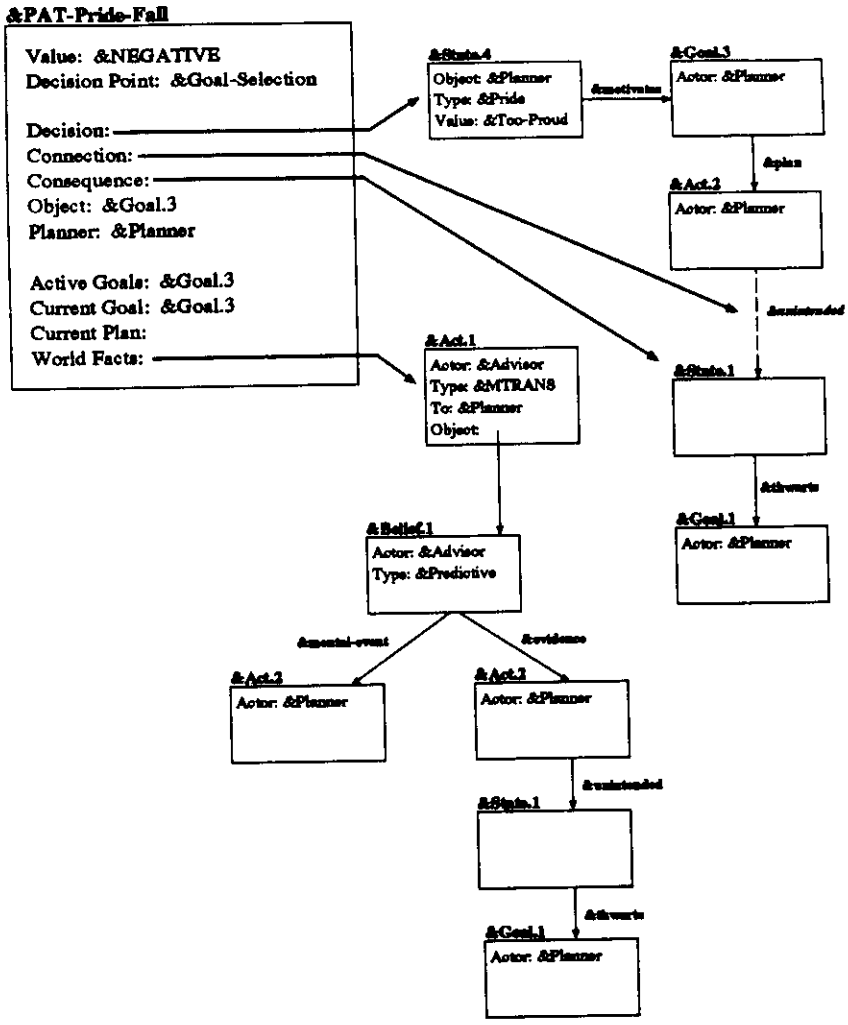
The Decision of this PAT is the planner deciding to use that plan despite the warning. The Consequence of this PAT is the predicted goal failure occurring. The Connection between the Decision and the Consequence is the unintended side-effect.

6.5 Using Themes in Storytelling

As the previous sections showed, Planning Advice Themes can be used to represent a variety of common story themes. To make use of these story themes to tell new stories, MINSTREL must perform two tasks:

(1) Select a theme.
(2) Create story scenes to illustrate the theme.

This section discusses the author-level goals and plans MINSTREL uses to achieve these two tasks.



Summaries	
Decision:	&Planner is warned by &Advisor not to try &Act.2, because it will result in an unintended goal failure.
Connection:	&Planners pride motivates him to try &Act.2 anyway.
Consequence:	&Act.2 causes the unintended goal failure &Planner was warned about.

Figure 6.10 PAT:Pride-Fall

6.5.1 Selecting a Theme

MINSTREL has two methods for selecting a theme.

First, the user can select a theme from amongst the PATs that MINSTREL knows, and ask MINSTREL to tell a story about the selected theme. This permits testing and demonstration of MINSTREL's storytelling abilities.

Second, MINSTREL can select a theme for storytelling by being reminded of one of the themes it already knows, in the same way a human author might be reminded of a theme and decide to tell a story based on the reminded theme.

Human authors are reminded of stories they've read and episodes from their lives by clues from their environment, both physical and mental. A human author who passes a mailman on the staircase might recall Ellison's "The Man Who Wasn't There" and be motivated to write a similar story. Autobiographical memories of this sort may be recalled involuntarily when cued by events in a person's environment (reminding), may be deliberately sought and retrieved, or may arise by a combination of reminding and deliberate retrieval [Cohen 1989].

Although one can imagine a robot writer who was able to move through the world and receive inputs from his environment, MINSTREL is only a computer program running on a workstation. However, input from the environment can be simulated, by asking the user to provide clues as if they came from MINSTREL's environment.

When MINSTREL is asked to find a story theme, it makes use of a pool of representation fragments (provided by the user) as initial indices into memory. For example, this pool may contain representations for "a woman" and the scene "someone does a favor for someone", as if MINSTREL had just seen a woman do a favor. These fragments are used as indices to episodic memory. If these indices can be used to recall an episode that was part of a story theme, then the associated theme can be used as the basis for a new story.

For example, the indices "a woman" and "someone does a favor for someone" might recall a scene in which a knight saves a princess, which itself is part of a story that illustrates PAT:Bird-In-Hand. MINSTREL will then use PAT:Bird-In-Hand as a basis for a new story.

This reminding process involves recall from episodic memory. Since MINSTREL's episodic memory is imaginative, these indices do not have to match a memory episode exactly. MINSTREL can use creativity heuristics to find episodes quite different from the initial clues. This is an example of how embedding creativity in the recall process makes it available to other cognitive processes.

MINSTREL's author-level plan for selecting a theme is shown in Figure 6.11. This plan applies to author-level goals of the type "&Tell-Story". The input to this plan is a list of indices supplied by the user. These are used to recall something from memory, and if the recalled scene has an associated theme, that theme is used as the basis for a new story.

ALP:Find-Story-Theme

Goal: &Tell-Story

Input: *AL-OBJ*, a list of story fragments such as "a woman" and "someone does a favor for someone", to be used as indices to memory.

Body:
1. Use *AL-OBJ* as an index for recall from imaginative memory.
2. If the recalled episode has an associated story theme, use that as the theme for a new story. Create a new author-level goal to &Tell-Story using the new theme.
3. Otherwise, this plan fails.

Figure 6.11 ALP:Find-Story-Theme

Once a theme has been selected, either by the user or by the reminding process in ALP:Find-Story-Theme, the new story must be begun by creating story events which illustrate the theme.

6.5.2 Illustrating a Theme

How can an author write a story that will communicate a theme to a reader? One way is to include story events that illustrate the theme – story events which act as an instance or example of the theme. In *Romeo and Juliet*, Juliet's use of a potion to appear dead, Romeo's mistaken belief that Juliet is dead, and the consequent tragedy are the story events that illustrate the theme PAT:Juliet. From the particular instance of the theme that appears in the story the reader can deduce the general, abstract theme. The story events that illustrate a theme are called the *plot* of that theme.

Although there are other ways that a story can communicate a theme, most rely on creating an example of the theme. For example, one character can repeat an adage corresponding to the theme to another character, but if this not supported with an example, or reflected in the second character's behavior in the story, the impact of the adage is lost. Similarly, a theme can be repeated in a story to improve its impact, but this depends on creating multiple examples of the theme. Consequently, the focus in MINSTREL has been on how an author can take a abstract, general principle like a Planning Advice Theme and create story events which illustrate that principle.

Planning Advice Themes can be illustrated by creating specific story events which correspond to each part of the theme: the Decision, Connection, Consequence and context parts of the PAT. To create a story that illustrates PAT:Juliet, MINSTREL must create story events in which a character uses a deception plan (the Decision), in which the deception fools some unintended character (the Connection), and in which the fooled character does something to thwart one of the main character's goals (the Consequence). These concrete story events, in which the abstract actors, goals, and other features from the theme have been replaced with specific values, illustrate the theme by presenting the reader with a particular example of the theme. In general, an author can incorporate an abstraction into a story by creating story events that act as a particular example of the abstraction. This process – taking an abstract description and creating an example – is called

instantiation. The problem of illustrating a theme - creating the plot of a story - is thus the problem of instantiating the schemas that make up the theme.

Instantiation is a pervasive process in storytelling. It is used not only to illustrate the theme, but also to accomplish many other author-level goals. Because of its importance to the storytelling process, instantiation merits a detailed discussion, which follows in the next chapter. For the moment it is sufficient to view instantiation as a black box that takes as input an abstract description of a story scene and produces as output a specific story scene that fits the description. MINSTREL's plan to illustrate a theme is then simply to instantiate the parts of the theme (Figure 6.12).

ALP:Tell-Story

Goal: &Tell-Story

Input: An uninstantiated instance of a Planning Advice Theme, *AL-OBJ*.

Body:
1. Create an author-level goal to instantiate the Decision of *AL-OBJ*.
2. Create an author-level goal to instantiate the Connection of *AL-OBJ*.
3. Create an author-level goal to instantiate the Consequence of *AL-OBJ*.
4. Create an author-level goal to instantiate the context of *AL-OBJ*.

Figure 6.12 ALP:Tell-Story

Once these instantiations have been achieved, MINSTREL is finished with the thematic level of the story. A theme has been chosen and story scenes created to illustrate that theme.

6.5.3 MINSTREL Example

Let's look now at an example of MINSTREL selecting and illustrating a story theme. This example looks at how the story theme for the "Richard and Lancelot" was chosen:[1]

Richard and Lancelot

It was the spring of 1089, and a knight named Lancelot returned to Camelot from elsewhere. Lancelot was hot tempered. Once, Lancelot lost a joust. Because he was hot tempered, Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed.

One day, a lady of the court named Andrea wanted to have some berries. Andrea went to the woods. Andrea had some berries because Andrea picked some berries. Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot saw Andrea.

1. Except for typography, "Richard and Lancelot" appears here exactly as output by MINSTREL.

Lancelot loved Andrea.

Some time later, Lancelot's horse moved Lancelot to the woods unintentionally, again causing him to be near Andrea. Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Lancelot hated Frederick. Because Lancelot was hot tempered, Lancelot wanted to kill Frederick. Lancelot went to Frederick. Lancelot fought with Frederick. Frederick was dead.

Andrea went to Frederick. Andrea told Lancelot that Andrea was siblings with Frederick. Lancelot believed that Andrea was siblings with Frederick. Lancelot wanted to take back that he wanted to kill Frederick. but he could not because Frederick was dead. Lancelot hated himself. Lancelot became a hermit. Frederick was buried in the woods. Andrea became a nun.

Moral: ''Done in haste is done forever.''

MINSTREL begins storytelling by selecting a story theme. This is done by using a set of initial story fragments provided by the user as indices to episodic memory. These initial story fragments represent chance remindings or clues from the environment that might remind MINSTREL of a theme about which to write a story. In telling "Richard and Lancelot", MINSTREL is given a single story fragment to use as an index. This fragment is "A man has a thwarted goal" (see Figure 6.13).

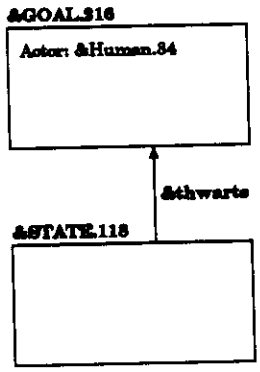


Figure 6.13 Example Index

Figure 6.14 illustrates the top level of MINSTREL's episodic memory of goals in the King Arthur domain. ALP:Find-Story-Theme uses the story fragment shown in Figure 6.13 (&Goal.216) as an index into this memory. If the story fragment recalls an episode with an asso-

ciated story theme, then MINSTREL will tell a story about that theme.

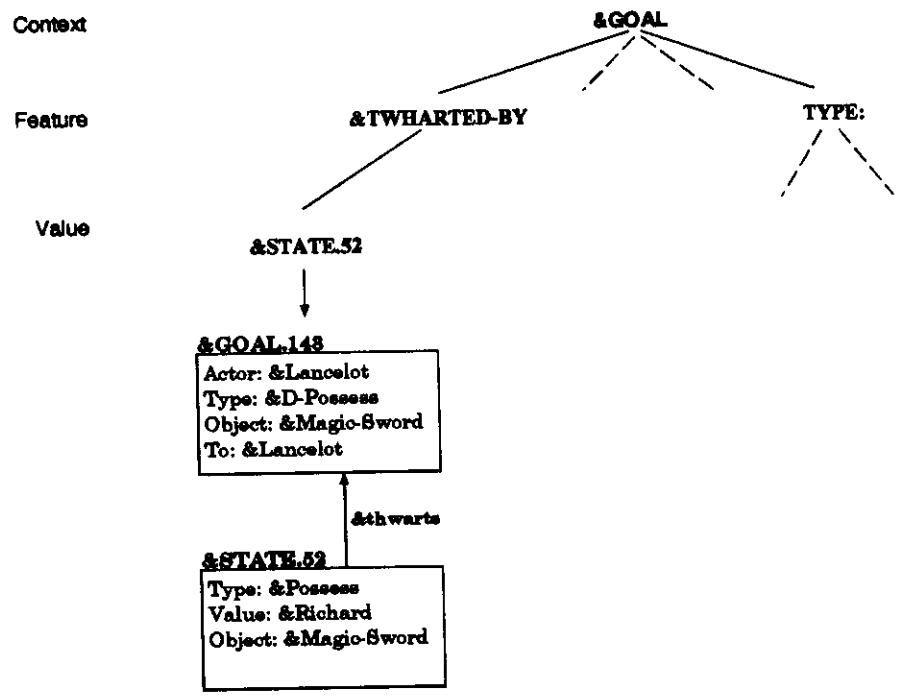


Figure 6.14 A Portion of MINSTREL's Episodic Memory

In this case, the context of the story fragment (a Goal schema) and a single feature (the goal was thwarted) are sufficient to recall an episode from memory. The value of the "&thwarted-by" link in the story fragment (&State.118) matches the value of the only branch of the "&thwarted-by" portion of the episodic memory tree (&State.52), and &Goal.143 is recalled.

Generally, episodic memory will have many levels, and each feature will have several possible values. If the story fragment provided to ALP:Find-Story-Theme does not possess sufficient features to recall a particular episode, additional features are elaborated by methods similar in function to those described in [Kolodner 1984]. Feature elaboration adds new features to a recall index that are likely to result in the recall of an episode. For this example, the new feature "the actor of the goal is a knight" might be added, because it is likely that story episodes in the King Arthur domain involve knights. In this way, episodic memory can recall episodes even when the recall index is underspecified. (For more discussion of feature elaboration in MINSTREL's episodic memory, see Chapter 2.)

If feature elaboration also fails to recall an episode, then creativity heuristics are applied. Creativity heuristics can change the recall index in more elaborate ways than feature adaptation, and can adapt any recalled episodes to match the original recall index. This use of creativity heuristics to extend the power of recall is another example of how embedding creativity in the recall process makes it available to a variety of cognitive processes.

It is important to note that in MINSTREL, both feature elaboration and the use of creativity heuristics are the normal processes of imaginative memory; no special mechanisms are needed to be reminded of a story theme.

If, despite feature elaboration and creativity, the story fragment provided by the user fails to remind MINSTREL of an episode in memory with an associated story theme, then ALP:Find-Story-Theme fails. The "clues from the environment" that might have reminded MINSTREL of a story theme and inspired it to tell a story have not, and so storytelling fails.

In this case, however, reminding succeeds. The initial clue "a man has a thwarted goal" reminds MINSTREL of a scene in which "Lancelot fails to possess a magic sword because Richard already possesses it" (&Goal.143). &Goal.143 is part of a larger story which illustrates the theme PAT:Hasty-Impulse-Regretted. As shown in Figure 6.15, &Goal.143 is part of the Decision of an instance of PAT:Hasty-Impulse-Regretted. &Goal.143 and &State.52 are also connected to a number of other schemas representing the rest of the story of Lancelot, Richard and the magic sword. These schemas are shown in dotted outline, because they do not enter this discussion.

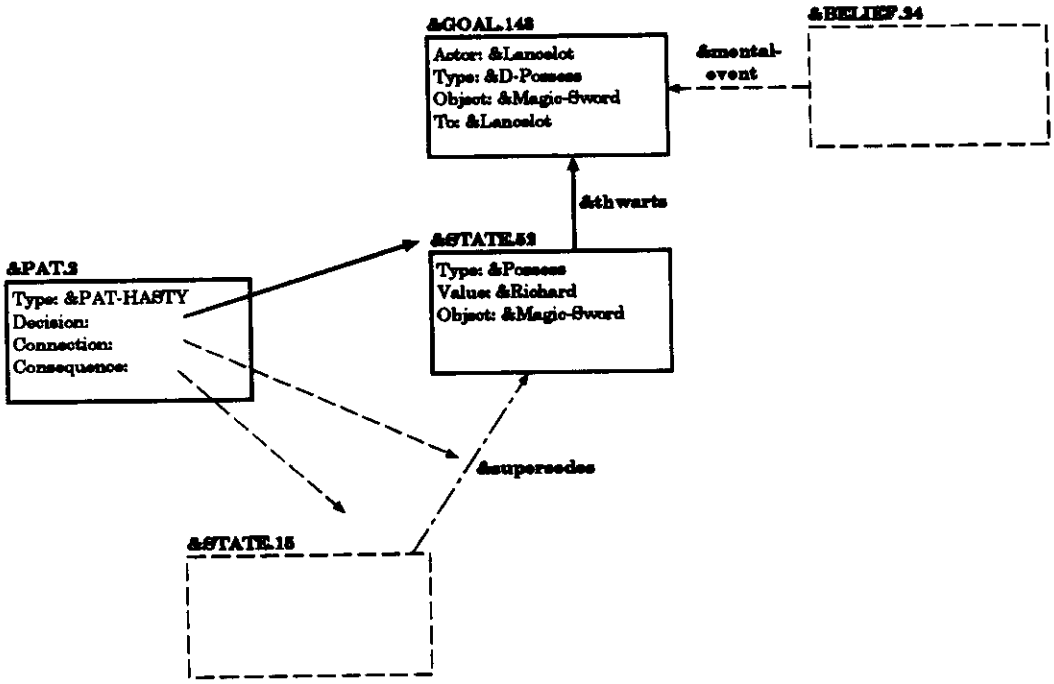


Figure 6.15 Recalled Episode

When ALP:Find-Story-Theme recalls &Goal.143, it notices the associated theme (&PAT.2) and makes a copy of the theme as the basis for a new story. This new copy of the theme is uninstantiated. That is, the new theme has none of the elements of &Goal.143 or any other specific story scenes. It is a blank, abstract specification of the theme PAT:Hasty-Impulse-Regretted, as shown earlier in Figure 6.9. Note that &Goal.143 will have no further role in the telling of this story unless MINSTREL is independently reminded of it later in the storytelling process. (In this case,

that does not happen, and so "Richard and Lancelot" has nothing to do with magic swords, despite this initial reminding.)

The following is a trace of MINSTREL's output as ALP:Find-Story-Theme takes the story fragment &Goal.216, uses it to recall &Goal.143, and makes a copy of PAT:Hasty-Impulse-Regretted for use in telling a new story:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &TELL-STORY applied to &GOAL.216.
Trying author-level plan ALP:TELL-STORY.
Trying author-level plan ALP:FIND-STORY-THEME.
Recalling &GOAL.216: (&GOAL.143).
Created theme: &PAT.8.
Created author-level goal to &TELL-STORY with &PAT.8.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Of note in this trace is that MINSTREL first tried the author-level plan ALP:Tell-Story. As described in an earlier section, ALP:Tell-Story is the author-level plan that takes a story theme and creates author-level goals to instantiate the parts of the theme (i.e., to create the plot of the story). But since at this point in MINSTREL's storytelling the theme of the story has not yet been selected, ALP:Tell-Story cannot be applied. So ALP:Tell-Story fails and MINSTREL next tries ALP:Find-Story-Theme, which succeeds and creates the new story theme &PAT.8, which is an instance of PAT:Hasty-Impulse-Regretted.

In summary, then, ALP:Find-Story-Theme takes a set of story fragments provided by the user which represent chance remindings or clues from MINSTREL's environment, and uses these as indices for a reminding from episodic memory. If MINSTREL does get reminded of an episode, and that episode is part of a story theme, then MINSTREL makes a fresh copy of the story theme and creates an author-level goal to tell a story about that theme.

This new goal is achieved by the author-level plan ALP:Tell-Story, which now succeeds because a theme has been selected. ALP:Tell-Story achieves the goal of telling a story about &PAT.8 by creating a number of author-level goals to instantiate the various parts of &PAT.8. The following is a trace of MINSTREL achieving this goal:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &TELL-STORY applied to &PAT.8.
Trying author-level plan ALP:TELL-STORY.
Created author-level goal to &INSTANTIATE with &BELIEF.28.
Created author-level goal to &INSTANTIATE with &STATE.202.
Created author-level goal to &INSTANTIATE with &STATE.201
Created author-level goal to &INSTANTIATE with &GOAL.225.
Created author-level goal to &INSTANTIATE with &STATE.206.
Created author-level goal to &INSTANTIATE with &BELIEF.29.
Created author-level goal to &INSTANTIATE with &GOAL.22.
[...]
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

&BELIEF.28, &STATE.202, &STATE.201, &GOAL.225, &STATE.206, &BELIEF.29 and &GOAL.22 are all parts of the theme &PAT.8 that must be instantiated to create the plot of the new story. These schemas represent the portions of the story that will illustrate the theme "Do not do in haste what you cannot undo." (In the edited portion of the trace indicated by "[...]", MINSTREL creates a number of goals to fulfill other storytelling goals, such as creating suspense. These goals are unrelated to the theme of the story, and so they are not discussed here, but we will return to these goals in following chapters.)

The schemas that make up the instantiation of &PAT:Hasty-Impulse-Regretted in this story are instantiated by a variety of author-level plans and creativity heuristics. MINSTREL's model of instantiation, and the various plans it uses to instantiate story scenes are discussed in the following chapter. For the moment, it is sufficient to say that MINSTREL is able to create the story scenes it needs to illustrate PAT:Hasty-Impulse-Regretted, and to look instead at the results of the instantiations and the role these scenes play in the final story.

The first part of theme to be instantiated is the Decision. In PAT:Hasty-Impulse-Regretted, the Decision is a complicated sequence of story scenes in which the main character (the planner to whom the advice of the theme applies) makes a false deduction about the state of the world and this motivates a hasty action. MINSTREL instantiates this as a sequence of scenes in which a knight thinks his love is stolen. MINSTREL expresses these scenes in English as follows:

> Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Because Lancelot was hot-tempered, he wanted to kill Frederick. Lancelot went to Frederick. Lancelot fought with Frederick. Frederick was dead.

Here the main characer is Lancelot. He sees Frederick kissing Andrea and makes a hasty deduction: Andrea loves Frederick. This thwarts Lancelot's goal of having Andrea love him. Spurred

on by his hot temper, Lancelot slays Frederick.

It's important to note that although these scenes fulfill the requirements of the story theme - that is, they describe a hasty decision that cannot be undone - they do not themselves form a complete story. Among other things, they lack any explanation of why Lancelot loves Andrea, how he came to see her kissing Federick, and other features that would make the story understandable and complete. At this stage of storytelling, MINSTREL is only trying to create the scenes necessary to illustrate the story theme. Other goals - such as making the story coherent - will be achieved later.

The next portion of the story theme to be instantiated is the Connection. In PAT:Hasty-Impulse-Regretted, the Connection is a state of the world that shows the planner (Lancelot) that his earlier belief was incorrect:

```
Andrea told Lancelot that Andrea was siblings with Frederick.
```

That Andrea and Frederick are siblings is an alternative explanation of what Lancelot had believed earlier, and will cause him to change his belief about Andrea loving Frederick romantically.

The final portion of the PAT to be instantiated is the Consequence. In PAT:Hasty-Impulse-Regretted, the Consequence is the planner's change of belief based upon the new evidence, and his regret for his earlier action:

```
Lancelot believed that Andrea was siblings with Frederick.
Lancelot wanted to take back that he wanted to kill Frederick.  But he could not because Frederick was dead.
```

Here Lancelot has superseded his earlier belief with a new belief, and wants to retract his earlier, hasty actions. But that is not possible.

The final part of the theme to be instantiate is the context. But PAT:Hasty-Impulse-Regretted has no context, so nothing is created.

MINSTREL has now completed its thematic-level development of this story. Using clues provided by the user, it has selected a story theme and instantiated the parts of that theme as a sequence of story events that illustrate the theme.

6.6 The Role of Theme in Storytelling

MINSTREL's primary purpose in storytelling is to tell a story that illustrates a particular story theme. In light of this, it is interesting to examine one of MINSTREL's stories to see how big a role the scenes that illustrate the theme play in the finished story.

In the following version of *Richard and Lancelot*, the portions of the story which illustrate the theme are shown in italics:

Richard and Lancelot

It was the spring of 1089, and a knight named Lancelot returned to Camelot from elsewhere. Lancelot was hot tempered. Once, Lancelot lost a joust. Because he was hot tempered, Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed.

One day, a lady of the court named Andrea wanted to have some berries. Andrea went to the woods. Andrea had some berries because Andrea picked some berries. Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot saw Andrea. Lancelot loved Andrea.

Some time later, Lancelot's horse moved Lancelot to the woods unintentionally, again causing him to be near Andrea. *Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick.* Lancelot hated Frederick. *Because Lancelot was hot tempered, Lancelot wanted to kill Frederick. Lancelot went to Frederick. Lancelot fought with Frederick. Frederick was dead.*

Andrea went to Frederick. *Andrea told Lancelot that Andrea was siblings with Frederick. Lancelot believed that Andrea was siblings with Frederick. Lancelot wanted to take back that he wanted to kill Frederick. but he could not because Frederick was dead.* Lancelot hated himself. Lancelot became a hermit. Frederick was buried in the woods. Andrea became a nun.

Moral: ''Done in haste is done forever.''

The portions of the story that illustrate the theme make up less than half the story. But what these scenes lack in bulk they make up in importance to the story, as we can see by comparing the story *without* the scenes that illustrate the theme:

It was the spring of 1089, and a knight named Lancelot returned

to Camelot from elsewhere. Lancelot was hot tempered. Once, Lancelot lost a joust. Because he was hot tempered, Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed.

One day, a lady of the court named Andrea wanted to have some berries. Andrea went to the woods. Andrea had some berries because Andrea picked some berries. Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot saw Andrea. Lancelot loved Andrea.

Some time later, Lancelot's horse moved Lancelot to the woods unintentionally, again causing him to be near Andrea. Lancelot hated Frederick.

Andrea went to Frederick. Lancelot hated himself. Lancelot became a hermit. Frederick was buried in the woods. Andrea became a nun.

to the story with *only* the scenes that illustrate the theme:

Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Because Lancelot was hot tempered, Lancelot wanted to kill Frederick. Lancelot went to Frederick. Lancelot fought with Frederick. Frederick was dead.

Andrea told Lancelot that Andrea was siblings with Frederick. Lancelot believed that Andrea was siblings with Frederick. Lancelot wanted to take back that he wanted to kill Frederick. but he could not because Frederick was dead.

Of these two versions of the story the second is clearly more "story"-like. Although the parts of the story that illustrate the theme may have less bulk than the remainder of the story, they are obviously more important to the success of the story.

At the beginning of this chapter, theme was defined as "the underyling concept or topic that organizes the story into a coherent whole". From there we defined one type of theme, and went on to show how MINSTREL represents that type of theme and creates stories that illustrate the theme. We have now come full circle. As the two versions above demonstrate, MINSTREL does indeed tell stories with an underlying concept that "organizes the story into a coherent whole".

6.7 Limitations of Planning Advice Themes

MINSTREL's Planning Advice Themes capture a specific type of planning advice. PATs represent advice in which (1) a planning decision is judged as either beneficial or harmful to the planner, and (2) that judgement is justified by a single causal outcome of the planning decision. As we have shown, this type of advice can represent many of the themes present in stories such as Aesop's fables and Shakespeare. There are, however, some limitations of Planning Advice Themes as both story themes and advice.

First, PATs are only able to capture themes about the planning process. Themes about how the world works, human emotions and many other categories are difficult or impossible to represent using PATs. Any abstract principle that does not impinge directly on the planning process cannot be represented: "There is life after death", "Behold the wonders of nature" and so on.

Of course, this is by design. As noted early in this chapter, themes like stories themselves have infinite variety. By focusing on a particular class of themes, it has been our hope that MINSTREL will illuminate larger issues. Still, it behooves us to be aware of the limitations of PATs, and not to stretch our conclusions too far.

A second limitation of PATs is in the representation of the Consequence. The Consequence is the justification or explanation of The advice encapsulated in the Value judgement (i.e., Positive or Negative) of the planning decision captured in the Decision. The Consequence can only represent abstracted, stereotypical planning situations. So while it is easy to represent a justification such as "the decision is bad because the violent act will unexpectedly hurt the planner" it is difficult to represent a justification such as "the decision is bad because the plan is inelegant." The first reason can be easily represented as a planning situation, but the second reason corresponds to a plan evaluation metric, which cannot be easily represented within the PAT framework. Further, the Consequence is a single justification, which means that PATs cannot represent a theme which has two or more reasons to support its advice. These shortcomings limit the range of advice that PATs can represent.

A possible solution to the first of these problems is to incorporate plan metrics, such as those proposed in [Dyer 1984]. Plan metrics are evaluations used to select between competing plans, and include measures such as cost, efficacy, risk and avaialability. Although [Dyer 1984] used plan metrics to categorize story themes, plan metrics could also be used to capture justifications for planning decisions that cannot be easily represented solely by the goal/plan structures used by MINSTREL.

A possible solution to the second of these problems is to explicitly represent the Consequence of a PAT as a series of one or more argument-style justifications, such as those presented in [Alvarado 1989]. Explicitly representing the justifications has several benefits: (1) It will permit more than one justification per theme, (2) It will make PATs more useful in other reasoning tasks, such as argumentation, (3) It may help identify common elements of tasks such as storytelling and argumentation, and (4) explicitly identifying the type of justification will permit MINSTREL to use that information to more fully develop the stories it tells.

6.8 Inventing New Story Themes

The six themes MINSTREL knows were represented and hand-coded by the author. How could MINSTREL learn new story themes?

One possibility is to learn new story themes from stories or a teacher. We can imagine a system that would read a story, analyze the actions of the characters in each story and posit new story themes. A storytelling program that could also read stories could expand its knowledge of themes by reading.

[Dolan 1988] presents a system called AESOP that learns new themes by reading. AESOP reads examples from "Aesop's Fables" and tries to recognize and learn new story themes. Connecting a program like AESOP to MINSTREL would allow MINSTREL to learn new themes by reading, but no work has been done yet on this possibility.

Another possibility is to learn new themes by inventing them. Using creativity to extend MINSTREL's knowledge about storytelling is an intriguing idea. What kinds of creativity heuristics would be needed to transform and adapt known story themes to create new story themes?

MINSTREL's existing themes hint at some possibilities. The observant reader may have noticed that PAT:Spite-Face and PAT:Good-Deeds-Rewarded have parallel structures. The Decision portion of each theme involves the &Planner subsuming another character's goal. PAT:Good-Deeds-Rewarded is positive advice about activating a &Favor goal, while PAT:Spite-Face is negative advice about what kinds of plans to use to achieve &Anti-Favor goals. The almost-opposite nature of these themes suggests that new themes might be invented by applying transformations to old themes.

In previous chapters we presented a model of problem-solving which explains how problem transformations can be applied to invent new problem solutions. Rather than describe in detail how this architecture could be applied to inventing new themes, we will concentrate instead on identifying problem transformations which are potentially useful for inventing new story themes. Four transformations that we have identified are:

1. Generalization
2. Specialization
3. Mutation
4. Recombination

One way to create new story themes is to use *generalization and specialization* on existing story themes. The idea in this transformation is to examine existing themes to see if two or more of them are specializations of a more general theme. If they are, we can deduce the generalization of these themes, and then use creative problem-solving to create a new specialization of the general

theme (i.e., create an *instantiation* of the general theme).

For example, suppose a storyteller knew the two themes "A fool rushes in where angels fear to tread" and "A fool and his money are soon parted." The first theme advises that a fool does not take care in selecting his planning situations; the second that a fool does not take care in selecting plans that involve money. These two themes are specializations of a more general theme about the planning abilities of fools, namely, "Fools make poor planning selections." A creative problem-solver can use this generalization to create new story themes involving fools and new types of planning selectionss. This might lead to a specialization concerning selecting goals to abandon: "A fool never knows when to quit" or "Only a fool quits with the end in sight."

Another useful type of transformation is *mutation*. In mutation, a specific limited change in structure is applied to an existing theme to create a new theme. One interesting type of mutation is suggested by an old adage: "The opposite of a great truth is another great truth." This suggests that the opposite of any piece of advice might also be good advice.

If we apply this transformation to PAT:Good-Deeds-Rewarded ("Good deeds do not go unrewarded") by reversing the Consequence and Value of the PAT, we get a new theme in which a planner does good deeds for people and when he finds himself in need, does *not* get repaid. This might be summarized as "A wise man helps himself" or "Do not sow where you cannot reap". Similarly, reversing PAT:Spite-Face ("Do not cut off your nose to spite your face") produces "No price too great for revenge."

Finally, new themes can also be formed by *recombination*. The idea behind this transformation is to combine parts from different themes to create a new theme. MINSTREL's planning advice themes are structured as advice, so it is simple to break themes down into their components as advice: Decision, Connection and Consequence. If two themes have similar Connections, the Decision of one theme can be connected to the Consequence of the other, and vice versa.

For example, PAT:Bird-In-Hand and PAT:Juliet ("Deception is a weapon difficult to aim") have similar Connections. In PAT:Bird-In-Hand a state in the Decision thwarts a goal in the Consequence. In PAT:Juliet, a state in the decision motivates a goal in the Consequence. By substituting the Consequence of one into the other, we can invent two new themes.

If the Consequence of PAT:Juliet is substituted into PAT:Bird-In-Hand, the result is a new PAT in which catching a bird motivates someone else to get revenge on the bird-catcher. This does not correspond directly to a common adage, but might be best summarized by "Don't make enemies by your actions."

Substituting the Consequence of PAT:Bird-In-Hand into PAT:Juliet results in a PAT which advises not to use deception plans because the deception might cause a goal failure for the planner. This is the vacuous theme: "Avoid deception plans because you might deceive yourself."

As this last theme suggests, a critical activity in inventing new story themes is in distinguishing between good themes and boring themes. One possibilty for measuring the "interestingness" of a

theme is to determine how useful it is as a abstraction of known stories and life episodes. If an author invents a story theme and can see immediately how it applies to many things in his life, that is some indication that the theme is interesting. Another possibility is to compare an invented theme with known themes, to see if they share components or structure. A theme similar to past themes is also likely to be interesting.

MINSTREL does not currently invent new story themes, nor does MINSTREL implement any methods for determining the "interestingness" of a theme. But these transformations and their similarities to creativity heuristics that MINSTREL does use suggest that inventing new themes is both possible and an interesting topic for further research.

6.9 Conclusions

MINSTREL's Planning Advice Themes are a knowledge structure that captures aspects of both story themes and planning advice. PATs are a structured representation of (1) a planning decision, (2) a value judgement of that decision and (3) a justification for that judgement. MINSTREL has used PATs to represent a variety of story themes, and to tell new stories based on those themes. MINSTREL has also taken a preliminary look at how new story themes can be created.

CHAPTER 7
Instantiation in Storytelling

7.1 Introduction

The process of writing a story is iterative and two-fold. The author must repeatedly (1) decide what he wants in the story and (2) create story scenes to fit those needs. In the first part of this process, the author builds abstract descriptions of story elements that will fulfill his goals. In the second part of this process, he uses these abstract descriptions to create story scenes.

The previous chapter illustrated the first part of this process for thematic goals. To tell a story based upon a particular theme, MINSTREL's author-level plans create abstract scene descriptions based upon a Planning Advice Theme. For example, to tell a story based on PAT:Good-Deeds-Rewarded, MINSTREL must create scenes which fit the following abstract descriptions:

1. A character (X) does a favor for another character (Y).
2. Y likes X because X helped Y.
3. Y returns the favor to X because Y likes X.

Obviously the first part of the storytelling process - deciding what needs to be in the story to achieve author-level goals - is very important. But the second part of the storytelling process - creating concrete, instantiated story scenes - is equally important. An author must be able to both formulate his needs and achieve them to tell a story.

This chapter discusses the process of instantiating abstract story scene descriptions. As with all problem-solving in MINSTREL, this process ultimately rests upon case-based creativity. But fleshing out a story scene has fundamental differences from finding an author-level plan, or inventing a way for a knight to kill a dragon, and so the creative process is used in differently than in author-level and character-level problem-solving.

7.2 The Instantiation Problem

The problem of creating an instantiation of an abstract scene description differs from other types of problem-solving in a fundamental way. In traditional problem-solving, the problem-solver is given a goal and a problem-solving situation and asked to find a plan for achieving the goal. In the instantiation problem, the problem-solver isn't trying to find a plan to achieve a goal; he is trying to find details to flesh out an abstract description. This difference requires the problem-solver to approach the problem of instantiation differently than traditional problem-solving.

MINSTREL uses traditional problem-solving at both the author and character levels. For example, MINSTREL's problem-solving component might be asked to find a plan to achieve the goal "Build suspense" in a particular story scene - author-level problem-solving. Or MINSTREL might need to find a plan to achieve a knight's goal to "Kill the dragon" given a certain story sit-

uation - character-level problem-solving. In both these cases, the result of problem-solving is a plan: a sequence of actions that is expected to achieve the original goal.

MINSTREL solves traditional planning problems in three steps. First, the original goal and problem situation are used to recall a similar past problem situation. This may require further specifying the original goal, or in some cases, modifying the original goal with a creativity heuristic. Second, the plan associated with the recalled problem situation is retrieved. If a creativity heuristic was applied to the original problem description, the transform part of the heuristic is applied to the recalled plan. Finally, the recalled plan is assessed using domain-specific plan assessments.

But the instantiation problem is different. There is no goal to be achieved, and no plan to find or invent. And because of this, there is no need to perform the second and third steps of problem-solving. Instantiation requires only the recall of a past scene similar to the abstract scene description. If a similar past scene can be recalled or invented, it can be used to "flesh out" the scene for the current story.

For example, if MINSTREL is trying to instantiate the abstract description of the first scene from PAT:Good-Deeds-Rewarded:

1. A character (X) does a favor for another character (Y).

and can recall a scene in which a knight helps a hermit by saving him from a dragon, then the recalled scene can be used to "fill out" or instantiate the abstract scene description.

In MINSTREL, this is achieved by imaginative memory. The abstract scene description is used as a set of recall indices. Imaginative memory recalls a matching scene, or if no scenes in memory match the abstract scene description, invents a new scene. Instantiation is thus imagination: using creativity and the knowledge in episodic memory to imagine scenes to fit an abstract scene description.

7.3 Using Imaginative Memory to Instantiate A Story Scene

To make the process of instantiation using imaginative memory more clear, this section looks at how MINSTREL instantiates the abstract scene description "A princess does something which kills a dragon."

MINSTREL's schema-based representation of this abstract scene description is shown in Figure 7.1. &ACT.47 is an action by a princess to a dragon that intentionally results in the dragon being dead. The death of the dragon is represented by &STATE.17.

To instantiate this scene, MINSTREL uses the scene as a set of recall indices to imaginative memory, to see if MINSTREL has previously invented or read a similar scene that it can use to help fill out this scene.

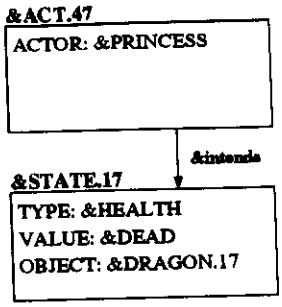


Figure 7.1 Representation of Abstract Scene Description

As it happens, none of the scenes in memory match this abstract scene description, so MINSTREL must use creativity to invent a scene. To begin, MINSTREL applies TRAM:Generalize-Object to the abstract scene description. This creativity heuristic suggests that an action that was applied to one kind of object might well be applied to another kind of object. In this case, the heuristic suggests looking for a scene in memory in which a princess kills anything, and then adapt that scene to the current scene in which a princess kills a dragon.

This too fails, because MINSTREL's memory does not contain any scenes in which a princess kills something. MINSTREL next applies TRAM:Similar-Outcomes, which suggests that an action which has one outcome might well have a similar outcome. In this case, this heuristic suggests looking for something similar to killing, i.e., if MINSTREL can recall a scene in which a princess wounds something, then perhaps that can be adapted to the current scene.

This also fails. Again, MINSTREL's memory does not contain any scenes in which a princess wounds something. MINSTREL applies another creativity heuristic, TRAM:Intention-Switch. This heuristic suggests looking for a scene in which a princess accidently hurt someone instead of a scene in which she intentionally hurt someone.

Again recall fails. MINSTREL applies another creativity heuristic, TRAM:Causal-Chain. This TRAM modifies the abstract scene description by replacing the immediate effect of the princess's action with a delayed effect. Instead of the princess taking an action which immediately, unintentionally hurts someone, TRAM:Causal-Chain suggests looking for a scene in which a princess does something which eventually (through a chain of causally-connected events) hurts someone.

Now recall is successful. The modified scene description: "A princess does something which eventually leads to someone accidently being hurt" recalls a scene in which a princess feeds a knight some spoiled food. The princess gives the knight some food, which motivates the knight to eat the food. The spoiled food then unintentionally causes the knight to become sick. The schema representation of this scene is shown in Figure 7.2. &Act.35 represents the princess giving (a Physical TRANSfer) the food (represented as a physical object of Type &Food) to Lancelot. This act intentionally results in Lancelot possessing the food, which motivates him to have a goal to satisfy his hunger. Lancelot eats the food, unintentionally becomes sick, and feels

unhappy about being sick.

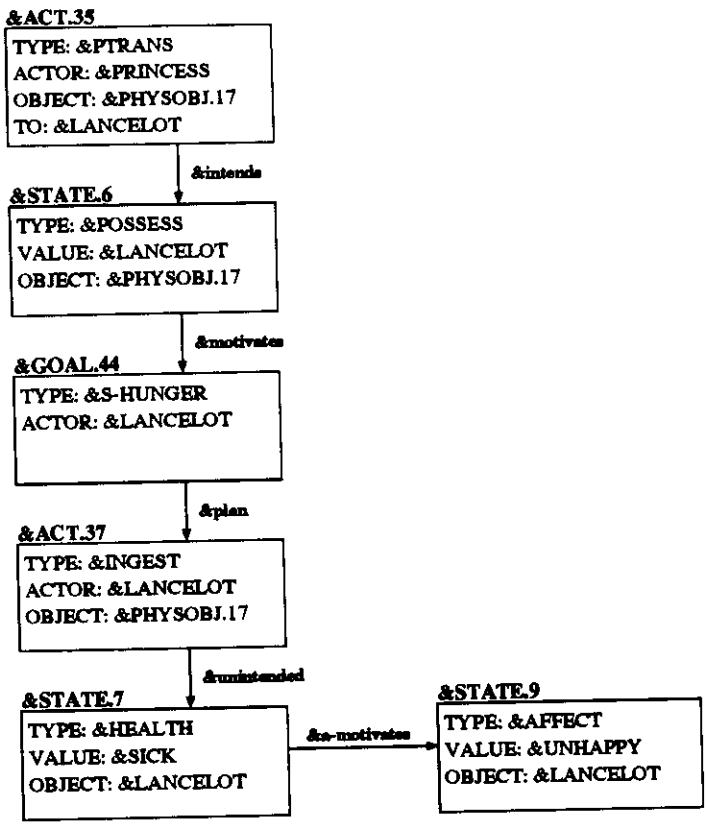


Figure 7.2 Recalled Scene

The recalled scene is adapted by the four creativity heuristics into a scene in which a princess kills a dragon by feeding it spoiled meat. This scene is then used to fill out the initial abstract scene description, resulting in the scene shown in Figure 7.3.

In the resulting scene, the princess gives the spoiled food to the dragon, creating the same chain of events that occurred in the recalled scene, except that this now leads to the death of the dragon rather than an illness.

There are several interesting features to this scene.

First, the instantiated scene is more complicated than the original abstract scene specification. Comparing Figure 7.3 with Figure 7.1 will show that the final scene is much different from the original scene specification. The instantiation process does more than simply replace roles in the abstract scene specification with values. By interacting with creativity and author-level plans for instantiation, instantiation can create scenes which are substantially different and expanded from the original specification.

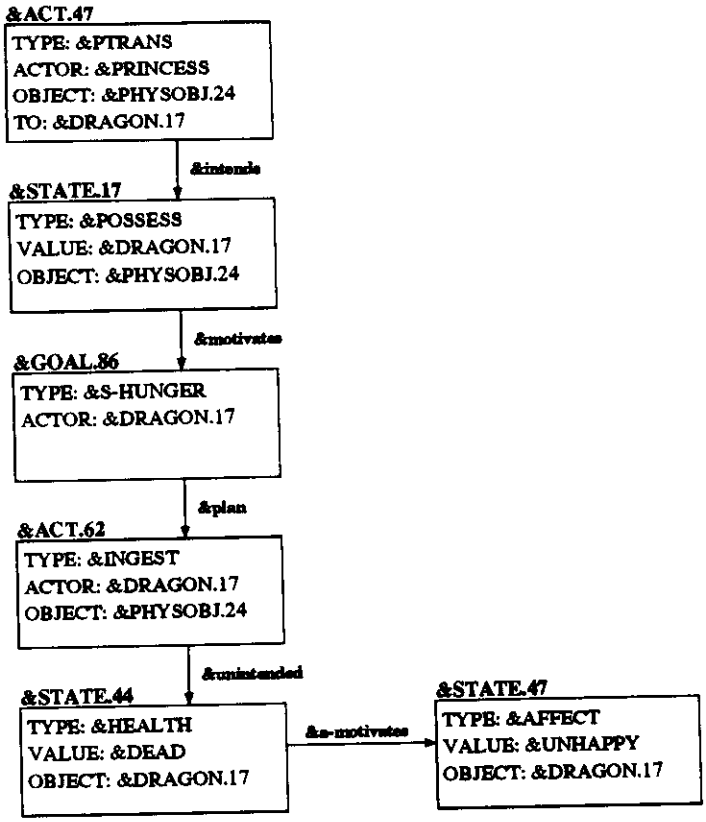


Figure 7.3 Instantiated Scene

Second, the instantiated scene in this case has a different structure than the original specification. The original specification called for an action by the princess that immediately resulted in the death of the dragon; the instantiated scene involves an intervening causal chain. This demonstrates how creativity can, within limits, change the author-level specifications of how to achieve an author-level goal.

Finally, it is interesting to note that the dragon has been given an emotional reaction to being poisoned - the dragon is unhappy about dying. This occurs because the dragon in the abstract description corresponds with the knight in the scene that is eventually recalled. The knight's reaction, after being suitably modified by the creativity heuristics, becomes the dragon's reaction. This illustrates how instantiation may result in the addition of unexpected features to a story.

7.4 Augmenting Imaginative Memory

Although imaginative memory forms the basis of instantiation, it is also useful to augment the instantiation process with additional knowledge specifically for instantiation. This is useful because imaginative memory is a *general* process that is used by many different cognitive processes and in many different problem domains. Consequently the creativity heuristics used by imaginative memory are intentionally generalized to apply to a range of problem domains. Each specific problem domain that uses imaginative memory may need to augment imaginative memory with specific knowledge about the problem domain.

In the MINSTREL model, knowledge about specific problem domains is captured in author-level plans. MINSTREL has ten author-level plans specifically for instantiation. These plans capture specific knowledge about how abstract story scenes in the King Arthur domain can be instantiated. Most of MINSTREL's author-level plans for instantiation capture knowledge about specific planning situations common to the King Arthur domain: revenge, deception, beliefs, and mistaken beliefs.

The following sections describe MINSTREL's author-level plans for instantiation in detail.

7.4.1 ALP:General-Instantiate

ALP:General-Instantiate is the author-level plan that implements the general instantiation process, i.e., using the scene specification as a recall index to imaginative memory and then merging the result. ALP:General-Instantiate is shown in Figure 7.4.[1]

Name:	ALP:General-Instantiate
Goal Type:	&Instantiate
Object:	*AL-OBJ*
Test:	None.
Body:	1. Use *AL-OBJ* as a recall specification.
	2. If nothing is recalled, this plan fails.
	3. Otherwise, randomly select one of the recalled episodes and merge that with *AL-OBJ*
	4. Return *AL-OBJ* as the instantiated episode.

Figure 7.4 ALP:General-Instantiate

ALP:General-Instantiate applies to any author-level goal of type &Instantiate. The scene to be instantiated is *AL-OBJ*; this is used as a recall specification, and if successful is merged with the recalled episode to form the instantiation. If there is more than one recalled episode, then one episode is chosen randomly to be merged with the recall specification. (As noted elsewhere, MINSTREL uses random selection only to choose between indistinguishable alternatives.)

1. For examples of MINSTREL's author-level plans in Lisp, see Appendix A.

7.4.2 ALP:Instantiate-Belief

Beliefs are a type of schema that MINSTREL uses to represent character beliefs about the world. A belief can be a deduction from evidence about something that is true, or a prediction from evidence of something that is likely to happen. Figure 7.5 shows an example of a belief schema.

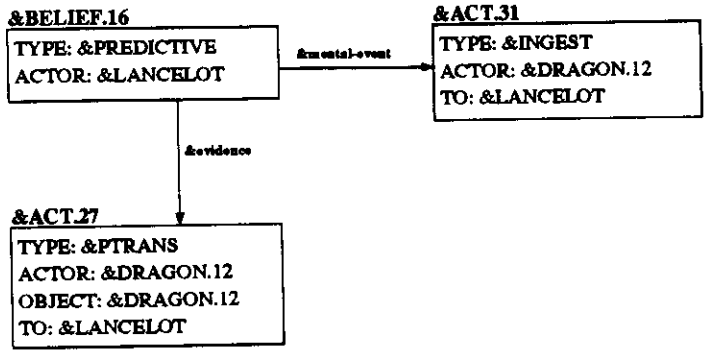


Figure 7.5 Example Belief

&Belief.16 is a predictive belief held by Lancelot. Based on the *evidence* that a dragon is moving to him (&PTRANS is a physical transfer), Lancelot predicts (i.e., has a *mental-event*) that the dragon will eat (i.e., &INGEST) him.

MINSTREL's representation of a belief has three parts. The belief schema itself captures the type of belief (either &Deductive or &Predictive) and the person who holds that belief (the Actor). The schema pointed to by the &Mental-Event link represents what the believer believes, either a state of the world that is true (in the case of a &Deductive belief), or a state of the world that will be true (in the case of a &Predictive belief). The schema pointed to by the &Evidence link represents why the believer holds this belief, i.e., the evidence that supports the belief. An optional fourth part of a belief is pointed to by a &Supersedes link. Schemas pointed to by &Supersedes links represent counter-evidence to a belief, i.e., reasons to think the belief false.

ALP:Instantiate-Belief captures the knowledge that if the type of a belief is known, then the belief can be instantiated by instantiating the &Evidence and &Mental-Event parts of the belief. ALP:Instantiate-Belief implements a simple form of the venerable "divide and conquer" problem-solving strategy for the King Arthur domain. ALP:Instantiate-Belief improves MINSTREL's chances of instantiating a belief by breaking a belief into its constituent parts and then instantiating those parts. ALP:Instantiate-Belief is shown in Figure 7.6.

Figure 7.7 illustrates how an author-level plan can create subgoals. The body of ALP:Instantiate-Belief makes two new author-level goals. Making a new author-level goal requires making a new, uninstantiated goal schema and then filling in the slots of the goal schema so that it represents the desired goal. Step 1 in Figure 7.6 illustrates the steps required to make a new author-level goal. Rather than repeat these steps wherever an ALP makes a new author-level goal, we

Name: ALP:Instantiate-Belief
Goal Type: &Instantiate
Object: *AL-OBJ*
Test: Is *AL-OBJ* a &Belief?
Body: 1. Make an author-level goal to &Instantiate the &Evidence of *AL-OBJ*:

a. Make an uninstantiated goal schema.

b. Fill the Actor slot of the goal schema with &MINSTREL, to indicate that this is an author-level goal.

c. Fill the Type slot of the goal schema with &Instantiate

d. Fill the Object slot of the goal schema with the story scene pointed to by the &Evidence link of the belief *AL-OBJ*.

e. Add the goal schema to the author-level goal queue.

2. Make an author-level goal to &Instantiate the &Mental-Event of *AL-OBJ* (see above).

Figure 7.6 ALP:Instantiate-Belief

will simply describe the goal as in Step 2 in Figure 7.6.

ALP:Instantiate-Belief is also an example of how knowledge about a problem domain can be used to instantiate a story scene. Without ALP:Instantiate-Belief, MINSTREL would only be able to instantiate a belief if a similar belief, complete with appropriate evidence and mental-event, was present in episodic memory. ALP:Instantiate-Belief uses knowledge about the problem domain – specifically about the structure of beliefs – to extend MINSTREL's instantiation capabilities. By revealing that the problem of instantiating a belief is really the problem of instantiating the evidence and the mental-event, ALP:Instantiate-Belief permits MINSTREL to instantiate story scenes that general instantiation alone could not.

7.4.3 ALP:Instantiate-Evidence

ALP:Instantiate-Evidence is the first of two author-level plans MINSTREL uses to instantiate the evidence part of a belief. ALP:Instantiate-Evidence does this by reasoning that one type of evidence for a belief is seeing someone take an action that would be motivated if the belief were true. For example, seeing Richard steal something is evidence for the belief that Richard is poor, because being poor would motivate Richard to possess money, and stealing is a way to achieve that goal. This type of reasoning is a *motivation chain.* Given a belief, ALP:Instantiate-Evidence tries to create evidence for the belief by inventing a motivation chain. ALP:Instantiate-Evidence is shown in Figure 7.7.

ALP:Instantiate-Evidence introduces a new tool in author-level planning: recursive use of author-

Name: ALP:Instantiate-Evidence
Goal Type: &Instantiate
Object: *AL-OBJ*
Test: If *AL-OBJ* is the evidence of a belief, and the mental-event of the belief has been instantiated, then apply this plan.
Body: 1. Try to invent a motivation chain leading from what the character believes about the world (the mental-event of the belief) to a state change. To do this:

a. Build an uninstantiated motivation chain from what the character believes to an uninstantiated state change: the belief motivates a goal, which has a plan, which intends a state change (see Figure 7.7).

b. Use instantiation recursively to try to instantiate this motivation chain.

2. If (1) succeeds, use the state-change as evidence for the belief.

Figure 7.7 ALP:Instantiate-Evidence

level planning. To find a state of the world that acts as evidence for the mental-event of a belief, ALP:Instantiate-Evidence uses instantiation recursively to create a motivation chain.

The first step of ALP:Instantiate-Evidence is to build an uninstantiated motivation chain that represents a line of reasoning from the (unknown) evidence to the (known) mental-event of the belief. The first column of Figure 7.8 shows MINSTREL's representation of the belief "Andrea loves Richard" and the second column shows the uninstantiated motivation chain built by step 1 of ALP:Instantiate-Evidence.

The motivation chain built for the belief that "Andrea loves Richard" is: Andrea loves Richard, and this motivates her to have a goal and to perform an act to achieve that goal. The motivation chain as built in step 1 of ALP:Instantiate-Evidence is uninstantiated. MINSTREL does not yet know what goal "Andrea loves Richard" will motivate in Andrea, nor how she will solve that goal. But if a solution can be found (i.e., if &STATE.33 in Figure 7.8 can be filled in), then the solution can be used as evidence for the belief that "Andrea loves Richard".

The motivation chain is instantiated by a recursive call to author-level instantiation. MINSTREL creates a new author-level goal to instantiate the motivation chain and immediately tries to solve that goal. This involves applying new ALPs to the new problem (instantiating &STATE.33) until one of them succeeds. In this case, ALP:General-Instantiate succeeds, recalling a scene in which a princess achieves a goal of achieving love by kissing the knight she loves. This recalled scene is then used to fill in the motivation chain, as shown in Figure 7.9.

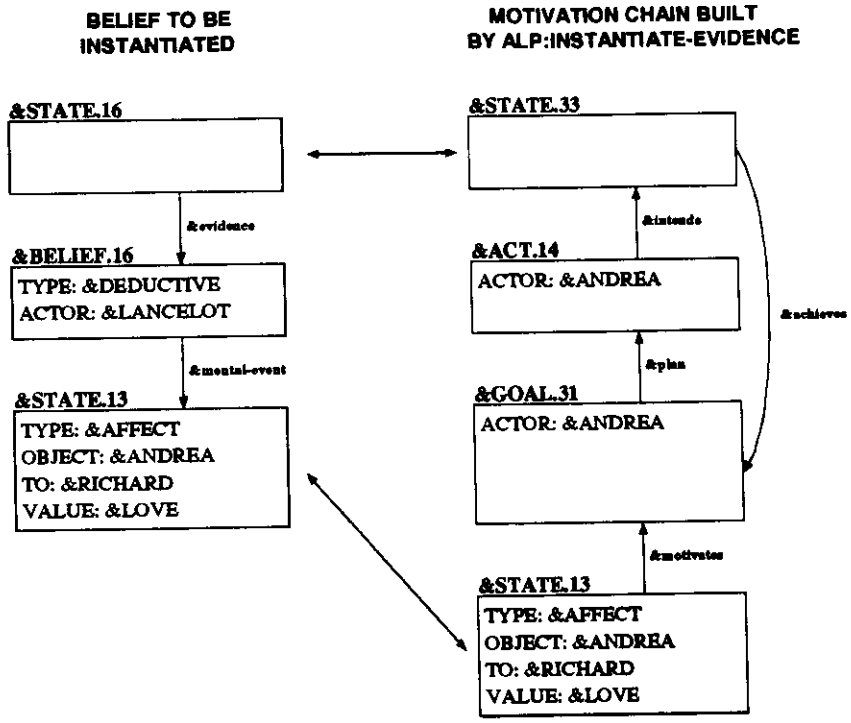


Figure 7.8 Example Motivation Chain

Andrea's kiss of Richard can now be used as evidence to support the original. ALP:Instantiate-Evidence has found something that might reasonably be expected to occur if the original belief (Andrea loves Richard) was true, namely, that Andrea would kiss Richard. To use this, step 2 of ALP:Instantiate-Evidence copies the instantiated state from the motivation chain (&STATE.33) into the &Evidence slot of the belief. This is illustrated in Figure 7.10. ALP:Instantiate-Evidence is interesting because it demonstrates how an author-level plan can recursively use author-level planning to achieve a goal. The ability to reformulate and resolve a problem is very powerful. This permits MINSTREL to "digress" to solve a problem that arises in the context of a larger problem, and to apply problem-solving techniques such as "divide and conquer".

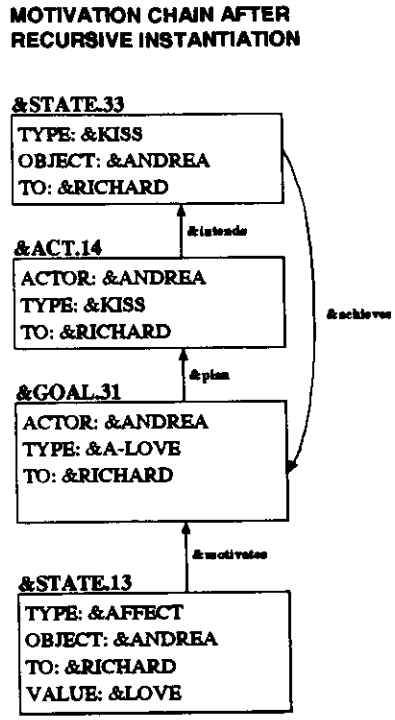


Figure 7.9 Instantiated Motivation Chain

7.4.4 ALP:Instantiate-Evidence-2

Another reason to believe something is because someone trustworthy - either an authority or one of the persons involved - tells you the belief is true. If Andrea says that she loves Richard, that's a reason to believe that she does. ALP:Instantiate-Evidence-2 creates evidence for a belief in which the character the belief is about tells the character who holds the belief that the belief is true. ALP:Instantiate-Evidence-2 is shown in Figure 7.11. The act of telling someone something is represented in MINSTREL by an &MTRANS act. The state of knowing something is represented by an &MBUILD state. ALP:Instantiate-Evidence-2 makes an &MTRANS representing the character the belief is about telling the holder of the belief that the belief is true, and an &MBUILD in which the holder of the belief hears and learns this information. An example of the evidence ALP:Instantiate-Evidence-2 creates for the belief that "Andrea loves John" is shown in Figure 7.12.

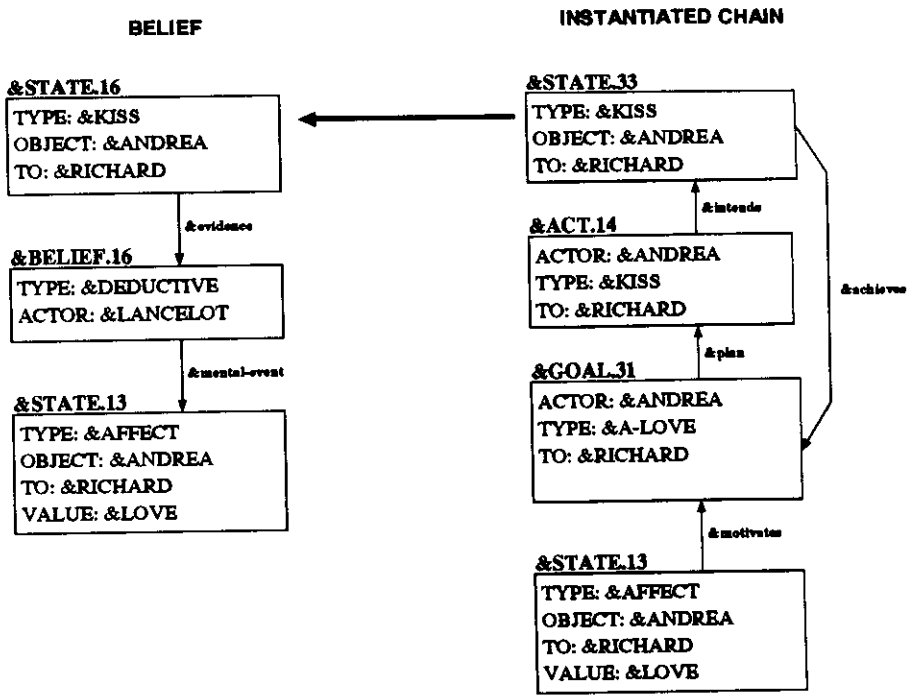


Figure 7.10 Instantiated Evidence

In this example, &ACT.15 represents Andrea telling Lancelot that she loves Richard. Andrea is the actor of this action because she is the character in the Object feature of the &mental-event of the belief. Like ALP:Instantiate-Evidence and ALP:Instantiate-Thwarting-State (below), this ALP takes advantage of the fact that representations of states concerning characters have the character in the Object feature. &STATE.15 in this example is the knowledge that Lancelot gains from what Andrea has told him, namely, &STATE.13. This in turn acts as evidence to support Lancelot's belief in &STATE.13.

7.4.5 ALP:Instantiate-Superseding-Belief

Just as there can be evidence *to* believe something, there can be evidence to *not* believe something. In MINSTREL, evidence that denies a belief is said to supersede the belief. ALP:Instantiate-Superseding-Belief is an author-level plan to instantiate a state that supersedes a belief. There are various forms of evidence (states of the world) that can supersede a belief.

If the belief is that a character has a particular goal, and the evidence to support that belief is an action or state that might plausibly been motivated by that goal (i.e., such as produced by ALP:Instantiate-Evidence), then that belief can be superseded by an alternative explanation of the evidence.

Name: ALP:Instantiate-Evidence-2
Type: &Instantiate
Object: *AL-OBJ*
Test: If the belief is about a character different than the character who holds the belief, then use this plan.
Body: 1. Make a story scene in which the character the belief is about tells the holder of the belief that the belief is true:

a. Make an uninstantiated act schema.

b. Fill the Actor slot of the act with the character the belief is about.

c. Fill the Type slot of the act with &MTRANS, indicating a transfer of information.

d. Fill the Object slot of the act with the information transferred: that the belief is true.

e. Fill in the To slot of the act with the holder of the belief.

2. Make a story scene in which the holder of the belief understands (knows) what he was told in (1):

a. Make an uninstantiated state schema.

b. Fill in the Type of the state schema as an &MBUILD, indicating the building of a mental concept.

c. Fill the Object slot with the actor who knows the concept, namely the holder of the belief.

d. Fill the Value slot with the information that was transferred in (1), namely, that the belief is true.

e. Connect this &MBUILD state to the &MTRANS act in (1) by an &Intends link.

f. Make the &MBUILD state the &Evidence for the belief.

Figure 7.11 ALP:Instantiate-Evidence-2

For example, suppose that Lancelot believes that a dragon is going to eat him because he sees the dragon flying towards him. If Lancelot later sees that the dragon is actually flying towards a baby dragon in a nearby bush, this new knowledge will supersede Lancelot's old belief about why the dragon is flying towards him, because it provides an alternate explanation for the evidence of the earlier belief.

ALP:Instantiate-Superseding-Belief creates this type of superseding state. The evidence for the original belief is used as the starting point to create an alternate explanation for the evidence, which is then used to create a superseding state. ALP:Instantiate-Superseding-Belief is shown in Figure 7.13. How ALP:Instantiate-Superseding-Belief creates superseding evidence will be

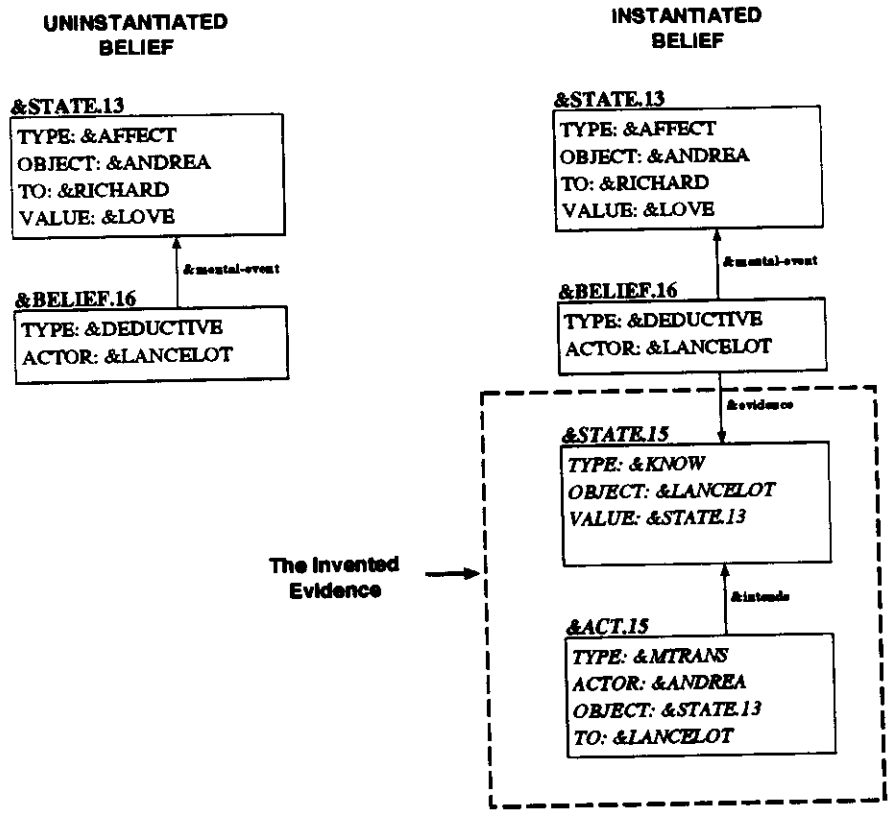


Figure 7.12 ALP:Instantiate-Evidence-2 Example

demonstrated using the example belief created by ALP:Instantiate-Evidence: "Lancelot believes that Andrea loves Richard because Andrea kissed Richard". This belief is shown in Figure 7.14. ALP:Instantiate-Superseding-Belief will be used to instantiate &STATE.16, the state that supersedes Lancelot's belief that Andrea loves Richard. The first step of ALP:Instantiate-Superseding-Belief is to see if the evidence for the original belief could also be evidence for an alternative belief. This is accomplished by making and instantiating a motivational chain, similar to the method used in ALP:Instantiate-Evidence. MINSTREL postulates that the original evidence might be motivation for a goal, and then uses instantiation (recursively) to fill in that goal. This is shown in Figure 7.15. In this example, author-level instantiation applied ALP:General-Instantiate to the goal of instantiating &GOAL.16, and ALP:General-Instantiate recalled a scene in which a princess kisses her brother to express affection. This is different from the original motivation chain for this evidence (Andrea kissed Richard to satisfy her romantic love for Richard), so the instantiated goal ("Andrea wants to express affection toward Richard") is an alternative to the original belief ("Andrea wants to achieve love with Richard").

At this point, MINSTREL now has two different beliefs that the original evidence (Andrea kissing Richard) supports: Andrea loves Richard romantically, or Andrea is expressing her (non-romantic) affection for Richard. To sway the believer towards the second belief, MINSTREL

Name: ALP:Instantiate-Superseding-Belief
Goal Type: &Instantiate
Object: *AL-OBJ*
Test: This plan applies to states that supersede beliefs.
Body: 1. To find a state that will supersede a belief, begin by finding another goal that could be achieved by the evidence for the belief:

a. Make a copy of the schema that represents the evidence for the belief.

b. Make a new, uninstantiated goal schema and connect this to the copy of the evidence by an &achieves link.

c. Use author-level instantiation recursively to instantiate the goal.

2. The instantiated goal represents an alternate belief supported by the original belief's evidence. Find additional evidence for the alternate belief by:

a. Make a new, uninstantiated state schema.

b. Connect the state schema to the instantiated goal (the alternate belief) by a &motivates link.

c. Use author-level instantiation recursively to instantiate the state schema.

3. The instantiated state schema represents additional evidence for the alternative belief, and hence, superseding evidence for the original belief. Connect the instantiated state schema to the original belief by a &supersedes link.

Figure 7.13 ALP:Instantiate-Superseding-Belief

now looks for something that will be additional evidence for the second belief. If something can be found and added to the story, it will supersede the original belief in favor of the second belief.

To do this, MINSTREL must find a new state of the world that would motivate the goal of the second belief, i.e., something else that would be evidence that Andrea has the goal of expressing her affection towards Richard. And in fact, MINSTREL has already found the required evidence. It is &State.37 in Figure 7.15, which represents Andrea and Richard being siblings. However, if the previous step of ALP:Instantiate-Superseding-Belief had not found the required evidence, then MINSTREL would use instantiation recursively to find it.

To complete the instantiation of the superseding state, the motivating state from the previous step is copied and attached to the original belief as a superseding state. This is shown in Figure 7.16. One interesting aspect of beliefs in MINSTREL is that they are represented in a compact way that discards much of the reasoning involved in their creation. Figure 7.16 succinctly represents a complex belief situation which required knowledge of goals, plans and character motivations to create. This illustrates how the apparently simple process of instantiation can involve very complex reasoning. Although the result of instantiation is often simple to understand, the process in-

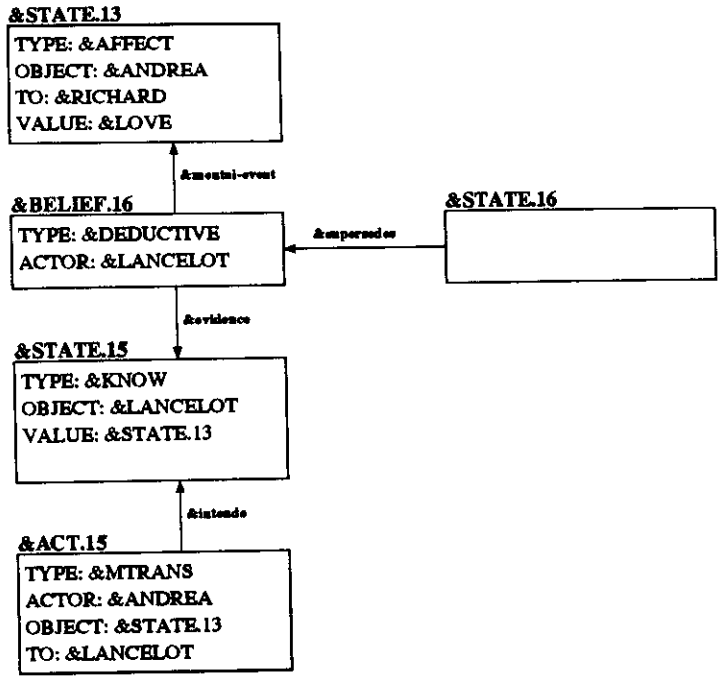


Figure 7.14 Example Belief

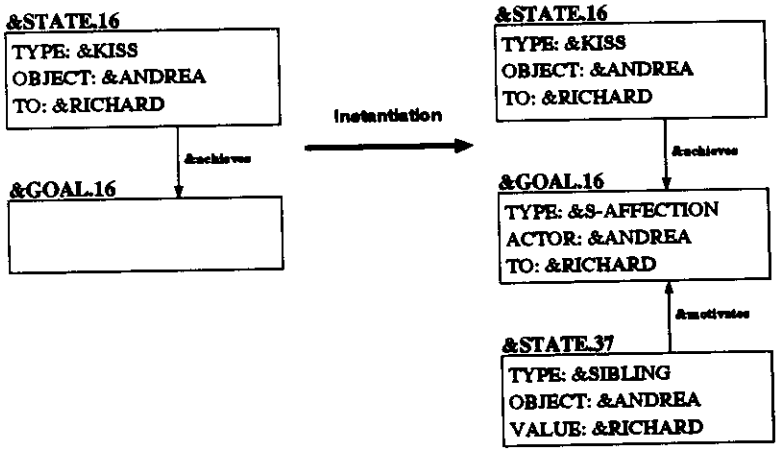


Figure 7.15 Instantiating the Evidence State

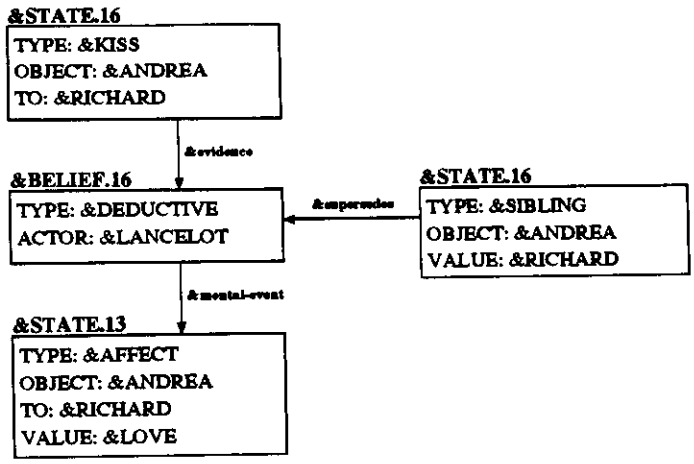


Figure 7.16 Example Superseded Belief

volved in creating that result may be very complex.

7.4.6 ALP:Instantiate-Revenge

Revenge is when one character retaliates for an insult or injury from another character. There are several interesting features about revenge that a storyteller can use to guide the instantiation. First, revenge is always motivated by a planning situation in which one character causes a goal failure for a second character. Second, a revenge goal always has an &Anti-Favor goal as a subgoal. The offended character gets revenge by causing a goal failure for the character that insulted or injured him. ALP:Instantiate-Revenge uses this knowledge to instantiate revenge goals.

When ALP:Instantiate-Revenge is applied to a goal, it tests to see if that goal could reasonably be a revenge goal. If it could, ALP:Instantiate-Revenge instantiates the goal as a revenge goal and then instantiates the related parts of a revenge situation (i.e., the motivating goal failure and the subgoal). ALP:Instantiate-Revenge is shown in Figure 7.17.

ALP:Instantiate-Revenge applies to goals that are motivated by a goal failure caused by a second party. If Lancelot suffers a goal failure because of something Richard does, then that is motivation for Lancelot to seek revenge on Richard. MINSTREL is careful to avoid creating a revenge goal in the situation where Lancelot causes his own goal failure: that would result in the nonsensical situation of Lancelot seeking revenge on himself. Figure 7.18 shows an example of a goal that ALP:Instantiate-Revenge recognizes as a revenge goal. In this example, &GOAL.16 is the goal MINSTREL recognizes as a revenge goal. &GOAL.31 is the failed goal that motivates the revenge. &GOAL.31 is Lancelot's goal to possess a magic sword; it is thwarted by Richard's act to possess the sword (&ATRANS stands for Abstract TRANSfer, and is used to represent changes of ownership).

Name: ALP:Instantiate-Revenge
Goal Type: &Instantiate
Object: *AL-OBJ*
Test: Applies to goals motivated by a goal failure caused by a different character.
Body: 1. Make the current goal a revenge goal by:

a. Set the Type slot of the current goal to &Revenge.

b. Set the To slot to the character who caused the goal failure that motivates this revenge goal ("the target").

2. Create a sub-goal of the revenge goal that will achieve the revenge goal by causing a goal failure for the other character:

a. Make a new, uninstantiated goal schema.

b. Fill in the Type slot of the new goal schema as &Anti-Favor.

c. Fill in the Actor slot of the new goal schema as the actor of the revenge goal.

d. Fill in the To slot of the new goal schema as the target character.

e. Connect the new goal schema to the revenge goal by a sub-goal link.

3. Create a new author-level goal to instantiate the sub-goal.

Figure 7.17 ALP:Instantiate-Revenge

The first step of ALP:Instantiate-Revenge is to instantiate the revenge goal by filling in the Type, Actor, and To features of the goal with &Revenge, the avenger, and the victim, respectively. The avenger is the character whose goal failure motivated the revenge, and the victim is the character who caused that goal failure. Figure 7.19 shows the example revenge goal after this step of the ALP. The actor of the revenge goal has been set to Lancelot, because he is the actor of the failed goal that is motivating the revenge. Similarly, the victim of the revenge has been set to Richard, because it was his action that led to the failure of the motivating goal. Note that ALP:Instantiate-Revenge must look through a chain of schemas to find the actor who caused the motivating goal failure.

The second step of ALP:Instantiate-Revenge is to create and instantiate the motivated &Anti-Favor sub-goal. This is accomplished by filling in the Actor, Type and To features of the motivated sub-goal as an &Anti-Favor, and then creating an author-level instantiation goal for the motivated sub-goal.

The final steps of instantiating the example revenge goal are accomplished by the author-level goal to instantiate the &Anti-Favor goal. MINSTREL accomplishes this goal (using ALP:Instantiate-Anti-Favor, and ALP:Instantiate-Thwarting-State below) by creating a scene in which Lancelot gets revenge on Richard by killing Richard, causing Richard to have a failure of

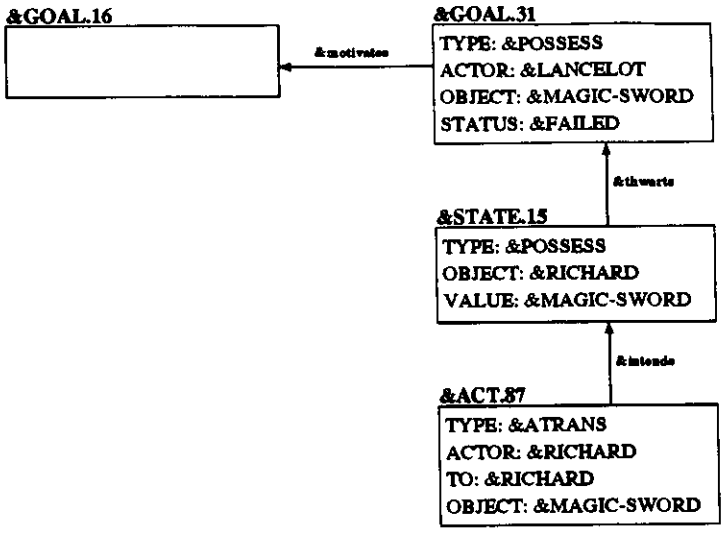


Figure 7.18 Example Revenge Goal

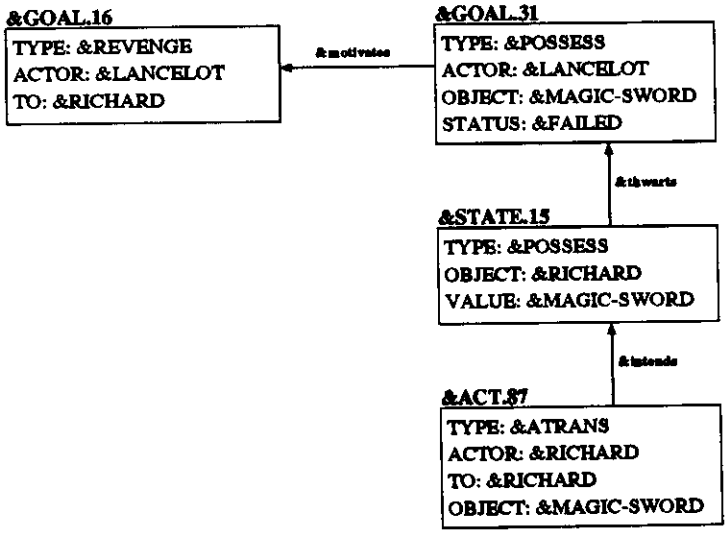


Figure 7.19 After Step 1 of ALP:Instantiate-Revenge

his goal to protect his life. This is shown in Figure 7.20.

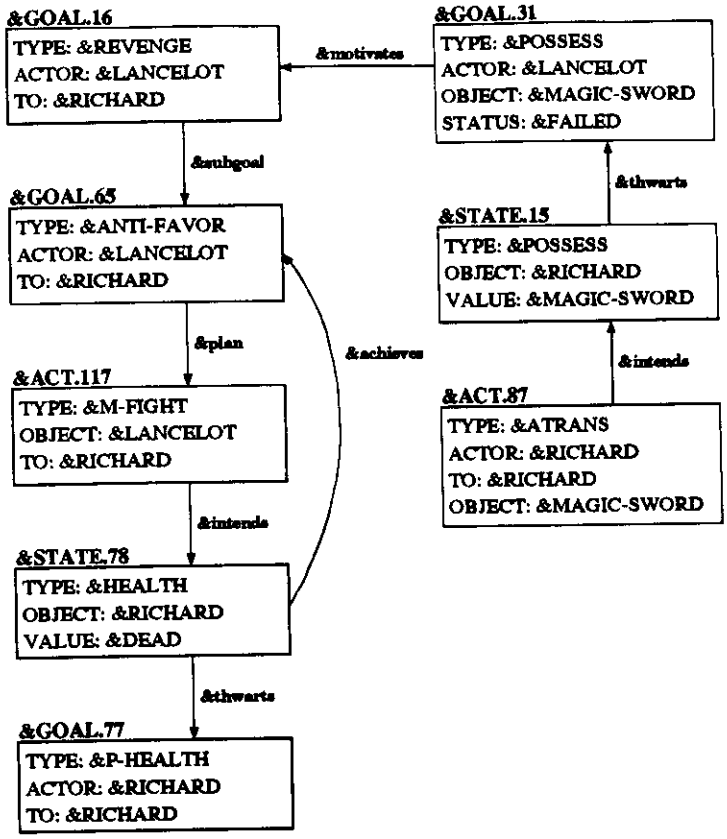


Figure 7.20 Final Result of ALP:Instantiate-Revenge

7.4.7 ALP:Instantiate-Favor and ALP:Instantiate-Anti-Favor

One does a favor by helping another achieve a goal; one does an anti-favor by thwarting another's goal. To instantiate a favor or anti-favor, MINSTREL must create the actions necessary to achieve or thwart another character's goal.

Suppose that MINSTREL is trying to create a scene in which Lancelot does a favor for Princess Jennifer. To do this, MINSTREL must (1) create a goal for Jennifer and (2) create the acts Lancelot does to achieve that goal (and consequently his own favor goal). Figure 7.21 shows the author-level plan to do this.

Figure 7.22 illustrates how ALP:Instantiate-Favor would be applied to the situation above, in which Lancelot wants to help Jennifer. The left side of this illustration shows the initial favor goal. The right side shows the favor goal after ALP:Instantiate-Favor has been executed. Note that after ALP:Instantiate-Favor has completed, the structure of the favor exists but the content – Jennifer's goal, how Lancelot helps her achieve that goal – has not been created. The author-level goals created by ALP:Instantiate-Favor to instantiate favor will create the content of the

Name: ALP:Instantiate-Favor
Goal Type: &Instantiate
Object: *AL-OBJ*
Test: Is *AL-OBJ* a favor goal?
Body: 1. Create the goal that the favor goal will achieve:

a. Make an uninstantiated goal schema.

b. Fill in the Actor slot of the new goal schema with the character the favor is being done for (which is in the To slot of the favor goal, *AL-OBJ*).

c. Connect the new goal to the favor goal by a &subsumes link.

d. Create an author-level goal to instantiate this new goal.

2. Create the act that achieves the new goal and the favor goal:

a. Make an uninstantiated act schema.

b. Fill in the Actor slot of the act schema with the Actor of the favor goal (i.e., the character doing the favor).

c. Connect the favor goal to the new act schema by a &plan link (i.e., the act is the plan to achieve the favor goal).

d. Create an author-level goal to instantiate the act.

e. Make an uninstantiated state schema.

f. Connect the new act schema to the new state schema by an &intends link (i.e., the act is intended to cause the state change).

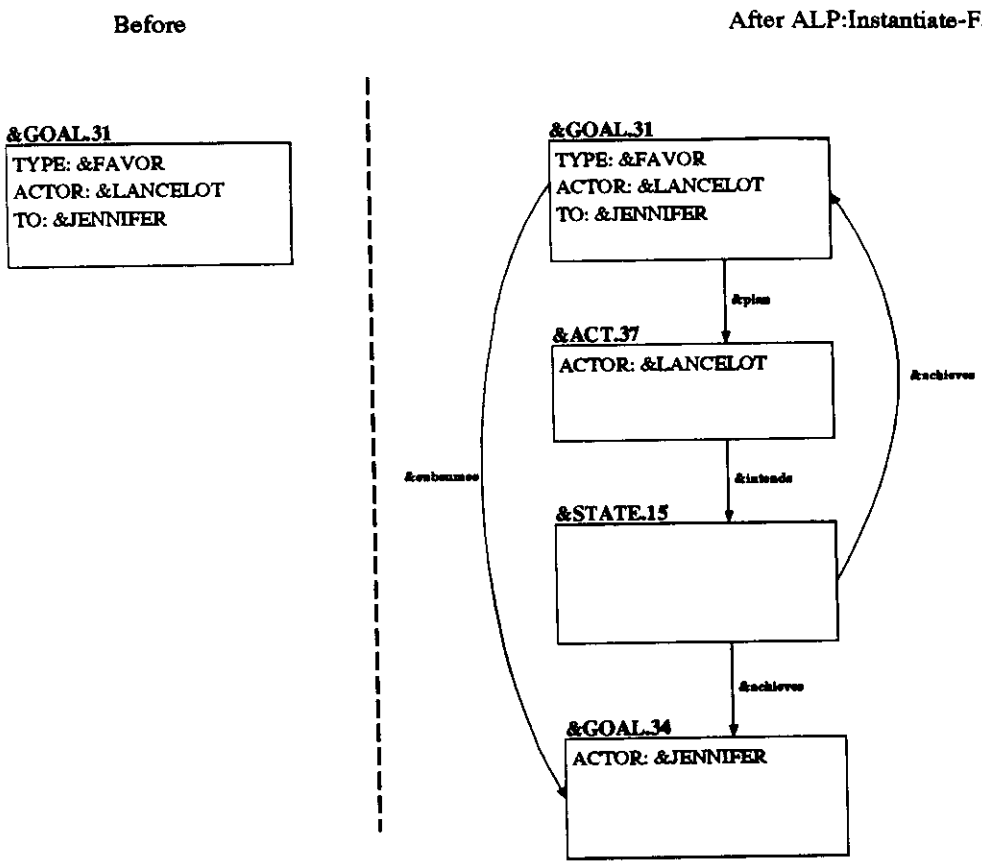
g. Connect the new state schema to both the favor goal and the new goal by &achieves links (i.e., the state change achieves the new goal and consequently the favor goal).

h. Create an author-level goal to instantiate the state schema.

Figure 7.21 ALP:Instantiate-Favor

favor (by instantiating Jennifer's goal and then Lancelot's actions to achieve that goal).

ALP:Instantiate-Anti-Favor is identical to ALP:Instantiate-Favor, except that the intention is to thwart another character's goal, rather than achieve it. This requires changing the link between the new state schema and the other character's goal to &thwarts instead of &achieves. Otherwise the two author-level plans are identical. Figure 7.23 shows how ALP:Instantiate-Anti-Favor would instantiate Lancelot doing an anti-favor to Jennifer. As with ALP:Instantiate-Favor, the content of the anti-favor will be created by other author-levels. ALP:Instantiate-Thwarting-State, below, is often used after ALP:Instantiate-Anti-Favor to instantiate the state which thwarts the



Pending author-level goals:

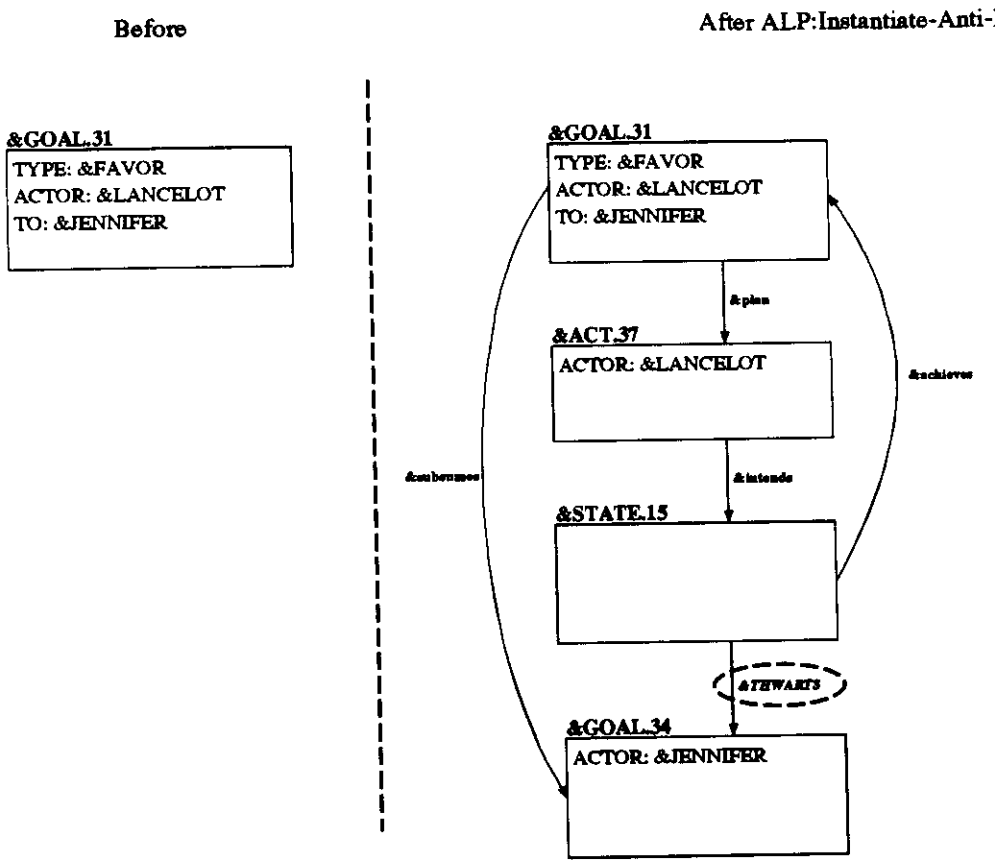
(1) Instantiate &GOAL.34
(2) Instantiate &ACT.37
(3) Instantiate &STATE.15

Figure 7.22 Example of ALP:Instantiate-Favor

other character's goal.

7.4.8 ALP:Instantiate-Thwarting-State

ALP:Instantiate-Thwarting-State is an author-level plan that knows how to instantiate a thwarting state as a conflict between two characters. Figure 7.24 shows an example of a thwarting state which can be instantiated in this way. In this example, &Goal.31 represents Lancelot's goal to possess a magic sword. &State.11 represents the new state of the world that would achieve this goal – Lancelot possessing the magic sword. The &new-state of a goal is an explicit representation of how a goal can be achieved. It is useful for representing goals for which the appropriate



Pending author-level goals:

(1) Instantiate &GOAL.34
(2) Instantiate &ACT.37
(3) Instantiate &STATE.15

Figure 7.23 Example of ALP:Instantiate-Anti-Favor

Type is unknown or difficult to determine, and it also useful for reasoning about how goals can be thwarted or achieved.

In this case, the new state represents a state of the world concerning the actor of the goal, namely, "Lancelot possesses the sword." Many goals which are achieved by the actor of the goal achieving a certain state can be thwarted by another character achieving that same state. In this example, Lancelot's goal of possessing the sword will be thwarted if another character achieves the state shown in &State.11, i.e., if (for example) Richard possesses the sword. Similarly, if Jennifer has the goal to express her love towards Lancelot by kissing him, then her goal will be thwarted if another princess kisses Lancelot. ALP:Instantiate-Thwarting-State uses this knowledge to instantiate thwarting states for goals with new-states of this type. ALP:Instantiate-

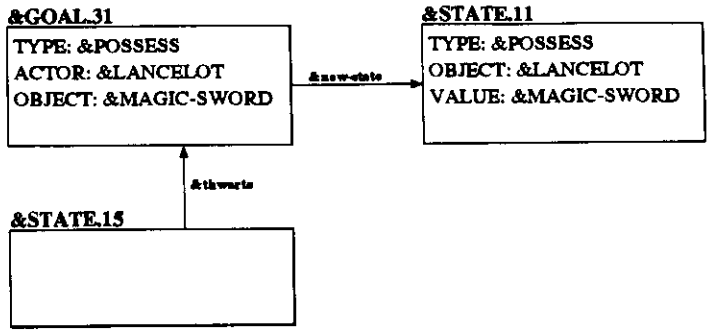


Figure 7.24 Example Thwarting State

Thwarting-State is shown in Figure 7.25.

Name:	ALP:Instantiate-Thwarting-State
Goal Type:	&Instantiate
Object:	*AL-OBJ*
Test:	If *AL-OBJ* is a state which thwarts a goal, and the Actor of the thwarted goal is the same as the Object of the &New-State of thwarted goal, then apply this plan.
Body:	1. Copy the &New-State of the thwarted goal into *AL-OBJ*. 2. Replace the Object of *AL-OBJ* with a new character. 3. Create an author-level goal to instantiate the new character.

Figure 7.25 ALP:Instantiate-Thwarting-State

The first step of ALP:Instantiate-Thwarting-State is to copy the features of the &New-State to the thwarting state. Figure 7.26 shows the example thwarting state after this step.

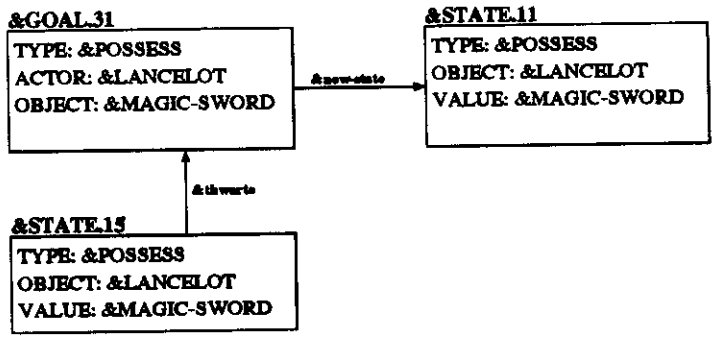


Figure 7.26 Thwarting State After Step 1

The second step of ALP:Instantiate-Thwarting-State is to create a new, uninstantiated character and make him the Object of the thwarting state. Figure 7.27 shows the example thwarting state

after this step.

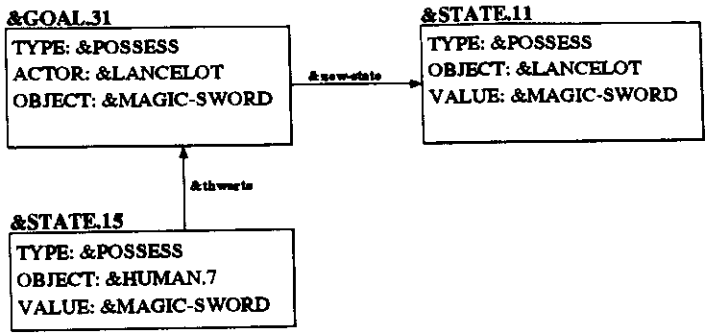


Figure 7.27 Thwarting State After Step 2

The final step of ALP:Instantiate-Thwarting-State is to create an author-level goal of instantiating the new character. Although a new character was introduced in creating the thwarting state, this does not necessarily mean that the new character will be present in the final story. The author might have an author-level plan to instantiate new characters by replacing them with an existing character, when appropriate. In this way, the author could "re-use" characters and create a tighter story. However, MINSTREL does not currently have any author-level plans that try to reuse characters, so MINSTREL does indeed create a new character in this case. (For example, see the story *The Princess and the Berries.*)

7.4.9 ALP:Instantiate-Unthwarts

When a character has a thwarted goal, this can motivate the character to unthwart the goal, i.e., remove the thwarting state, find an alternate goal, etc. MINSTREL represents this as a sub-goal of the thwarted goal. ALP:Instantiate-Unthwarts is an author-level plan for instantiating this type of sub-goal.

One way to unthwart a goal is to change or remove the state of the world that is thwarting the goal, and if the thwarting state involves a story character, one way to eliminate the thwarting state is to kill that character. For example, if Lancelot's goal of possessing a magic sword is being thwarted by the fact that Richard possesses the sword, then one solution is to kill Richard.

Obviously, ALP:Instantiate-Unthwarts is an example of an author-level plan very specific to the King Arthur domain. Outside of storytelling, killing someone is rarely a good plan to achieve a goal. Even in storytelling domains, this plan is only acceptable in certain genres.

Figure 7.28 illustrates ALP:Instantiate-Unthwarts.

ALP:Instantiate-Unthwarts tests to see if the actor of the goal being instantiated is a violent character. MINSTREL has semantic knowledge about the characteristics of roles in the King Arthur

Name: ALP:Instantiate-Unthwarts
Type: &Instantiate
Object: *AL-OBJ*
Test: This plan applies when a violent character has the goal to unthwart a goal that was thwarted by another character. *AL-OBJ* is a subgoal of the thwarted goal.
Body: 1. Make the sub-goal of the thwarted goal be the goal to kill the character who thwarted the goal:

a. Set the Type slot of *AL-OBJ* to &Destroy.

b. Set the Actor slot of *AL-OBJ* to the actor of the thwarted goal.

c. Set the To slot of *AL-OBJ* to the character who thwarted the original goal.

2. Create a state schema representing the death of the thwarting character.

a. Make a new, uninstantiated state schema.

b. Set the Type slot of the schema to &Health.

c. Set the Object feature of this schema to the thwarting character.

d. Set the Value feature of this schema to &Dead.

e. Connect this schema to the goal to destroy the thwarting character by an &achieves link.

Figure 7.28 ALP:Instantiate-Unthwarts

domain. One aspect of this knowledge is whether or not a role is violent, i.e., whether or not a character of that type uses plans involving violence. Knights and monsters use violent plans; princesses and hermits do not. For more on MINSTREL's class hierarchies of character roles, see the discussion of TRAM:Similar-Outcomes in Chapter 3.

Figure 7.29 shows how ALP:Instantiate-Unthwarts would create an unthwarting goal for the thwarted goal created in the previous section.

ALP:Instantiate-Unthwarts creates a goal for Lancelot to destroy &Human.7, the character who thwarted Lancelot's goal of possessing the magic sword by possessing it himself. The new-state for this goal is set to be the death of &Human.7.

Note that ALP:Instantiate-Unthwarts does not create a plan for achieving this new goal. The plan for achieving this goal, if necessary, will be created by another author-level goal. But it's certainly possible that the story MINSTREL is creating requires only that Lancelot have the goal to kill &Human.7, not that he act upon it. For example, this scene may be created to illustrate Lancelot's quick temper, in which case it isn't necessary that Lancelot act upon his goal.

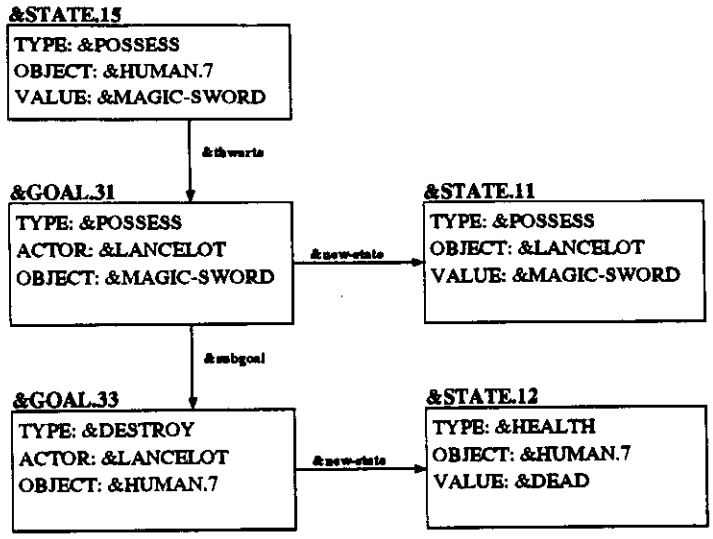


Figure 7.29 Unthwarting Goal

This illustrates an important feature of MINSTREL's storytelling model. Instantiation is a process that is subordinate to the author-level goals that use it to create story scenes. It may seem "obvious" that MINSTREL should complete the scene it is creating by finding a plan for Lancelot's new goal, but in fact it would be incorrect to do so. The instantiation process must be driven by the abstract scene descriptions created by higher-level author goals.

7.4.10 ALP:Instantiate-Deception

In the discussion of ALP:Instantiate-Superseded-Belief, we saw how MINSTREL creates scenes in which a character has a mistaken belief. In a mistaken belief, a character believes something about the world when in fact something contradictory is true. For example, in the story *Richard and Lancelot*, Richard forms a mistaken belief that Andrea loves Lancelot.

Deceptions are similar to mistaken beliefs. However, in deception, the mistaken belief occurs because a character does something to intentionally present a false state to the world. For example, in *Romeo and Juliet*, Juliet purposely takes a potion that makes her appear to be dead in order to deceive her family.

Representing deception is complex, because we must capture both the deceptive and true states of the world as well as the intentional nature of the deception. In MINSTREL, a deception is represented as a goal to create a deceptive state of the world. This goal is achieved by an action that intends two different states of the world: one of these states is the deception state and the other is the "true" state. MINSTREL's representation of this scene (which is part of MINSTREL's episodic memory) is shown in Figure 7.30.

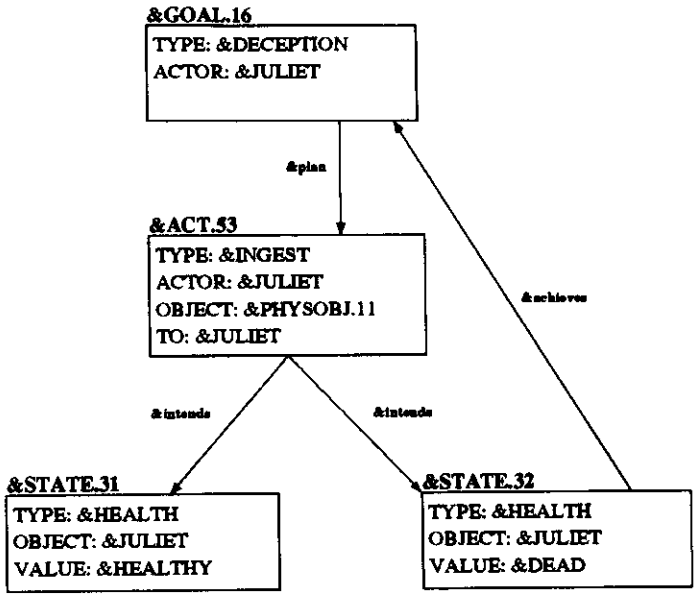


Figure 7.30 Representation of Juliet's Deception

In this representation, &Goal.16 represents Juliet's goal to be deceptive. Her plan for achieving this goal is &Act.53 - drinking a magic potion. This action intends two results. The first, shown to the left in Figure 7.30, is the true state of the world - Juliet is healthy. The second, shown to the right, represents the deceptive state - Juliet appearing dead. This deceptive state achieves Juliet's goal of deceiving the world.

Because deception is complex, instantiating deception goals is also complex. MINSTREL's author-level plan for instantiating deceptions is called ALP:Instantiate-Deception.

There are several steps to instantiating a deception goal.

First, MINSTREL must decide what the deceptive state is going to be. This requires examining how the deceptive state is used in the story to determine what an appropriate deception would be. For example, if the deception state is used to motivate a character to be hungry, then the deception state should probably be some appealing display of food. The goal that the deception state motivates (another character being hungry) is called the secondary goal. Once the deception state has been determined, MINSTREL then determines what the true state is (by reasoning about opposite states, as in ALP:Instantiate-Thwarting-State). Finally, MINSTREL must try to instantiate the act that intends these states. ALP:Instantiate-Deception is shown in detail in Figure 7.31.

Figure 7.32 shows an example storytelling situation involving a deception. In this example, a princess named Jennifer plans to get revenge on another princess by making her think she's being attacked by a monster. Jennifer's plan is to have a monster move towards the other princess.

Name:	ALP:Instantiate-Deception
Goal Type:	&Instantiate
Object:	*AL-OBJ*
Test:	If *AL-OBJ* is a deception goal, then apply this plan.
Body:	1. The state that achieves *AL-OBJ* is the deception state. Call the goal that the deception state achieves the secondary goal. 2. Use instantiation recursively to instantiate the secondary goal: 3. Use instantiation recursively to instantiate the deception state. 4. If (3) fails, instantiate the deception state as the existence of the actor of the secondary goal: a. Set the Type feature of the deception state to &Exists. b. Set the Object feature of the deception state to the actor of the secondary goal. 4. Use reasoning about opposite states to instantiate the "true" state as the opposite of the deception state. 5. Create an author-level goal to instantiate the action that intends the true state and the deception state. If it is missing, fill in the actor of the action with the actor of the original deception goal.

Figure 7.31 ALP:Instantiate-Deception

Since Jennifer can't make a monster do this, she hopes to achieve her goal using a deception. ALP:Instantiate-Deception will invent an appropriate deception to achieve this goal.

The first three steps of ALP:Instantiate-Deception are to determine and instantiate the secondary goal or act. In this example, &Act.15 is the secondary act, and it has already been instantiated as a monster moving towards a princess. However, the deception state isn not instantiated, so instantiation is used recursively to instantiate the deception state in step (2). In this case, that fails, because MINSTREL cannot recall any precondition for a monster to move towards a princess. ALP:Instantiate-Deception falls through to step three, where the deception state is instantiated as the existence of the actor of &Act.15. The deception after the first three steps are complete is shown in Figure 7.33.

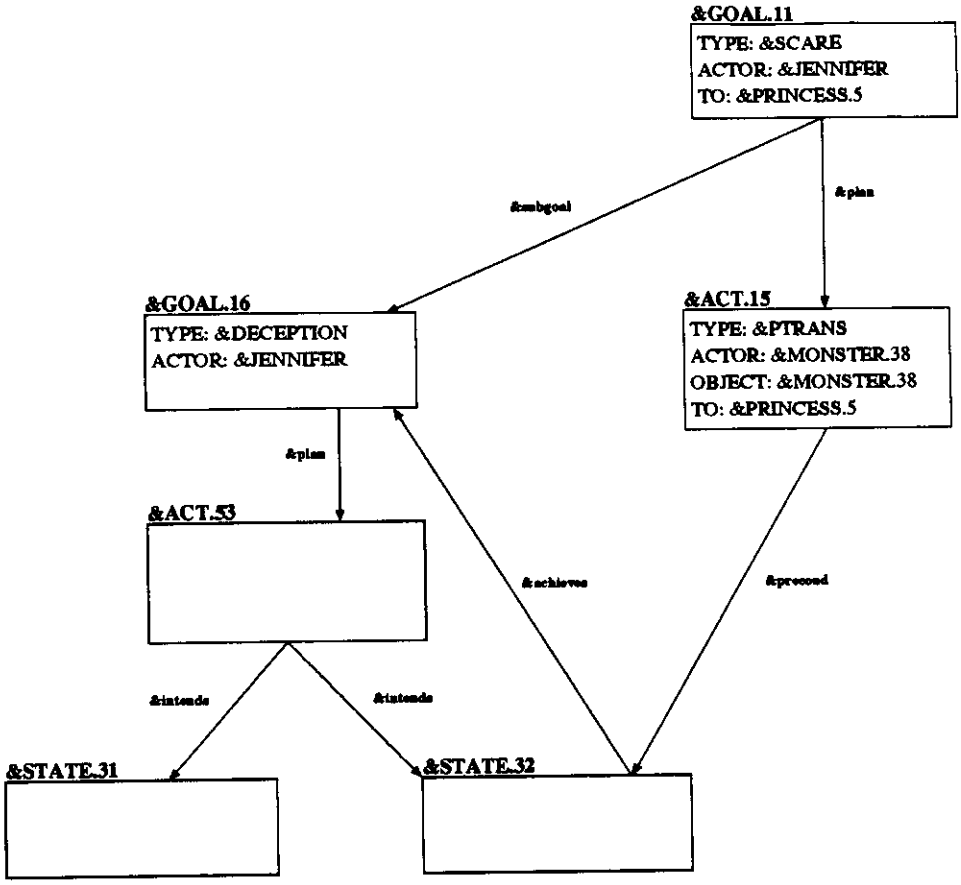


Figure 7.32 Example Deception Situation

The intention of the first three steps of ALP:Instantiate-Deception is to find a reasonable deception state. The first method is to hope that instantiating the secondary goal/act will also instantiate the deception state. The next method is to use instantiation recursively. This permits MINSTREL to use its author-level plans and creativity heuristics to attempt to invent a reasonable deception state. If this also fails, the last resort is to create a state that is a default precondition for any act, namely that the object of the act must exist. In this example, MINSTREL can find no other preconditions for &Act.15, so it uses the default precondition.

The next step in this example is to instantiate the true state of the deception, &State.31, as the "opposite" of the deception state, &State.32. The opposite of the existence of something is the non-existence of that thing, so MINSTREL instantiates &State.31 as the non-existence of &Monster.38. This is shown in Figure 7.34. What remains is to instantiate the action that intends the deception state and the true state. This is accomplished by creating an author-level goal to instantiate the act. In this example, MINSTREL uses TRAM:Intention-Swith and TRAM:Use-Magic to recall the scene in which Juliet quaffs a potion to appear dead, and uses that to instantiate this act as Jennifer quaffing a potion to appear to be a monster. The final form

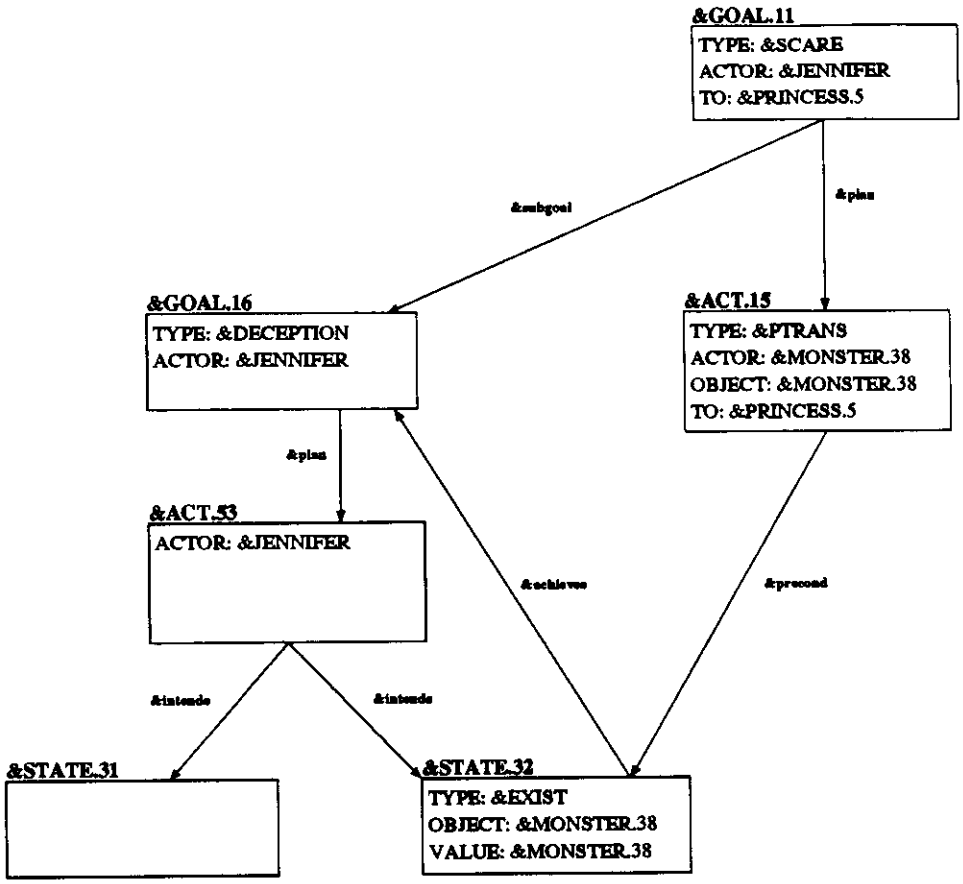


Figure 7.33 Instantiated Deception State

of the deception is shown in Figure 7.35. ALP:Instantiate-Deception combines in one author-level plan many of the techniques MINSTREL uses to augment the power of imaginative memory, including divide-and-conquer, domain-specific knowledge, and recursive problem-solving.

7.4.11 ALP:Dont-Instantiate

ALP:Dont-Instantiate is an author-level plan that helps MINSTREL avoid unnecessary instantiating. Scenes which already have all or most of their important details filled in don't need to be instantiated. ALP:Dont-Instantiate detects these kinds of scenes and avoids instantiating them. This is primarily an efficiency measure. Nothing untoward happens if MINSTREL tries to instantiate a scene unnecessarily; but it does waste time and effort.

There are several reasons MINSTREL might have an unnecessary author-level goal to instantiate a story scene. When story scenes are created and added to the current story, MINSTREL creates an author-level goal to instantiate each part of the new scenes. But some of these parts might not require instantiation. Or a story scene might have been instantiated as a side-effect of a plan that

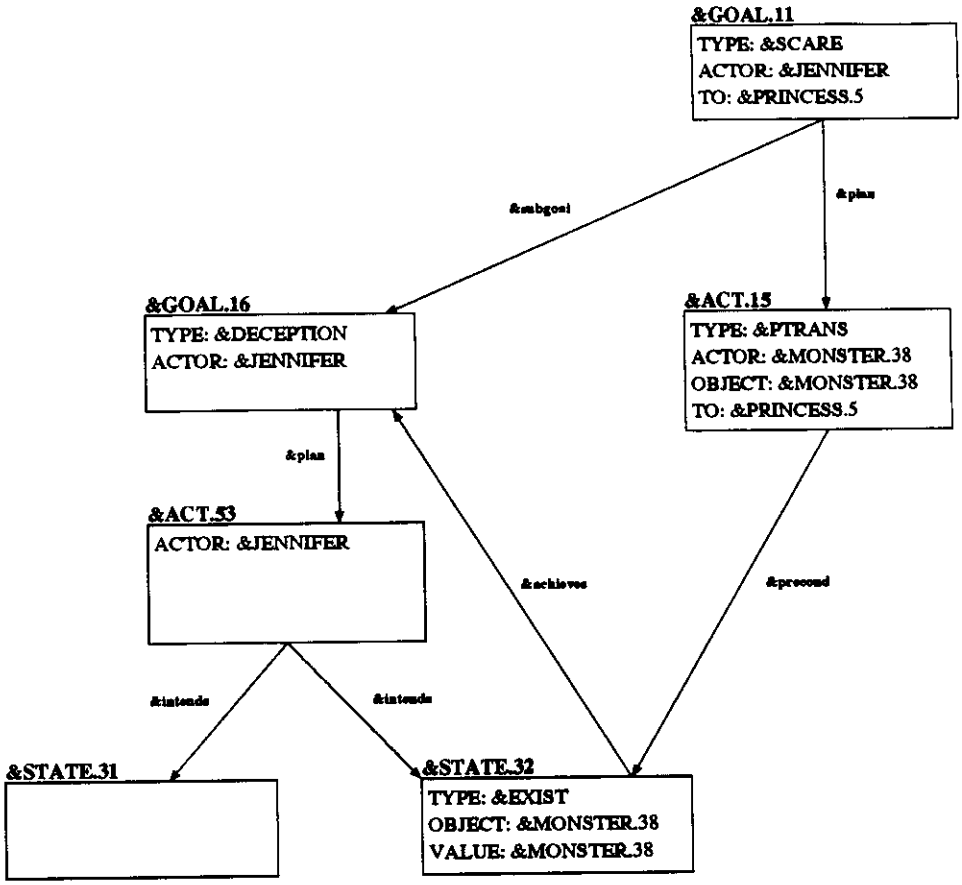


Figure 7.34 Instantiated True State

was executed while the instantiation goal for that goal waited on the planning queue for higher level goals to execute.

ALP:Dont-Instantiate detects already-instantiated scenes by checking the Type feature of the schemas representing the scene. The Type feature is usually set as part of instantiation, and so is a good indicator of whether or not a scene has already been instantiated. ALP:Dont-Instantiate is shown in Figure 7.36.

7.5 Conclusions

Instantiation is the process of creating complete knowledge structures from partially specified knowledge structures. In general, instantiation is achieved by using imaginative memory to recall a known structure similar to the specification, and then using a merging process to instantiate the specification from the recalled structure.

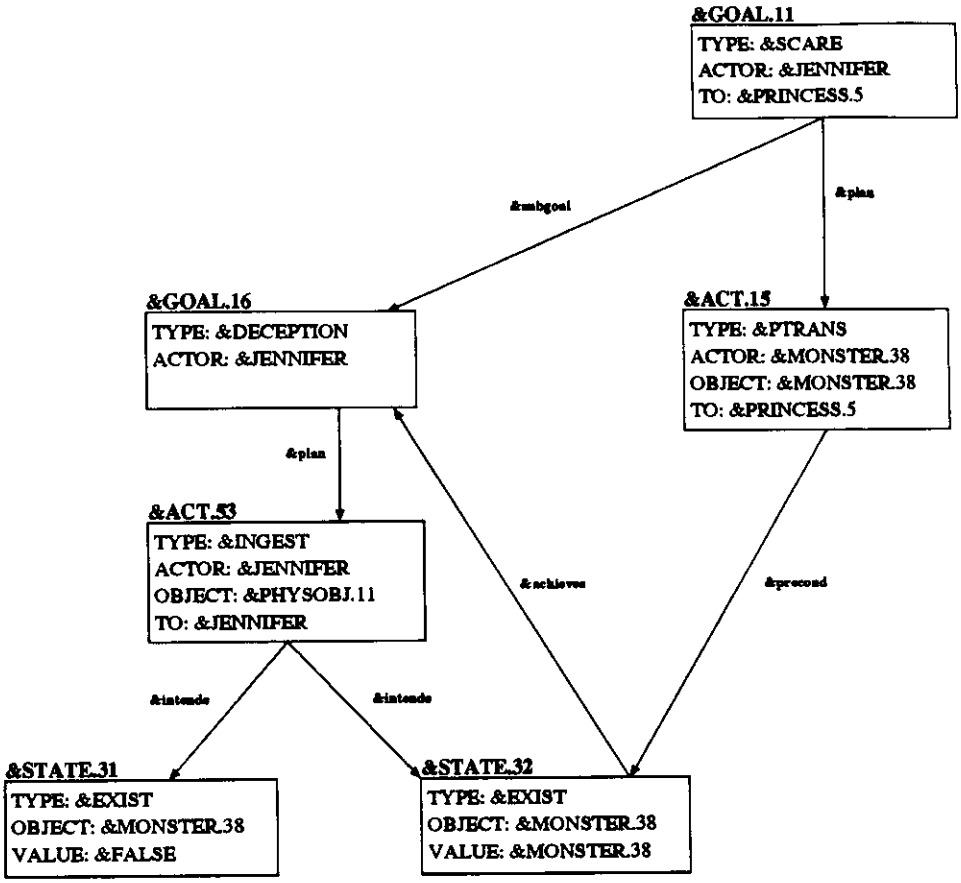


Figure 7.35 Final Deception

Name: ALP:Dont-Instantiate
Goal Type: &Instantiate
Object: *AL-OBJ*
Test: *AL-OBJ* is already instantiated if the Type feature has a value.
Body: 1. Succeed if *AL-OBJ* is already instantiated.

Figure 7.36 ALP:Dont-Instantiate

For particular problem domains, instantiation is augmented with domain-specific knowledge. Instantiation and creativity are general processes that perform well across a wide range of problem domains, but which consequently cannot be maximally effective in every domain. Augmenting instantiation with knowledge for a particular problem domain permits the solution of problems which could not be solved using only domain-independent knowledge. In MINSTREL, such domain-dependent knowledge is captured in author-level plans.

CHAPTER 8
Dramatic Writing Goals in Storytelling

8.1 Introduction

The primary concern of MINSTREL is to tell a story that illustrates a particular story theme. The previous two chapters have shown how MINSTREL selects a theme and actively plans to tell a story based upon that theme.

But authoring involves more than just the bald statement of the events that illustrate the story theme. Good authors make use of a variety of techniques to improve the literary quality of their stories: pacing, characterization, dialogue, suspense, foreshadowing, description, and many others. These flourishes "dress up" the story, adding literary appeal to the story's basic message. A very good author uses these techniques not only to create literary value, but also to support the theme of his story.

Human authors have many literary writing techniques. A look at the creative writing section of any bookstore will reveal that there are as many writing techniques as there are authors to expound them. Every human author develops a combination of literary writing goals and techniques that create a particular writing style.

It would be impossible to model all these techniques and their myriad combinations in MINSTREL. Instead, we have chosen to select and implement a few representative techniques: suspense, tragedy, characterization, and foreshadowing. This particular combination of techniques represents, if you will, MINSTREL's writing style. More importantly, these techniques demonstrate that the MINSTREL model is capable of capturing the knowledge and processes needed to implement typical literary writing techniques.

The remainder of this chapter is divided into four sections: suspense, tragedy, foreshadowing and characterization. Each section contains a brief discussion of the writing technique, an example of how MINSTREL used the writing technique in a finished story, and a description of the author-level goals and plans used to implement the writing technique. The final section of the chapter summarizes the lessons learned in implementing these techniques.

8.2 Suspense

The Random House Dictionary of the English Language defines suspense as "a state or condition of mental uncertainty or excitement, as from awaiting a decision or outcome." Building suspense – that is, inducing this feeling in the reader – is a key element of good writing in adventure and mystery stories. By prolonging the resolution of an uncertain outcome, the author emphasizes the eventual resolution. And human readers find the feeling of suspense followed by relief pleasurable, even when taken to great extremes, as in horror films.

8.2.1 Suspense in *The Hermit and the Knight*

In *The Hermit and the Knight*, MINSTREL has the author-level goal to build suspense in a scene in which a hermit is threatened by a dragon:

The Hermit and the Knight

Once upon a time, there was a hermit named Bebe and a knight named Cedric. One day, Cedric was wounded when he was attacked by a dragon. Bebe, who was in the woods picking berries, healed Cedric. Cedric was grateful and vowed to return the favor.

Later, Bebe believed that he would die because he saw a dragon moving towards him and believed it would eat him. *Bebe was very scared. Bebe tried to run away but failed!...*

In the second paragraph of this story, Bebe the hermit is endangered by a dragon. Because Bebe is a sympathetic character, the reader does not want to see Bebe harmed. This tension is the basis of suspense. MINSTREL recognizes this and attempts to increase the suspense in this situation by inventing story scenes in which demonstrate Bebe's fear and in which Bebe attempts to escape his fate but fails.

As with all writing techniques, there are two fundamental questions to ask about the use of suspense in writing: (1) When is it appropriate to use this writing technique? and (2) How can this writing technique be achieved?

8.2.2 When is Suspense Appropriate?

In *The Hermit and the Knight*, MINSTREL decides to build suspense in the scene in which Bebe is threatened by the dragon. Why did MINSTREL choose to build suspense here and not, say, in the scene in which Bebe was picking berries? More interestingly, why didn't MINSTREL also choose to build suspense in the scene in which Cedric was attacked by a dragon - a scene which shares many features with the Bebe scene?

According to the dictionary, suspense can result from any undetermined decision or outcome. In reality, people feel suspense in porportion to the importance of the decision or outcome. Whether or not a character will die is inherently more suspenseful than whether or not he will have to walk to the castle, because preserving one's life is a more important goal that preserving one's feet. It is apparent, then, that suspense is most appropriate for story scenes that involve important goals.

There have been various efforts to classify common human goals according to their importance. Schank [] suggested that goals are interesting in porportion to their importance, and other authors have made similar suggestions or rankings of common goals. What is important to an author is not the precise ranking of goals - which will undoubtedly vary from person to person - but that

he understand that the appropriateness of suspense is porportional to the importance of the goal involved, so that he can correctly identify appropriate uses of suspense. This is the first principle of suspense:

> **Scenes are appropriate for suspense in porportion to the importance of the character-level goals involved.**

MINSTREL has a simple ranking of character goals it uses to determine when suspense is reasonable. Only one goal is important enough to warrant suspense: the goal to preserve one's life. When MINSTREL looks at story scenes to determine if they are reasonable candidates for suspense, it accepts only scenes involving the possible loss of a character's life.

Beyond the question of whether it is *reasonable* to build suspense in a scene is the question of whether it is *desirable*. As noted above, suspense focuses the reader's attention on the outcome of the undecided situation. Consequently, the author should use suspense only in scenes he wants the reader to focus upon. Consider, for example, the following story fragment:

> John knew that he had to get to the hospital if he wanted to save his little brother's life. He jumped onto his bicycle and pedalled down the hill as fast as he could. The trees of his neighborhood flew by on either side as he sped down the hill towards the hospital.
>
> At the bottom of the hill, John rode passed the city park. As he passed the park, John saw a young woman being attacked by a mugger. She tried to run away, but the mugger jumped a bush and grabbed her again. She struggled and screamed, but no one heard her. Finally she got loose and ran towards the front of the park, where she collided with a patrolman. The mugger, seeing this, darted off into the bushes.
>
> John continued pedalling past the park, turning onto the narrow side street that led to the hospital. He risked a glance at his watch. Now he had only five minutes left! He pedalled even harder, his chest pounding with the effort...

Most readers find the part of this story concerning the mugger and the young lady distracting and irritating. Although suspenseful, it turns the reader's attention away from the main focus of the story: John's effort to save his little brother. By using suspense in a secondary scene, the author has distracted the reader from the main point or purpose of the story, and consequently weakened his story rather than strengthened it. This is the second principle of supense:

> **Scenes are appropriate for suspense in porportion to their importance to the story.**

The main purpose of the stories MINSTREL tells are to illustrate a particular Planning Advice Theme (PAT). Consequently, MINSTREL limits suspense to scenes which directly illustrate the theme of the story. Thus MINSTREL is assured that the reader's attention will be focused on the main point of the story.

These two principle explain why MINSTREL chose to build suspense in the scene in which Bebe

was being threatened by the dragon. The scene in which Bebe picks berries isn't suitable for supense because the goal of the involved character is unimportant; the scene in which Cedric is injured is not suitable for suspense because it does not directly illustrate the theme of the story. The two principles of suspense focus MINSTREL's attempts to build suspense where they will be most successful and most effective.

MINSTREL's author-level plan for determining whether to build suspense in a scene is shown in Figure 8.1. During the course of storytelling, MINSTREL has a goal to check every scene created to see whether it is a suitable candidate for building suspense. ALP:Check-Scene-For-Suspense is the author-level plan that determines whether a scene is suitable for suspense.

Name:	ALP:Check-Scene-For-Suspense
Goal Type:	&Check-Scene-For-Suspense
Object:	*AL-OBJ*
Test:	1. Is *AL-OBJ* a &P-HEALTH or &C-HEALTH goal?
	2. Is *AL-OBJ* part of the plot of the story?
	3. If yes to both questions, succeed.
Body:	1. Create an author-level goal to &Add-Suspense-to-Scene to *AL-OBJ*.

Figure 8.1 ALP:Check-Scene-For-Suspense

ALP:Check-Scene-For-Suspense first determines whether the part of the story it is looking at is a &P-Health or &C-Health goal schema. If it is not, ALP:Check-Scene-For-Suspense fails, because MINSTREL tries to build suspense only in story scenes involving the possible death of a character. Second, ALP:Check-Scene-For-Suspense checks to see whether the goal schema is part of the plot of the story, i.e., the schemas which illustrate the Planning Advice Theme of the story. If it is not, ALP:Check-Scene-For-Suspense fails, because MINSTREL uses suspense only to focus the reader's attention on the elements of the story which illustrate the theme of the story. If a story scene passes both these tests, then MINSTREL tries to add suspense to that scene by applying one or more writing techniques for building suspense.

Two types of knowledge must be used in deciding where to apply a writing technique such as suspense. First, the author must decide if the technique is suitable to a particular scene, i.e., whether or not the scene has characteristics suited for this technique. Secondly, the author must decide whether or not the changes the technique will make in the story are desirable, i.e., whether they will improve the story.

8.2.3 Writing Techniques for Building Suspense

How does an author make a scene suspenseful? There are many methods, and inventive writers are always discovering more. L. Sprague De Camp, advising writers on how to write imaginative literature, suggested:

> *Suspense is effected by strong emotions on the part of the characters, by threats to their*

well-being, by bits of atmospheric description slyly dropped into the action...[1]

MINSTREL implements two methods to build suspense. The first corresponds to De Camp's first suggestion of having the story characters display strong emotions. The second is a technique often seen in modern horror films, which we call the Horror Film Principle.

8.2.3.1 ALP:Add-Suspense-Via-Character-Emotion

MINSTREL's first author-level plan for adding suspense to a scene is to have the character whose life is threatened react by being scared. This is implemented by the author-level plan ALP:Add-Suspense-Via-Character-Emotion. This plan was used in the telling of *The Hermit and the Knight*, in the scene in which the hermit is threatened by a dragon:

> Later, Bebe believed that he would die because he saw a dragon moving towards him and believed it would eat him. *Bebe was very scared.* Bebe tried to run away but failed! ...

ALP:Add-Suspense-Via-Character-Emotion examines the scene to which suspense is being added and determines the character whose life is being threatened. ALP:Add-Suspense-Via-Character-Emotion then changes the story so that the event that motivates the character's goal to protect his life also motivates the character to be scared. ALP:Add-Suspense-Via-Character-Emotion is shown in Figure 8.2.

The result of ALP:Add-Suspense-Via-Character-Emotion is a short scene in which the character whose life is threatened is described as being fearful.

8.2.3.2 ALP:Add-Suspense-Via-Failed-Escape

A technique for building suspense that is commonly used in horror films is the failed escape. The young heroine, menaced by a masked, knife-wielding maniac, runs down a hallway towards a door and certain escape, only to discover at the last moment that the door is locked. The reader's tension is redoubled when the apparent resolution is reversed.

MINSTREL implements this technique in ALP:Add-Suspense-Via-Failed-Escape. This author-level plan tries to increase the suspense in a story scene in which a character's life is threatened by constructing a plan by which the threatened character can avoid dying. This plan is then executed, only to fail in the final step. ALP:Add-Suspense-Via-Failed-Escape is shown in Figure 8.3. This plan was used in *The Knight and the Hermit* to increase the suspense in the scene in which the hermit's life is threatened by the dragon:

> Later, Bebe believed that he would die because he saw a dragon moving towards him and believed it would eat him. Bebe was

1. De Camp, L. Sprague and De Camp, Catherine C., *Science Fiction Handbook, Revised*, 1975, pp. 147.

Name: ALP:Add-Suspense-Via-Character-Emotion
Goal Type: &Add-Suspense-to-Scene
Object: *AL-OBJ*
Test: None.
Body: 1. Determine the actor of *AL-OBJ*, Actor. Determine the event motivating *AL-OBJ*, Motivation.

2. Create a new state representing Actor being fearful:

1. Make a new, uninstantiated state schema.
2. Set the Type feature of the new state to &Affect.
3. Set the Object feature of the new state to the Actor.
4. Set the Value feature of the new state to &Neg (negative).
5. Set the Scale feature of the new state to &Normal.
6. Connect the new state (representing Actor's fear) to the event that caused the fear by a &motivates link.

Figure 8.2 ALP:Add-Suspense-Via-Character-Emotion

Name: ALP:Add-Suspense-Via-Failed-Escape
Goal Type: &Add-Suspense-to-Scene
Object: *AL-OBJ*
Test: Use if the character doesn't already have a plan to avoid dying.
Body: 1. Create a plan to avoid dying:

a. Make a new, uninstantiated act schema.
b. Set the Actor feature of the new act to the threatened character.
c. Connect the new act to the threatened character's &P-Health goal by a &plan link (i.e., the new act is a plan to achieve the &P-Health goal).
d. Use author-level planning recursively to instantiate the new act.

2. Mark the final step of the plan as failed.

Figure 8.3 ALP:Add-Suspense-Via-Failed-Escape

very scared. *Bebe tried to run away but failed!* . . .

In this case, MINSTREL finds a plan to avoid being eaten by running away. This is added to the story and the final step of this plan is marked as a failure. (In this case, the plan has only one step, fleeing, so that is marked as the failure.) The result is a scene in which Bebe tries to flee but fails to outrun the dragon.

8.3 Tragedy

Like suspense, tragedy depends upon creating an emotion in the reader. To create suspense, the writer evokes anticipation. To create tragedy, the writer evokes pity and regret.

As expounded by Aristotle, the classical basis for tragedy is the character flaw: a single weakness such as pride or envy in an otherwise empathatic character which leads inevitably to downfall or destruction ([Muller 1956]). From his omnipotent third-person viewpoint, the reader can recognize the character flaw and see how it leads inevitably to disaster. It is this knowledge, combined with the helplessness of the reader to affect events in the story, that makes the story events "tragic" and leads to the reader's feelings of pity and regret.

Since Aristotle, the tragic motif has been developed and expanded in many ways. [Muller 1956] follows the development of tragedy through the ages with sections such as "Greek Tragedy", "Elizabethan Tragedy", "Neo-Classical Tragedy" and "Modern Tragedy". In developing MINSTREL, we have focused on classical tragedy as outlined by Aristotle. As the basis and precursor of all tragedy, classical tragedy has a rich usage and history that makes it an excellent vehicle for demonstrating MINSTREL's use of dramatic writing techniques to improve a story's literary quality.

8.3.1 Tragedy in *The Mistaken Knight*

One of the stories MINSTREL creates which has a tragic element is "The Mistaken Knight". In this story, a knight accidently kills the princess he loves:

The Vengeful Princess

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 moved 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. Grun-
feld moved towards the woods. Grunfeld was near the woods.
Grunfeld fought a dragon. The dragon died. Jennifer wanted to
live. Jennifer tried to drink a magic potion but failed.
Grunfeld was filled with grief.

          Moral: Deception is a weapon difficult to aim.
```

The tragedy in this story arises from Jennifer's character flaw: temper. She seeks revenge on another princess for a trivial reason. This leads inevitably to her downfall, an accidental death at the hands of Grunfeld. MINSTREL, recognizing the element of tragedy in this story, increases the impact of the tragedy by making Jennifer and Grunfeld lovers. The fact that Jennifer's tragic flaw leads to her death at the hands of someone who loves her intensifies the tragic aspect of this story.

One interesting aspect of this story is that the initial tragedy arises independent of the theme of the story. The theme of this story is to avoid using deception plans, and this theme can be illustrated without any element of tragedy, as for instance:

> Jennifer decided to scare her little brother by playing dead. She poured ketchup on herself and laid on the floor. But it was her mother who came into the room and was scared. Later, Jennifer was punished for her prank.

But in "The Mistaken Knight", MINSTREL's instantiation of the theme leads inadvertantly to a tragic situation. MINSTREL doesn't intentionally instantiate Jennifer's motivation for deception as a trivial affront. But once instantiated in this way, MINSTREL recognizes the elements of tragedy and take steps to further develop this aspect of the story. Thus MINSTREL is able to react to and capitalize on a fortuitous aspect of the story it is telling.

This demonstrates once again how an author with a variety of goals, some of them active (such as the goal to tell a story about a particular theme) and some of them opportunistic (such as the goal to improve the tragic aspect of a story which contains the seeds of tragedy), can create stories with unexpected layers of meaning and complexity.

8.3.2 When is Tragedy Appropriate?

A tragedy in the classical form involves a character with a tragic flaw who suffers a downfall as a direct, but unintended, result of the tragic flaw. To recognize this situation in a story, MINSTREL looks for three elements:

(1) A character suffers a goal failure.
(2) The goal failure is the result of unanticipated side-effects of the character's own plans.
(3) The plans are motivated by a character flaw.

If all three of these elements are found, then MINSTREL recognizes the tragic element in the story and creates a goal to augment the tragedy. MINSTREL's author-level plan to recognize tragedy is shown in Figure 8.4.

Name:	ALP:Check-Scene-For-Tragedy
Goal Type:	&Check-Scene-For-Tragedy
Object:	*AL-OBJ*
Test:	1. Is *AL-OBJ* a thwarted &P-HEALTH or &C-HEALTH goal?
	2. Is *AL-OBJ* thwarted by an unintended side-effect of an action by the actor of the goal?
	3. Does the actor of *AL-OBJ* have an explicitly marked character flaw? Or does he perform a socially unacceptable action motivated by a character trait?
Body:	1. Create an author-level goal to &Add-Tragedy-to-Scene to *AL-OBJ*.

Figure 8.4 ALP:Check-Scene-For-Tragedy

There are two interesting features of this plan.

First, note that in tragedy as in suspense, MINSTREL limits itself to important goals, so as to avoid trying to create a tragedy concerning a small, meaningless goal. Not only would tragedy revolving around an unimportant goal be ineffective, it would likely distract from the theme of the story, rather than augment the theme as the author desires.

Second, this plan requires determining if a character has a "character flaw". The idea and implementation of character flaws in MINSTREL is a topic with several interesting facets.

Character flaws are personality traits such as greed, pride, or envy which cause a character to act outside the bounds of acceptable behavior. For instance, a greedy character uses normally unacceptable plans to achieve money, such as theft and cheating. Character flaws can be contrasted to other character traits, which cause characters to act in unusual but acceptable ways. For instance, a compassionate character might give a large sum of money to a beggar, an unusual but socially acceptable act.

There are two ways MINSTREL can determine if a character has a character flaw.

First, the flaw can be marked explicitly. In stories based upon the theme PAT:Pride ("Pride goes before a fall"), the main character is explicitly marked as having a character flaw (pride).

Second, the character flaw can be implicit in the character's actions. If a character is motivated by a character trait to use socially unacceptable plans to achieve his goals, that character trait can be recognized as a flaw. For example, MINSTREL can recognize that a character who is motivated by "greed" to cheat and steal has a character flaw. Whether or not MINSTREL knows anything further about "greed", it can recognize that in this case "greed" is a character flaw.

Currently, MINSTREL recognizes socially unacceptable plans as those which cause goal fail-

ures for other characters. For example, cheating and stealing are both recognized as socially unacceptable because they cause goal failures (failure to protect one's money) in other characters when used. Compare this to selling, where both characters involved have goal successes. Although this definition of socially acceptability works well for the King Arthur domain, more complete models of the valuation of goals and plans have been defined (see for example [Reeves 1991]).

In summary, then, MINSTREL recognizes a tragic situation when a character with a character flaw performs an action which has unintended bad results for the character.

8.3.3 Writing Techniques for Building Tragedy

There are a variety of common patterns in classical tragedy which build upon the basic element of the flawed character. One such is the "mythical hero", in which the flawed character is a hero who arises from humble or unknown origins to ascend to kingship, only to die because of a tragic flaw, a pattern which is present in stories from Oedipus Rex to Moses to Siegfried to King Arthur to Robin Hood. Another is the "fortunate fall", in which the tragic character's fall leads to some greater good. This is nowhere more evident than in the story of Adam, whose fall into sin sets the stage for the coming of Christ. Another common pattern, present also in Oedipus Rex, is "destroying what one loves", in which the flawed character not only causes his own downfall, but unwittingly also destroys or hurts someone he loves. It is this last technique that MINSTREL implements.

"Destroying what one loves" increases the tragic effect of a story by sharing the tragic character's downfall with a character he loves. The reader's sense of justice is offended when an innocent character suffers, and is even more outraged when a loved character suffers at the hands of the one he loves, for shouldn't love be a protection against tragedy and injustice? Thus "destroying what one loves" increases the tragic effect of a story by combining the downfall of the tragic character with what is seen by the reader as the unfair loss of an innocent character.

8.3.3.1 ALP:Add-Tragedy-Via-Loved-One

MINSTREL's author-level plan for involving a loved one in the downfall of a tragic character is limited. It applies only if the tragic character dies inadvertantly at the hands of another character. If this is true, then ALP:Add-Tragedy-Via-Loved-One increases the tragedy of this event by making the flawed character and the character responsible for the downfall lovers. ALP:Add-Tragedy-Via-Loved-One is illustrated in Figure 8.5.

Two things should be noted about ALP:Add-Tragedy-Via-Loved-One.

First, ALP:Add-Tragedy-Via-Loved-One requires that the death of the tragic character at the hands of the loved one be inadvertant. This avoids the paradoxical situation where a character knowingly kills someone he loves. In the hands of skilled writer, this paradox can be resolved,

Name: ALP:Add-Tragedy-Via-Loved-One
Goal Type: &Add-Tragedy-To-Scene
Object: *AL-OBJ*
Test: Use if the flawed character's downfall is his death, and the death was caused by another character.
Body: 1. Determine the character who caused the death of the flawed character.
2. Create a scene that establishes that these two characters loved each other:

a. Create an uninstantiated state schema.

b. Set the Type feature of the new state to &Affect.

c. Set the To feature to the flawed character.

d. Set the Object feature to the other character.

e. Set the Value feature to &Pos (positive).

f. Set the Scale feature to &Strong.

g. Add the new state to the establishing scenes of the story.

3. Create a scene in which the character who causes the death of the flawed character, and who loved the flawed character, grieves over the loss:

a. Create an uninstantiated state schema.

b. Set the Type feature of the new state to &Affect.

c. Set the Object feature to the other character.

d. Set the Value feature to &Neg (positive).

e. Set the Scale feature to &Strong.

f. Connect the new state schema to the death of the flawed character by a &reaction link.

Figure 8.5 ALP:Add-Tragedy-Via-Loved-One

and, indeed, form the basis of an even more powerful tragedy. But that is beyond the scope of MINSTREL, and so ALP:Add-Tragedy-Via-Loved-One avoids that situation.

Second, the version of "destroying what one loves" that ALP:Add-Tragedy-Via-Loved-One creates has a layered complexity. The tragic character's death does indeed destroy someone he loves, because the loved character is destroyed by the knowledge that he has killed someone he loved. But on a second level, that destruction rebounds back to the tragic character, because he is literally destroyed by the loved character. And so ALP:Add-Tragedy-Via-Loved-One creates

a complex situation in which the tragedy and destruction are intertwined on several levels.

8.4 Characterization

The art of creating and portrayal of convincing characters is called *characterization*. Poor characterization in a story causes critics to speak of "cardboard" or "one-dimensional" characters: characters whose personalities are very simple and shallow, and who act on the simplest and most obvious of motivations. To portray a convincing, deep character, the author must both create a character with the aspects of a real person and show how those aspects impact the character's life.

8.4.1 Characterization in *The Proud Knight*

Characterization is particularly important in stories which turn upon a character trait. For a story of this sort to be successful, the author must be able to portray and establish the character trait in sufficient detail to justify the story development involving the trait.

The previous section showed how a character flaw drove the development of tragedy in *The Mistaken Knight*. Another story MINSTREL tells which is driven by a character flaw is *The Proud Knight*. In this story, Lancelot is a proud man whose hot temper eventually leads him to a hasty killing. In the early part of this story, MINSTREL establishes Lancelot's hot temper:

```
                    The Proud Knight

    It was the spring of 1089, and a knight named Lancelot
returned to Camelot from elsewhere. Lancelot was hot-
tempered. Once, Lancelot lost a joust. Lancelot wanted to
destroy his sword. Lancelot struck his sword. His sword was
destroyed...
```

MINSTREL establishes Lancelot's hot temper in two ways. First, MINSTREL simply states Lancelot's character trait: "Lancelot was hot-tempered." Secondly, MINSTREL invents a story episode which illustrates Lancelot's hot temper and shows how it leads him to make hasty and irrational decisions.

8.4.2 When Should Characterization be Used?

At first thought, it may seem like an author should always try to characterize his story characters as fully as possible. But that is not the case. There are several cases where characterization is not only unnecessary, but harmful.

For instance, it often harms a story if a minor character is given a full characterization. By spending time and detail on a character, the author creates in the reader an expectation that this character will be important. The reader, like the author, has a rough heuristic that the time and

detail spent upon a portion of a story reflects its importance to the story. To go against this expectation creates "red herrings" which weaken the story. Skilled mystery writers sometimes take advantage of this bit of psychology to mislead the reader with false clues, but for less talented authors (like MINSTREL) it is best to avoid creating red herrings.

Characterization is also misplaced if it emphasizes the wrong aspect of the character. For example, if, in the beginning of *The Proud Knight*, MINSTREL had developed Lancelot as a very handsome character, Lancelot's later hot-tempered actions would have been inexplicable and unfounded. Thus characterization simply for the sake of characterization can have a detrimental effect on a story.

To avoid these types of problems, MINSTREL only tries to create characterization when a character has an explicitly marked character trait that is part of the story theme. In this way MINSTREL is assured that (1) the character is a major character, because he is part of the story theme, and (2) the character trait is important to the story development, because it is explicitly marked as necessary to the story theme. MINSTREL's author-level plan for determining when to apply characterization is shown in Figure 8.6.

Name:	ALP:Check-Story-For-Characterization
Goal Type:	&Check-Story-For-Characterization
Object:	*AL-OBJ*
Test:	Is the story based upon a theme?
Body:	1. Look at all the characters involved in the theme. If any of them have an explicitly marked character trait, create an author-level goal to &Add-Characterization for that character trait.

Figure 8.6 ALP:Check-Story-For-Characterization

One limitation of MINSTREL's current implementation of characterization is that, unlike the similar heuristic for building tragedy in a story, it cannot detect character traits unless they are explicitly marked in the story theme.

There are two reasons that ALP:Check-Story-for-Characterization is limited to finding explicitly marked character traits.

First, finding character traits in general is more difficult than finding character flaws. ALP:Check-Scene-for-Tragedy uses a simple definition of character flaws (using plans that cause goal failures for others) that is not sufficient to detect many other kinds of character traits (such as beauty, kindness, indecisiveness, etc.). The general problem of examining a character's behavior and determining whether or not he is exhibiting a consistent character trait, and if so, what trait, is quite difficult.

Second, for characterization, MINSTREL must be able to manipulate the character traits it detects. To build tragedy, MINSTREL needs only to detect character flaws as one element of a tragic situation. But for characterization, MINSTREL must be able to both detect character traits *and* manipulate them, as in *The Proud Knight*, where MINSTREL detects that Lancelot

is hot-tempered and then creates an anecdote to illustrate that character trait. Detecting a character trait and understanding it well enough to create further examples of it are two different tasks, and the latter is much more difficult.

For these reasons, MINSTREL is currently limited to detecting explicitly marked character traits.

8.4.3 Developing Characterization

Edgar Roberts, a professor of literature at City University of New York, writing about the role of characterization in literature, identifies four ways by which an author can indicate character to the reader:

1. By what the character himself says (and thinks, from the author's third-person omniscient point of view).
2. By what the character does.
3. By what other characters say about him.
4. By what the author says about him, speaking as either the storyteller or an observer of the action.[1]

MINSTREL has implemented plans for characterization based on (2) and (4).

8.4.3.1 ALP:Add-Characterization-Statement

Of the characterization methods that Roberts identifies, author statement of a character trait is perhaps the simplest. In this method, the author simply states, third person, that a character possesses a particular trait, as MINSTREL does in *The Proud Knight*:

```
...Lancelot was hot-tempered...
```

ALP:Add-Characterization-Statement is the author-level plan that achieves this. ALP:Add-Characterization-Statement creates the character trait as an establishing scene in the story. Later, when the story is generated as English, this scene is generated as a third-person statement of the character trait. ALP:Add-Characterization-Statement is illustrated in Figure 8.7.

Details on how MINSTREL generates English from a conceptual story structure can be found in Chapter 10.

1. [Roberts 1977], page 55-56.

Name:	ALP:Add-Characterization-Statement
Goal Type:	&Add-Characterization
Object:	*AL-OBJ*
Test:	None.
Body:	1. Copy the character trait into the establishing scenes for the story being told.

Figure 8.7 ALP:Add-Characterization-Statement

8.4.3.2 ALP:Add-Characterization-Example

A more subtle method of characterization is to have the character perform actions motivated by or predicated upon the character trait to be developed. Unlike author statement, this method of characterization does not baldly state the underlying character trait. Rather, it requires that the reader deduce from the character's actions the underlying character trait. Although this is more subtle, it can also be more effective, because it gives the reader a concrete example of the character's behavior.

In *The Proud Knight*, this method is used to illustrate Lancelot's hot temper:

> ...Once, Lancelot lost a joust. Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed...

By showing how Lancelot's hot temper affects his behavior, this method makes the characterization more immediate to the reader.

In MINSTREL, this method of characterization is implemented by the author-level plan ALP:Add-Characterization-Example. One might expect this plan to be much more complicated than the simple plan used to implement characterization through author statement. In fact, however, Add-Characterization-Example is quite simple. ALP:Add-Characterization-Example is illustrated in Figure 8.8.

ALP:Add-Characterization-Example is simple because MINSTREL can use author-level planning recursively to accomplish the task of inventing an example. All that is needed for ALP:Add-Characterization-Example to do is to specify the important features of the example, i.e., that the goal be motivated by the character trait. The rest is simply instantiation, which can be accomplished by MINSTREL's various author-level plans for instantiation.

ALP:Add-Characterization-Example is another example of how embedding creativity at the lowest level of cognitive processes permits other, higher-level processes to make use of creativity in a simple and flexible manner.

Name: ALP:Add-Characterization-Example
Goal Type: &Add-Characterization
Object: *AL-OBJ*
Test: None.
Body: 1. Create a new, uninstantiated scene in which the character trait motivates a goal:

a. Make a new, uninstantiated goal schema.

b. Set the Actor feature of the goal schema to the character being characterized.

c. Connect the new goal schema to the character trait by a &motivates link.

2. Use author-level planning recursively to instantiate this goal.
3. If successful, add the instantiated goal to the story.

Figure 8.8 ALP:Add-Characterization-Example

8.5 Foreshadowing

Foreshadowing is a literary technique in which story incidents introduce, repeat, give casual allusion to or hint at later story incidents. The purpose of foreshadowing is to build a sense of inevitability in the later story events ([de Camp 1975]). By foreshadowing important story elements, the author avoids having those elements appear contrived, and in addition, creates a sense of unity in the story.

8.5.1 Foreshadowing in *The Mistaken Knight*

MINSTREL uses foreshadowing in The Mistaken Knight. The climax of this story turns upon Lancelot unexpectedly seeing the woman he loves kissing another knight. MINSTREL accomplishes this by having Lancelot's horse act willful – pulling him into the woods where he sees Andrea with Frederick. To prevent this from appearing contrived, and to help unify the story, MINSTREL uses the same type of scene earlier in the story, when Lancelot meets Andrea:

The Mistaken Knight

It was the spring of 1089, and a knight named Lancelot returned to Camelot from elsewhere. Lancelot was hot tempered. Once, Lancelot lost a joust. Because he was hot tempered, Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed.

One day, a lady of the court named Andrea wanted to have some berries. Andrea wanted to be near the woods. Andrea moved to

the woods. Andrea was at the woods. Andrea had some berries because Andrea picked some berries. *Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea.* Because Lancelot was near Andrea, Lancelot loved Andrea.

Some time later, Lancelot's horse moved Lancelot to the woods unintentionally, again causing him to be near Andrea. Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick...

One interesting aspect of foreshadowing is that it requires the author to create early story events *after* he has created later events. In this case, MINSTREL must create the scene in which Lancelot meets Andrea after the scene in which Lancelot kills Frederick, i.e., out of story order.

Like a human author, MINSTREL does not necessarily write a story "from beginning to end". Indeed, MINSTREL's representation of a story does not even possess a linear structure; one of the tasks of presenting the story to the reader is to determine the order in which to tell the story events. Instead, MINSTREL creates the events of the story in an order roughly determined by the importance of the events to the story. MINSTREL may create important story events near the end of the story (such as the resolution of the story theme) before it even considers lower priority events near the beginning of the story (such as scenes that introduce characters).

So there is no particular difficulty involved in creating earlier story events after later story events, as must happen in foreshadowing. If necessary, MINSTREL can "jump back" to an earlier part of the story and create a new story event, or if necessary, modify an existing story event.

8.5.2 Detecting Opportunities for Foreshadowing

As with other dramatic writing goals, the first step in using foreshadowing is to determine when it is appropriate. MINSTREL uses two criteria to determine when to apply foreshadowing.

First, MINSTREL applies foreshadowing only to story events which illustrate the story theme. As with other dramatic writing techniques, MINSTREL restricts foreshadowing to the story theme in order to focus the reader's attention on the story theme and to avoid creating "red herrings".

Second, MINSTREL applies foreshadowing only to unique story events. When deciding whether to foreshadow a story scene, MINSTREL compares that story scene to others in its memory. Only if the story scene has a unique feature or combination of features will MINSTREL choose to foreshadow the scene.

Because the purpose of foreshadowing is to create a sense of inevitability, foreshadowing is

best applied to events which *might* seem contrived. Story events which are commonplace (i.e., similar to those MINSTREL has read about in the past) are unlikely to be seen as contrived or unexpected, and consequently there is no need to foreshadow these types of events. Story events which are unusual (i.e., unlike anything MINSTREL has previously encountered) might well be seen as contrived, and so it is to these types of events that MINSTREL applies foreshadowing.

MINSTREL's model of contrivance is admittedly weak. Not everything uncommon is contrived. Furthermore, if MINSTREL uses a contrived scene in several stories, it will begin to see the scene as common and hence uncontrived. Despite these flaws, choosing to foreshadow uncommon scenes remains a useful heuristic for several reasons. First, it is a domain-independent heuristic. It does not rely on any specific knowledge of the King Arthur domain to determine what might seem contrived. Rather, it uses the author's own experience in the story domain. Second, a simple form of learning can improve this heuristic. As the author reads other stories in the story domain, his knowledge of what is common and acceptable will increase. Finally, foreshadowing is not strictly limited to eliminating contrivance. It also serves to create a feeling of inevitability and to create a unifying story element. Both of these functions can be served even if the story event chosen for foreshadowing is not contrived. For these reasons, this model of how scenes for foreshadowing are selected has proven adequate for the kinds of stories MINSTREL currently tells.

MINSTREL's author-level plan to determine when foreshadowing is appropriate is shown in Figure 8.9.

Name:	ALP:Check-Story-For-Foreshadowing
Goal Type:	&Check-Story-For-Foreshadowing
Object:	*AL-OBJ*
Test:	None.
Body:	1. Look at all the act/state combinations in the story. Use each combination as an index for recall. Save each combination that does not recall anything as a candidate. 2. Select one candidate at random from the available candidates. Create an author-level goal to foreshadow the candidate event.

Figure 8.9 ALP:Check-Story-For-Foreshadowing

ALP:Check-Story-For-Foreshadowing differs in two ways from the criteria described above.

First, ALP:Check-Story-For-Foreshadowing looks only for unique combinations of act and state schemas, rather than unique combinations of any schemas. In the King Arthur domain, stories are "action driven", meaning that the most important story events are typically direct character actions. Rather than examine all the combinations of schemas present in a story, ALP:Check-Story-For-Foreshadowing concentrates instead on character actions (act schemas) and their results (state schemas). This allows MINSTREL to find foreshadowing opportunities much more efficiently, at the cost of possibly overlooking a foreshadowing opportunity that does not involve a character action.

A second addition that ALP:Check-Story-For-Foreshadowing makes to the criteria listed above is that ALP:Check-Story-For-Foreshadowing selects only one of the suitable candidates for foreshadowing. The purpose of this is to limit the amount of foreshadowing used in a story. This is purely an aesthetic judgement, the kind of "rule of thumb" a beginning author might learn to avoid overusing literary techniques such as foreshadowing[1].

8.5.3 Creating Foreshadowing

Once the story scene to be foreshadowed has been selected, the next task is to create the foreshadowing scene. This is accomplished in two steps.

In the first step, the story is searched for points where a foreshadowing scene can be inserted. Because the scene to be foreshadowed consists of an act and a related state, MINSTREL looks through the remainder of the story for other act and state schemas similar to the foreshadowed schemas. If a match is found, it is saved as a possible candidate, and when all candidates are found, one is selected randomly as the point for foreshadowing.

For example, in *The Mistaken Knight*, the foreshadowed scene consists of Lancelot's horse taking an action which inadvertantly caused Lancelot to be in a location. To find a point in the story where this can be foreshadowed, MINSTREL looks through the story for schemas representing (1) Lancelot being in a location, or (2) Lancelot's horse taking an action. A story event that matches the former is found: Lancelot must be in the same location as the Princess Andrea in order to meet her and fall in love with her. This is the only story event that matches the foreshadowed scene, so it becomes the candidate location for foreshadowing.

The second step is to copy the foreshadowed scene into the foreshadowing location. Human authors foreshadow subtly, by repeating only one or two key elements of the foreshadowed scene. MINSTREL's technique is more unrefined: it copies as much of the foreshadowed scene as is usable into the foreshadowing scene.

In the *The Mistaken Knight*, this results in a scene in which Lancelot meets Andrea inadvertantly when his horse moves him unexpectedly into the woods. The entire act schema (representing Lancelot's horse moving unexpectedly into the woods) is copied from the foreshadowed scene. Consequently, Lancelot not only meets Andrea in the same unusual manner that he later discovers her with another man, but it also occurs in the same location.

MINSTREL's author-level plan which implements foreshadowing is shown in Figure 8.10.

1. In fact, opportunities for foreshadowing are rare in the stories that MINSTREL tells, and it is only in *The Mistaken Knight* that a suitable candidate is found and foreshadowing achieved, so this test could be dropped with no change in MINSTREL's behavior.

Name:	ALP:Add-Foreshadowing
Goal Type:	&Add-Foreshadowing
Object:	*AL-OBJ*
Test:	None.
Body:	1. Look at all the act and state schemas in the story. Save each schema that matches the schemas that represent the scene to be foreshadowed. When all candidates have been gathered, select one randomly. 2. If a candidate foreshadowing location was found, copy the foreshadowed scene into the foreshadowing location.

Figure 8.10 ALP:Add-Foreshadowing

8.6 Conclusions

As the examples in this chapter have shown, even the use of simple plans to achieve literary goals can improve the quality of the stories that MINSTREL tells. These stories suggest that one source of complexity, beauty, and quality in human storytelling is the interaction of a variety of opportunistic writing goals. This in turn suggests that opportunistic planning may be as important to a model of storytelling as active, directed planning.

In MINSTREL, opportunistic planning is implemented as a series of author-level goals that watch the developing story for opportunities to achieve various literary goals. In fact, MINSTREL was not designed with opportunistic planning in mind, and the support for opportunistic planning is rudimentary. Future storytelling programs will need more direct support for opportunistic planning, perhaps based on techniques similar to those used in expectation-based parsing ([Dyer 1982]).

CHAPTER 9
Consistency Goals in Storytelling

9.1 Introduction

Here's an inexplicable little story:

The Story of John

> One day, John passed a stranger on the street. John pulled a gun and shot him. The neighborhood priest saw John kill the man. The priest stopped his car and gave John a hug. John went home and gave his dog to his neighbor.

Although individually the events of this story are fine – even interesting – the story as a whole is confusing. The characters act almost randomly, pedestrians carry guns and no one seems to react properly to anything that happens. The story and the world it describes lack rhyme and reason. But the most curious thing about this story is that a little explanation changes its complexion entirely:

The Story of John, Redux

> One day, John was walking home from his night job as a security guard. John passed a stranger on the street. John saw a tattoo on the man's hand which he remembered from the night his parents died. This man had to die. John drew his gun and shot the man. The neighborhood priest was driving by and saw John kill the man. The priest stopped his car and looked at the man. Then he turned to his brother and hugged him. They both cried. Then John went home, gave his dog to his neighbor, and waited for the police to arrive.

The Story of John – or rather, the two stories of John – illustrate the importance of *story consistency*. Readers expect a story to be understandable. The characters should act in ways that are familiar and easily understood; the world should operate as the reader expects. When they don't, the author must explain why not. The *Story of John* is unacceptable not because of what happens, but because of what is left unexplained.

In MINSTREL, story consistency is maintained by a collection of opportunistic goals that watch the developing story for inconsistencies and correct them. For example, MINSTREL has a story consistency goal that watches for situations where a character uses a plan without first fulfilling all the preconditions of the plan. When this occurs, MINSTREL adds scenes that fulfill the preconditions. In **The Story of John**, this goal would notice that John uses a plan (shooting someone with a gun) without fulfilling one of the preconditions of that plan (possessing a gun). MINSTREL would then add scenes explaining how John came to have a gun (perhaps by revealing that John was a security guard). In this way, MINSTREL assures that the stories it tells have internal consistency and plausibility.

9.2 How Story Inconsistencies Arise

One of the difficulties of storytelling is that it requires the author to solve a "problem" (writing a story) with many constraints. The author must create a story which fulfills a story theme, has literary value, is internally consistent, and so on. One method for problem-solving under many constraints is to sub-divide the problem and attack each set of constraints separately and serially.

For example, the writer can decide to work first on the parts of the story that illustrate the theme, without worrying about literary value or story consistency. Then he might move on to increasing the dramatic tension of his story while ignoring the other elements, or the characterization of the main actors. By moving back and forth between his various goals and constraints, the author reduces a very difficult monolithic problem to a series of more reasonable smaller problems.

But one consequence of this technique is that the author must be careful to review his story in light of all his constraints. A scene created to fulfill one author-level goal might unwittingly violate another. If so, the problem must be corrected. This is how story inconsistencies arise, and this is why author-level goals and plans to detect and correct story inconsistencies are required.

As an example of how fulfilling one author-level goal can violate another, consider this scene, created by MINSTREL when telling *The Hermit and the Knight*:

```
Once upon a time, a knight named Cedric was wounded.  A hermit
named Bebe healed Cedric.  Cedric was grateful and vowed to re-
turn the favor.
```

This scene is created to illustrate the first part of the story theme for *The Hermit and the Knight*: "A favor earned is soon returned." Bebe's kindness in healing Cedric will be returned when Bebe is in need. But although it fulfills the author-level goal of illustrating the story theme, it violates several author-level goals of telling a consistent story. It lacks both an explanation of how Cedric came to be injured, and an explanation of how Bebe happened to find the injured knight in order to heal him.

MINSTREL's author-level consistency goals detect and correct these and other types of story inconsistencies.

9.3 Organization of Consistency Goals

MINSTREL's consistency goals fall into three categories:

- Planning Inconsistencies
- Story World Inconsistencies
- Emotional Inconsistencies

The first category deals with inconsistencies in the way characters plan. These goals assure that characters are using appropiate plans to achieve their goals, creating new goals in response to changes in the story world, and so on. The second category deals with the description of the story world. These goals make sure that things don't happen in the world without proper reason. The final category deals with how characters react emotionally to the events in their world. These goals provide characters with appropriate emotional reactions to goal failures and successes.

Every time MINSTREL creates a new story scene, it also creates goals to check the consistency in each of the above categories. So every part of the stories MINSTREL tells is checked for consistency. If a created scene is inconsistent, MINSTREL tries various author-level plans to correct the inconsistency. Altogether, MINSTREL has nine plans for maintaining story consistency. Most capture commonsense knowledge about how people plan and how the world works; a few are more complex.

9.4 Planning Inconsistencies

As mentioned above, the scenes that MINSTREL creates to fulfill thematic and literary goals often contain incomplete planning sequences. Consider, for example, this scene from *Lancelot and Frederick*:

```
One day, Lancelot wanted to kill Frederick.  Lancelot fought
with Frederick.  Frederick was dead.
```

In this scene, Lancelot's goals are unclear. Why should Lancelot want to kill Frederick? A character who decides "out of the blue" to kill another character violates the reader's expectations of how people behave, and results in an inconsistent story. The reader expects characters in the story world to have understandable goals and pursue those goals in expected ways. If they do not, explanation is required.

MINSTREL has author-level goals and plans to detect and correct several types of planning inconsistencies.

9.4.1 Unmotivated Goals

The difficulty with the above story fragment is that Lancelot's goal to kill Frederick is unmotivated. An unmotivated goal disrupts the story by causing the reader to ruminate over the unspoken reasons why the character would have this goal. A good author provides motivations for the goals that the characters in his story pursue.

In MINSTREL, there are three possible motivations for a goal:

(1) it can be a role goal,
(2) it can be a sub-goal of another goal, or

(3) it can be motivated by a state of the world.

Role goals are the "life goals" associated with character types such as knight and princess. These are goals that both the author and the reader assume that the character has based on his position and role in society. Role goals for knights, for example, include achieving status with their king and killing monsters. (To a certain extent, role goals define the character type: a character is a knight *because* he kills monsters, and tries to impress his king.) Role goals need no further motivation because the reader accepts them as typical and understandable goals for the character. The role goals used by MINSTREL are shown in Figure 9.1.

King	none[1]
Knight	Achieve Status (&A-Status) Destroy Monsters (&Destroy) Achieve Love (&A-Love)
Princess	Pick Berries (&Atrans) Achieve Love (&A-Love)
Hermit	Pick Berries (&Atrans)
Monster	Satisfy Hunger (&S-Hunger)

Figure 9.1 Role Goals in MINSTREL

Role goals are an artifact of the shared culture between the author and the reader. They work because the author and the reader share a common understanding about roles in the story genre. Of course, if the author and reader don't share a common culture, the goals of characters in the story may be incomprehensible. Lack of a shared culture is one reason modern readers often struggle over stories from earlier times. MINSTREL assumes that the reader has enough familiarity with stories in the King Arthur genre that he won't be puzzled to read of a knight killing a monster. But if the reader lacks that background he may well be confused.

Role goals are an example of a reader model. Who will read a story is an important consideration in how a story is told ([Smith 1982]). The difference between children's stories and stories for adults - in content, subject matter, style and explanation - is an excellent example of how a reader model affects the writing process.

MINSTREL assumes that the reader has the same knowledge that MINSTREL itself has, i.e., MINSTREL writes for itself as a reader. This is the simplest and easiest reader model. It require no special reasoning about what the reader knows or understands. Future work in computer models of storytelling may examine in more detail the role of the reader model in storytelling, but for MINSTREL, this simple reader model has proven effective and convenient.

A second possible motivation for a goal is for it to be a sub-goal of another goal. The sub-goals of a goal form a plan for achieving the goal. For instance, if Lancelot has a goal to become rich, his sub-goals might include to be where Frederick is, to kill Frederick, and then to take Freder-

1. MINSTREL uses creativity to invent role goals for Kings when necessary (see Chapter 15).

ick's money. Together, the sub-goals form a plan for achieving the goal of being rich. Sub-goals are explained by their purpose in achieving the super-goal.

The third possible motivation for a goal is a change in the state of the world. Characters react to changes in their world by creating new goals for themselves. For example, if Frederick drew his sword and charged at Lancelot, Lancelot might well react to this by creating the goal to kill Frederick. Lancelot's goal to kill Frederick is explained by the fact that Frederick is charging Lancelot with a drawn sword.

MINSTREL has one author-level plan for detecting unmotivated character goals, and two plans for correcting unmotivated character goals.

9.4.1.1 ALP:Check-Consistency-Goal

ALP:Check-Consistency-Goal determines if a goal is unmotivated by checking to see that it is not (1) a role goal, (2) a sub-goal or (3) motivated by an story event. If none of these conditions are true, ALP:Check-Consistency-Goal creates an author-level goal to create a motivation for this goal. ALP:Check-Consistency-Goal is illustrated in Figure 9.2.

Name:	ALP:Check-Consistency-Goal
Goal Type:	&Check-Consistency
Object:	*AL-OBJ*
Test:	1. Is *AL-OBJ* of type &Goal?
	2. Is *AL-OBJ* a role goal for the actor of the goal?
	3. Is *AL-OBJ* the sub-goal of another goal?
	4. Is *AL-OBJ* motivated by a state of the world?
Body:	1. If none of the tests are true, create an author-level goal to make *AL-OBJ* consistent (&Make-Consistent).
	2. Otherwise, the goal is consistent and no further work needs to be done.

Figure 9.2 ALP:Check-Consistency-Goal

9.4.1.2 ALP:Make-Consistent-Supergoal

ALP:Make-Consistent-Supergoal tries to make an unmotivated goal consistent by creating a super-goal, i.e., by making the unmotivated goal a step in a plan to achieve another goal. To do this, MINSTREL uses author-level planning recursively to try to recall or invent a plan that involves the unmotivated goal. ALP:Make-Consistent-Supergoal is illustrated in Figure 9.3.

One place ALP:Make-Consistent-Supergoal is used is in *The Hermit and the Knight*. This story is based upon the theme "Good deeds do not go unrewarded." In the first part of this story, a hermit named Cedric does a favor for a knight named Bebe when Bebe is injured fighting a dragon. Here is an initial version of the story:

Name:	ALP:Make-Consistent-Supergoal
Goal Type:	&Make-Consistent
Object:	*AL-OBJ*
Test:	1. Make *AL-OBJ* the sub-goal of a new, uninstantiated goal:
	a. Make a new, uninstantiated goal schema.
	b. Connect the new goal schema to *AL-OBJ* by a &sub-goal link.
	2. Use author-level planning recursively to try to instantiate the super-goal:
	a. Make a new author-level goal to &Instantiate the new goal schema.
	b. Call author-level planning to achieve the new author-level goal.
Body:	1. If instantiation succeeds, add the instantiated super-goal to the story.

Figure 9.3 ALP:Make-Consistent-Supergoal

```
Once upon a time, there was a hermit named Bebe and a knight
named Cedric.  One day, Cedric was in the woods when he was at-
tacked by a dragon and wounded.  Bebe wanted to be in the woods.
Bebe went to the woods.  Bebe healed Cedric.  Cedric was grate-
ful and vowed to return the favor.
```

The difficulty with this initial version of the story lies in Bebe's trip to the woods. The theme of this story requires that Bebe heal Cedric, and that in turn requires that Bebe be in the same location as Cedric. Consequently, Bebe goes to the woods. But although Bebe's goal to be in the woods serves an author purpose, it is unmotivated at the character level, resulting in an inconsistent story.

ALP:Make-Consistent-Supergoal corrects this inconsistency by inventing a reason Bebe to be in the woods. MINSTREL uses author-level planning recursively to find a goal for Bebe which has a sub-goal of being in the woods. To do this, MINSTREL makes an uninstantiated goal schema and connects it to the inconsistent goal by a &sub-goal link. The new goal schema is now the supergoal of the inconsistent goal, but it is uninstantiated.

To instantiate the supergoal, MINSTREL makes an author-level goal to instantiate the supergoal and calls author-level planning recursively to solve this new goal. At the recursive level, MINSTREL tries to instantiate the supergoal using its author-level plans for instantiation (see Chapter 7). In this case, MINSTREL uses ALP:General-Instatiate and recalls a previous story scene in which a Hermit went to the woods as a sub-goal of picking berries. This is then used to instantiate the supergoal in the current scene as "Bebe wanted to pick some berries".

If the recursive instantiation succeeds, the instantiated scene is added to the story as an explanation of the originally inconsistent scene:

Once upon a time, there was a hermit named Bebe and a knight named Cedric. One day, Cedric was in the woods when he was attacked by a dragon and wounded. Bebe, *who was in the woods picking berries*, healed Cedric. Cedric was grateful and vowed to return the favor.

Unlike Bebe's goal to be in the woods, Bebe's goal to pick berries is consistent. Picking berries is a role goal for hermits. It is something readers expect hermits to be doing, and no further explanation is needed.

9.4.1.3 ALP:Make-Consistent-Motivating-State

An alternate explanation for an unmotivated goal is a motivating state. Goals can arise as characters react to the state of the world. Consider this fragment from *Lancelot and Frederick*:

Lancelot believed that Andrea loved Frederick. Lancelot wanted Andrea to love him. But Andrea loved Frederick. Lancelot hated Frederick... Lancelot wanted to kill Frederick.

This fragment contains an unmotivated goal: Lancelot wants Andrea to love him. Most readers of this fragment "fix" this problem by assuming that Lancelot loves Andrea. This state of the world explains why Lancelot would want Andrea to love him and be upset if she did not. Unwittingly, the reader is following the same strategy as ALP:Make-Consistent-Motivating-State.

ALP:Make-Consistent-Motivating-State makes an unmotivated goal consistent by trying to invent or recall a state of the world that would motivate the goal. In the case of this example, ALP:Make-Consistent-Motivating-State finds the same solution most readers find - "Lancelot loves Andrea":

Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Lancelot hated Frederick... Lancelot wanted to kill Frederick.

ALP:Make-Consistent-Motivating-State tries to repair an unmotivated goal by creating a new state of the world that would motivate the goal, i.e., it tries to make the character's goal a reaction to something that happened in the story world. Like ALP:Make-Consistent-Supergoal, ALP:Make-Consistent-Motivating-State achieves this by using author-level planning recursively to recall or invent a state that can motivate the unmotivated goal. ALP:Make-Consistent-Motivating-State is illustrated in Figure 9.4.

ALP:Make-Consistent-Motivating-State creates an uninstantiated state schema which it uses as a "placeholder" motivation for the unmotivated goal. Author-level planning is then used recursively to try to fill in the placeholder. At the recursive-level, author-level plans for instantiation (see Chapter 7) are used to fill in the state schema. If instantiation succeeds, the instantiated state

Name: ALP:Make-Consistent-Motivating-State
Goal Type: &Make-Consistent
Object: *AL-OBJ*
Test:
1. Create an uninstantiated motivating state:
   a. Make a new, uninstatiated state schema.
   b. Connect the new state to the unmotivated goal by a &motivates link. (The new state represents an unknown motivation for the goal.)
2. Use author-level planning recursively to try to instantiate the new state:
   a. Make a new author-level goal to &Instantiate the new state schema.
   b. Call author-level planning to achieve the new author-level goal.

Body:
1. If instantiation succeeds, add the instantiated state to the story.

Figure 9.4 ALP:Make-Consistent-Motivating-State

can be used to motivate the inconsistent goal.

9.4.2 Motivating States

Just as a state of the world *can* be used to motivate a goal, so some states *should* motivate goals. Changes in the world - especially ones which affect important character goals - should motivate new goals in the characters they affect. A world in which characters did not react to their environment would be strange indeed:

> John was walking to work. He saw a thousand dollar bill on the sidewalk. John kept walking to work. John walked by the bank just as some robbers came out. John was hit by a stray bullet. John kept walking to work...

MINSTREL's initial solution to these problems was to use an author-level plan very specific to the King Arthur story world. In the King Arthur story world, the primary story events that characters must react to are dangers to their health. This knowledge was captured in an author-level plan called ALP:Make-Consistent-P-Health. Later, a more general author-level plan which used episodic memory to reason about how states motivate goals was created.

9.4.2.1 ALP:Make-Consistent-P-Health

ALP:Make-Consistent-P-Health is an author-level plan that creates a goal for a character to protect his health whenever he is injured, killed, or his life threatened. This assures that characters will react properly to life-threatening situations. ALP:Make-Consistent-P-Health is shown in Figure 9.5.

Name: ALP:Make-Consistent-P-Health
Goal Type: &Make-Consistent
Object: *AL-OBJ*
Test: 1. *AL-OBJ* is a state which represents a character being injured, killed, possibly as an outcome of an intended action by another character.
Body: 1. Create a goal for the character being injured to protect his health:

a. Make a new, uninstantiated goal schema.

b. Set the Actor feature of the new goal schema to the character who was injured.

c. Set the Type feature to &P-Health ("protect one's health").

d. Connect the new goal to the state representing the injury by a &motivates link (i.e., being injured motivates a character to have the goal to protect his health).

Figure 9.5 ALP:Make-Consistent-P-Health

The effect of ALP:Make-Consistent-P-Health can be seen in stories such as *The Lady or the Dragon*. In this story, Princess Jennifer has used a magic potion to change herself into a dragon and is subsequently attacked by Grunfeld:

```
...Grunfeld fought a dragon.  The dragon died.  Jennifer wanted
to live.  Jennifer tried to drink a magic potion to save her
life but failed.
```

In this story, ALP:Make-Consistent-P-Health creates the scene in which Jennifer, when attacked by Grunfeld, has the goal to save herself. (Subsequently, Jennifer attempts to achieve that goal by drinking another magic potion.) Without ALP:Make-Consistent-P-Health, Jennifer would have placidly accepted her death.

9.4.2.2 ALP:General-Motivating-State

Although ALP:Make-Consistent-P-Health is effective, it is very specific to the King Arthur domain. It is also quite inflexible. ALP:Make-Consistent-P-Health can make a character react correctly to having his life threatened, but not make him react correctly to finding a pot of gold. To generalize ALP:Make-Consistent-P-Health and add the ability to learn new motivating states, a new author-level plan was created which uses episodic memory to recognize motivating states.

The idea behind ALP:General-Motivating-State is to use episodic memory to determine if a state can motivate a goal. To do this, ALP:General-Motivating-State uses the state as an index to imaginative memory. If it can recall or invent a past situation in which a similar state motivated a goal, then that information can be used to improve the consistency of the current story.

For example, suppose that MINSTREL is telling a story in which a knight finds a pot of gold. ALP:General-Motivating-State uses that state as an index for recall, and remembers this story fragment from a story that MINSTREL had previously read:

> Joshua was wandering through the desert when he found a hoard of silver and jewels. Joshua wanted to be rich, so he took the hoard.

In this story fragment, finding something valuable motivates a character to have a goal of being rich and accomplishing that by taking the valuable item. This knowledge can then be applied to the current story by making the pot of gold motivate the knight to want to be rich.

Because ALP:General-Motivating-State is driven by episodic memory, it can adapt to different story genres and change as the author learns more about a story genre. Thus ALP:General-Motivating-State is flexible in ways that ALP:Make-Consistent-P-Health was not.

However, ALP:General-Motivating-State has one problem that ALP:Make-Consistent-P-Health did not. Just because a state *could* motivate a goal does not necessarily mean that it *should*. For example, seeing a beautiful princess can cause a knight to fall in love, but not every knight should fall in love with every princess he sees. The purpose of ALP:General-Motivating-State is to maintain story consistency. Rather than add every possible motivated goal, it should add only those without which the story would be inconsistent.

Determining this, however, is difficult. Ultimately, the author must rely on his acquired knowledge of the story genre to know what goals are important to each type of character. This can be modeled by semantic knowledge of important character goals, much in the same way that role goals were modeled. For the King Arthur domain, MINSTREL assumes that only protection goals (i.e., protect one's health, protect one's possessions, protect one's status, etc.) are important enough to warrant inclusion in the story purely for consistency reasons. ALP:General-Motivating-State is illustrated in Figure 9.6.

Like ALP:Make-Consistent-Supergoal and ALP:Consistent-Motivating-State, ALP:General-Motivating-State uses author-level planning recursively to find knowledge it needs to make the story consistent. The power of this technique is that it permits MINSTREL to apply all of the knowledge it has about instantiation towards achieving a different type of author-level goal: building consistency.

Name:	ALP:General-Motivating-State
Goal Type:	&Make-Consistent
Object:	*AL-OBJ*
Test:	1. *AL-OBJ* is a state which does not motivate a goal.
Body:	1. Create an uninstantiated goal motivated by *AL-OBJ*:
	a. Make a new, uninstantiated goal schema.
	b. Connect the original state (*AL-OBJ*) to the new goal schema by a &motivates link (i.e., the original state motivates the new, uninstantiated goal).
	2. Use author-level planning recursively to instantiate the uninstantiated goal:
	a. Make a new author-level goal to &Instantiate the new goal schema.
	b. Call author-level planning to achieve the new author-level goal.
	3. If instantiation succeeds and the instantiated goal is a protection goal, then add the new goal to the current story.

Figure 9.6 ALP:General-Motivating-State

9.4.3 Missing Preconditions

A precondition for an action is a state of the world that must be true before the action can be performed. For example, a precondition for fighting someone is to be colocated with that person. Similarly, possessing something is a precondition for giving it away.

Inconsistency arises if an action is performed without first achieving all of its preconditions. To prevent this type of inconsistency from appearing in its stories, MINSTREL has an author-level plan to detect and correct missing preconditions.

9.4.3.1 ALP:Check-Act-Preconds

ALP:Check-Act-Preconds is an author-level plan that examines each act schema in a story for missing preconditions, and creates the missing preconditions. ALP:Check-Act-Preconds is shown in Figure 9.7.

ALP:Check-Consistent-Precond relies upon semantic knowledge of what the necessary preconditions are for a given action. While this knowledge could be deduced from episodic memory by recalling past uses of an action, the assumption is that knowledge about preconditions is used frequently enough and changes infrequently enough that it is compiled into a more convenient form (i.e., semantic memory). MINSTREL's knowledge of preconditions is shown in Figure 9.8. ALP:Check-Consistent-Precond operates by looking up the preconditions needed for an action in

Name: ALP:Check-Act-Preconds Goal
Type: &Make-Consistent
Object: *AL-OBJ*
Test: 1. Is *AL-OBJ* an act schema?
Body: 1. Determine what the preconditions for this type of action are. (The preconditions of an action is a list of states that must be true before the action can occur.)
2. For each precondition, search the story to see if the precondition is already true.
3. For each precondition that is not true:

a. Make a new, uninstantiated state schema.

b. Copy the current precondition into the new state schema.

c. Connect the new state schema to the action by a &precond link.

Figure 9.7 ALP:Check-Consistent-Precond

Identifier	Meaning	Preconditions
&M-Heal	Healing a character	Healer in same location as patient.
&M-Fight	Fighting someone	Combatants in same location.
&Mtrans	Witnessing something	Witness in same location as witnessed event.
&Ptrans	Physical movement	Physical control of object.
&Atrans	Change of ownership	Object to be transferred must be possessed.

Figure 9.8 Table of Preconditions

the table shown in Figure 9.8. If a precondition for an action is missing from the story, ALP::Check-Consistent-Precond creates a state of the world that fulfills the precondition and adds it to the story.

An example of the use of ALP:Check-Consistent-Precond occurs in the story *The Proud Knight*. In this story, a knight is warned that fighting a dragon will result in doom. Being proud, he fights the dragon nonetheless, and is hurt in the process. The fight scene as MINSTREL initially creates it is:

```
Grunfeld fought with the dragon in the woods.  The dragon was
destroyed but Grunfeld was hurt.
```

After this scene is created, ALP:Check-Act-Preconds notices that the precondition for fighting the dragon has not been met: Grunfeld is not in the same location as the dragon. ALP:Check-Act-Preconds creates a state that achieves this precondition and adds it to the story:

```
Grunfeld was near the dragon.  Grunfeld fought with the dragon
in the woods.  The dragon was destroyed but Grunfeld was hurt.
```

The preconditions for fighting the dragon have now been satisfied. Grunfeld and the dragon are now colocated.

Of course, this new addition to the story itself seems inconsistent. There's no explanation of how Grunfeld came to be near the dragon, or why he wanted to be there. This type of inconsistency - an inconsistent state of the world - and how to correct it, is discussed in the following section.

9.5 Story World Inconsistencies

Just as the reader expects characters to act in familiar ways, he also expects the world to act in familiar ways. Unusual events should not happen at random or without explanation, and the ways in which the characters interact with the world should not violate the reader's common-sense understanding of the world.

In MINSTREL's stories, the world is represented by state schemas. Each state schema represents a fact or "state" of the world. Like goals, most states require an explanation - a reason why they exist. In MINSTREL, there are two possible explanations for a state:

(1) it can be a default state of the world, or
(2) it can be the result of a character action.

Default states are analagous to role goals. They are states of the world that both the author and the reader accept as reasonable without further explanation. Default states represent any typical state of the world, such as the existence of an object or place. If the author assumes that a forest exists in the story world, the reader will not object, because the existence of forests is a commonplace state of the real world.

Story events which are default states require no further explanation, because the reader already accepts them as reasonable and consistent. As with role goals, default states depend upon the shared culture between the author and the reader. To limit MINSTREL's dependency upon this assumption, MINSTREL uses a very limited set of default states. The default states that MINSTREL recognizes are the existence of objects and locations. All other states must have an explicit explanation.

The second possible explanation for a state is that it results from a character's actions. The purpose of action is to change the state of the world. That a character's actions should change the world is both expected and consistent.

(A third possible explanation for a state is that it can be caused by an inanimate actor, such as "Nature" or gravity. Although this is a commonly used explanation in human storytelling, MINSTREL does not currently use inanimate actors to explain story events. Adding inanimate actors to MINSTREL would not, however, pose any difficulties.)

MINSTREL has three author-level plans for maintaining story world consistency. The first plan, ALP:Check-Consistency-State, determines whether or not a state is consistent. If it is not, the

state is made consistent by one of the other two author-level plans: ALP:Make-Consistent-State or ALP:Make-Consistent-Colocation.

9.5.1 ALP:Check-Consistency-State

ALP:Check-Consistency-State determines if a state is unexplained by checking to see if it is (1) a default state, or (2) explained by a character action. If neither of these conditions are true, ALP:Check-Consistency-State creates an author-level goal to create an explanation for this state. ALP:Check-Consistency-State is illustrated in Figure 9.9.

Name:	ALP:Check-Consistency-State
Goal Type:	&Check-Consistency
Object:	*AL-OBJ*
Test:	1. Is *AL-OBJ* a state schema?
	2. Does *AL-OBJ* represent the existence of an object or location? (I.e., is *AL-OBJ* a common state?)
	3. Is *AL-OBJ* the result of a character action?
Body:	1. If neither 2 nor 3 is true, then create an author-level goal to &Make-Consistent-State *AL-OBJ*.
	2. Otherwise, *AL-OBJ* is consistent.

Figure 9.9 ALP:Check-Consistency-State

9.5.2 ALP:Make-Consistent-State

ALP:Make-Consistent-State tries to make an unexplained state consistent by creating a character action which results in the state. To do this, MINSTREL uses author-level planning recursively to try to recall or invent an action that causes the state. ALP:Make-Consistent-State is illustrated in Figure 9.10.

To see how ALP:Make-Consistent-State works, let's return to the example used in describing ALP:Check-Consistent-Preconds. In that example, MINSTREL had started with a scene in which a knight fights a dragon:

```
Grunfeld fought with the dragon in the woods.  The dragon was
destroyed but Grunfeld was hurt.
```

At this point, ALP:Check-Act-Preconds noticed that a precondition for fighting the dragon had not been met: Grunfeld is not in the same location as the dragon. ALP:Check-Act-Preconds then created a state that achieves this precondition and added it to the story:

```
Grunfeld was near the dragon.  Grunfeld fought with the dragon
in the woods.  The dragon was destroyed but Grunfeld was hurt.
```

Name: ALP:Make-Consistent-State
Goal Type: &Make-Consistent
Object: *AL-OBJ*
Test:
1. Make *AL-OBJ* the result of a new, uninstantiated character action:
   a. Make a new, uninstantiated act schema.
   b. Connect the new act schema to the inconsistent state by an &intends link (i.e., the state is an intentional result of the action).
2. Use author-level planning recursively to try to instantiate the character action:
   a. Make a new author-level goal to &Instantiate the new act schema.
   b. Call author-level planning to achieve the new author-level goal.

Body:
1. If instantiation succeeds, add the instantiated action to the story.

Figure 9.10 ALP:Make-Consistent-State

Now ALP:Make-Consistent-State comes into play. Because the state created to fulfill the pre-condition is not a default state (the existence of an object or location), and not the result of a character action, ALP:Make-Consistent-State recognizes it as inconsistent. ALP:Make-Consistent-State is then used to make the state consistent.

ALP:Make-Consistent-State uses author-level planning to find an action that a knight can do which would result in the knight being in the woods. When it finds a suitable action it adds it to the story:

```
Grunfeld wanted to be near the dragon.  Grunfeld moved to the
dragon.  Grunfeld was near the dragon.  Grunfeld fought with the
dragon in the woods.  The dragon was destroyed but Grunfeld was
hurt.
```

The inconsistent state has now been explained by making it the result of a character action. Grunfeld being near the dragon is no longer puzzling. It is the direct result of Grunfeld's desire to be near the dragon, and his subsequent move in that direction.

But interestingly enough, the new explanation is itself inconsistent. Although Grunfeld's goal of being near the dragon explains his move towards the dragon, it lacks motivation. This inconsistency is detected by ALP:Check-Consistent-Goal. ALP:Make-Consistent-Supergoal corrects the problem by making the inconsistent goal a step in a larger plan:

```
Grunfeld wanted to kill the dragon.  Grunfeld wanted to be near
the dragon.  Grunfeld moved to the dragon.  Grunfeld was near
the dragon.  Grunfeld fought with the dragon in the woods.  The
dragon was destroyed but Grunfeld was hurt.
```

However, the new goal is still inconsistent! Grunfeld's goal to kill the dragon is neither a role goal for a knight nor motivated by a change in the world. So ALP:Make-Consistent-Supergoal is used again, making the inconsistent goal a sub-goal of a larger plan:

> Grunfeld wanted to impress the King. Grunfeld wanted to kill the dragon. Grunfeld wanted to be near the dragon. Grunfeld moved to the dragon. Grunfeld was near the dragon. Grunfeld fought with the dragon in the woods. The dragon was destroyed but Grunfeld was hurt.

Now (finally) the scene is consistent. Impressing the king is a role goal for knights and so needs no further explanation. As this example has illustrated, correcting the inconsistencies in a story scene may require several steps, and each correction may introduce inconsistencies of its own, which must also be corrected.

9.5.3 ALP:Make-Consistent-Colocation

MINSTREL's second author-level plan for making a state consistent is used when ALP:Make-Consistent-State fails, i.e., when MINSTREL cannot recall or invent an act that intends the inconsistent state. ALP:Make-Consistent-Colocation guesses that a state may occur because all the elements of the state come together in a location. In other words, ALP:Make-Consistent-Colocation explains states of the world as simply "happy coincidences".

For example, in telling *Richard and Lancelot*, MINSTREL creates the following story scene (see ALP:Make-Consistent-Motivating-State):

> Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Lancelot hated Frederick... Lancelot wanted to kill Frederick.

This scene is inconsistent because it lacks an explanation of why Lancelot loves Andrea. What's more, ALP:Make-Consistent-State fails to make the scene consistent, because MINSTREL knows of no action that results in one person loving another. So ALP:Make-Consistent-Colocation is applied. ALP:Make-Consistent-Colocation suggests that Lancelot may love Andrea simply because the two came together at some point:

> One day, a lady of the court named Andrea wanted to have some berries. Andrea wanted to be near the woods. Andrea moved to the woods. Andrea was at the woods. Andrea had some berries because Andrea picked some berries. *Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot loved Andrea.*

ALP:Make-Consistent-Colocation works for two reasons. First, humans understand that people

and the world are sometimes mysterious. The thought process of people and the causal processes of the world are complicated and often unfathomable. It *does* sometimes seem that people fall in love with each other simply because they happen to meet. Second, readers strive to make sense of what they read. A reader encountering a scene such as:

```
Lancelot met Andrea.  They fell in love.
```

will use his considerable knowledge of people and the world to construct explanations for these happenings. So in some cases, the author need not provide an explanation; the reader will invent one of his own.

There are, however, two difficulties with ALP:Make-Consistent-Colocation. The first is that a story filled with such happy coincidences would strain the reader's credulity. The second is that there are many states that *aren't* explainable by colocation, for example:

Lancelot met Andrea. They both turned into spiders.

Consequently, it is best to use ALP:Make-Consistent-Colocation only when other, better explanations cannot be found. For this reason, ALP:Make-Consistent-Colocation is a low-priority plan that is attempted only when ALP:Make-Consistent-State fails.

9.6 Emotional Inconsistencies

The final class of story inconsistencies that MINSTREL detects and corrects are emotional inconsistencies. Just as readers expect story characters to be rational, goal-seeking actors, they also expect story characters to have appropriate emotional reactions to events in their lives. Characters should be happy when they succeed and unhappy when they fail.

9.6.1 The Tone Plus Goal Situation Model of Emotions

MINSTREL uses a "tone plus goal situation" model of emotions based upon models developed in [Dyer 1982], [Dyer 1987] and [Oatley 1987]. There are two fundamental theoretical claims made by tone plus goal situation models of emotion.

The first claim is that emotions are caused by goal resolutions, i.e., the failure or success of a goal. The term "resolution" is used rather broadly here, because emotions are often caused by the *anticipated* resolution of a goal. We feel happy not only when we succeed at some goal, but also when something happens that suggests we will succeed.

In the case of self goals, the relationship between goal resolutions and the resulting emotions is specific: goal successes cause euphoric emotions and goal failures cause dysphoric emotions. That is to say, we feel happy when we succeed and sad when we fail. Further, the strength of emotion is related to the importance and difficulty of the goal resolved. Although both are goal successes, the birth of a first child is cause for more celebration than finding a lost comb. Emo-

tional reactions to the goal resolutions of other people are more problematic. We may feel happy, sad or indifferent, depending upon our relationship to that other person.

The second theoretical claim of the tone plus goal situation model is that all emotions can be represented as a basic tone (happiness, sadness, anxiety, anger and disgust in [Oatley 1987]; positive and negative in [Dyer 1987] and MINSTREL) coupled with an abstract goal situation. Consider, for example, the emotion "feeling grateful". When we say we are feeling grateful, we mean that we are happy towards someone because they have helped us in some way. In the tone plus goal situation model, this is represented as a positive (happy) basic tone caused by having a goal achieved by a third party. The claim of the tone plus goal situation model is that *all* emotions can be represented this way. Figure 9.11 shows a lexicon of emotions and their tone plus goal situation representations based upon a similar lexicon in [Dyer 1987].

Emotion	Character	Tone	Toward	Goal Situation
happy	x	pos		G(x) achieved
sad	x	neg		G(x) thwarted
grateful	x	pos	y	G(x) achieved by y
angry-at	x	pos	y	G(x) thwarted by y
scared	x	neg	y	G(x) thwarted by y, G(x) a health goal
proud	x	pos	y	G(y) achieved by x
guilty	x	neg	y	G(x) thwarted by y
gloating	x	pos		G(y) thwarted
envious	x	neg		G(y) achieved
felicitating	x	pos		G(y) achieved
condoling	x	neg		G(y) thwarted.

Figure 9.11 Sample Lexicon of Emotions

In addition to these fundamental claims, both [Dyer 1987] and [Oatley 1987] make further claims about the role of emotions in a generalized cognitive system, but inasmuch as these claims are not germane to MINSTREL and lack agreement, they will not be discussed here.

MINSTREL's tone and goal situation model is based primarily on [Dyer 1987]. Each emotion as represented as a state schema with the following slots:

1. TYPE - The type slot of state schemas which represent emotions is set to &AFFECT. &AFFECT is a symbol which acts only to identify the representation of emotions; it has no other meanings.

2. VALUE - The value slot contains &POS or &NEG to identify the basic tone of the emotion.

3. OBJECT - The character who is feeling the emotion.

4. TO - The character the emotion is directed at, if any.

5. SCALE - The strength of the emotion, either &STRONG, &NORMAL, or &WEAK.

State schemas which represent emotions may also have named links to other schemas:

1. &G-SITU - This link is a pointer to the goal situation which caused this emotion.

2. &MOTIVATES - Like any state, an emotion can be motivation for a goal, i.e., being angry at a character might be motivation for having the goal to harm them. The &MOTIVATES link of an emotion points to any goals the emotion has motivated.

3. &SUPERSEDES - An emotion can supersede another emotion.

Figure 9.12 shows how MINSTREL represents the situation "Andrea was grateful when Lancelot killed the dragon." Andrea's grateful emotion is captured as a state schema with a &G-SITU pointer to the representation of "Lancelot killed the dragon". Note that the representation includes Andrea's unmentioned goal of wanting the dragon to be killed.

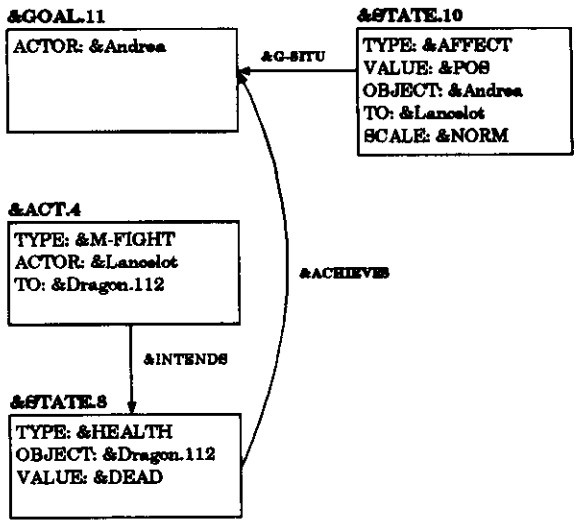


Figure 9.12 Representation of an Example Emotion

The following sections show how MINSTREL uses the two fundamental features of the tone plus goal situation model to detect and correct emotional inconsistencies in the stories it tells.

9.6.2 Maintaining Emotional Consistency

MINSTREL's primary concern with emotions is to maintain the emotional consistency of the story characters, i.e., assure that they have the proper emotional reactions to events. To do this requires (1) detecting when a character should feel an emotion, and (2) building the proper schema-based representation for the emotion[2]. The tone plus goal situation model of emotions has several features which make these tasks easy to achieve.

First, the relationship between goal resolutions and emotions provides a method to detect when characters should feel emotions. Whenever a goal is resolved, MINSTREL knows that the character who held the goal should feel either euphoric (if the goal was successful) or dysphoric (if the goal was unsuccessful). For example, if Richard thwarts Lancelot's goal of possessing the sword Excalibur by stealing it, then Richard should feel happy (because his goal to thwart Lancelot succeeded) and Lancelot should feel unhappy (because his goal failed).

Second, once the goal resolution has been found, creating the appropriate emotion is simple because the emotion is already implictly represented in the goal resolution:

- The resolution of the goal (success or failure) defines the basic tone (positive or negative).

- The goal, it's resolution, and related schemas are the "goal situation" part of the representation of the emotion.

- The character feeling the emotion is the actor of the resolved goal.

- The character toward whom the emotion should be directed is the actor who caused the goal resolution.

These four elements can be combined to create the schema-based representation of the proper emotion. In the case of Richard and Lancelot above, the emotion for Lancelot consists of a basic tone, negative, which is determined by the goal resolution (a failure), the causative goal situation (Lancelot's goal thwarted by Richard), and the character the emotion is directed towards (Richard, because he was the actor who caused the goal failure). Combined into a state schema, this is the representation for "Lancelot was angry at Richard".

Maintaining emotional story consistency is thus a matter of (1) detecting goal resolutions, and (2) building representations for the emotions caused by the goal resolutions using the information already present in the goal situation. The author-level plan which implements these tasks is called ALP:Check-Affects. (The problem of third-person reactions to goal resolutions is more difficult and is discussed below.)

---

2. In addition, MINSTREL must (1) make judgements about which emotions should be included in the story, i.e., the author may choose to express only strong emotions, and (2) decide how to express emotions in language. These tasks are discussed in Chapter 10, Presentation Goals in MINSTREL.

9.6.3 ALP:Check-Affects

The purpose of ALP:Check-Affects is to look at every goal resolution in a story (including goal resolutions of secondary characters and monsters) and determine whether the actor of the goal has an emotional reaction to the goal resolution. If he does not, then a state schema representing his reaction is created and added to the story. ALP:Check-Affects is shown in Figure 9.13.

Name: ALP:Check-Affects
Goal Type: &Check-Consistency
Object: *AL-OBJ*
Test: Is *AL-OBJ* a resolved goal schema that lacks an emotional reaction?
Body: 1. Create the state schema:

a. Make a new, uninstantiated state schema.

b. Set the Type feature to &Affect.

c. Set the Object feature to the actor of the resolved goal.

d. Connect the new state schema to the resolved goal by a &g-situ link. (The &g-situ link points from an affect to the *goal situ*ation that caused the emotion.)

2. Determine the strength of the emotion:

a. If *AL-OBJ* is an important goal (&P-HEALTH, &C-HEALTH, or &A-LOVE), then the strength of the emotion is &STRONG.

b. Otherwise, the strength is &NORMAL.

c. Set the Scale feature of the new state schema to the selected strength.

3. Determine the character the emotion is directed towards:

a. If *AL-OBJ* was achieved or thwarted by an intentional action, then the emotion is directed towards the actor of the action.

b. If *AL-OBJ* was achieved or thwarted by an unintentional action, then the emotion is directed towards the actor of the action, but the strength of the emotion is &WEAK.

c. Set the To feature of the state schema to the selected character.

Figure 9.13 ALP:Check-Affects

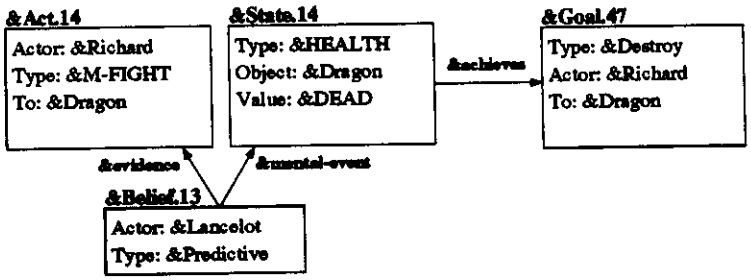
ALP:Check-Affects proceeds in three steps. First it creates the state schema that represents the emotional reaction to the goal resolution, filling in the character who is feeling the emotion, and the link to the goal resolution. Then it determines the strength of the emotion, and finally, the character towards whom the emotion is directed (if any).

ALP:Check-Affects uses knowledge about the relative importance of character goals to determine how strong the emotional reaction should be. Currently, MINSTREL uses a fixed scale of goal importance in which three goals, &P-HEALTH, &C-HEALTH and &A-LOVE are important enough to warrant a strong emotional reaction. These are the goals to, respectively, protect one's health, save one's life and achieve love. A more sophisticated approach would be to model goal priorities for each character (by using, for instance, goal trees [Carbonell 1978]) and use this information to determine the strength of emotional reactions. Currently, however, MINSTREL uses the fixed scale of goal importance outlined above.

The strength of emotion is also affected by how the goal was resolved. The third step of ALP:Check-Affects determines which character the emotion should be directed at by looking at how the goal was achieved or thwarted. If the goal is achieved or thwarted unintentionally, the strength of the goal is reduced to &WEAK. If Richard accidently breaks Lancelot's magic sword, Lancelot may still be angry at Richard, but that anger will be less than if Richard had broken the sword intentionally.

An interesting aspect of ALP:Check-Affects is that it also maintains emotional consistency for *anticipated* goal resolutions. To understand how this happens, it is necessary to describe how MINSTREL represents an anticipated goal resolution.

In MINSTREL, anticipated events are represented as predictive beliefs. For example, when Princess Andrea sees a dragon moving towards her, she has the predictive belief that the dragon will eat her, thwarting her goal of protecting her health (Figure 9.14). Andrea's belief (&Belief.13) has two parts: the evidence, which is the dragon moving towards her, and the event that she predicts, the dragon eating her.

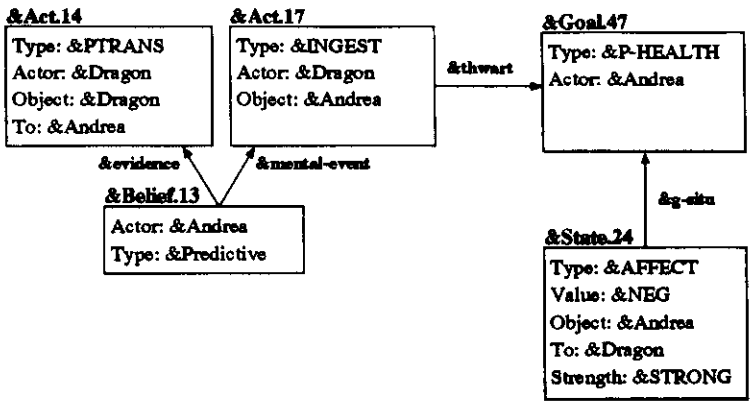



```
Lancelot seee Richard fighting a dragon and believes
that Richard will kill the dragon.
```

Figure 9.14 Representation of Anticipated Goal Resolution

What happens when the scene represented in Figure 9.14 appears in a story? The mental-event portion of Andrea's belief contains a goal resolution (Andrea's thwarted &P-HEALTH goal). This triggers ALP:Check-Affects, which builds an emotion based upon this goal resolution. Because there is nothing in ALP:Check-Affects to detect goal resolutions which are part of beliefs, it treats Andrea's anticipated goal resolution just as if it had actually occurred. The emotion

ALP:Check-Affects creates is shown in Figure 9.15.




```
Andrea saw a dragon moving towards her and believed that she
would be eaten.  Andrea was very scared of the dragon.
```

Figure 9.15 Anticipated Goal Resolution and Emotion

The emotion ALP:Check-Affects creates is "Andrea was very scared of the dragon." ALP:Check-Affects acts as if the anticipated goal resolution had actually occurred, creating the emotion that Andrea would feel[3] if she were eaten. This behavior captures our intuitive understanding of how people react to anticipated outcomes: they react as if the event anticipated had occurred. This phenomenon is an important part of daydreaming [Mueller 1987], and probably serves to reduce the effect of strong emotions by habituation.

Thus ALP:Check-Affects not only maintains the proper reactions to resolved goals, it also maintains the proper reactions to anticipated goal resolutions.

9.6.4 Third Person Emotions

To this point, our discussion of emotional consistency in MINSTREL has focused on the reactions of characters to the resolution of their personal goals. This task is relatively straightforward because the basic tone of the emotion is determined by whether the goal was achieved or thwarted. The same cannot be said, however, about third person reactions to goal resolutions. When Lancelot achieves a goal, the emotional reaction of Andrea depends upon her emotional relationship to Lancelot. To illustrate this point, consider the following story fragments:

> Andrea {loved, admired, was grateful to} Lancelot. When Andrea heard that Lancelot lost an important joust with Richard, she felt sad.
>
> Andrea {hated, was angry at, was envious of} Lancelot. When Andrea heard that

3. At least momentarily.

Lancelot lost an important joust with Richard, she felt happy.

Although Andrea had heard of Lancelot, she didn't know him. When Andrea heard that Lancelot lost an important joust with Richard, she was neither upset nor happy.

Andrea's reaction to Lancelot's goal failure is determined by her emotional relationship with Lancelot. If she has a positive relationship with Lancelot, then she shares empathically in his emotions. If she has a negative relationship with Lancelot, then she feels the opposite of his emotions. And if Andrea and Lancelot have no relationship, or a neutral one, then Andrea feels nothing in response to Lancelot's goal resolutions.

What's most interesting about this theory is that the emotional reaction of a third person to a goal resolution depends upon the emotions he is feeling toward the actor of the goal. This is all the more apparent when we realize that character relationships like "love" can be viewed as long-lasting emotions. Thus emotions beget emotions in kind. If someone we view positively succeeds at something, we feel happy for him. If someone causes him to fail, we are angry at that person ("The enemy of my friend is my enemy"). This cycle of emotions begetting emotions serves to reinforce emotional relationships by making positive relationships more positive and negative relationships more negative.

MINSTREL's author-level plan which captures this knowledge is called ALP:Check-Affects-Others[4].

9.6.5 ALP:Check-Affects-Others

The purpose of ALP:Check-Affects-Others is to look at each goal resolution in a story and determine whether any third party characters in the story should have a reaction to the goal resolution. As with ALP:Check-Affects, if an emotion is missing it is created. ALP:Check-Affects-Others is shown in Figure 9.16.

ALP:Check-Affects-Others adds one feature to the model of third-person emotional reactions. In ALP:Check-Affects-Others, third persons have empathic emotional reactions only to strong emotions. The idea here is that emotions felt empathically should be weakened in porportion to the strength of the empathy. Since MINSTREL has only one level of empathy, it arbitrarily reduces the strength of empathic emotions one level. A more detailed and robust model might have several levels of empathy (aquaintance, friend, comrade, love) and corresponding effects on empathic emotions.

4. Although described here in some detail, ALP:Check-Affects-Others was never implemented.

Name:	ALP:Check-Affects-Others Goal
Type:	&Check-Consistency
Object:	*AL-OBJ*
Test:	1. Is *AL-OBJ* a resolved goal schema? Does the actor of *AL-OBJ* have a &STRONG emotional reaction to *AL-OBJ*?
Body:	1. For each third-person character in the story, look through the story to see if there exists any emotional relations between the character and the actor of the goal *AL-OBJ*. 2. If a positive relationship exists, then create a normal strength emotional reaction to this goal resolution equivalent to the reaction of the actor of *AL-OBJ*. 3. If a negative relationship exists, then create a normal strength emotional reaction to this goal resolution opposite to the reaction of the actor of *AL-OBJ*.

Figure 9.16 ALP:Check-Affects-Others

CHAPTER 10
Presentation Goals in MINSTREL

10.1 Introduction

The primary focus of MINSTREL is on the processes an author uses to create stories. But an author must not only invent a story, he must also present the story to a reader. He must express the story in language, in a manner that is both pleasing and understandable.

This involves a number of tasks. At the very least, the author must be able to convert his internal representation of the story into language. But there is more to presenting a story than simply rattling off the scenes of the story at random.

To begin with, the author must order the scenes of his story into an intelligible whole. For simple narratives this may require no more than telling the scenes involving the main character in causal order. For more complicated stories, decisions about the ordering of each scene must be based on many factors.

Beyond simply ordering the story, human authors have many techniques they use to improve the presentation of their stories. They create story scenes for presentation purposes and use language in powerful ways to effect their author-level goals.

To tell a story in English, MINSTREL achieves three tasks:

1. *Create* story events to improve the presentation of the story.

2. *Order* the events in the story.

3. *Generate* the story events in English.

Although MINSTREL does not have the presentation expertise of a human author, these tasks are intended to be indicative of the kinds of processes and knowledge an author needs to present a story in language.

10.2 Presentation Story Scenes

The goal of presentation is to communicate the story to the reader. If the story was created to illustrate a theme, then the goal of presentation is to communicate that illustration lucidly and effectively to the reader. Primarily this goal is achieved by expressing the events of the story in language. But the author has other presentation tools available. The most important of these tools is the ability to create new story scenes to improve the presentation of the story.

If we view a story as a complicated web of interconnected characters, goals, plans and events, it should be apparent that one difficulty in telling a story is communicating the interconnections between the story pieces. One way this communication can be improved is by creating new story

scenes to guide the reader through the transitions from one story event to another. In technical writing, the reader is guided from topic to topic by "transition sentences"or "transition paragaphs". Authors of fiction use similar devices to help guide their readers through the story.

Of course, not all story transitions are difficult to understand. For instance, the transition between two causally-related story events is usually easy to comprehend:

```
Lancelot fought the dragon.  The dragon died.
```

The causal relationship between the two story events provides the reader with a framework to understand the scenes. No additional presentation is needed, because the reader expects and easily understands this type of transition. Other story transitions are more difficult, and the author can improve the presentation of his story by providing an explicit marking or explanation of the transition.

One type of story transition that MINSTREL explicitly guides the reader through is the beginning of the story. This transition is difficult because the reader has not yet created any framework to guide his understanding of the story. He may not even be aware of the genre or the topic of the story. By providing an explicit transition scene that introduces the story, MINSTREL eases the effort the reader must make to understand the story. A similar transition occurs at the end of the story, and MINSTREL also creates an explicit transition scene for that transition.

(For an example of how MINSTREL uses other devices to ease transition, see the use of paragraphing in ALP:Generate-Theme, below.)

10.2.1 Introduction Scenes

To help the reader begin the story, MINSTREL includes an introductory scene at the beginning of each story. This scene introduces the reader to the genre, the time frame, and the main character of the story. The introductory scene may also contain scenes to establish the main character's personality:

```
It was the spring of 1089, and a knight named Lancelot returned
to Camelot from elsewhere.  Lancelot was hot tempered.  Once,
Lancelot lost a joust.  Because he was hot tempered, Lancelot
wanted to destroy his sword.  Lancelot struck his sword.  His
sword was destroyed.
```

The mention of the year and "Camelot" establish immediately the time frame and genre of the story. Similarly, the early mention of Lancelot alerts the reader to the importance of that character. Together, this information permits the reader to quickly and easily establish a framework for understanding the story, thereby improving the communication between the reader and the author.

Although human authors have developed a variety of types and styles of story introductions, MINSTREL knows only one type of story introduction. MINSTREL's plan for creating a story introduction mentions the time, location and main character. This is captured in an author-level plan called ALP:Create-Story-Intro (Figure 10.1).

Name: ALP:Create-Story-Intro

Goal Type: &Add-Story-Intro

Object: *AL-OBJ*

Test: None

Body:

1. Create a state schema representing the story time frame (a random year in the Middle Ages) and add it to the introductory scenes of the story.

2. Create a state schema representing the main character of the theme being located at Camelot, and have that state schema supersede an earlier location "elsewhere". Add these state schemas to the introductory scenes of the story.

Figure 10.1 ALP:Create-Story-Intros

MINSTREL's plan for creating a story introduction is specific for the King Arthur domain. The appropriate location and time frame for the story (Camelot, the Middle Ages) are built into the author-level plan. An author with more experience in writing in a number of different domains might have several more general plans for introducing a story.

10.2.2 Denouement Scenes

When a reader picks up a text, he has expectations about what he will find ([Smith 1982]). The precise expectations vary from genre to genre, as well as from reader to reader and culture to culture. A common expectation for modern readers of fictional text is that the writer will resolve all important questions before ending the story. In common terms, the author will "tie up the loose ends".

In theme-based stories of the sort that MINSTREL tells, the primary issue that the author must resolve before ending the story is the illustration of the story theme. This, of course, MINSTREL does, because its highest priority author-level goal is to illustrate the theme. However, the presentation of the story is improved if other, secondary issues are also resolved.

For example, the presentation of a story is improved if the author resolves all important character goals. A story that ended with a princess being menaced by a dragon would leave the reader feeling uneasy and unfulfilled, because of his failed expectation that the author would resolve the princess's fate before the story ended. Or he might expect that the story wasn't truly ended yet – perhaps the author intended a sequel.

To avoid leaving the reader with such failed expectations, MINSTREL has the author-level goal

to provide denouements that resolve any outstanding story issues. MINSTREL's author-level plan for achieving this goal recognizes two types of issues:

1. Long-term character fates.

2. Important, unresolved character goals.

At the end of each story it tells, MINSTREL provides denouement scenes that resolve these issues. For instance, at the end of *The Mistaken Knight*, MINSTREL resolves the fates of the three story characters:

```
...Lancelot became a hermit.  Frederick was buried in the
woods.  Andrea became a nun.
```

MINSTREL has three author-level plans for creating story denouements. Two of these, ALP:Burial-Denouement and ALP:Tragic-Denouement create story scenes that explain the long-term fates of characters whose future is in doubt. The third, ALP:Important-Goal-Denouement, creates story scenes to resolve any important unresolved character goals.

10.2.2.1 Character Fates

One type of denouement scene that MINSTREL creates explains the long-term fate of a character. But MINSTREL does not create these scenes for all the characters in a story. Denouement scenes are created only for characters with unhappy fates. Characters which who achieve their goals and live happily ever after are not explicitly mentioned in the story denouements.

Denouement scenes for happy characters do little to improve the presentation of the story. The reader already has an (optimistic) expectation that all characters will achieve their goals and live happily ever after; confirming this adds little to the story. Of course, this is purely a stylistic choice. MINSTREL could well create scenes of the form:

```
...Richard and Jennifer lived happily ever after.
```

But since these scenes add little to the story they are omitted.

Denouement scenes for unhappy characters, on the other hand, do improve the story presentation. There are several reasons for this. First, readers generally expect story characters to succeed at their goals. If they do not the failure of the expectation raises cognitive dissonance in the reader that can be reduced by confirming the failure of the expectation. Second, readers expect goal failures to have some effect on the behavior of the story characters, and the denouement scene provides an opportunity to show this effect.

10.2.2.1.1 ALP:Burial-Denouement

One type of unhappy fate that can befall a character is to die. When a character dies in the course of a story, ALP:Burial-Denouement provides a denouement for the character by creating a scene in which the character is buried. ALP:Burial-Denouement is shown in Figure 10.2.

Name: ALP:Burial-Denouement

Goal Type: &Make-Denouement

Object: *AL-OBJ*

Test: Is *AL-OBJ* a character who died during the story?

Body:
1. Create a state schema representing the character being buried.
2. Try to recall where previous characters of the same role (knight, princess, etc.) were buried. If recall is successful, then bury this character in the same location. If recall fails, then bury the character in the woods.

Figure 10.2 ALP:Burial-Denouement

ALP:Burial-Denouement results in a scene of the form:

```
Frederick was buried in the woods.
```

for each character that died during the course of the story.

10.2.2.1.2 ALP:Tragic-Denouement

Another type of unhappy fate that can befall a character is to have an important goal thwarted, such as trying to achieve love and failing. When this situation occurs, MINSTREL creates a denouement in which the character changes his or her role:

```
...Andrea loved Frederick...  Lancelot fought with Frederick.
Frederick was dead...  Andrea became a nun.
```

This denouement satisfies the reader's expectation that important goal failures should have a long-term effect on a character's behavior.

These denouements are created by the author-level plan ALP:Tragic-Denouement. ALP:Tragic-Denouement is shown in Figure 10.3.

The key step in ALP:Tragic-Denouement is finding an appropriate new role for the tragic character. This is achieved by searching MINSTREL's semantic knowledge of roles.

Each of the character roles that MINSTREL knows is described in terms of a small set of funda-

Name:	ALP:Tragic-Denouement
Goal Type:	&Make-Denouement
Object:	*AL-OBJ*
Test:	Is *AL-OBJ* a character who suffered an important goal failure during the story?
Body:	1. Create a state schema representing a role change from the character's current role to a new role. 2. Search the semantic knowledge of character roles to find the role most completely different from the character's current role. Make this role the character's new role.

Figure 10.3 ALP:Tragic-Denouement

mental dispositions (Figure 10.4). To find a role different from the current role, MINSTREL searches this knowledge for a role which has different fundamental dispositions.

Sex	Role	Dispositions
Female	Nun	Peaceful, Non-Social.
	Princess	Peaceful, Social.
Male	Hermit	Peaceful, Non-Social.
	Knight	Violent, Social.
	King	Peaceful, Social.

Figure 10.4 Roles and Fundamental Dispositions

As Figure 10.4 shows, there are two fundamental dispositions for roles in MINSTREL: Peaceful/Violent, and Social/Non-Social. Roles are also divided into male and female, a politically incorrect division that is nonetheless appropriate for the King Arthur genre. To find a new role for a tragic character, MINSTREL searches the roles it knows for a role which differs on as many fundamental dispositions as possible. Consequently, tragic knights become hermits, tragic princesses become nuns, and so on.

10.2.2.2 ALP:Important-Goal-Denouement

In addition to resolving character fates, MINSTREL also creates denouements to resolve any important, undecided character goals. If a story contained the unresolved scene:

```
Andrea saw a dragon move towards her.  Andrea feared the dragon
would eat her.
```

MINSTREL would create a denouement to resolve Andrea's goal of protecting her health:

```
Andrea saw a dragon move towards her.  Andrea feared the dragon
would eat her...  Andrea ran away from the dragon and escaped.
```

Unresolved character goals can occur for several reasons. First, a character goal may be necessary to achieve an author-level goal (such as illustrating the theme) without needing a resolution. In this case, the author-level goal will create the goal without creating a resolution. Second, an unresolved character goal can occur as an unexpected side-effect of some other character action. For example, if a dragon moves towards the woods, and a princess is in the woods, then MINSTREL's consistency goals will notice that the princess should have a goal to protect her life. This goal will be created and added to the story, and might still be unresolved by the end of the story. In these cases, ALP:Important-Goal-Denouement will create a resolution for the unresolved character goal.

Whether the denouement is a successful resolution of the goal or a thwarted one depends upon the character's other goal resolutions. If the character has had other goals thwarted during the course of the story, then this goal will be thwarted as well (thereby making a tragic character more tragic). Otherwise, a successful resolution to the goal will be sought.

The author-level plan which achieves this is ALP:Important-Goal-Denouement. ALP:Important-Goal-Denouement is shown in Figure 10.5.

Name: ALP:Important-Goal-Denouement

Goal Type: &Make-Denouement

Object: *AL-OBJ*

Test: Is *AL-OBJ* an unresolved &P-Health, &C-Health or &A-Love character goal?

Body:

1. Search the story to determine if the actor of *AL-OBJ* has any thwarted goals.
2. If so, then create an uninstantiated state schema that thwarts *AL-OBJ*. Use author-level planning recursively to instantiate the schema.
3. Otherwise, create an uninstantiated state schema that achieves *AL-OBJ*. Use author-level planning recursively to instantiate the schema.

Figure 10.5 ALP:Important-Goal-Denouement

As with the consistency goals ALP:General-Motivating-State and ALP:Check-Affects (see Chapter 9), ALP:Important-Goal-Denouement uses a static evaluation of goal importance to determine what unresolved character goals require resolution. As discussed previously, this model can be extended to use a dynamic evaluation of goal importance, such as character goal trees [Carbonell 1978].

10.3 Ordering of Story Events

The second major task MINSTREL must undertake in order to present a story in English is to order the events in the story. When MINSTREL creates a story, it does not create it in a serial, beginning-to-end fashion. Rather, it jumps about, creating scenes where its author-level goals direct. Like a human author, MINSTREL may create scenes near the beginning of the story for a while, then jump to the end of the story, only later filling in the scenes between. Indeed, MINSTREL's first goal is to create scenes to illustrate the theme - scenes which will be scattered throughout the final story.

To present the story to the reader, MINSTREL must impose an order upon the story. Some of this order is implicit in the story, in the form of causal relationships between events. Other scenes must be explicitly ordered by the roles they take in larger structures such as the story theme.

10.3.1 Implicit Ordering

The primary knowledge MINSTREL relies upon to order the events of the story is causality. Most of the events in the story are character actions and the states they cause. These actions arise out of character goals and plans, which must also be expressed. All of these are causally related: goals intend plans, which intend actions, which intend changes in the world, which achieve goals. Much of the story thus already has an implicit order.

The straightforward way to express two events which have a causal relationship is to express the cause before the effect:

```
Lancelot fought the dragon.  The dragon was dead.
```

Telling causally related events in this order is pleasing and easy for the reader to understand. It also has the advantage of making the causal relationship explicit in the text.

Of course, it possible to change this ordering. An author can place the effect first if he desires:

```
The dragon was dead because Lancelot fought the dragon.
```

This second expression obscures the causal relationship between the two events (and consequently requires a clue word to alert the reader to the relationship), but has the advantage of emphasizing the result of the action over the action itself. Expressing causal events out-of-order is thus useful when the author wishes to emphasize the effect instead of the cause.

MINSTREL's default behavior is to express causally-related events in the cause-effect order. To express two causally related events, or a series of causally-related events, MINSTREL first expresses the cause and then the effect:

```
Because he was hot tempered, Lancelot wanted to destroy his
sword.  Lancelot struck his sword.  His sword was destroyed.
```

However, MINSTREL is able to express events in the other order when it *explicitly* wishes to emphasize a cause instead of an effect. For example, when MINSTREL wishes to emphasize that Andrea possesses berries, it writes:

```
Andrea had some berries because Andrea picked some berries.
```

MINSTREL's knowledge about how causality can be used to order story events is captured in the lexicon of its language generation component. The language generation component, the lexicon, and the role of implicit ordering in story generation is discussed in more detail below (see Section 10.4).

10.3.2 Explicit Ordering

Although the ordering implied by causal relationships is useful to guide the presentation of the story to the reader, there are situations where the author must provide explicit ordering. To begin with, there may be story scenes with no causal relationship. In the stories that MINSTREL tells, this is the case with the scenes that illustrate the theme. These scenes have a thematic relationship, but not necessarily a causal one. And even if a causal relationship is present between two scenes, the author may wish to provide an explicit ordering to emphasize a particular story event.

MINSTREL has author-level plans that provide explicit ordering for the introductions, denouements and moral of the story, for the events in the theme of the story, and for events in character beliefs. The first two plans provide explicit ordering for events that have no causal relationship; the last plan provides an explicit ordering that emphasize an evidence-belief relationship instead of a causal relationship.

10.3.2.1 ALP:Generate-Story

MINSTREL's stories have a simple overall structure, consisting of introductory scenes, theme-related scenes, denouement scenes, and a moral:

The Vengeful Princess

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

Theme	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. Jennifer wanted to live. Jennifer tried to drink a magic potion but failed. Grunfeld was filled with grief.
Denouements	Jennifer was buried in the woods. Grunfeld became a hermit.
Moral	MORAL: Deception serves the devil.

The author-level plan responsible for ordering the presentation of these scenes is called ALP:Generate-Story.

At the top level, MINSTREL represents a story as a story schema. The story schema contains pointers to the characters in the story, the introduction and denouement scenes, the abstract theme of the story, and to the events that make up the body of the story. The story schema for **The Vengeful Princess** is shown in Figure 10.6. ALP:Generate-Story uses the structure of the story schema to generate the parts of the story in proper order.

Given a story schema, ALP:Generate-Story creates author-level goals to generate in English the introductory scenes, the theme-related scenes, the denouement scenes and the moral. These goals are given priorities to ensure that they will be achieved in the proper order, resulting in a story in which the introductory scenes come first, the moral last, and the intervening scenes are also in proper order. ALP:Generate-Story is shown in Figure 10.7.

10.3.2.2 ALP:Generate-Theme

The bodies of the stories that MINSTREL tells are made up of scenes that illustrate a Planning Advice Theme (PAT). These scenes do not necessarily have any causal relationships, so the author must provide an explicit ordering. This is achieved by an author-level plan called ALP:Generate-Theme.

Figure 10.8 shows an example story theme, &PAT-Good-Deeds-Rewarded. This theme has a complicated structure, consisting of a schema with eleven slots. These slots are filled with abstract goal/plan structures that represent the advice of the theme. In this case, the structures represent the advice "Good deeds do not go unrewarded." The purpose of ALP:Generate-Theme is

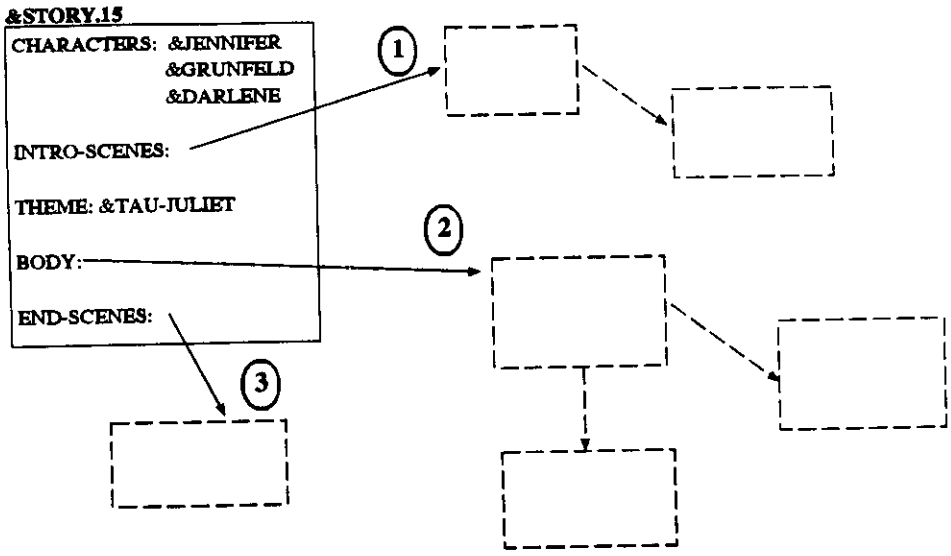


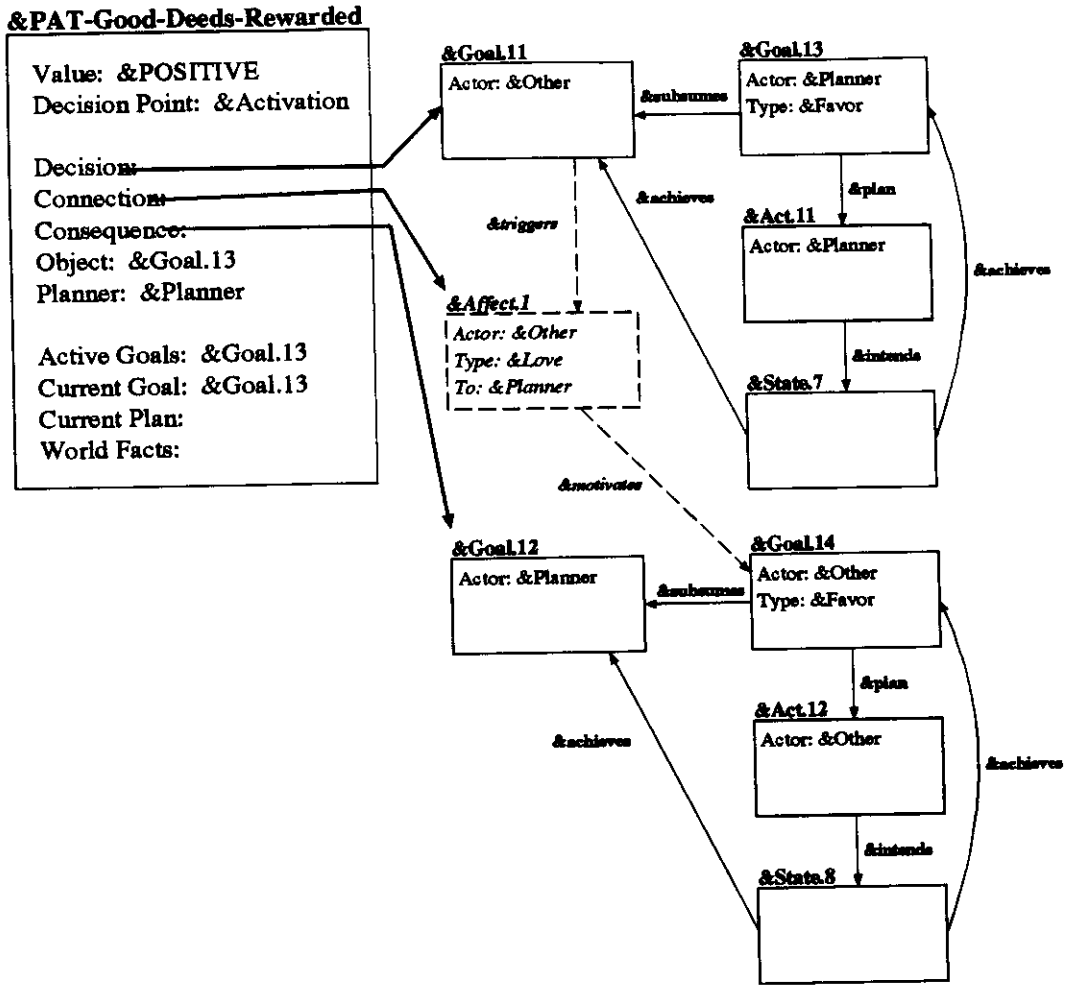
Figure 10.6 Example Story Schema

Name: ALP:Generate-Story
Goal Type: &Generate-English
Object: *AL-OBJ*
Test: Is *AL-OBJ* a story schema?
Body:
1. Create a goal to generate in English the introductory scenes of the story, with a priority of 100.
2. Create a goal to generate in English the theme-related scenes (i.e., the body of the story), with a priority of 90.
3. Create a goal to generate in English the denouement scenes, with a priority of 80.
4. Create a goal to generate in English the moral of the story, with a priority of 70.

Figure 10.7 ALP:Generate-Story

to generate the parts of the instantiated story theme in the proper order, so that a reader can recognize the advice implicit in the illustration of the theme.

The core of the advice represented by a Planning Advice Theme is captured in the "Decision", "Connection" and "Consequence" slots. The Decision represents a planning choice, the Consequence represents either a good or bad result of that choice, and the Connection represents how the Decision and the Connection are related. In the case of PAT:Good-Deeds-Rewarded, the Decision is to do a good deed for another, the Consequence is that someday that favor is returned, and the Connection is that the first good deed creates a thankful emotion that later motivates the returned favor.



Summaries	
Decision:	&Planner does a favor for &Other, helping him achieve a goal.
Connection:	&Other loves &Planner for his good deed.
Consequence:	&Other returns &Planner's good deed.

Figure 10.8 PAT:Good-Deeds-Rewarded

For the reader to recognize this advice, the connection between the planning decision and its consequence must be clear. There are several ways a writer might present this, but the easiest is to present the decision, connection and consequence as a simple sequence:

A hermit named Bebe healed a knight named Cedric. Cedric was grateful. Later, Cedric killed a dragon that was threatening Bebe.

The sequencing of these events makes it simple for the reader to deduce the relationship between them, and hence the story theme.

A Planning Advice Theme can also contain events other than Decision, Connection and Consequence. In particular, a PAT can contain "World Facts" which provide a context in which to understand the theme's advice. Figure 10.9 shows an example of a PAT which contains World Facts, PAT:Pride-Fall. PAT:Pride-Fall represents the advice "Pride goes before a fall." The Decision, Connection and Consequence represent a planner who pursues a goal motivated by excessive pride and suffers as a result. The World Facts represent a context in which this advice can be understood, namely that the planner was warned against pursuing his goal and ignored that advice.

For the reader to make use of the context that the World Facts provide, it should be presented before the remainder of the theme:

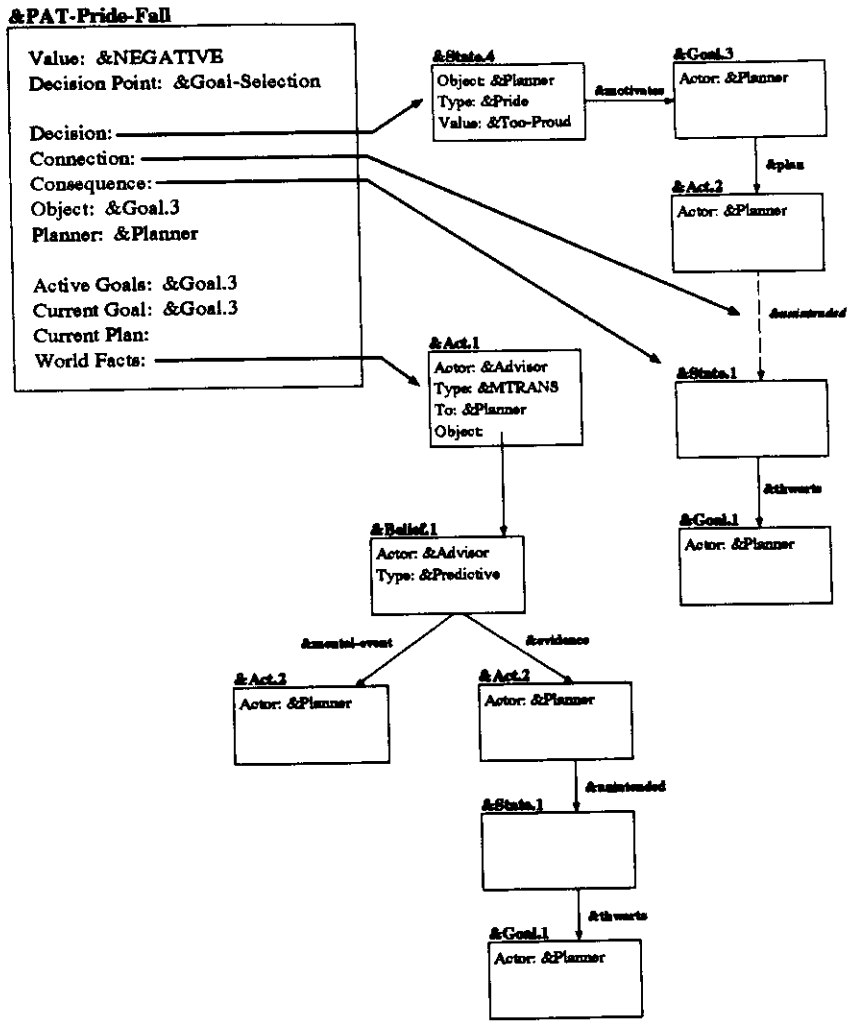
```
A hermit named Bebe told Grunfeld that Bebe believed that if
Grunfeld fought with the dragon then he would be hurt.

Grunfeld was very proud.  Because he was very proud...
```

ALP:Generate-Theme is shown in Figure 10.10. It generates the parts of theme by first generating the World Facts (if any), followed by the Decision, the Connection and the Consequence. In addition to creating an explicit ordering for the parts of the theme, ALP:Generate-Theme also aids the reader in transitioning between the parts of the theme by the use of paragraphing. The World Facts, Decision and Consequence of the theme are explicitly marked in the text by paragraph breaks. By making the structure of the theme explicit in the structure of the text, MINSTREL helps the reader discover and understand the theme implicit in the text.

10.3.2.3 ALP:Generate-Belief

MINSTREL's final author-level plan for explicit ordering is ALP:Generate-Belief. ALP:Generate-Belief provides an explicit order for presenting character beliefs to the reader. Character beliefs do have a causal structure, but rather than emphasize the causal structure, MINSTREL uses explicit ordering to emphasize the argument structure of the belief.

Character beliefs have an argument-like structure consisting of a believer, what he believes, the evidence for his belief, and how the believer found the evidence. Figure 10.11 shows the representation for the belief that Lancelot has when he sees a dragon move towards Jennifer. The belief is represented by the belief schema &Belief.16. The type of this belief is predictive and the prediction involved is pointed to by the &Mental-Event link. The prediction is that Jennifer will be eaten by the dragon. The evidence for this is pointed to by the &Evidence link. In this case, the evidence is the dragon moving towards Jennifer. The &B-Precond link points to a representation of how the believer achieved the evidence for his belief. In this case, the belief precondition is that Lancelot was in the same location as the dragon and saw the dragon move towards



Summaries	
Decision:	&Planner is warned by &Advisor not to try &Act.2, because it will result in an unintended goal failure.
Connection:	&Planners pride motivates him to try &Act.2 anyway.
Consequence:	&Act.2 causes the unintended goal failure &Planner was warned about.

Figure 10.9 PAT:Pride-Fall

Name: ALP:Generate-Theme
Goal Type: &Generate-English
Object: *AL-OBJ*
Test: Is *AL-OBJ* a Planning Advice Theme?
Body:
1. Begin a new paragraph. Create a goal to generate in English the World Facts slot of the PAT, with a priority of 100.
2. Begin a new paragraph. Create a goal to generate in English the Decision slot of the PAT, with a priority of 90.
3. Create a goal to generate in English the Connection slot of the PAT, with a priority of 80.
4. Begin a new paragraph. Create a goal to generate in English the Consequence slot of the PAT, with a priority of 70.

Figure 10.10 ALP:Generate-Theme

Jennifer (literally, paid attention with his eyes to the event).

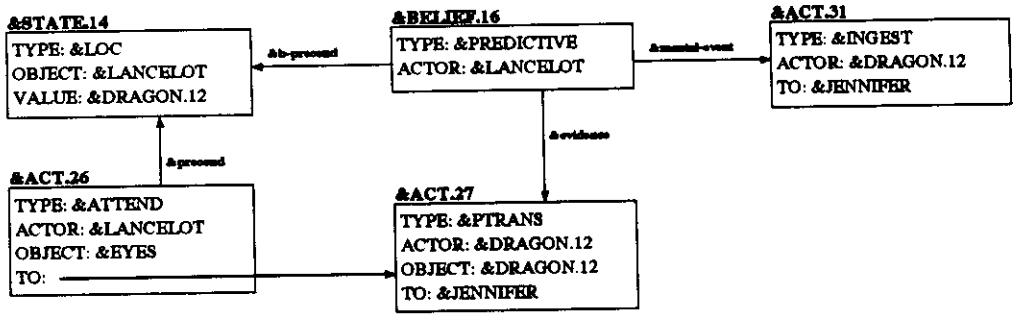


Figure 10.11 Example Belief

There is a causal relationship between the belief precondition and the belief. The acts in the belief precondition cause the believer to acquire the evidence for his belief. Following its implicit ordering of cause before effect, MINSTREL would therefore generate the belief shown in Figure 10.11 in the following way:

```
Lancelot was in the same location as a dragon.  Lancelot saw
the dragon move towards Jennifer.  Lancelot believed that the
dragon would eat Jennifer.
```

This expression emphasizes the belief precondition by placing it first. However, the author would like in this case to emphasize the argument structure of what he is expressing, rather than the causal structure. The argument structure can be better expressed by placing the evidence first:

```
Lancelot saw the dragon move towards Jennifer because Lancelot
was in the same location as the dragon.  Lancelot believed that
the dragon would eat Jennifer.
```

MINSTREL's plan for generating beliefs overrides the implicit ordering of the belief as a causal structure with an explicit ordering as an argument structure. ALP:Generate-Belief is shown in Figure 10.12.

Name:	ALP:Generate-Belief
Goal Type:	&Generate-English
Object:	*AL-OBJ*
Test:	Is *AL-OBJ* a belief schema?
Body:	1. Create a goal to generate the &Evidence portion of the belief, followed by the clue word "because", followed by the &B-Precond portion of the belief, with priority 100. 2. Create a goal to generate the &Mental-Event portion of the belief, with priority 90.

Figure 10.12 ALP:Generate-Theme

10.4 Natural Language Generation in MINSTREL

After creating presentation scenes and ordering the story events, the final step in presenting the story to the reader is to express the story in natural language. The process of expressing concept representations in natural language is called natural language generation. In MINSTREL, this task is handled by a component called RAP.

To translate concepts into language, RAP uses a knowledge base about language encoded as *phrases*. Each phrase associates a conceptual object with a linguistic structure, such as a word, idiom or syntactic form. RAP takes as input a conceptual structure and by repeatedly matching it against phrases, outputs a list of natural language words.

Phrasal generation is a variant of phrasal parsing. Phrasal parsing is a methodology that integrates syntactic and semantic information into a lexicon of phrases. Phrasal parsing evolved from case grammar ([Fillmore 1968]) where syntactic rules were used to fill in semantic case frames. Becker ([Becker 1975]) recognized that idioms could be treated as individual syntactic units, and thus that the base unit for the lexicon should not be single words but phrases. Functional grammar ([Kay 1979]) extended the approach by using a single kind of formal structure to specify patterns of features, function assignments, lexical items, and constituent orderings. Different variations of functional grammar have been proposed, depending on the focus of research and type of formal sturcture used. For example, lexical-functional grammar ([Bresnan 1982]) emphasizes the role of the lexicon within a transformational grammar approach, and definite-clause grammars ([Pereira 1980]) focus on how to specify lexical and syntactic information in logic.

10.4.1 RAP Overview

MINSTREL's phrasal generator is inspired by PGEN (Phrasal GENerator), a tool developed by John Reeves for use with Rhapsody knowledge representation package ([Reeves 1991][Turner 1987]). In turn, PGEN was based upon the PHRAN phrasal parser ([Wilensky 1980][Arens 1986]). As in PGEN and PHRAN, entries in RAP's lexicon are composed of an input pattern and an output pattern:

```
(rap:phrase knight
  (input &knight)
  (output knight))
```

Each phrase represents a "translation" from a pattern of input concepts into a pattern of output words. The above phrase translates the conceptual structure which represents the knight role (&Knight) into the word "knight". When given a conceptual structure to express in language, RAP tries to match the structure against the input patterns of the phrases in lexicon. When a match is found, the input concept is replaced with the output pattern of the phrase.

In MINSTREL, concept instances are represented by schemas. In the input and output patterns of phrases, schemas are represented in list form. The type of the schema is the head of the list, the name of the schema is the second element of the list (or nil, if the schema is anonymous) and the remaining items in the list are slot/value or link/value pairs. For example:

```
(human &human.57
  :type &knight
  :name 'lancelot)
```

is the list representation of a human schema named &human.57. This schema has a slot named :type with the value &knight and a slot named :name with the value "lancelot". A very specific phrase to translate the above schema into the word "Lancelot" would look like this:

```
(rap:phrase knight-lancelot
  (input (human nil
           :type &knight
           :name 'lancelot))
  (output the knight lancelot))
```

The anonymous human schema in the input pattern of this phrase will match any human schema which has a :type of &Knight and a :name of "lancelot", translating the schema into the phrase "the knight lancelot".

Patterns may also contain logical variables. Logical variables are represented using **?variable-name** notation. When applying a phrase, variable matching is done using unification, so variables of the same name must match exactly. Variables in the output pattern substitute their values. Logical variables are useful to create generic phrases and to copy parts of the input pattern into the output pattern:

```
(rap:phrase knight-generic
  (input (human nil
           :type &knight
           :name ?name))
  (output the knight ?name))
```

The input pattern of this phrase matches any human schema with the :type slot &knight. The logical variable **?name** will match the name of the knight and substitute it into the output pattern. If the substituted value is a word, it is output as language just as if the word itself had been present in the output pattern. Applied to &human.57, this pattern also generates "the knight lancelot".

If the output pattern of a phrase contains a concept, generation is recursively applied to the output concept to translate it into words. Thus it is possible to generate a concept by recursively generating the parts of the concept:

```
(rap:phrase generic-character
  (input (human nil
           :type ?role
           :name ?name))
  (output a ?role named ?name))
```

When this phrase is applied to the human schema shown above, the logical variable **?role** matches &Knight and **?name** matches "lancelot." These are substituted into the output pattern, resulting in:

```
a &knight named lancelot
```

&Knight is a concept so generation is recursively applied. &Knight matches the very first phrase shown above, which translates &Knight into "knight". This is substituted into the output from the firsqt phrase, resulting in:

```
a knight named lancelot
```

This process is known as *recursive descent* generation.

In addition to schemas, words, and logical variables, patterns may contain logical conjunctions and disjunctions. These are represented as lists with the special heads "*or*" and "*and*". These have the obvious meanings:

```
(rap:phrase example-or
  (input (*or* &hermit &monk))
  (output an old man))
```

translates &Hermit or &Monk into the phrase "an old man".

Finally, phrases may have two additional features called "test" and "proc". The test feature is a predicate that is evaluated after matching the input pattern but before a phrase is applied. If the predicate returns true, then the phrase is applied. If the predicate fails, the phrase is ignored. The test feature enables RAP to test conditions that cannot be expressed as patterns or logical variables. For example:

```
(rap:phrase pronoun-he
  (comment "he")
  (input (human nil
           :name ?name
           :sex &male))
  (test (rap:mrr? ?name))
  (output he))
```

RAP:MRR? is a predicate that returns true if its argument was the most recently referenced name in the story output. This phrase will be applied only if the concept being generated matches the input pattern and the name of the human was the most recently referenced name in the story. This permits RAP to to produce sentences such as:

```
A hermit named Bebe told Grunfeld that Bebe believed that if
Grunfeld fought with the dragon then he would be hurt.
```

The last reference to Grunfeld in this sentence can be generated as "he" because the immediate previous reference to Grunfeld causes the test portion of the pronoun-he phrase to succeed.

The proc feature is a procedure that is evaluated after a phrase is applied. This enables RAP to perform actions other than outputting a pattern, such as saving a reference:

```
(rap:phrase generic-character
  (input (human nil
           :type ?role
           :name ?name))
  (output a ?role named ?name)
  (proc (rap:save-reference ?name)))
```

Here RAP:SAVE-REFERENCE is a procedure that saves its argument as the most recently referenced character name.

Given a lexicon of phrases and a conceptual structure to be generated, the process of generation is straightforward. The input conceptual structure is matched against each of the phrases in the lexicon. When a match succeeds, logical variables in the output pattern are replaced by their values and the resulting pattern is placed in the output buffer. The buffer is then scanned from left to right. If the leftmost object in the buffer is a word, it is output and removed from the buffer. If the leftmost object is a concept, generation is recursively applied. This process continues until the output buffer is empty.

To illustrate this process, suppose that MINSTREL has this additional phrase in its lexical memory:

```
(rap:phrase fight
  (input (act nil
           :type &m-fight
           :actor ?actor
           :to ?victim))
  (output ?actor fought ?victim *period*))
```

This phrase describes how to generate an act schema representing a fight. With this and the above phrases in its lexical memory, MINSTREL is asked to generate the following concept:

```
(act &act.57
  :type &m-fight
  :actor (human &human.37
           :type &knight
           :name 'lancelot
           :sex 'male)
  :to (human &human.18
        :type &knight
        :name 'richard
        :sex 'male))
```

Generation begins by trying to match the input concept, &Act.57, against phrases in the lexicon. &Act.57 matches the input pattern of the lexical phrase "fight", binding **?actor** to the schema &Human.37 and **?victim** to the schema &Human.18. These values are substituted into the output pattern and placed in the output buffer:

```
&human.57 fought &human.18 *period*
```

Now the leftmost object in the output buffer is examined. It is a concept (&Human.57), so generation is recursively applied.

&Human.57 matches the lexical phrase "generic-character". **?Role** is bound to &knight and **?name** is bound to "lancelot". The output pattern of "generic character" is then substituted into the output buffer in place of &Human.57:

```
a &knight named lancelot fought &human.18 *period*
```

The leftmost member of the output buffer is a word ("a") so it is output. The next member is the concept &knight. Generation is recursively applied, this concept matches the lexical phrase "knight", and the concept &knight is replaced by the word "knight":

```
knight named lancelot fought &human.18 *period*
```

Now the first four members of the output buffer are words, and they are successively output, until &Human.18 is encountered. In a manner similar to &Human.37, &Human.18 is translated to:

```
a knight named richard *period*
```

This is then output, ending in the special word "*period*" which tells the phrasal generator to add a period to the previous word. (RAP uses similar keywords to capitalize proper names, capitalize the first word of a sentence, and indent for paragraphs.) The final output is thus:

```
A knight named Lancelot fought a knight named Richard.
```

Figure 10.13 shows the generation tree created in this example.

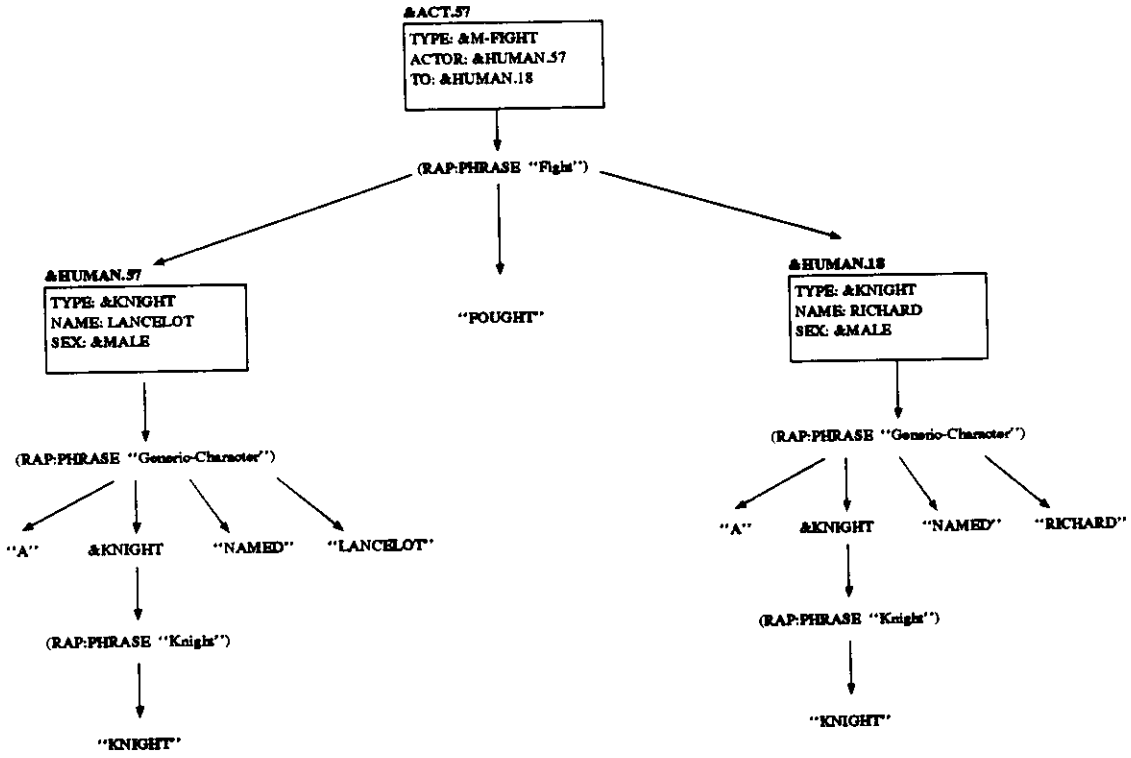


Figure 10.13 Example Generation Tree

10.4.2 Syntactic Information in RAP

One issue which RAP must address is the need to process syntactic information during the generation process. For example, the phrase shown earlier for generating a pronoun:

```
(rap:phrase pronoun-he
  (comment "he")
  (input (human nil
           :name ?name
           :sex &male))
  (test (rap:mrr? ?name))
  (output he))
```

is not completely correct, because it can lead to the production of an incorrect pronoun if the character it is applied to appears in the object position of a direct verb, i.e.,

John gave the sword to *he*.

This occurs because "pronoun-he" lacks any knowledge of the proper syntactic category of the pronoun.

There are two possibilities for integrating syntactic information into the language generation process. The first is to express syntactic information in the same framework that expresses semantic information, i.e., in the conceptual structures being generated. For example, the schema that represents a story character could be augmented with a slot to identify its syntactic category, permitting a revised pronoun phrase which would use "he" only in the subject position:

```
(rap:phrase pronoun-he
  (comment "he")
  (input (human nil
           :name ?name
           :sex &male
           :syntax-category &subject))
  (test (rap:mrr? ?name))
  (output he))
```

Although this has the advantage of providing a uniform representation for both syntactic and semantic information, it has several problems. First, this representation mixes the language-independent conceptual knowledge of the story with the language-dependent expression of the story, defeating any notion of a language-independent conceptual representation. Second, this representation mixes long-lasting and ephemeral knowledge. While the semantic representation of a story is long-lasting, the syntactic information is necessarily ephemeral. A character may be the subject in one sentence and the object in the next.

The second possibility is to encode syntactic information separately from semantic information. Although this has the disadvantage of requiring a second representation for syntactic knowl-

edge, it has the advantages of maintaining a language-independent conceptual representation of the story while encapsulating syntactic knowledge with other language knowledge in the phrasal lexicon.

In MINSTREL, syntactic knowledge is represented by keywords. These keywords are "attached" to semantic representations by adjacency in the pattern portions of lexical phrases. Syntactic information can then be used to control which lexical phrases are applied during the generation process.

For example, the "fight" phrase used above can be augmented with keywords to mark the subject and object positions of the sentence being generated:

```
(rap:phrase fight
  (input (act nil
           :type &m-fight
           :actor ?actor
           :to ?victim))
  (output :subject ?actor fought :object ?victim *period*))
```

The ":subject" and ":object" keywords in the output pattern identify the following concepts as occupying subject and object positions in the sentence being generated. These keywords are placed into the output buffer along with the rest of the output pattern. Other lexical phrases can then use this syntactic knowledge to control generation. The "pronoun-he" phrase can be augmented to match only characters in the subject position by requiring that the character being generated be proceeded by the :subject keyword:

```
(rap:phrase pronoun-he
  (comment "he")
  (input :subject (human nil
                   :name ?name
                   :sex &male))
  (test (rap:mrr? ?name))
  (output he))
```

The new version of "pronoun-he" matches a two object pattern consisting of the keyword :subject followed by a male human schema. This properly restricts the use of the pronoun "he" to the subject position.

10.4.3 Implicit Ordering by Causal Relationships

As discussed earlier, the sequence in which story events are presented to the reader is determined by (1) explicit author-level goals and (2) implicit ordering by causal relationships between the story events. The latter knowledge is encoded in the phrasal lexicon.

There are several reasons to encode knowledge about how to speak about causal relationships in

the phrasal lexicon. First, causal relationships are common to many different problem domains. Capturing this knowledge at a low level in the language generation process permits the easy sharing of this knowledge between domains. Second, this encoding reflects the intuitive difference between explicit, reasoned decisions about how a story will be presented to the reader and the implicit, unconscious use of language. In general, we assume that the author's language generation capability is at least partly at the conscious level and at least partly below the conscious level. Explicit author-level presentation goals capture conscious reasoning about ordering and other presentation concerns (i.e., "I'd better mention this event first to set up the foreshadowing...") while knowledge in the phrasal lexicon captures the author's fluent, automatic language generation capability. Speaking about causally-related events is so common to language use that it can be expected to be captured at the subconscious, fluent level.

When MINSTREL's language generation component is asked to generate a story event which is causally-related to other story events, lexical phrases are applied which generate not only the requested story event, but the causally-related events as well, using ordering and clue words to make the causal relationships evident.

The story events shown in Figure 10.14 represent typical causal relationships. &Goal.15 is Lancelot's goal to destroy a monster. It has causal links to an act intended to achieve the goal (the &Plan link to &Act.17) and to a state of the world that achieves the goal (the &Achieved-by link to &State.8). This particular configuration is called a "goal triad" because it consists of the three legs of a planning sequence: a goal, the plan to achieve the goal, and the state that achieved the goal.

When asked to generate &Goal.15, MINSTREL produces the following output:

```
Lancelot wanted to kill the dragon.  Lancelot fought the dragon
with his sword.  The dragon was dead.
```

Not only is the goal expressed, but also the causally-related events which achieved the goal. Furthermore, these events were expressed in a way that makes their relationships evident. The knowledge needed to generate and order these events based upon their causal relationships is captured in the phrasal lexicon.

The top-level lexical phrase used to generate the above example is called "goal-triad":

```
(rap:phrase goal-triad
  (input :sentence (*and* ?goal
                          (goal nil
                                &plan ?plan
                                &achieved-by ?as)))
  (output :simple-sentence ?goal
          :simple-sentence ?plan
          :simple-sentence ?as))
```

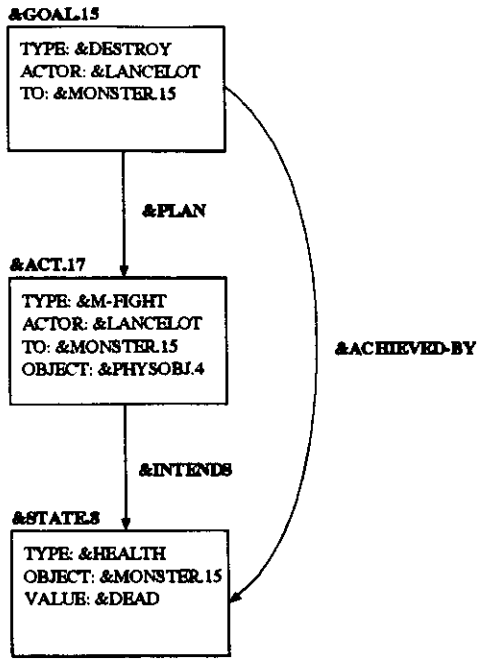


Figure 10.14 Example Causal Relationship

The input pattern of this phrase consists of the keyword :sentence followed by a goal schema. The keyword :sentence indicates that the goal can be generated as a sentence, a sentence with clauses, or a series of sentences. (The keyword :simple-sentence indicates that a story event should be generate as a simple sentence without any causally-related events.) The goal schema has a &plan link and an &achieved-by link. The values of these links will be bound to the logical variables **?plan** and **?as**, respectively. The logical "and" enclosing the goal schema will result in the goal schema itself being bound to the logical variable **?goal**.

When the input pattern of this phrase is matched against the example above, **?goal** is bound to &Goal.15, **?plan** is bound to &Act.17, and **?as** is bound to &State.8.

The output breaks the goal triad into three sentences in the order: goal, plan, achievement. The first pattern generates the goal again with the keyword :simple-sentence, which generates the goal as a single sentence ignoring all causal links. (This prevents the generator from repeatedly applying the "goal-triad" phrase.) The lexical phrase "generic-goal-sentence" is applied to this new pattern.

```
(rap:phrase generic-goal-sentence
  (input :simple-sentence
         (goal nil
           :actor ?who
           :type ?type
           :object ?object))
  (output :subject ?who wanted ?type :object ?object *period*))
```

After the various parts of this output pattern have been generated, the result is the sentence:

```
Lancelot wanted to kill the dragon.
```

Similar lexical phrases are applied to the plan and the achievement of the goal to create the second and third sentences:

```
Lancelot wanted to kill the dragon.
Lancelot fought the dragon with his sword.
The dragon was dead.
```

Implicit ordering by causality can be overridden by explicit author-level goals. One reason to override the implicit ordering is to emphasize a particular event by placing it first in a series of causally-related events. This can be achieved by beginning the generation process at the event to be emphasized.

In the above example, if MINSTREL wished to emphasize the action rather than the goal, it would generate &Act.17 instead of &Goal.15. Consequently, the first pattern applied would be "act-triad":

```
(rap:phrase act-triad
  (input :sentence (*and* ?act
                          (act nil
                               &plan-of ?goal
                               &intends ?is)))
  (output :clause ?plan because
          :simple-sentence ?goal
          :simple-sentence ?is))
```

The keyword :clause is similar to :simple-sentence except that the trailing period is omitted. The result is:

```
Lancelot fought the dragon with his sword because he wanted to
kill the dragon.  The dragon was dead.
```

The different ordering of the clauses and the clue word "because" reflect the change in focus of the language generation from the goal to the act.

10.5 Summary

The final task in telling a story is to present the story to the reader. At the very least this involves selecting an order in which to present the events of the story and expressing those events in language. At the most, the author may create new story events and use language in subtle ways to improve the presentation of the story.

MINSTREL has both explicit and implicit author-level goals to order story events for presentation. RAP, MINSTREL's language generation component, encodes common knowledge about causal relationships and how they can be clearly expressed by use of ordering and clue words. At the level of explicit author goals, MINSTREL has knowledge about how to order the events that illustrate the theme of the story, beliefs, and other non-causal structures common to the stories it tells. MINSTREL also has the ability to explicitly override the implicit ordering used by RAP.

MINSTREL also uses other techniques to improve the presentation of its stories. MINSTREL uses paragraphing to make evident the underlying thematic structure of the stories it tells, and uses transition scenes to ease the reader's cognitive effort in understanding the story.

CHAPTER 11
Annotated Trace

11.1 Introduction

This chapter contains an annotated trace of MINSTREL as it tells the story *The Mistaken Knight*:

The Mistaken Knight

It was the spring of 1089, and a knight named Lancelot returned to Camelot from elsewhere. Lancelot was hot tempered. Once, Lancelot lost a joust. Because he was hot tempered, Lancelot wanted to destroy his sword. Lancelot struck his sword. His sword was destroyed.

One day, a lady of the court named Andrea wanted to have some berries. Andrea wanted to be near the woods. Andrea moved to the woods. Andrea was at the woods. Andrea had some berries because Andrea picked some berries. Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot loved Andrea. Some time later, Lancelot's horse moved Lancelot to the woods unintentionally, again causing him to be near Andrea. Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick. Lancelot believed that Andrea loved Frederick. Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Lancelot hated Frederick. Andrea loved Frederick. Because Lancelot was hot tempered, Lancelot wanted to kill Frederick. Lancelot wanted to be near Frederick. Lancelot moved to Frederick. Lancelot was near Frederick. Lancelot fought with Frederick. Frederick was dead.

Andrea wanted to be near Frederick. Andrea moved to Frederick. Andrea was near Frederick. Andrea told Lancelot that Andrea was siblings with Frederick. Lancelot believed that Andrea was siblings with Frederick. Lancelot wanted to take back that he wanted to kill Frederick. But he could not because Frederick was dead. Lancelot hated himself. Lancelot became a hermit. Frederick was buried in the woods. Andrea became a nun.

MORAL: Done in haste is done forever.

The purpose of this trace is to show how MINSTREL's various author-level goals come together to create a finished story. This trace will also identify places in the storytelling process where MINSTREL's mechanisms fail. This is important not only in order to present this research hon-

estly, but also to identify areas of future research.

In telling *The Mistaken Knight*, MINSTREL attempts 654 author-level goals and achieves 182. The trace of this processing is nearly 200 single-spaced pages; far too long to include here in its entirety. In order to reduce the length of this trace and highlight the interesting sections, the trace has been edited. This has consisted primarily of removing failed author-level goals, failed attempts at creativity and repetetive portions of the trace.

11.2 The Mistaken Knight

```
USER> (goal nil
         :actor (human nil
                       :type &knight)
         :new-state (state nil)
         &thwarted-by (state nil))

&GOAL.216

USER> (goal nil
         :actor &MINSTREL
         :type &tell-story
         :object &goal.216
         :priority 100)

&GOAL.217
```

The trace begins with the human user of the program suggesting a topic. The user specifies (using list notation for a goal schema) a character goal (&GOAL.216). The actor of the goal is a knight, and the goal is thwarted. What the goal is, how it is achieved, and how it is thwarted are left unspecified. The user then creates an author-level goal for MINSTREL to tell a story about the suggested character-level goal (&GOAL.217).

The purpose of this is to simulate "exterior input" to MINSTREL. Since MINSTREL does not have the life experience that human authors have, it has no source of inputs to suggest stories or jog its memory. To simulate this, the user provides a scene or fragment of a scene to act as an initial "seed" for the storytelling process. This scene won't necessarily be incorporated as part of the final story. Rather, as we shall see below, it serves as a index for recall. The shape and content of the final story will depend not so much upon this scene fragment as upon what the fragment recalls.

The next step is to begin MINSTREL's storytelling process on the initial storytelling goal:

```
User        USER>(alp:run &goal.217)
            ++++++++++++++++++++++++++++++++++++++++++++++++++
Goal        Author-level goal &TELL-STORY applied to &GOAL.216.

Plan        TRAM Cycle: &GOAL.217.
Recall        Executing TRAM:AL-STANDARD-PROBLEM-SOLVING.
                Recalling:    NIL.
              ...TRAM failed.
              Executing TRAM:GENERALIZED-AL-PLANS.
                Recalling:    (&GOAL.171 &GOAL.170).
              ...TRAM succeeds: (&GOAL.223 &GOAL.222).
            TRAM Cycle succeeds: (&GOAL.223 &GOAL.222).
Plans       Found plans: ALP:TELL-STORY ALP:FIND-STORY-THEME.

Plan        Trying author-level plan ALP:TELL-STORY.
Execution   Trying author-level plan ALP:FIND-STORY-THEME.
              Recalling using &GOAL.216 as an index: (&GOAL.143).
              Created theme &PAT.8 from &PAT-HASTY.
              Creating author-level goal &TELL-STORY applied to
              &PAT.8.
            Author-level planning succeeded.
            ++++++++++++++++++++++++++++++++++++++++++++++++++
```

`alp:run` is the top-level function used to begin the author-level problem-solving process. `alp:run` repeatedly takes the highest priority goal off the author-level goal agenda and applies author-level problem-solving. In this case, the only goal on the author-level goal agenda is the initial goal to tell a story about the user suggestion, so that is popped off the goal agenda and solved. This process repeates until the goal agenda is empty.

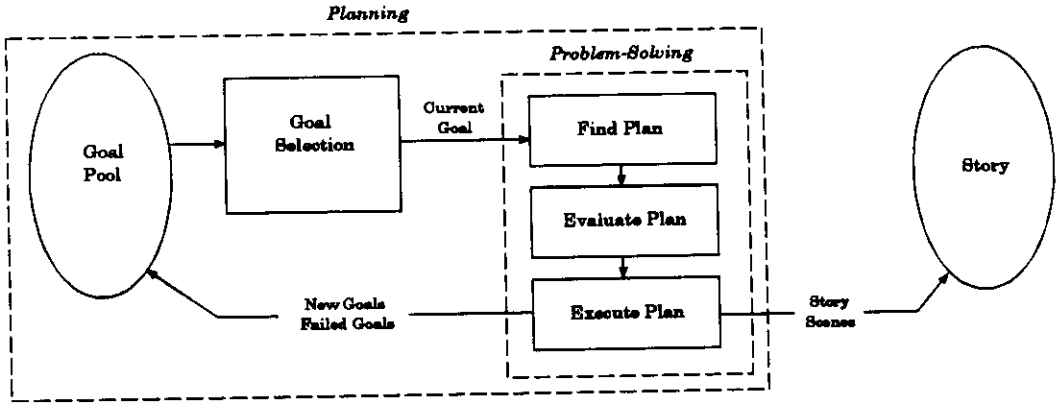


Figure 11.1 Author-Level Planning and Problem-Soving

Figure 11.1 shows the steps of the author-level planning and problem-solving process. The current goal is used to recall similar past problem-solving situations. If recall succeeds, any past plans found are applied to the current problem. If no plans are found, or if all plans fail, then

problem-solving has failed. When problem-solving fails, the priority of the failed goal is reduced and it is placed back on the goal agenda. (For a more complete description of MINSTREL's storytelling process, see Chapter 5.)

The trace of each cycle of the problem-solving process is bordered by two lines of plusses. The first line inside the borders identifies the highest priority goal on the goal agenda. This is the goal MINSTREL is trying to achieve during the current problem-solving cycle:

```
Author-level goal &TELL-STORY applied to &GOAL.216.
```

This line identifies the type of the goal (&TELL-STORY) and the part of the story it is applied to (&GOAL.216). In this case, the goal is the one just created by the user - to tell a story about the user suggestion.

The next portion shows MINSTREL using the current goal to recall possible plans. As discussed in Chapter 5, this involves creativity at the author level. The trace of the creative recall process begins with the line "TRAM Cycle..." and ends with the line "TRAM Cycle succeeds...":

```
TRAM Cycle: &GOAL.217.
   Executing TRAM:AL-STANDARD-PROBLEM-SOLVING.
     Recalling:    NIL.
   ...TRAM failed.
   Executing TRAM:GENERALIZED-AL-PLANS.
     Recalling:    (&GOAL.171 &GOAL.170).
   ...TRAM succeeds: (&GOAL.223 &GOAL.222).
TRAM Cycle succeeds: (&GOAL.223 &GOAL.222).
Found plans: ALP:TELL-STORY ALP:FIND-STORY-THEME.
```

At the author-level, MINSTREL has available two creativity heuristics, TRAM:AL-STANDARD-PROBLEM-SOLVING, and TRAM:GENERALIZED-AL-PLANS. These are used to recall similar past problem-solving situations and the plans that were used on those occasions. TRAM:AL-STANDARD-PROBLEM-SOLVING recalls very similar past problem-solving situations while TRAM:GENERALIZED-AL-PLANS recalls more generalized past situations. (See Chapter 5 for a detailed explanation of TRAM:AL-STANDARD-PROBLEM-SOLVING and TRAM:GENERALIZED-AL-PLANS.) If either heuristic succeeds, the plans associated with the past goals are applied to the current goal.

In this case, TRAM:GENERALIZED-AL-PLANS succeeds in recalling two past problem-solving situations (&GOAL.223 and &GOAL.222). The plans from these past situations (ALP:TELL-STORY and ALP:FIND-STORY-THEME) will now be applied to the current goal.

The creative recall process at the author-level is similar for all of MINSTREL's author-level goals. TRAM:AL-STANDARD-PROBLEM-SOLVING is applied first, and if that heuristic fails to recall any similar past goals, then TRAM:GENERALIZED-AL-PLANS is applied. Although the past problem-solving situations recalled vary depending upon the author-level goal

being solved, the process and the trace of this process remains largely the same. For this reason, the trace of author-level creative recall has been supressed for the remainder of this chapter, leaving only the line that indicates which plans were found.

The final step in the author-level problem-solving process is to apply the plans found until one is found that achieves the current goal (i.e., the goal named in the first line of the trace, in this case, the goal to &TELL-STORY about the user's suggestion):

```
Trying author-level plan ALP:TELL-STORY.
Trying author-level plan ALP:FIND-STORY-THEME.
  Recalling using &GOAL.216 as an index: (&GOAL.143).
  Created theme &PAT.8 from &PAT-HASTY.
  Creating author-level goal #{TELL-STORY} applied to &PAT.8.
Author-level planning succeeded.
```

Lines beginning "Trying..." indicate that MINSTREL is applying the named plan to the current goal situation. Plans are tried in the order they are recalled[1] until one succeeds.

In this case, ALP:TELL-STORY is applied without success. Generally, MINSTREL's author-level plans do not produce any trace when they fail. However, plan failure should be apparent to the reader because another plan is immediately applied. Had the original plan succeeded, author-level planning would have succeeded and there would be no need to try another plan.

Next, ALP:FIND-STORY-THEME is tried. This author-level plan uses the story fragment suggested by the user as an index for recall. If the story fragment recalls a past story, MINSTREL uses the theme from the past story as the theme for a new story.

In this case, the story fragment suggested by the user ("A knight has a thwarted goal" - represented as &GOAL.216) recalls a similar scene from memory (&GOAL.143), as shown by the line:

```
  Recalling using &GOAL.216 as an index: (&GOAL.143).
```

&GOAL.143 represents a small part of a story based upon the theme "Done in haste is done forever":

Thwarted Love[2]

> Once upon a time, Arthur loved Jennifer and wished that she loved him also. Jennifer said that she didn't love Arthur. Arthur was upset. Arthur wanted to kill himself. Arthur hit himself with his sword. When Jennifer saw that Arthur was dying, she told Arthur that she did love him. Arthur regretted that he had killed himself. Arthur died.

1. Plans can also be executed in random order.

The original user suggestion ("A knight has a thwarted goal") recalls the first event in this story, in which Arthur has his goal of being Jennifer's love thwarted. In turn, this story recalls the theme PAT:Hasty-Impulse-Regretted. ALP:FIND-STORY-THEME makes a copy of this abstract theme (&PAT.8) to use as the theme for a new story, and merges the original user suggestion with the recalled story to create the beginnings of a new story:

```
                    The Mistaken Knight 1

    Once upon a time, a knight named Lancelot loved a princess
    named Andrea but his love was thwarted.  This caused him to
    make a hasty decision.

    Later, he discovered that his hasty decision was incorrect.  He
    wished he could take back what he did.  But he couldn't.
```

MINSTREL's internal representation of the story theme is shown in Figure 11.2. For the reader's convenience, the actors are shown as &Lancelot and &Andrea; in fact they are represented by schemas named &HUMAN.138 and &HUMAN.137. As this figure shows, the initial story skeleton consists of a largely uninstantiated theme. The merging of the reminded story, the user's initial suggestion and the theme has resulted in the knight's thwarted goal being instantiated as a goal to achieve love with Andrea (&GOAL.224). Otherwise, the theme remains uninstantiated.

After creating this new story skeleton based upon the user's suggestion and the recalled theme, MINSTREL creates an author-level goal to tell a story based upon this skeleton:

```
  Creating author-level goal &TELL-STORY applied to &PAT.8.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++
```

This concludes this cycle of the author-level problem-solving process. However, ALP:FIND-STORY-THEME created another author-level goal (to tell a story about &PAT.8), so the author-level goal queue is no longer empty. The top priority goal in the queue is popped and problem-solving repeats on this new goal:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &TELL-STORY applied to &PAT.8.
Found plans: ALP:TELL-STORY ALP:FIND-STORY-THEME.
Trying author-level plan ALP:TELL-STORY.
  Creating goal &INSTANTIATE applied to &BELIEF.28.
  Creating goal &CONNECT applied to &PAT.8.
  Creating goal &INSTANTIATE applied to &BELIEF.29.
  Creating goal &CHECK-STORY-FOR-SUSPENSE applied to &STORY.1.
  Creating goal &CHECK-STORY-FOR-TRAGEDY applied to &STORY.1.
```

2. This story has a conceptual representation in MINSTREL's episodic memory. The English version shown here was prepared by the author.

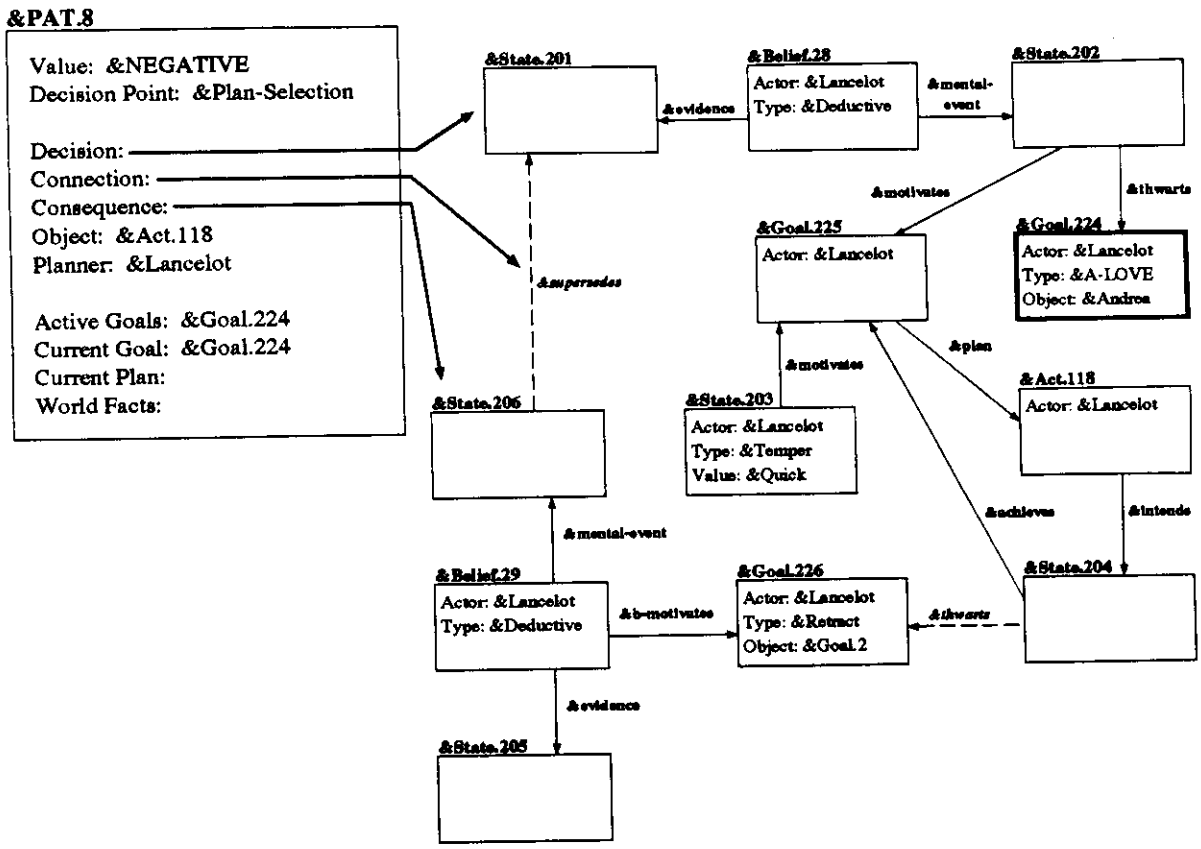


Figure 11.2 The Mistaken Knight 1

```
  Creating goal &CHECK-STORY-FOR-FORESHADOWING to &STORY.1.
  Creating goal &CHECK-STORY-FOR-CHARACTERIZATION to &STORY.1.
  Creating goal &ADD-STORY-INTROS applied to &STORY.1.
  Creating goal &ADD-DENOUEMENTS applied to &STORY.1.
  Creating goal &GENERATE-ENGLISH applied to &STORY.1.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

This goal is the same type as the previous goal (&TELL-STORY) but applied to a new type of object: a story theme (&PAT.8). The same author-level plans are applied, but this time the first plan (ALP:TELL-STORY) succeeds. ALP:TELL-STORY tells a story about a particular story theme by creating goals to:

1. Instantiate the parts of the theme (&BELIEF.28, &BELIEF.29).

2. Achieve dramatic writing goals (suspense, tragedy, etc.)

3. Achieve presentation goals (add story introductions, denouements, etc.)

These goals are prioritized in the order listed and are added to the author-level goal queue.

11.3 Instantiating the Theme

ALP:TELL-STORY creates three goals to instantiate the story theme. Two of these goals are to instantiate the Decision (&BELIEF.28) and Consequence (&BELIEF.29) of the theme, and the third is to create the Connection between the Decision and the Consequence.

As shown in Figure 11.2, the Decision portion of the theme "Done in haste is done forever" involves an actor who makes a hasty decision based upon a mistaken belief. MINSTREL's first goal in instantiating the theme is to instantiate this belief. The first plan tried to achieve this goal is ALP:GENERAL-INSTANTIATE:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &BELIEF.28.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE...
Trying author-level plan ALP:DONT-INSTANTIATE.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
TRAM Cycle: &BELIEF.28.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    NIL.
   ...TRAM failed.
   Executing TRAM:GENERALIZE-ACTOR.
     Recalling:    (&FIGHT2 &KNIGHT-FIGHT).
     Recalling:    NIL.
   [TRAM Recursion: &BELIEF.59.]
     [...]
   ...TRAM failed.
   Executing TRAM:LIMITED-RECALL.
     Recalling:    NIL.
   [TRAM Recursion: &BELIEF.73.]
      Executing TRAM:STANDARD-PROBLEM-SOLVING.
        Recalling: NIL.
      ...TRAM failed.
   ...TRAM failed.
TRAM Cycle fails.
ALP:GENERAL-INSTANTIATE failed.
```

ALP:GENERAL-INSTANTIATE uses the object being instantiated as an index for creative recall. If something similar can be recalled or invented, the recalled object can be used to fill in (instantiate) the object from the current story. In this case, despite applying a number of TRAMs, MINSTREL can neither recall nor invent a suitable belief. So ALP:GENERAL-INSTANTIATE fails. (For the remainder of this chapter, when creativity fails the detailed trace of the creative cycle will be deleted to save space.)

```
Trying author-level plan ALP:INSTANTIATE-AFFECT.
```

```
Trying author-level plan ALP:INSTANTIATE-DECEPTION.
Trying author-level plan ALP:INSTANTIATE-REVENGE.
Trying author-level plan ALP:INSTANTIATE-BELIEF.
  Creating goal &INSTANTIATE applied to &STATE.202.
  Creating goal &INSTANTIATE applied to &STATE.201.
  Creating goal &CHECK-NEW-SCENE applied to &BELIEF.28.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.202.
  Creating goal &CHECK-NEW-SCENE applied to &GOAL.225.
  Creating goal &CHECK-NEW-SCENE applied to &ACT.118.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.204.
  Creating goal &CHECK-NEW-SCENE applied to &HUMAN.138.
  Creating goal &CHECK-NEW-SCENE applied to &GOAL.224.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.207.
  Creating goal &CHECK-NEW-SCENE applied to &HUMAN.137.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.203.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.201.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

A number of other plans are recalled but do not apply to the current goal. These plans are ALP:INSTANTIATE-AFFECT, ALP:INSTANTIATE-DECEPTION and ALP:INSTANTIATE-REVENGE. Each of these plans is a specialized author-level plan for instantiating a particular kind of story situation (i.e., a deception, revenge, or affect). In this case, none of these plans apply because the particular story scene being instantiated (&BELIEF.28) is not a deception, revenge or affect.

It should be noted that MINSTREL does not execute inapplicable plans. Each plan has a test condition which determines whether or not the plan is applicable to a particular author-level goal. Before plans are applied, the test condition is executed to determine whether or not the plan is applicable to the current goal. If the test condition fails, the plan is discarded without being executed. In this way, MINSTREL avoids doing unnecessary work. In this situation, the test conditions of ALP:INSTANTIATE-AFFECT, ALP:INSTANTIATE-DECEPTION and ALP:INSTANTIATE-REVENGE all fail (because none of these plans apply to the goal of instantiating a belief).

Eventually, ALP:INSTANTIATE-BELIEF is applied and succeeds. ALP:INSTANTIATE-BELIEF is an author-level plan specifically for instantiating beliefs. Like ALP:TELL-STORY, ALP:INSTANTIATE-BELIEF instantiates a belief by creating author-level goals to instantiate the parts of the belief. The parts of a belief are the mental-event (i.e., the thing believed) and the evidence for the belief. In this case, the mental-event is represented by &STATE.202 and the evidence by &STATE.201, so ALP:INSTANTIATE-BELIEF creates author-level goals to instantiate these two schemas. The remainder of the goals created by ALP:INSTANTIATE-BELIEF are author-level goals to check the consistency of all of the schemas which make up &BELIEF.28. This ensures that after the belief is instantiated it will be checked for consistency.

The new author-level goal to instantiate the mental-event of the belief is now the top goal in the

author-level goal queue:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &STATE.202.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE...
Trying author-level plan ALP:DONT-INSTANTIATE.
 [...]
Trying author-level plan ALP:INSTANTIATE-THWARTING-STATE.
  Copying achieving state into thwarting state.
  Replacing &HUMAN.138 with &HUMAN.274.
  Instantiated object is &STATE.202.

  (STATE &STATE.202
     :TYPE       &AFFECT
     :VALUE      &LOVE
     :OBJECT     &HUMAN.137
     :TO         &HUMAN.274
     &MOTIVATES <==>   (&GOAL.225)
     &EVENT-IN <==>    (&BELIEF.28)
     &THWARTS <==>     (&GOAL.224))

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

&STATE.202 is the mental-event portion of &BELIEF.28 (see Figure 11.2). &STATE.202 represents the state of the world that thwarts Lancelot's initial goal to have Andrea love him. After a number of plans fail to instantiate this schema, a plan called ALP:INSTANTIATE-THWARTING-STATE succeeds.

ALP:INSTANTIATE-THWARTING-STATE is an author-level plan that instantiates a state which thwarts a goal by "reversing" the goal. That is, MINSTREL knows that many goals can be thwarted by states which are the reverse of the state which would achieve the goal. For example, possession of an object is thwarted if someone else possesses the object, the goal to be somewhere is thwarted by being somewhere else, and so on. ALP:INSTANTIATE-THWARTING-STATE instantiates a thwarting state by making a copy of the state that would achieve the goal and then "reversing" the copy.

The first step of ALP:INSTANTIATE-THWARTING-STATE is to determine what state would achieve the thwarted goal. In this case the thwarted goal is "Lancelot wants to achieve love with Andrea" and MINSTREL determines that the state "Andrea loves Lancelot" would achieve this goal[3]. Once this is determined, the achieving state is copied into the thwarting state (i.e., &STATE.202, the state being instantiated).

3. There are a number of ways to determine this: (1) from the :NEW-STATE slot of the goal being thwarted, which contains an explicit representation of the state which would achieve the goal, (2) from semantic knowledge of goals or (3) by using episodic memory to recall what states have achieved similar goals in the past. In this case, MINSTREL uses method (1).

The next step is to turn the achieving state into a thwarting state by "reversing" the state. ALP:INSTANTIATE-THWARTING-STATE knows that if a state is directed toward something, then the state can be reversed by directing it toward a different but similar object. In this case, the achieving state represents Andrea's love directed toward Lancelot, so the achieving state can be turned into a thwarting state by replacing Lancelot with another character. MINSTREL invents a new, uninstantiated character (&HUMAN.274) and replaces Lancelot with this new character, creating the new thwarting state "Andrea loves some other person."

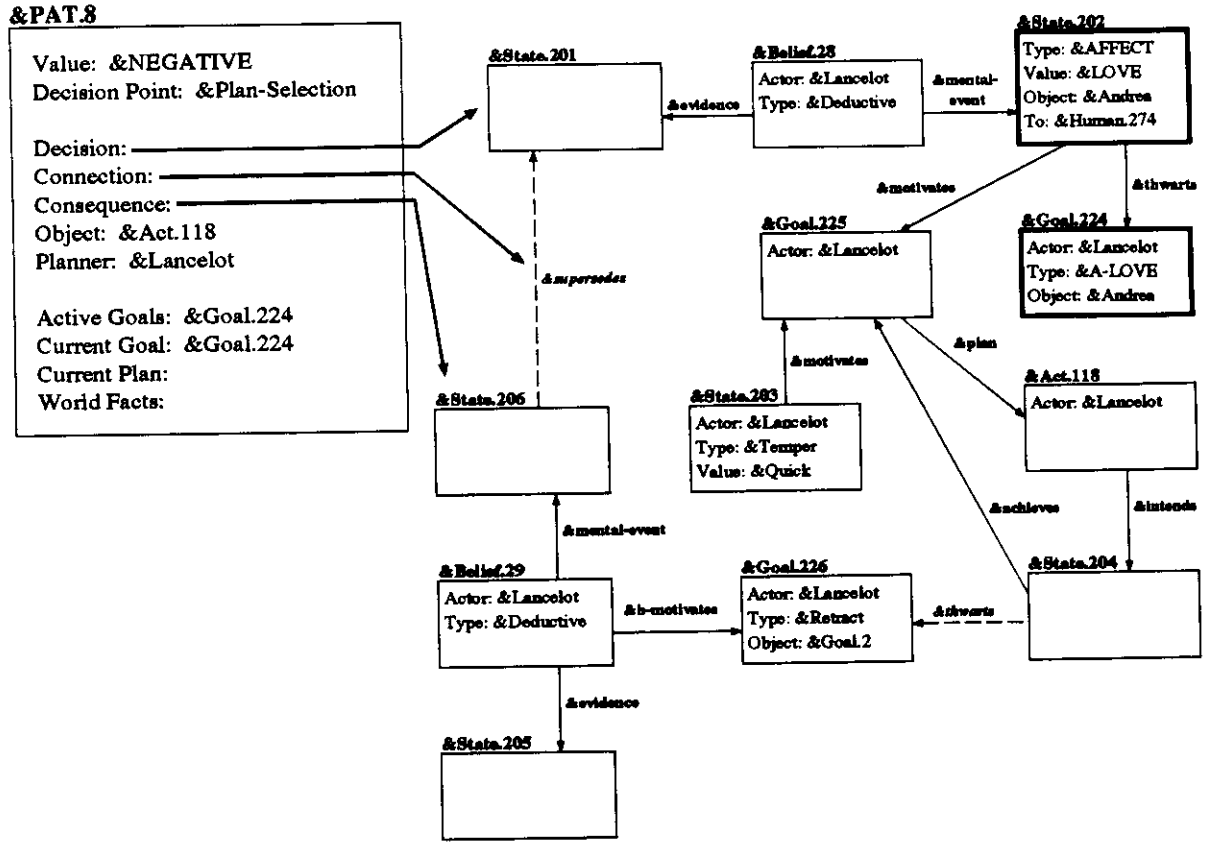


Figure 11.3 The Mistaken Knight 2

The end result of this plan is that the state that thwarts Lancelot's goal to achieve love with Andrea has been instantiated as "Lancelot loves some other person". This is shown in Figure 11.3.

The thwarting state was the mental-event portion of the belief which makes up the Decision portion of PAT:Hasty-Impulse-Regretted. The next author-level goal is to instantiate the evidence for this belief:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &STATE.201.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE...
```

```
Trying author-level plan ALP:DONT-INSTANTIATE.
 [...]
Trying author-level plan ALP:INSTANTIATE-EVIDENCE.
Initial motivation chain is &GOAL.483.

(GOAL &GOAL.483
   :ACTOR          &HUMAN.137
   &MOTIVATED-BY <==>   (STATE &STATE.694
                               :TYPE       &AFFECT
                               :VALUE      &LOVE
                               :OBJECT     &HUMAN.137
                               :TO         &HUMAN.274))
```

To instantiate the evidence for the belief, MINSTREL applies ALP:INSTANTIATE-EVIDENCE. ALP:INSTANTIATE-EVIDENCE is an author-level plan that invents evidence for a belief by generating a motivational chain from the belief to some action that can be observed. For example, to generate evidence for the belief "The dragon is going to eat Andrea" ALP:INSTANTIATE-EVIDENCE might reason that "The dragon is going to eat Andrea" could motivate the goal "Andrea wants to be away from the dragon" which in turn could be achieved by the action "Andrea runs away". From this it can be deduced that seeing Andrea running away from the dragon is reasonable evidence to support the belief "The dragon is going to eat Andrea":

Belief	A dragon is going to eat Andrea.
↓	↓
Intermediate Goal	Andrea wants to be away from the dragon.
↓	↓
Evidence	Andrea runs away from the dragon.

Thus ALP:INSTANTIATE-EVIDENCE invents reasonable evidence by starting at the belief and building a motivational chain until it reaches an action or result that can be observed.

In this case, ALP:INSTANTIATE-EVIDENCE is trying to create a motivation chain beginning with "Andrea loves a person". To do this, ALP:INSTANTIATE-EVIDENCE tries to find a goal this state could motivate, and then an act that would achieve that goal. If this is possible the act (not the intermediate goal) can serve as evidence for the belief.

The first step is to find an intermediate goal that could be motivated by the belief. To find a goal that "Andrea loves a person" could motivate, ALP:INSTANTIATE-EVIDENCE begins by building an uninstantiated goal motivated by this state. In the trace, this is shown as &GOAL.483:

```
Initial motivation chain is &GOAL.483.

(GOAL &GOAL.483
   :ACTOR          &HUMAN.137
```

```
&MOTIVATED-BY <==>   (STATE &STATE.694
                          :TYPE       &AFFECT
                          :VALUE      &LOVE
                          :OBJECT     &HUMAN.137
                          :TO         &HUMAN.274))
```

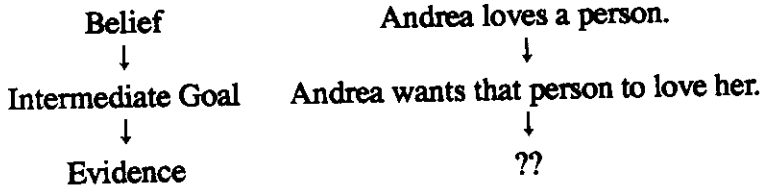
&GOAL.483 is a goal belonging to &HUMAN.137 (Andrea). The goal is motivated by Andrea's love for &HUMAN.274, but is otherwise uninstantiated. To instantiate this goal, ALP:INSTANTIATE-EVIDENCE uses this goal as an index to imaginative memory. If a similar past goal (i.e., a goal motivated by love) can be recalled or invented, then the past goal can be used to instantiate this intermediate goal in the motivation chain. The next portion of the trace shows the recall process:

```
Imaginative recall using index &GOAL.483:
 TRAM Cycle: &GOAL.483.
    Executing TRAM:STANDARD-PROBLEM-SOLVING.
      Recalling:     (&AR-AFFECTION).
    ...TRAM succeeds: (&AR-AFFECTION).
 TRAM Cycle succeeds: (&AR-AFFECTION).
Reminding is &AR-AFFECTION.

(GOAL &AR-AFFECTION
   :TYPE          &A-LOVE
   :ACTOR         &HUMAN.361
   :OBJECT        &HUMAN.362
   :NEW-STATE     &STATE.698
   &MOTIVATED-BY <==>   (&STATE.699)
   &PLAN <==>           (&ACT.212)
   &ACHIEVED-BY <==>    (&STATE.698))
```

&GOAL.483 recalls &AR-AFFECTION from MINSTREL's episodic memory. &AR-AFFECTION represents the following scene:

> Motivated by her love of a knight, a princess wanted to achieve the love of a knight, so she kissed him.

Recall succeeded in finding a past goal motivated by love. This reminding can now be used to instantiate the goal motivated by "Andrea loves a person" as "Andrea wants the person to love her", i.e., loving someone motivates you to want them to love you back.



The next step in building the motivational chain is to find an action that can achieve the intermediate goal. Fortuitously, the reminding found for the previous step already contains a plan. In the reminding, the plan for achieving the goal of having someone love you is to kiss that person. This same action can be used in the current motivation chain:

Belief	Andrea loves a person.
↓	↓
Intermediate Goal	Andrea wants that person to love her.
↓	↓
Evidence	Andrea kisses that person.

The trace shows the representation of evidence after this instantiation:

```
Evidence after instantiation:

(STATE 'STATE.201
  :OBJECT  '&HUMAN.137
  :TO  '&HUMAN.274
  :TYPE  &KISS
  &EVIDENCE-FOR  '&BELIEF.28 )

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 11.4 shows the state of theme after this instantiation.

There are two interesting things about this instantiation. The first is that the intermediate goal does not appear as part of the story. It exists only temporarily during the execution of ALP:INSTANTIATE-EVIDENCE. In the final story there is no mention of Andrea's goal to have Frederick love her, or how that goal connects the evidence and the belief. Instead, the reader is presented with the belief and the evidence and reconstructs the motivation chain himself.

The second item of note is that instantiating the evidence has the side-effect of instantiating the character that Andrea loves. In the reminding, the princess loves a knight, and this information is copied over into the person Andrea loves in the story, resulting in &HUMAN.274 being instantiated as a knight:

```
(HUMAN 'HUMAN.274
  :SEX   'MALE
  :NAME  'FREDERICK
  :TYPE  &KNIGHT)
```

(The name "Frederick" is chosen randomly from a list of male character names.)

MINSTREL has now instantiated the belief that makes up the first part of the plot of *The Mis-*

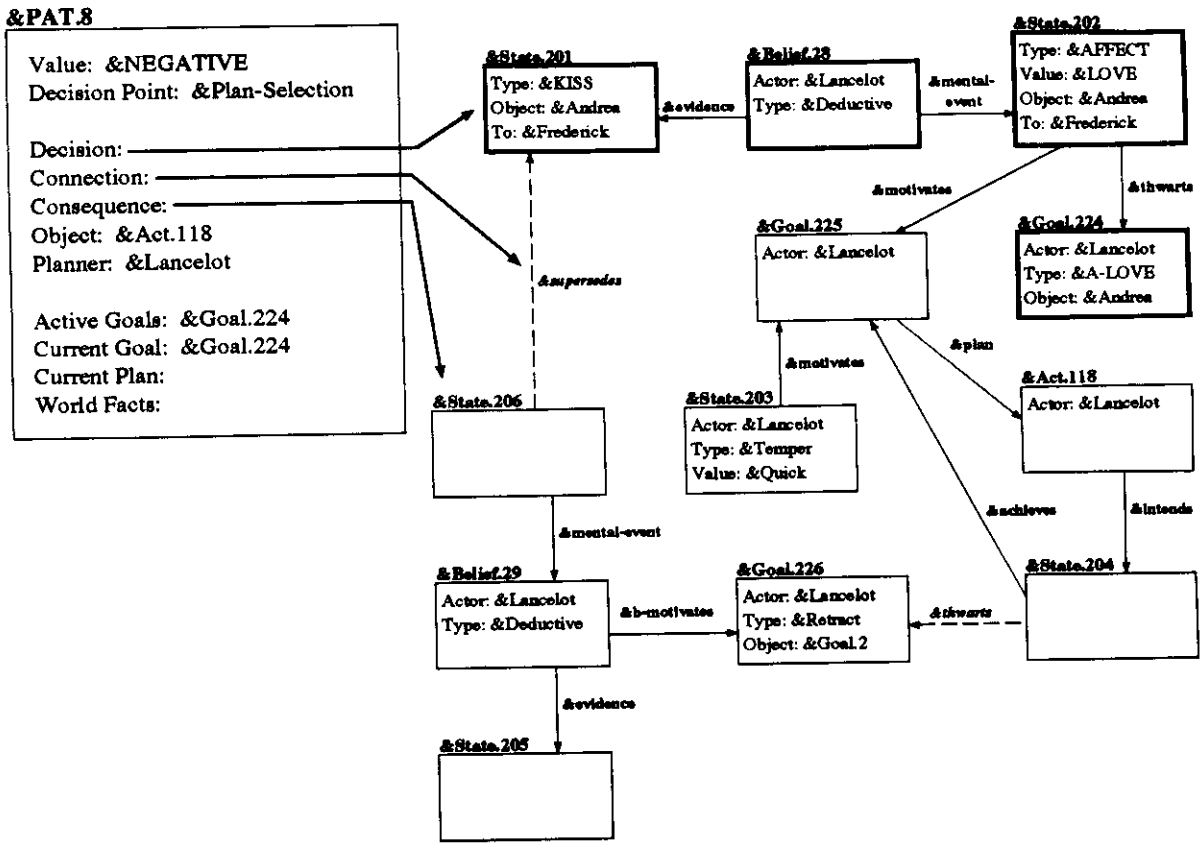


Figure 11.4 The Mistaken Knight 3

taken Knight. The story so far is:

The Mistaken Knight 3

Once upon a time, a knight named Lancelot loved a princess named Andrea. Lancelot saw Andrea kiss a knight named Frederick. Lancelot believed that Andrea loved Frederick. Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. This caused him to make a hasty decision.

Later, he discovered that his hasty decision was incorrect. He wished he could take back what he did. But he couldn't.

With the completion of the goals to instantiate the belief that makes up the first half of the theme, the lower priority goals to check the consistency of the instantiated belief now rise to the top of the author-level goal queue. These goals are achieved by ALP:CHECK-NEW-SCENE:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-NEW-SCENE applied to &BELIEF.28.
Found plans: ALP:CHECK-NEW-SCENE
Trying author-level plan ALP:CHECK-NEW-SCENE.
  Creating goal &CHECK-CONSISTENCY applied to &BELIEF.28.
  Creating goal &CHECK-AFFECTS applied to &BELIEF.28.
  Creating goal &CHECK-SCENE-FOR-SUSPENSE applied to &BELIEF.28.
  Creating goal &CHECK-SCENE-FOR-TRAGEDY applied to &BELIEF.28.
  Creating goal &CHECK-SCENE-FOR-FORESHADOWING applied to
     &BELIEF.28.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-NEW-SCENE applied to &STATE.202.
Found plans: ALP:CHECK-NEW-SCENE
Trying author-level plan ALP:CHECK-NEW-SCENE.
  Creating goal &CHECK-CONSISTENCY applied to &STATE.202.
  Creating goal &CHECK-AFFECTS applied to &STATE.202.
  Creating goal &CHECK-SCENE-FOR-SUSPENSE applied to &STATE.202.
  Creating goal &CHECK-SCENE-FOR-TRAGEDY applied to &STATE.202.
  Creating goal &CHECK-SCENE-FOR-FORESHADOWING applied to
     &STATE.202.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-NEW-SCENE applied to &STATE.201.
Found plans: ALP:CHECK-NEW-SCENE
Trying author-level plan ALP:CHECK-NEW-SCENE.
  Creating goal &CHECK-CONSISTENCY applied to &STATE.201.
  Creating goal &CHECK-AFFECTS applied to &STATE.201.
  Creating goal &CHECK-SCENE-FOR-SUSPENSE applied to &STATE.201.
  Creating goal &CHECK-SCENE-FOR-TRAGEDY applied to &STATE.201.
  Creating goal &CHECK-SCENE-FOR-FORESHADOWING applied to
     &STATE.201.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

In each case, ALP:CHECK-NEW-SCENE checks the consistency of a story scene by creating a number of subgoals. These goals include a goal to instantiate the scene (if it has not already been instantiated), to check the (causal) consistency of the scene, to check the affect consistency of the scene, and to check various author-level dramatic writing goals.

This plan is applied to &BELIEF.28 and all its parts. These include the mental-event and the evidence (&STATE.201 and &STATE.202 as shown above) as well all other schemas attached to the evidence and mental-event via causal links (see Figure 11.4: &GOAL.225, &STATE.203, &GOAL.224, &ACT.118, and &STATE.204). One of these attached schemas is &GOAL.225:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-NEW-SCENE applied to &GOAL.225.
Found plans: ALP:CHECK-NEW-SCENE
Trying author-level plan ALP:CHECK-NEW-SCENE.
  Creating goal &INSTANTIATE applied to &GOAL.225.
  Creating goal &CHECK-CONSISTENCY applied to &GOAL.225.
  Creating goal &CHECK-AFFECTS applied to &GOAL.225.
  Creating goal &CHECK-SCENE-FOR-SUSPENSE applied to &GOAL.225.
  Creating goal &CHECK-SCENE-FOR-TRAGEDY applied to &GOAL.225.
  Creating goal &CHECK-SCENE-FOR-FORESHADOWING
      applied to &GOAL.225.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

&GOAL.225 is Lancelot's goal that is motivated by the state "Andrea loves Frederick" (which in turn thwarts Lancelot's goal of achieving love with Andrea). The top priority goal created by ALP:CHECK-NEW-SCENE (above) is to instantiate &GOAL.225, so this goal is popped off the priority queue and solved:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &GOAL.225.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE...
Trying author-level plan ALP:DONT-INSTANTIATE.
 [...]
Trying author-level plan ALP:INSTANTIATE-UNTHWARTS.

To unthwart a goal, kill the actor
who is thwarting the goal: &HUMAN.274.

(GOAL &GOAL.225
   :TYPE          &DESTROY
   :ACTOR         &HUMAN.138
   :OBJECT        &HUMAN.274
   :NEW-STATE     &STATE.204
   &PLAN <==>             (&ACT.118)
   &ACHIEVED-BY <==>      (&STATE.204)
   &SUBGOAL-OF <==>       (&GOAL.224)
   &MOTIVATED-BY <==>     (&STATE.202 &STATE.203))

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ALP:INSTANTIATE-UNTHWARTS is applied. ALP:INSTANTIATE-UNTHWARTS is an author-level plan that knows how to unthwart a goal. It applies in this case because the goal being instantiated (&GOAL.225) is motivated by a state (&STATE.202) that thwarts a goal (&GOAL.224). If something occurs that thwarts an actor's goal, then that can motivate the actor to unthwart his goal. ALP:INSTANTIATE-UNTHWARTS will try to instantiate

&GOAL.225 as something that will unthwart &GOAL.224.

ALP:INSTANTIATE-UNTHWARTS captures the knowledge that one way to unthwart a goal is to kill the person who is thwarting the goal, i.e., to unthwart Lancelot's goal to have Andrea love him, kill the actor who is thwarting the goal: Frederick. So ALP:INSTANTIATE-UNTHWARTS instantiates &GOAL.225 as a goal to &DESTROY Frederick, which (by means of some as yet uninstantiated action) results in Frederick being dead. The state of the theme after this instantiation is shown in Figure 11.5.

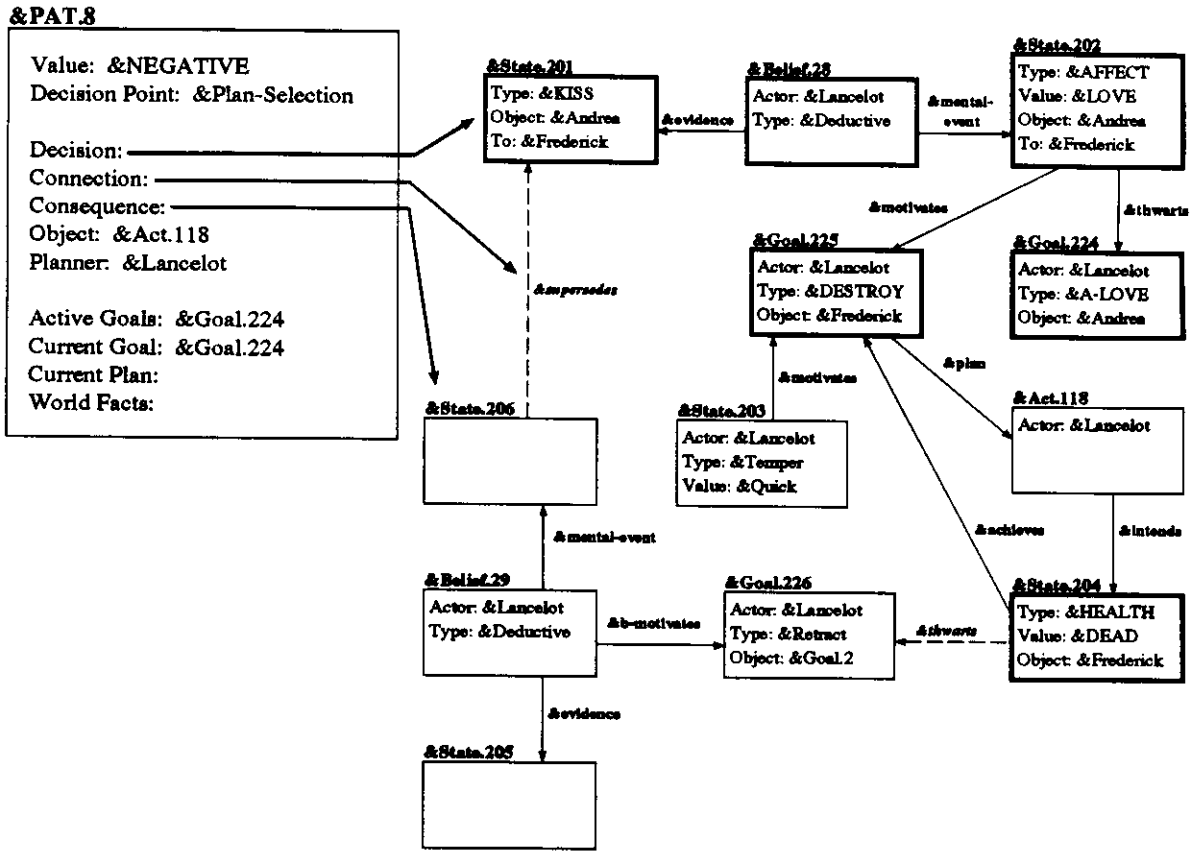


Figure 11.5 The Mistaken Knight 4

MINSTREL's attention now turns to the act that achieves Lancelot's newly instantiated goal to destroy Frederick. The top goal on the author goal queue is to check this scene. Achieving this puts the goal to instantiate this scene next on the queue:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &ACT.118.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE...
Trying author-level plan ALP:DONT-INSTANTIATE.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
```

```
Imaginative recall using index: &ACT.118.
 TRAM Cycle: &ACT.118.
   [...]
   Executing TRAM:GENERALIZE-ACTOR.
      Recalling: NIL.
      [TRAM Recursion: &ACT.280.]
        [...]
        Executing TRAM:SIMILAR-OUTCOMES.
        ...TRAM succeeds: (&ACT.375).
TRAM Cycle succeeds: (&ACT.376).

Found a reminding for instantiation: &ACT.376.

(ACT &ACT.376
   :AT       &SETTING.42
   :TO       &HUMAN.706
   :ACTOR    &HUMAN.705
   :TYPE     &M-FIGHT
   &PLAN-OF <==>   (&GOAL.756)
   &INTENDS <==>   (&STATE.1083))
```

&ACT.118 is instantiated by the author-level plan ALP:GENERAL-INSTANTIATE. This plan tries to instantiate a schema by using it as an index to creative recall. If a similar schema can be recalled or invented, the recalled schema can be used to fill out the current schema.

In this case, MINSTREL is not initially able to recall any schemas similar to &ACT.118. But by using two creativity heuristics (TRAM:GENERALIZE-ACTOR and TRAM:SIMILAR-OUTCOMES), MINSTREL is able to invent a similar schema, and use this to fill in &ACT.118.

The initial index for recall is the schema being instantiated, &ACT.118, which has links to two other schemas, &GOAL.225 and &STATE.204:

```
(GOAL &GOAL.225
  :TYPE   &DESTROY
  :ACTOR   &HUMAN.138
  :OBJECT   &HUMAN.274
  &PLAN   &ACT.118)

(ACT &ACT.118
  :ACTOR   &HUMAN.138
  &PLAN-OF   &GOAL.225
  &INTENDS   &STATE.204)

(STATE &STATE.204
  :TYPE   &HEALTH
  :VALUE   &DEAD
  :OBJECT   &HUMAN.274
```

```
  &INTENDED-BY  &ACT.118)
```

&ACT.118 is Lancelot's plan to achieve the goal of destroying Frederick, which will be achieved by Frederick's death. Together these schemas represent "Lancelot does something to achieve his goal of killing Frederick by making Frederick dead." But when &ACT.118 is used as an index for recall, recall fails, because MINSTREL does not know anything about knights killing other knights.

When recall fails, creativity is tried. A creativity heuristic is applied which transforms the recall index. If recall is successful, the same creativity heuristic will adapt any recalled solutions back to the original problem. The heuristic applied in this case is TRAM:GENERALIZE-ACTOR.

TRAM:GENERALIZE-ACTOR is a creativity heuristic which generalizes the actor of the recall index. If recall succeeds with the new, modified index, then TRAM:GENERALIZE-ACTOR adapts any recalled episodes back to the original problem by replacing the generalized actor with the original actor. (For a more detailed description of TRAM:GENERALIZE-ACTOR, see 4.3.2.)

In this example, TRAM:GENERALIZE-ACTOR replaces Lancelot (the actor of &ACT.118) with a monster. (By noticing that both knights and monsters engage in violent acts, TRAM:GENERALIZE-ACTOR reasons that they are similar enough to interchange in many situations.) This replacement is also done on all the schemas connected to &ACT.118:

```
(GOAL &GOAL.225
  :TYPE  &DESTROY
  :ACTOR  &MONSTER.221
  :OBJECT  &HUMAN.274
  &PLAN  &ACT.118)

(ACT &ACT.118
  :ACTOR  &MONSTER.221
  &PLAN-OF  &GOAL.225
  &INTENDS  &STATE.204)

(STATE &STATE.204
  :TYPE  &HEALTH
  :VALUE  &DEAD
  :OBJECT  &HUMAN.274
  &INTENDED-BY  &ACT.118)
```

The new recall index represents "A monster wanted to destroy Frederick and did something to achieve that goal." This also fails to recall anything, because MINSTREL does not have any episodes in memory in which a monster kills a knight.

Since recall has failed, creativity is again applied. A new TRAM is used, TRAM:SIMILAR-OUTCOMES. TRAM:SIMILAR-OUTCOMES is a creativity heuristic which modifies the in-

tended result of an action. TRAM:SIMILAR-OUTCOMES replaces the intended result of an action with a similar but different outcome. (For a more detailed description of TRAM:SIMILAR-OUTCOMES, see 4.4.2.)

In this example, TRAM:SIMILAR-OUTCOMES reasons that being wounded is similar to being killed, and replaces the death of Frederick with the wounding of Frederick:

```
(GOAL &GOAL.225
  :TYPE  &DESTROY
  :ACTOR  &MONSTER.221
  :OBJECT  &HUMAN.274
  &PLAN  &ACT.118)

(ACT &ACT.118
  :ACTOR  &MONSTER.221
  &PLAN-OF  &GOAL.225
  &INTENDS  &STATE.204)

(STATE &STATE.204
  :TYPE  &HEALTH
  :VALUE  &WOUNDED
  :OBJECT  &HUMAN.274
  &INTENDED-BY  &ACT.118)
```

The twice-modified index represents "A monster wanted to kill a knight and did something which wounded the knight". This index succeeds in recalling a scene from MINSTREL's episodic memory:

> One day, a dragon wanted to kill a knight so it attacked the knight. The dragon wounded the knight but was killed by the knight.

In this scene, a dragon tries to kill a knight and only wounds him, matching the modified index. The next step is for TRAM:SIMILAR-OUTCOMES and TRAM:GENERALIZE-ACTOR to adapt this recalled episode to the original problem.

TRAM:SIMILAR-OUTCOMES begins the adaptation by changing the wounding in the recalled episode back to death:

> One day, a dragon wanted to kill a knight so it attacked the knight in the woods. The dragon killed the knight but was killed by the knight.

Next TRAM:GENERALIZE-ACTOR replaces the generalized actor (a monster) with the actor from the original index for recall:

> One day, Lancelot wanted to kill a knight so he attacked the knight in the woods. Lancelot killed the knight but was killed by the knight.

Now the adapted recalled episode can be used to fill in &ACT.118 as an attack on Frederick by Lancelot. The extraneous events in the recalled episode (namely Lancelot also dying in the attack) can be ignored because they do not match events already existing in the story. The instantiated result of &ACT.118 is:

```
After instantiation: &ACT.118.

(ACT &ACT.118
   :AT       &SETTING.43
   :TO       &HUMAN.274
   :TYPE     &M-FIGHT
   :ACTOR    &HUMAN.138
   &PLAN-OF <==>   (&GOAL.225)
   &INTENDS <==>   (&STATE.204))
```

This represents "Lancelot fought Frederick in the woods." By using creative recall, ALP:GENERAL-INSTANTIATE is able to invent an act by which Lancelot can kill Frederick, even though prior to telling this story MINSTREL knew nothing about knights fighting knights.

The final step of ALP:GENERAL-INSTANTIATE is to create a number of author-level goals to check the scenes just created:

```
  Creating goal &CHECK-NEW-SCENE applied to &STATE.204.
  Creating goal &CHECK-NEW-SCENE applied to &HUMAN.274.
  Creating goal &CHECK-NEW-SCENE applied to &GOAL.224.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.201.
  Creating goal &CHECK-NEW-SCENE applied to &HUMAN.137.
  Creating goal &CHECK-NEW-SCENE applied to &HUMAN.138.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.207.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.203.
  Creating goal &CHECK-NEW-SCENE applied to &SETTING.43.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 11.6 shows the state of the theme after this instantiation.

MINSTREL has now finished instantiating the Decision portion of the theme. At the top of the queue now are a number of author-level goals to check the scenes in the Decision portion of the queue. The top goal happens to be a goal to check &STATE.204, which represents Frederick's death:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-NEW-SCENE applied to &STATE.204.
Found plans: ALP:CHECK-NEW-SCENE
Trying author-level plan ALP:CHECK-NEW-SCENE.
  Creating goal &INSTANTIATE applied to &STATE.204.
  Creating goal &CHECK-CONSISTENCY applied to &STATE.204.
```

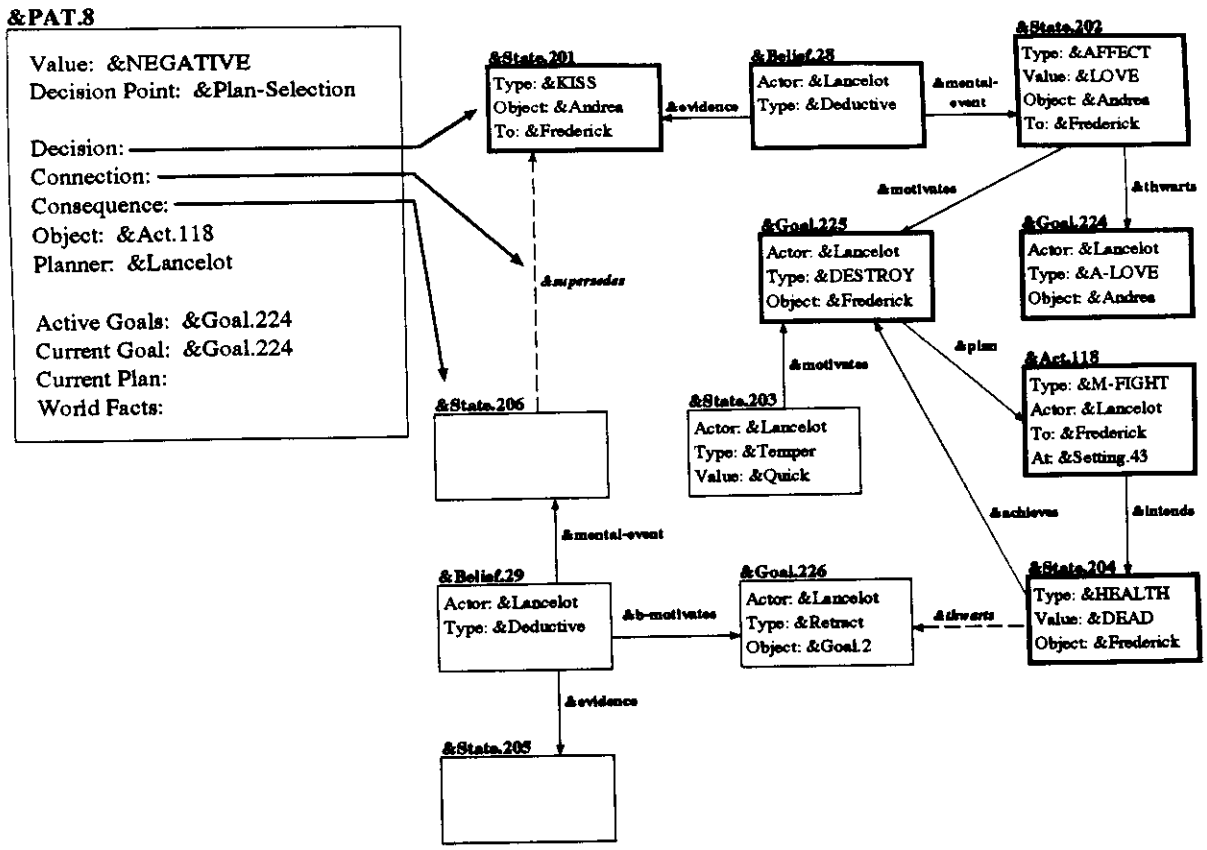


Figure 11.6 The Mistaken Knight 5

```
  Creating goal &CHECK-AFFECTS applied to &STATE.204.
  Creating goal &CHECK-SCENE-FOR-SUSPENSE applied to &STATE.204.
  Creating goal &CHECK-SCENE-FOR-TRAGEDY applied to &STATE.204.
  Creating goal &CHECK-SCENE-FOR-FORESHADOWING
            applied to &STATE.204.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &STATE.204.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE...
Trying author-level plan ALP:DONT-INSTANTIATE.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

The goal to check this scene is achieved by ALP:CHECK-NEW-SCENE, which creates a number of subgoals. The first of these is to instantiate &STATE.204. But &STATE.204 has been previously instantiated (as a side-effect of instantiating &GOAL.225), so this goal is trivially achieved by ALP:DONT-INSTANTIATE, an author-level goal that simply checks to see

whether or not a scene has already been instantiated. The remainder of the subgoals for &STATE.204 are lower priority, and will rise to the top of the queue later in this trace.

Similar author-level goals are achieved for all the other schemas in the Decision part of the theme, including: &HUMAN.138, &GOAL.224, &STATE.207, &HUMAN.137, &STATE.203, &STATE.201, &HUMAN.274, &SETTING.43, and &STATE.204. The author-level goal to check each scene rises to the top of the queue, a subgoal to instantiate the scene is created, and the subgoal is achieved by ALP:DONT-INSTANTIATE. The trace of this is omitted.

After these goals are achieved, the author-level goals to check the consistency of the scenes in the Decision portions of the theme rise to the top of the author-level goal queue. The trace of these goals will be temporarily delayed so that we can show how the remainder of the theme becomes instantiated.

The next theme-related goal MINSTREL achieves is the goal to connect the two halves of the theme. This is achieved by a simple author-level plan that creates the causal links connecting the two halves of the theme. In this theme, there are two connections: a supersedes link between the mental-events of the two beliefs and a thwarts link between &STATE.204 and &GOAL.226 (see Figure 11.6).

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CONNECT applied to &PAT.8.
Found plans: ALP:CONNECT
Trying author-level plan ALP:CONNECT.
  Creating &SUPERSEDES link between &STATE.206 and &STATE.201.
  Creating &THWARTS link between &STATE.204 and &GOAL.226.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Next, MINSTREL begins instantiating the Consequence portion of the story theme. As it happens, the Consequence portion of this theme is also a belief. This new belief will supersede the belief just instantiated as the Decision of the theme. As with that belief, the goal to instantiate this belief is achieved by ALP:INSTANTIATE-BELIEF:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &BELIEF.29.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE...
Trying author-level plan ALP:DONT-INSTANTIATE.
 [...]
Trying author-level plan ALP:INSTANTIATE-BELIEF.
  Creating goal &INSTANTIATE applied to &STATE.206.
  Creating goal &INSTANTIATE applied to &STATE.205.
  Creating goal &CHECK-NEW-SCENE applied to &BELIEF.29.
  Creating goal &CHECK-NEW-SCENE applied to &GOAL.226.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.2479.
  Creating goal &CHECK-NEW-SCENE applied to &GOAL.1786.
```

```
  Creating goal &CHECK-NEW-SCENE applied to &ACT.1036.
  Creating goal &CHECK-NEW-SCENE applied to &SETTING.215.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.206.
  Creating goal &CHECK-NEW-SCENE applied to &STATE.205.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ALP:INSTANTIATE-BELIEF creates a number of subgoals, including the goals to instantiate the mental-event and evidence of the belief: &STATE.206 and &STATE.205. The goal to instantiate the mental-event is the next goal achieved:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &STATE.206.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE..
[...]
Trying author-level plan ALP:INSTANTIATE-SUPERSEDING-BELIEF.

Try to recall an alternate explanation for the following state:

(STATE &STATE.4403
   :TYPE      &KISS
   :TO        &HUMAN.274
   :OBJECT    &HUMAN.137
   &ACHIEVES <==>   (&GOAL.2787))
```

&STATE.206 is instantiated by ALP:INSTANTIATE-SUPERSEDING-BELIEF. This author-level plan instantiates a state that supersedes a belief by finding an alternate explanation for the evidence of the belief. In this case, Lancelot's belief that Andrea loves Frederick is based upon the evidence that Andrea kissed Frederick, so ALP:INSTANTIATE-SUPERSEDING-BELIEF tries to find an alternate explanation for this evidence, i.e., it tries to find another reason why Andrea might have kissed Frederick.

To do this, ALP:INSTANTIATE-SUPERSEDING-BELIEF creates a copy of the evidence of the first belief ("Andrea kisses Frederick") and uses this as an index for creative recall, in the hopes of finding an alternate goal that the evidence might achieve:

```
Imaginative recall using index: &STATE.4403.
 TRAM Cycle: &STATE.4403.
    [...]
    Executing TRAM:GENERALIZE-OBJECT.
      Recalling:     (&SHOW-AFFECTION).
    ...TRAM succeeds: (&STATE.4423).
 TRAM Cycle succeeds: (&STATE.4423).

 Reminding is:

 (STATE &STATE.4423
```

```
  :TYPE  &RELATION
  :VALUE  &SIBLINGS
  :OBJECT  &HUMAN.1878
  :TO  &HUMAN.1876
  &ACHIEVES (&GOAL.2807))

Alternate explanation is:

(GOAL &GOAL.2807
   :TYPE          &A-AFFECTION
   :OBJECT        &HUMAN.1878
   :TO            &HUMAN.1876
   &ACHIEVED-BY <==>      (&STATE.4423))
```

Using a creativity heuristic called TRAM:GENERALIZE-OBJECT, imaginative memory invents an episode in which two people kiss to show their affection for one another because they are siblings. The recalled episode is then used to instantiate the mental-event of Lancelot's new belief as Andrea and Frederick being siblings:

```
Instantiated state:

(STATE &STATE.206
  :TYPE  &RELATION
  :VALUE  &SIBLINGS
  :OBJECT  &HUMAN.137
  :TO  &HUMAN.274)
```

This alternate explanation is the mental-event of the superseding belief. Andrea and Frederick are siblings, which motivates them to show affection for one another, which they achieve by kissing one another, which supersedes the previous belief that Andrea and Frederick are lovers. Note that as with the previous belief, the intermediate steps in this reasoning chain (in particular the goal to express affection) are not expressed in the story. The instantiated theme to this point is shown in Figure 11.7.

The next author-level goal is to instantiate the evidence for the new belief. The first plan tried to achieve this goal is ALP:INSTANTIATE-EVIDENCE, the author-level plan used to create the evidence for Lancelot's original belief that Andrea and Frederick were lovers:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &STATE.205.
Found plans: ALP:DONT-INSTANTIATE ALP:GENERAL-INSTANTIATE...
[...]
Trying author-level plan ALP:INSTANTIATE-THWARTING-STATE.
Trying author-level plan ALP:INSTANTIATE-EVIDENCE.

Initial motivation chain is &GOAL.2999.
```

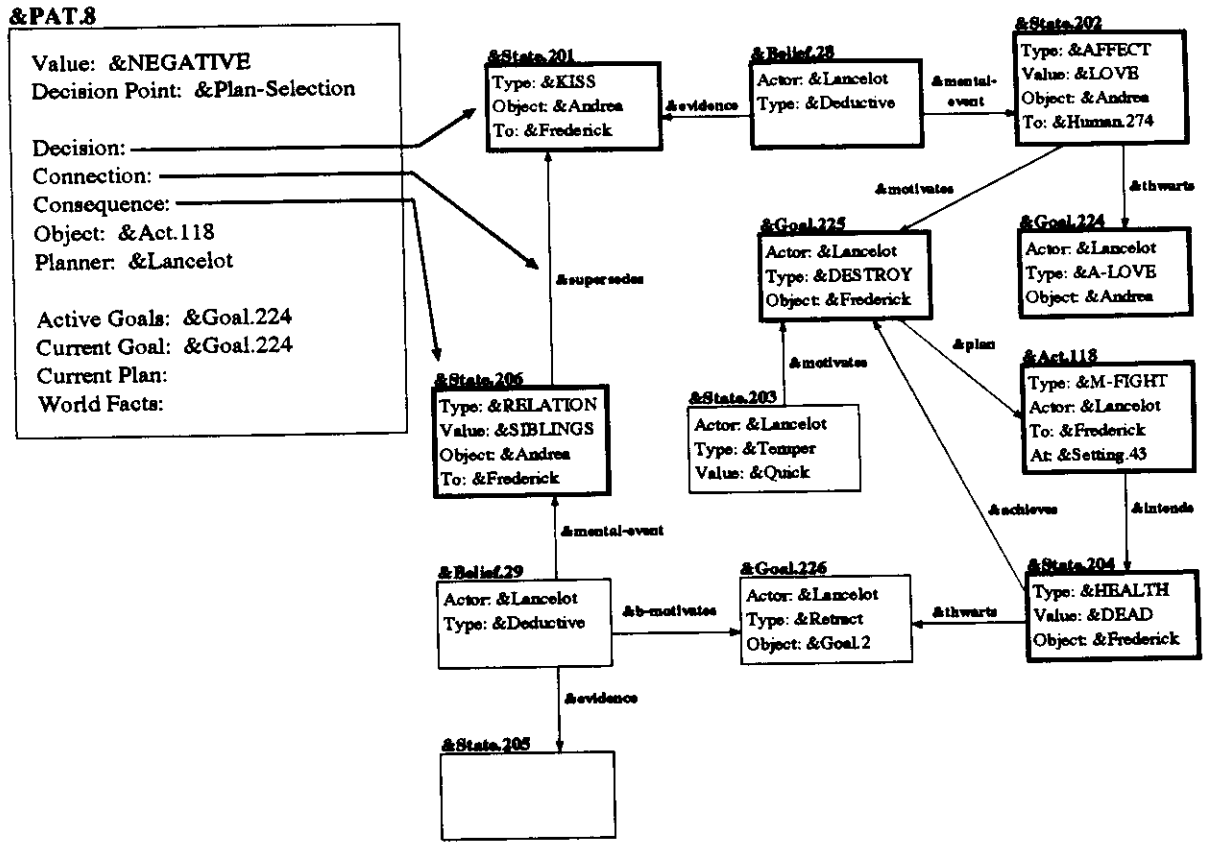


Figure 11.7 The Mistaken Knight 6

```
(GOAL &GOAL.2999
   :NEW-STATE      &STATE.4826
   :ACTOR          &HUMAN.137
   &PLAN <==>              (&ACT.2166)
   &ACHIEVED-BY <==>       (&STATE.4826)
   &MOTIVATED-BY <==>      (&STATE.4825))
```

As with the instantiation of the evidence for Lancelot's first belief, ALP:INSTANTIATE-EVIDENCE is applied to find a motivation chain from the belief ("Andrea and Frederick are siblings") to some evidence to support that belief. This time, however, ALP:INSTANTIATE-EVIDENCE fails, because MINSTREL cannot invent any goals that might be motivated by Andrea and Frederick being siblings:

```
Imaginative recall with index: &GOAL.2999
 TRAM Cycle: &GOAL.2999.
    Executing TRAM:STANDARD-PROBLEM-SOLVING.
      Recalling:     NIL.
    ...TRAM failed.
```

```
    [...]
 TRAM Cycle fails.

Reminding is NIL.
Instantiation fails: cannot invent motivation chain.
```

Problem-solving falls through to a second plan, ALP:INSTANTIATE-EVIDENCE-2. This plan says that another form of evidence for a belief is being told that the belief is true by an authority. ALP:INSTANTIATE-EVIDENCE-2 creates evidence for a belief by having the believer be told the belief is true by an authority figure. The authority can be a character involved in the belief or an authority figure such as a king.

In this case, ALP:INSTANTIATE-EVIDENCE-2 instantiates the evidence of the belief as Lancelot knowing that Andrea and Frederick are siblings after being told so by Andrea. Since Andrea is one of the characters involved in the belief, being told it is true by her carries weight.

```
Trying author-level plan ALP:INSTANTIATE-EVIDENCE-2.
Making the evidence a &know event: &STATE.205.

(STATE &STATE.205
   :TYPE          &KNOW
   :TO            &HUMAN.138
   :OBJECT        &STATE.4915
   &EVIDENCE-FOR <==>  (&BELIEF.29))

(STATE &STATE.4915
  :TYPE  &RELATION
  :VALUE  &SIBLINGS
  :OBJECT  &HUMAN.137
  :TO  &HUMAN.274)

Making the evidence the intended result of an MTRANS: &ACT.2218.

(ACT &ACT.2218
   :TYPE     &MTRANS
   :ACTOR    &HUMAN.137
   :TO       &HUMAN.138
   :OBJECT   &STATE.4915
   &INTENDS <==>  (&STATE.205))
```

In MINSTREL, a character knowing something is represented as a &KNOW state, while a character telling another character something is represented as a mental transfer - &MTRANS - of the information. In the trace above, &STATE.205 represents Lancelot's new knowledge (that Andrea and Frederick are siblings) and &ACT.2218 represents Andrea's act of telling Lancelot that knowledge. In both cases the knowledge that Andrea and Frederick are siblings is represented by &STATE.4915. The result of this instantiation on the theme is shown in Figure 11.8.

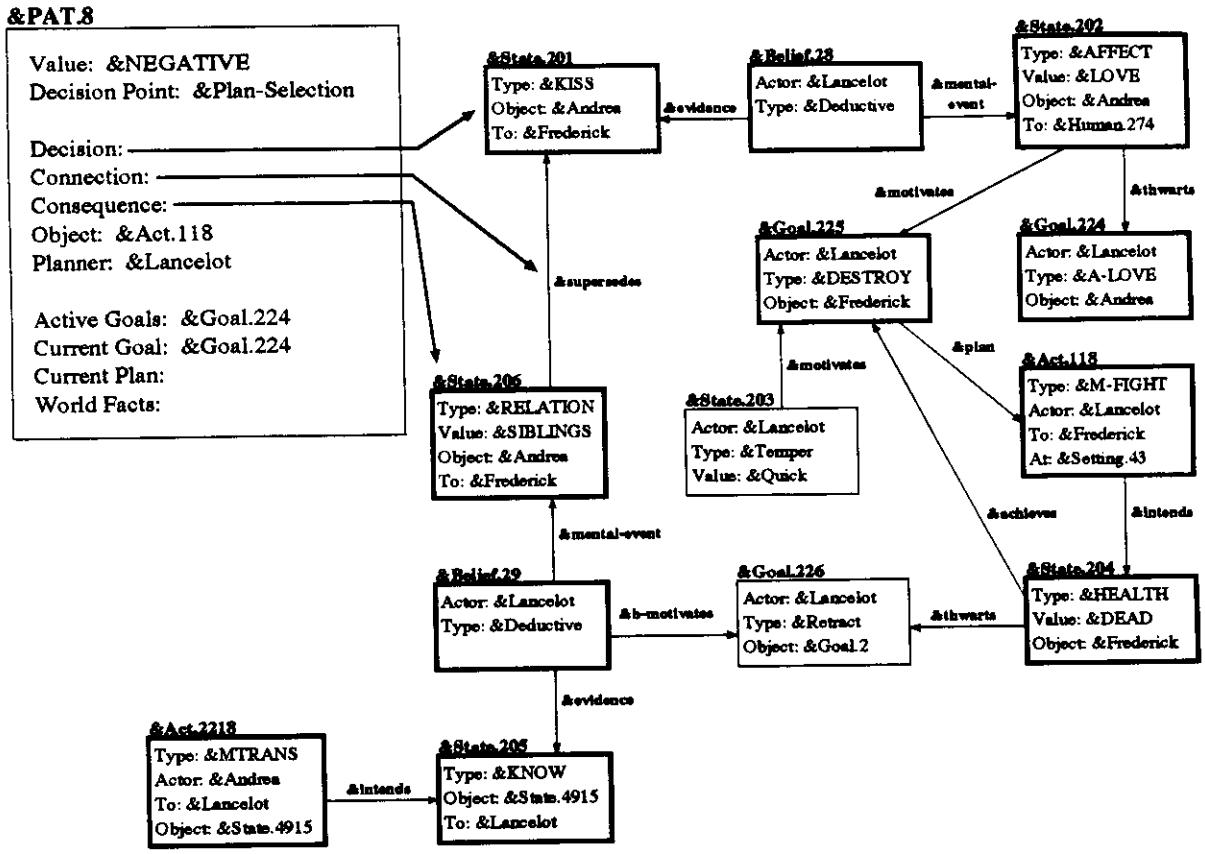


Figure 11.8 The Mistaken Knight 7

One interesting thing to note about the instantiation of &STATE.206 is that it has resulted in an unmotivated action. There is no explanation as to why Andrea told Lancelot that she and Frederick are siblings. As with other inconsistencies that arise during storytelling, this inconsistency will be detected and corrected by an author-level goal to maintain the story consistency.

At this point, instantiation of the theme is complete. MINSTREL has created the elements of the story that illustrate the story theme:

The Mistaken Knight 7

Once upon a time, a knight named Lancelot loved a princess named Andrea. Lancelot saw Andrea kiss a knight named Frederick. Lancelot believed that Andrea loved Frederick. Lancelot wanted to be the love of Andrea. But he could not because Andrea loved Frederick. Lancelot decided to kill Frederick. Lancelot fought with Frederick. Frederick was dead.

```
Later, Andrea told Lancelot that Andrea and Frederick were sib-
lings.  Lancelot believed that Andrea and Frederick were sib-
lings.  Lancelot wanted to take back that he wanted to kill
Frederick.  But he could not because Frederick was dead.
```

Now that the theme has been instantiated, MINSTREL's remaining tasks are to (1) check the story for consistency, (2) apply dramatic writing techniques, and (3) present the story to the reader.

11.4 Consistency Checking

As author-level goals are achieved and the story modified, MINSTREL must continuously check the story for inconsistencies, and try to correct any that it finds. Depending upon the priorities of the pending author-level goals, this checking may occur immediately after a scene is created or later, after more important intervening goals have been addressed. The previous section illustrated how several consistency goals contributed to the development of the theme. In this section we'll look at some additional consistency goals and how they are achieved.

The first problem MINSTREL's consistency goals detect in the story so far concerns &GOAL.224:

```
(GOAL &GOAL.224
  :TYPE   &A-LOVE
  :ACTOR   &HUMAN.138
  :OBJECT   &HUMAN.137)
```

This represents Lancelot's goal to achieve love with Andrea. MINSTREL finds this goal inconsistent because there's no explanation in the story so far of why Lancelot has this goal. There's no state that motivates this goal, and this not one of the type of goals that MINSTREL expects knights to normally have (i.e., it is not a "role goal"). So this goal is inconsistent, and MINSTREL creates an author-level goal to make the goal consistent:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &MAKE-GOAL-CONSISTENT applied to &GOAL.224.
Found plans: ALP:MAKE-CONSISTENT-SUPERGOAL
                  ALP:MAKE-CONSISTENT-OBJECT
                  ALP:MAKE-CONSISTENT-MOTIVATING-STATE
Trying author-level plan ALP:MAKE-CONSISTENT-SUPERGOAL.
Trying author-level plan ALP:MAKE-CONSISTENT-OBJECT.
Trying author-level plan ALP:MAKE-CONSISTENT-MOTIVATING-STATE.

Recalling a motivating state:
 TRAM Cycle: &GOAL.1183.
    Executing TRAM:STANDARD-PROBLEM-SOLVING.
      Recalling:     (&AR-AFFECTION).
    ...TRAM succeeds: (&AR-AFFECTION).
```

```
 TRAM Cycle succeeds: (&AR-AFFECTION).

A motivating state for &GOAL.224 has been created:

(STATE &STATE.1470
  :TYPE  &AFFECT
  :VALUE  &LOVE
  :OBJECT  &HUMAN.138
  :TO  &HUMAN.137
  &MOTIVATES  &GOAL.224)

Creating author-level goal &CHECK-NEW-SCENE
             applied to &STATE.1470.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

MINSTREL knows three plans for making a goal consistent. The first two fail, but the third succeeds. The third plan is ALP:MAKE-CONSISTENT-MOTIVATING-STATE, which tries to make a goal consistent by inventing a state of the world that could motivate the goal. To do this, ALP:MAKE-CONSISTENT-MOTIVATING-STATE uses the goal as an index for creative recall. If a past scene can be recalled with a similar goal, the motivation for that past scene can be adapted to the current scene.

In this case, recall succeeds in finding a similar past scene: &AR-AFFECTION. In &AR-AFFECTION, a princess is motivated to achieve love with a knight because she loves the knight. This is adapted to the current scene by substituting Lancelot for the princess and Andrea for the knight, resulting in a motivating state for Lancelot's goal to achieve love with Andrea: "Lancelot loves Andrea".

(It is interesting to note at this point that &AR-AFFECTION was recalled and used earlier to instantiate a different part of the story, &STATE.201. &STATE.201 represents the kiss between Andrea and Frederick. MINSTREL's creative recall enables it to make maximum use of what it knows, adapting the same knowledge for a variety of tasks and situations.)

MINSTREL has now created a motivating state for Lancelot's goal of achieving love with Andrea. But his new motivating state is itself inconsistent, because there is no explanation for why Lancelot loves Andrea. The author-level goal created at the end of ALP:MAKE-CONSISTENT-MOTIVATING-STATE detects this inconsistency and creates an author-level goal to make the motivating state consistent:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-CONSISTENCY applied to &STATE.1470.
Found plans: ALP:MAKE-CONSISTENT-STATE
  ALP:MAKE-CONSISTENT-STATE-COLOCATION
  ALP:MAKE-CONSISTENT-P-HEALTH
Trying author-level plan ALP:CHECK-CONSISTENCY-STATE.
```

```
Trying author-level plan ALP:MAKE-CONSISTENT-STATE-COLOCATION.
```

MINSTREL knows several plans for making a state consistent. The primary plan, ALP:MAKE-CONSISTENT-STATE, uses creative recall to try to find an act that could have the inconsistent state as a result. In this case, ALP:MAKE-CONSISTENT-STATE tries to find an action that could result in Lancelot loving Andrea. This fails, because MINSTREL can neither recall or invent an action which has the intended result of causing someone to be in love.

The next plan tried is ALP:MAKE-CONSISTENT-STATE-COLOCATION. ALP:MAKE-CONSISTENT-STATE-COLOCATION tries to explain a state by making it an unintended result of the principal elements of the state being colocated. For example, if ALP:MAKE-CONSISTENT-STATE-COLOCATION were used to explain the state "Lancelot possesses a magic sword" it would try to invent a scene where Lancelot and the magic sword became unexpectedly colocated. In this case, ALP:MAKE-CONSISTENT-STATE-COLOCATION tries to invent a scene in which Lancelot and Andrea become unexpectedly colocated.

To do this, ALP:MAKE-CONSISTENT-STATE-COLOCATION uses creative recall to try to recall or invent a scene in which a knight unexpectedly becomes colocated with a princess:

```
Attempting to recall an unintentional colocation:
 TRAM Cycle: &ACT.673.
     Executing TRAM:STANDARD-PROBLEM-SOLVING.
       Recalling:    NIL.
     ...TRAM failed.
     Executing TRAM:CROSS-DOMAIN-REMINDING.
       Recalling:    (&SPOT-DART).
     ...TRAM succeeds: (&ACT.690).
 TRAM Cycle succeeds: (&ACT.690).

Created unintended colocation to explain state:

(ACT &ACT.673
  :TYPE  &PTRANS
  :ACTOR  &ANIMAL.5
  :OBJECT  &HUMAN.138
  :TO  &SETTING.215
  &UNINTENDED  '&STATE.1470
  &UNINTENDED  '&STATE.1800)
```

MINSTREL's episodic memory does not contain any scenes where a knight and a princess unexepectedly meet, so non-creative recall (using TRAM:STANDARD-PROBLEM-SOLVING) fails. However, a creativity heuristic called TRAM:CROSS-DOMAIN-REMINDING succeeds in inventing a suitable episode. TRAM:CROSS-DOMAIN-REMINDING is a creativity heuristic that tries to recall a parallel scene from a different domain and adapt it to the current domain. In this case, TRAM:CROSS-DOMAIN-REMINDING succeeds in recalling the following scene from the "modern" domain:

A Willful Dog

One day, a businessman named Tim was out walking his dog Spot. Suddenly, Spot darted through a hole in a hedge, dragging Tim with him. At first Tim was angry with Spot, but on the other side of the hedge he saw his long-lost friend Tom. What luck that Spot had darted through the hedge!

TRAM:CROSS-DOMAIN-REMINDING and ALP:MAKE-CONSISTENT-STATE-COLOCATION adapt this reminding to the King Arthur domain, creating a scene in which Lancelot's horse (represented by &ANIMAL.5) carries Lancelot to a location where he unexpectedly meets and falls in love with Andrea:

> Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot loved Andrea.

Note that MINSTREL has already determined that Lancelot loves Andrea, and is explaining here how they came to love one another. MINSTREL doesn't make every character which comes near another character fall in love. Indeed, MINSTREL doesn't have any knowledge at all about what makes characters fall in love. It is only by using ALP:MAKE-CONSISTENT-STATE-COLOCATION and creativity that MINSTREL is able to invent a reason for Lancelot and Andrea to fall in love.

```
Creating author-level goal &CHECK-NEW-SCENE: &ACT.673.
Creating author-level goal &CHECK-NEW-SCENE: &STATE.1800.
Creating author-level goal &CHECK-NEW-SCENE: &STATE.1819.
Creating author-level goal &CHECK-NEW-SCENE: &GOAL.1394.
Creating author-level goal &CHECK-NEW-SCENE: &ACT.692.
Creating author-level goal &CHECK-NEW-SCENE: &ANIMAL.5.
Creating author-level goal &CHECK-NEW-SCENE: &STATE.1478.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

These goals detect another inconsistency: What is Andrea doing in the woods where Lancelot unexpectedly meets her? This inconsistency is corrected by inventing a reason for Andrea to be in the woods. Several other inconsistencies are also corrected, and eventually, the explanation for why Lancelot wants Andrea to love him results in the following story scenes:

> One day, a lady of the court named Andrea wanted to have some berries. Andrea wanted to be near the woods. Andrea moved to the woods. Andrea was at the woods. Andrea had some berries because Andrea picked some berries. Lancelot's horse moved Lancelot to the woods. This unexpectedly caused him to be near Andrea. Because Lancelot was near Andrea, Lancelot loved Andrea. Because Lancelot loved Andrea, Lancelot wanted to be the love of Andrea.

The schema representation for these scenes is shown in Figure 11.9.

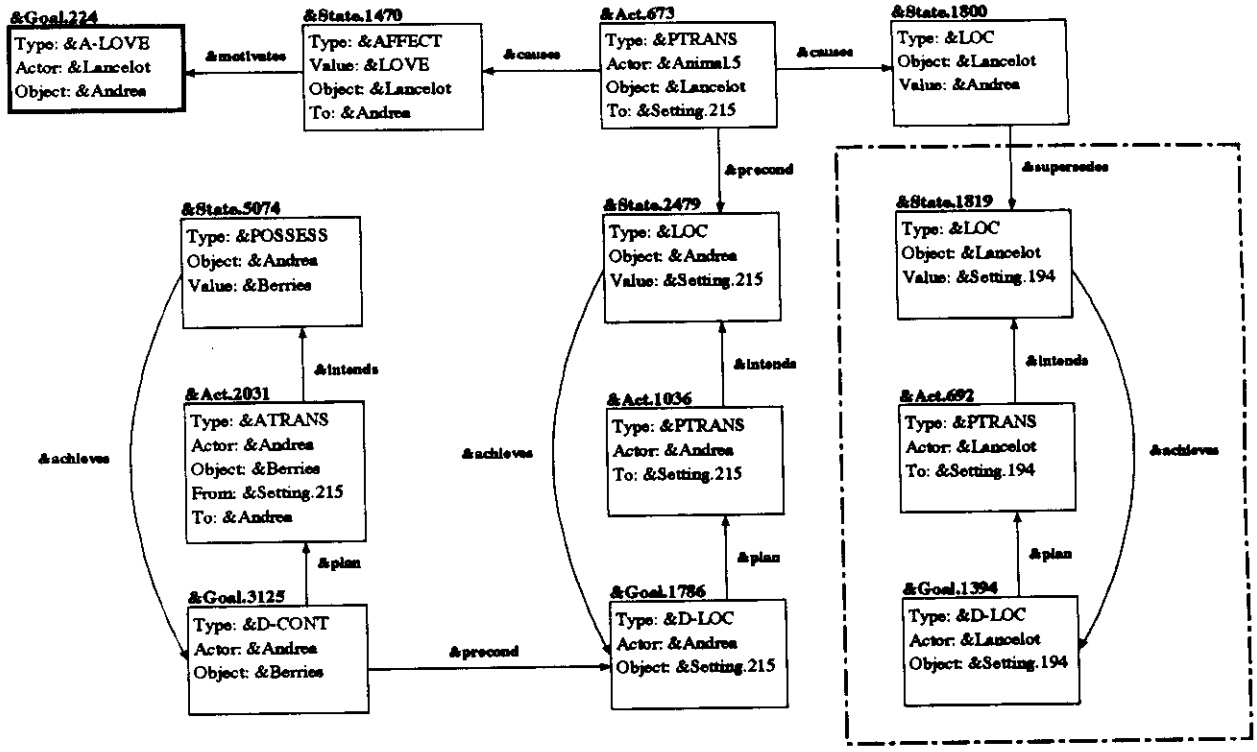


Figure 11.9 Illustration of Consistency Scenes

This example illustrates two interesting features of MINSTREL.

First, this example shows how storytelling is achieved by a large number of interacting author-level goals. Achieving one author-level goal can give rise to a number of new author-level goals, and adding one element to the story can require the addition of many more elements. This example started with the single author-level goal to make Lancelot's goal to achieve love with Andrea consistent, and ended only after the invention of a number of new story events. As this example illustrates, storytelling cannot be understood as a single, straightforward process, but rather as a complicated interaction between large numbers of competing goals.

Second, it is interesting to note that not all of what MINSTREL invents during storytelling will necessarily be expressed in the final story. The dotted box in Figure 11.9 encloses an explanation of where Lancelot was before he came to the woods and met Andrea. MINSTREL creates the events in the dotted box to improve the consistency of the story, but later decides not to express them when presenting the story to the reader in natural language. This illustrates another facet of a system involving competing goals. In a system with many goals, the goals must necessarily be semi-independent. Each goal cannot take into account all the other goals. Therefore, goal conflicts will arise. One goal may make a decision only to be overruled later by another

goal.

Before we look at author-level dramatic writing goals, we will examine one other type of consistency goal. The example above dealt with causal inconsistencies: problems in the way characters planned or in the way the world behaved. Another type of inconsistency is emotional inconsistency. Not only should characters plan properly, they should also have the proper emotional reactions to events in the story world.

During the processing of *The Mistaken Knight*, MINSTREL discovers an emotional inconsistency in &GOAL.224. &GOAL.224 is Lancelot's thwarted goal to achieve love with Andrea. Important thwarted goals should cause an emotional reaction in the actor of the goal. This inconsistency is detected by the author-level goal &CHECK-AFFECTS:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-AFFECTS applied to &GOAL.224.
Found plans: ALP:CHECK-AFFECT-THWARTED-GOAL
Trying author-level plan ALP:CHECK-AFFECT-THWARTED-GOAL.

Creating emotional reaction:

(STATE &STATE.1478
  :VALUE    &HATE
  :TO       &HUMAN.274
  :OBJECT   &HUMAN.138
  :TYPE     &AFFECT
  &TRIGGER &GOAL.224)

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ALP:CHECK-AFFECT-THWARTED-GOAL is an author-level plan which checks a thwarted goal to see if the actor of the goal had an emotional reaction. If he did not, a negative emotional reaction directed toward the character who thwarted the goal. In this case, ALP:CHECK-AFFECT-THWARTED-GOAL creates the reaction "Lancelot hates Frederick".

Maintaining causal and emotional consistency during storytelling is important to creating a believable story. For a more detailed discussion of MINSTREL's goals and plans for maintaining story consistency, see Chapter 9.

11.5 Dramatic Writing Goals

After the story events which illustrate the theme have been created and checked for consistency, MINSTREL's next priority is to apply dramatic writing techniques to improve the literary quality of the story. In this story, MINSTREL is able to create foreshadowing and characterization. We'll look first at how foreshadowing is achieved.

Foreshadowing is a literary technique in which story incidents introduce, repeat, give casual allusion to or hint at later story incidents. The purpose of foreshadowing is to build a sense of inevitability in the later story events ([de Camp 1975]). By foreshadowing important story elements, the author avoids having those elements appear contrived, and in addition, creates a sense of unity in the story.

MINSTREL's author-level plan for creating foreshadowing looks through the story theme for unique events and then checks to see if those events can be repeated elsewhere in the story. By repeating events, MINSTREL hopes to improve the unity of the story; by repeating *unique* events, MINSTREL hopes to make this repetition noticeable to the reader.

MINSTREL's author-level plan for foreshadowing begins with any element of theme and looks through *all* the theme-related story events for foreshadowing possibilities:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-STORY-FOR-FORESHADOWING applied to
  &BELIEF.28.
Found plans: ALP:CHECK-STORY-FOR-FORESHADOWING
Trying author-level plan ALP:CHECK-STORY-FOR-FORESHADOWING.

Checking possible candidates:
  Recalling &STATE.1800: NIL.
  Recalling &STATE.1470: NIL.
Candidates = (&STATE.1800 &STATE.1470).
```

The first step in ALP:CHECK-STORY-FOR-FORESHADOWING is to find possible candidates for foreshadowing. To do this ALP:CHECK-STORY-FOR-FORESHADOWING looks through the story events which illustrate the theme for unique combinations of states and acts. To determine whether or not a particular combination is unique, MINSTREL uses the combination as an index for recall.

If a particular combination of states and acts recalls something from episodic memory, then it is clearly not unique, because MINSTREL has encountered similar combinations before. On the other hand, if it recalls nothing, then it is unique in MINSTREL's experience. This is the same type of test that MINSTREL uses to determine whether a problem solution is creative (see Chapter 3), and indicates again how a properly-organized episodic memory can be useful for determining the originality of an episode.

In this case, MINSTREL finds two combinations that do not recall similar past episodes: &STATE.1800 and &STATE.1470. Both states have to do with Lancelot's willful horse. &STATE.1470 represents Lancelot's willful horse unintentionally causing Lancelot to fall in love with Andrea; &STATE.1800 represents the related unintentional meeting of Lancelot and Andrea. MINSTREL's episodic memory does not contain any states similar to these two scenes (recall that this scene was invented using the TRAM:CROSS-DOMAIN-REMINDING creativity heuristic), so they are judged suitable for foreshadowing.

Once acceptable candidates for foreshadowing have been found, the next step is to see if these candidates can be inserted elsewhere in the story. This is achieved by looking through the story for states similar to the candidate states. In this case, that means looking for states representing two people being unexpectedly in the same location, or one person unexpectedly loving another person.

```
Insertions for &STATE.1800 are: (&STATE.6950).
New foreshadowing state is &STATE.17630, new foreshadowing
act is &ACT.9768.

Insertions for &STATE.1470 are NIL.
No foreshadowing created.

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

One match is found: &STATE.6950. This state represents Lancelot and Andrea being in the same location so that Lancelot can see Andrea kissing Frederick. &STATE.6950 was created by a combination of consistency goals which noticed that (1) Lancelot must see the evidence for his belief, i.e., he must witness the kiss, and (2) Lancelot must be in the same location as the event in order to witness it. These consistency goals also created an explanation for how Lancelot comes to unexpectedly witness the kiss:

> Some time later, Lancelot wanted to kill a dragon. Lancelot rode to the woods, causing him to be near Andrea. Lancelot knew that Andrea kissed with a knight named Frederick because Lancelot saw that Andrea kissed with Frederick.

This explanation is created by finding a typical goal for a knight that would take him to the location where he should unexpectedly be. This is accomplished by the same consistency goals that invented the scenes in which Andrea goes to the woods to pick berries (so that she can unexpectedly be where Lancelot sees her and falls in love).

However, this explanation is about to be replaced by a new explanation. ALP:CHECK-STORY-FOR-FORESHADOWING has noticed that &STATE.6950 (which represents Lancelot unexpectedly meeting Andrea when he sees her kissing Frederick) matches &STATE.1800 (which represents Lancelot unexpectedly meeting Andrea when he falls in love with her). In order to create foreshadowing, ALP:CHECK-STORY-FOR-FORESHADOWING deletes the existing explanation for &STATE.6950 ("Lancelot was in the woods to kill a dragon") and replaces it with the same explanation used for the early scene ("Lancelot's willful horse took him to where Andrea was"). The result is that the early scene in which Andrea and Lancelot meet foreshadows the later scene in which Lancelot sees Andrea and Frederick kiss:

> One day, a lady of the court named Andrea wanted to have some berries. Andrea wanted to be near the woods. Andrea moved to the woods. Andrea was at the woods. Andrea had some berries

```
because Andrea picked some berries.  Lancelot's horse moved
Lancelot to the woods.  This unexpectedly caused him to be near
Andrea.  Because Lancelot was near Andrea, Lancelot loved An-
drea...

Some time later, Lancelot's horse moved Lancelot to the woods
unintentionally, again causing him to be near Andrea.  Lancelot
knew that Andrea kissed with a knight named Frederick because
Lancelot saw that Andrea kissed with Frederick.
```

It is interesting to note that just as MINSTREL earlier created story scenes that were not expressed in the final version of the story, here MINSTREL creates story scenes only to discard them later in favor of different scenes. As these examples illustrate, the final story is not an accurate record of the storytelling process. Just as human authors revise and discard, so MINSTREL sometimes creates story scenes only to later discard, revise, or choose not to express them.

The second dramatic writing goal MINSTREL achieves during the telling of this story is characterization. The purpose of characterization is to create a convincing portrayal of a character's personality and other traits. Most importantly, characterization is used to present to the reader the facets of a character's personality that have direct impact on his actions in the story.

MINSTREL attempts to develop characterization for personality traits that (1) belong to a major character and (2) are necessary for the development of the story theme. (For more on MINSTREL's author-level goals for characterization, see Chapter 8.) The theme of *The Mistaken Knight* contains the information that the main character (Lancelot) has a quick temper (see &STATE.203 in Figure 11.1). Lancelot's quick temper is one of his motivations for deciding to kill Frederick. Since Lancelot's quick temper impacts directly upon his actions in the story, it is a suitable candidate for characterization:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &CHECK-STORY-FOR-CHARACTERIZATION applied to
  &PAT.8.
Found plans: ALP:CHECK-STORY-FOR-CHARACTERIZATION
Trying author-level plan ALP:CHECK-STORY-FOR-CHARACTERIZATION.

Creating author-level goal &ADD-CHARACTERIZATION applied to
  &STATE.203.

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

The author-level plan ALP:CHECK-STORY-FOR-CHARACTERIZATION creates the new author-level goal to add characterization of Lancelot's hot temper. This is achieved by the author-level plan ALP:ADD-CHARACTERIZATION-EXAMPLE:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &ADD-CHARACTERIZATION applied to &STATE.203.
Found plans: ALP:ADD-CHARACTERIZATION-EXAMPLE
             ALP:ADD-CHARACTERIZATION-STATEMENT.

Trying author-level plan ALP:ADD-CHARACTERIZATION-EXAMPLE.

Trying to recall a goal motivated by &STATE.203:
TRAM Cycle: &GOAL.10608.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    (&KILL-SWORD).
   ...TRAM succeeds: (&KILL-SWORD)
TRAM Cycle succeeds: (&KILL-SWORD).
```

ALP:ADD-CHARACTERIZATION-EXAMPLE tries to characterize a personality trait by inventing a new goal motivated by the character trait. To do this, an uninstantiated goal motivated by the character trait is used as an index for recall. In this case, the goal recalls a previous story scene in which a knight became angry and broke his sword:

The Angry Knight

> Godwin was a knight with a hot temper. One time he lost a joust and he was so upset that he broke his sword.

Happily, this is almost exactly what is needed to illustrate Lancelot's hot temper. By replacing Godwin with Lancelot, ALP:ADD-CHARACTERIZATION-EXAMPLE creates the scene which is later generated at the beginning of the story to illustrate Lancelot's hot temper:

```
Created goal motivated by &STATE.203:

(GOAL &GOAL.10608
  :TYPE  &DESTROY
  :ACTOR  &HUMAN.138
  :OBJECT  &PHYSOBJ.736
  &ACHIEVED-BY  &STATE.17721
  &PLAN  &ACT.9818
  &MOTIVATED-BY  &STATE.17722
  &MOTIVATED-BY  &STATE.17716 )

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

When the story is presented to the reader, this is expressed as:

```
It was the spring of 1089, and a knight named Lancelot returned
to Camelot from elsewhere.  Lancelot was hot tempered.  Once,
Lancelot lost a joust.  Because he was hot tempered, Lancelot
```

```
wanted to destroy his sword.  Lancelot struck his sword.  His
sword was destroyed.
```

If the &KILL-SWORD scene had been unavailable (or if it had been used several times and consequently judged uninteresting), MINSTREL would have tried to invent a new episode illustrating Lancelot's hot temper, and failing that, fallen through to another plan for illustrating a character trait, ALP:ADD-CHARACTERIZATION-STATEMENT. (See Chapter 8 for a description of ALP:ADD-CHARACTERIZATION-STATEMENT.)

11.6 Presentation Goals

The final step in the storytelling process is presenting the story to the reader. This involves (1) creating any new scenes needed to improve the presentation of the story, such as character introductions and denouements, and (2) generating the story in language.

MINSTREL creates two types of story introductions. The first introduction scene establishes the location and time of the story. The second scene identifies the main character of the story. (For the type of theme-based stories MINSTREL tells, the main character is the character who makes the planning decision identified by the Decision portion of the theme.) In this story, the main character is Lancelot. Both introductory scenes are created by ALP:CREATE-STORY-INTROS:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &ADD-STORY-INTROS applied to &STORY.1.
Found plans: ALP:CREATE-STORY-INTROS
Trying author-level plan ALP:CREATE-STORY-INTROS.

Creating establishing scene.
Year is 1089.

(STATE &STATE.24407
  :VALUE  &SPRING
  :OBJECT  '1089
  :TYPE  &DATE)

Creating introduction scene.
Location is &CAMELOT.
Main character is: &HUMAN.138.

(STATE &STATE.24406
  :VALUE  &CAMELOT
  :OBJECT  &HUMAN.138
  :TYPE  &LOC
  &SUPERSEDES &STATE.24405 )

Author-level planning succeeded.
```

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

The first introductory scene is a state schema representing the current date, Spring of 1089. (The year is generated randomly in range of credible years for the King Arthur genre.) The second introductory scene is a state schema representing the main character's return to the current location from elsewhere. This serves both to establish the current location and introduce the main character. These scenes are later generated as the first sentence of the story:

```
                    The Mistaken Knight

    It was the spring of 1089, and a knight named Lancelot returned
    to Camelot from elsewhere...
```

Note that MINSTREL's language generation goals have combined the two state schemas into a single sentence to produce a more fluent introduction.

As well as creating introduction scenes, MINSTREL's presentation goals also create denouement scenes. Denouement scenes are used by MINSTREL to resolve character fates. By telling the reader briefly how the events of the story affected the characters in the story, MINSTREL provides the reader with a feeling of closure. For reasons discussed in Chapter 10, MINSTREL creates denouement scenes only for characters that have unhappy fates. In *The Mistaken Knight*, that includes Lancelot, Andrea and Frederick.

The first denouement created is for Andrea. Andrea is a tragic character because she suffers some important goal failures during the course of the story. For a tragic character of this sort, MINSTREL creates a denouement scene in which the tragedy motivates the character to change roles:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &MAKE-DENOUEMENT applied to &HUMAN.137.
Found plans: ALP:BURIAL-DENOUEMENT ALP:TRAGIC-DENOUEMENT.
Trying author-level plan ALP:BURIAL-DENOUEMENT.
Trying author-level plan ALP:TRAGIC-DENOUEMENT.

&HUMAN.137 is tragic because of &GOAL.2808.
&HUMAN.137 is currently a &PRINCESS.
Most different role from &PRINCESS is &HERMIT.
Created denouement scene:

(STATE &STATE.24494
  :VALUE  &HERMIT
  :TYPE  &ROLE-CHANGE
  :OBJECT  '&HUMAN.138 )

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ALP:TRAGIC-DENOUEMENT begins by checking to make sure that the character it is being applied to is in fact a tragic character. For Andrea, ALP:TRAGIC-DENOUEMENT finds a thwarted goal to express affection toward her brother (thwarted because Lancelot has killed her brother). This establishes Andrea as a tragic character.

The next step is to determine a suitable role change for Andrea. Currently Andrea is a princess, which has the features of being &SOCIABLE and &NON-VIOLENT. By comparing these features to the features of other roles, MINSTREL is able to determine that the most different role is &HERMIT, which has the features &UNSOCIABLE and &NON-VIOLENT. ALP:TRAGIC-DENOUEMENT then creates a state schema representing a role change for Andrea to a hermit and adds this to the story. This will later be generated as:

```
Andrea became a nun.
```

MINSTREL natural language component generates female hermits as "nuns".

A similar process occurs for Lancelot, who is considered tragic because of his failed goal to retract his goal of killing Frederick. Lancelot also becomes a hermit, because as a knight his role features are &SOCIABLE and &VIOLENT.

Next a denouement is created for Frederick. Unlike Andrea and Lancelot, Frederick has been killed during the course of the story, so a different author-level plan applies:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &MAKE-DENOUEMENT applied to &HUMAN.274.
Found plans: ALP:BURIAL-DENOUEMENT ALP:TRAGIC-DENOUEMENT.
Trying author-level plan ALP:BURIAL-DENOUEMENT.

Trying to recall burial site:
 TRAM Cycle: &STATE.24463.
   [...]
 TRAM Cycle fails.
Recall fails.

Using default burial site: &WOODS.
Created denouement:

(STATE 'STATE.24463
  :VALUE  '&SETTING.3724
  :OBJECT  '&HUMAN.274
  :TYPE  &BURIED)

(SETTING 'SETTING.3724
  :TYPE  &WOODS)

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

ALP:BURIAL-DENOUEMENT creates a burial scene for characters that have died during the course of the story. To determine where to bury a character, ALP:BURIAL-DENOUEMENT uses episodic memory to try to recall where similar past characters have been buried. In this case, recall fails, because MINSTREL does not know of any past stories in which knights were buried.

When a setting cannot be recalled a default setting is used, and a scene is created in which Frederick is buried in the woods (&STATE.24463). This is later generated as:

```
Frederick was buried in the woods.
```

After the story introductions and character denouements have been created, all that remains is to generate the story in English. This is accomplished by author-level goals which control the order in which the story scenes will be generated, and a phrasal generator which does the actual translation of schemas to English. Rather than trace all of the author-level goals and phrasal generator calls involved in expressing *The Mistaken Knight* in English, we will look here at a few representative examples. MINSTREL's natural language capabilities are discussed in more detail in Chapter 10.

ALP:GENERATE-BELIEF is an example of an author-level plan for generation. ALP:GENERATE-BELIEF generates the parts of a character belief in an order that emphasizes the argument structure of the belief:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &GENERATE-ENGLISH applied to &BELIEF.28.
Found plans: ALP:GENERATE-BELIEF.
Trying author-level plan ALP:GENERATE-BELIEF.

Generating establishing scene from belief:

    (ONE DAY *COMMA* A LADY OF THE COURT NAMED ANDREA WANTED TO
         HAVE SOME BERRIES *PERIOD* ANDREA WANTED TO BE NEAR THE
         WOODS *PERIOD* ANDREA MOVED TO THE WOODS *PERIOD* ANDREA
         WAS AT THE WOODS *PERIOD* ANDREA HAD SOME BERRIES
         BECAUSE ANDREA PICKED SOME BERRIES *PERIOD* LANCELOT
         *POSSESSIVE* HORSE MOVED LANCELOT TO THE WOODS *PERIOD*
         THIS UNEXPECTEDLY CAUSED HIM TO BE NEAR ANDREA *PERIOD*
         BECAUSE LANCELOT WAS NEAR ANDREA *COMMA* LANCELOT LOVED
         ANDREA *PERIOD*)
```

ALP:GENERATE-BELIEF begins by generating the scenes need to establish the belief. These scenes represent the preconditions of the belief, and were created by consistency goals to explain why Lancelot wanted Andrea to love him (the goal which is thwarted by Lancelot's belief that Andrea loves Frederick). The trace shows the actual output from the RAP phrasal generator. Note that punctuation is represented with keywords such as "*COMMA*". These are later filtered into their proper form by a separate program.

```
Generating b-precond:

   (SOME TIME LATER *COMMA* LANCELOT *POSSESSIVE* HORSE MOVED
         LANCELOT TO THE WOODS UNINTENTIONALLY *COMMA* AGAIN
         CAUSING HIM TO BE NEAR ANDREA *PERIOD*)
```

Next ALP:GENERATE-BELIEF generates the belief preconditions. These represent the preconditions for attaining the evidence of the belief. In this case, the evidence is "Andrea kisses Frederick", and the preconditions are that Lancelot must be in the same location as Andrea to witness the kiss.

```
Generating evidence:

   (LANCELOT KNEW THAT ANDREA KISSED WITH A KNIGHT NAMED
         FREDERICK BECAUSE LANCELOT SAW THAT ANDREA KISSED WITH
         FREDERICK *PERIOD*)
```

Now ALP:GENERATE-BELIEF generates the evidence for the belief. This consists of Lancelot viewing the kiss.

```
Generating mental-event:

   (LANCELOT BELIEVED THAT ANDREA LOVED FREDERICK *PERIOD*
       LANCELOT LOVED ANDREA *PERIOD* BECAUSE LANCELOT LOVED
       ANDREA *COMMA* LANCELOT WANTED TO BE THE LOVE OF ANDREA
       *PERIOD* BUT HE COULD NOT BECAUSE ANDREA LOVED FREDERICK
       *PERIOD* LANCELOT HATED FREDERICK *PERIOD* ANDREA LOVED
       FREDERICK *PERIOD* BECAUSE LANCELOT WAS HOT TEMPERED
       *COMMA* LANCELOT WANTED TO KILL FREDERICK *PERIOD*
       LANCELOT WANTED TO BE NEAR FREDERICK *PERIOD* LANCELOT
       MOVED TO FREDERICK *PERIOD* LANCELOT WAS NEAR FREDERICK
       *PERIOD* LANCELOT FOUGHT WITH FREDERICK *PERIOD*
       FREDERICK WAS DEAD *PERIOD*)

Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Finally, ALP:GENERATE-BELIEF generates the mental-event of the belief (i.e., what Lancelot believes) and the story events which are causally dependent upon this belief.

The actual generation of each part of the belief is handled by MINSTREL's natural language component, RAP. RAP is a phrasal parser ([Reeves 1991][Arens 1986][Wilensky 1980]) which uses a lexicon of phrases that pair concepts and words. To generate a concept in English, it is matched against concepts in the lexicon, and when a match is found, the corresponding words are output. RAP is described in detail in Chapter 10.

To illustrate how RAP produces English language from conceptual representation, we will trace

the production of the evidence portion of &BELIEF.28:

```
(LANCELOT KNEW THAT ANDREA KISSED WITH A KNIGHT NAMED
      FREDERICK BECAUSE LANCELOT SAW THAT ANDREA KISSED WITH
      FREDERICK *PERIOD*)
```

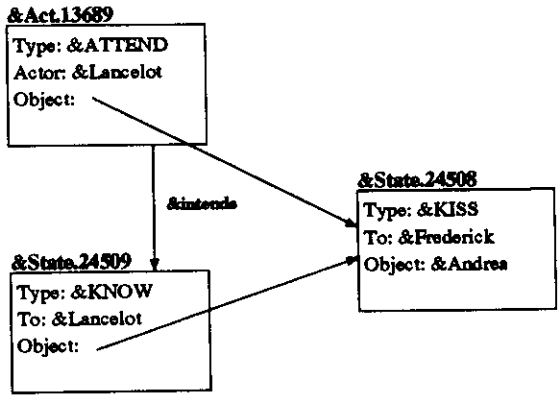


Figure 11.10 Conceptual Representation of Evidence of &BELIEF.28

Figure 11.10 shows the conceptual representation of the evidence for Lancelot's belief that Andrea loves Frederick. &STATE.24508 represents Andrea kissing Frederick. &ACT.13689 represents Lancelot's viewing of the kiss, which intends his knowledge of the kiss, represented by &STATE.24509. For technical reasons[4], the identifiers for the schemas shown in Figure 11.10 are different from the ones shown previously. Similarly, although we have labeled the characters "&Lancelot", "&Frederick" and "&Andrea" in Figure 11.10, these characters are actually represented by schemas &HUMAN.5369, &HUMAN.5370, and &HUMAN.5368.

To begin generation, the RAP generator is called with the keyword :sentence and the act representing Lancelot's seeing the kiss. The keyword :sentence indicates that RAP is to generate this concept as a full sentence, including any necessary clauses or secondary sentences to convey the full meaning of the concept:

```
RHAPSODY-USER>(rap:gen (list :sentence &act.13689))

Applying rule RAP:INTENTIONAL-RESULTS:

  (:SENTENCE
      &ACT.13689) -> (:CLAUSE &STATE.24509 BECAUSE :SENTENCE
                                  &ACT.13747).
```

Each step of the trace of RAP shows the phrase applied and the action that phrase took. The

4. The RAP generator creates copies of the concepts it generates so that any changes made to the conceptual structures during the course of generation won't affect the actual conceptual structure of the story being told.

first phrase applied is RAP:INTENTIONAL-RESULTS, which rewrites the initial pattern into a new pattern in which the intentional result of Lancelot's action is generated as a clause followed by "because" and then the action that intended the result. RAP:INTENTIONAL-RESULTS appears this way in the phrasal lexicon:

```
(rap:phrase intentional-results
   (input :sentence
        (*and* ?act
               (act nil
                    &intends (*and* ?ul (*value* ?ul)))))
   (output
     :clause ?ul because :sentence (:wo ?act &intends)))
```

The input portion of this phrase tests to see if the input is an act being generated as a sentence, and tests to see if the act has an "&intends" link to a non-nil value. During the matching of the input portion of the phrase, the logical variable "?act" is bound to the input act schema (i.e., &ACT.13689) and the variables "?ul" is bound to the state schema the act intends (i.e., &STATE.24509). If these tests succeed, the input is rewritten to an output consisting of the keyword ":clause" followed by the intended state, followed by "because" and then the original act generated as a sentence. The ":wo" function removes the link between &ACT.13689 and &STATE.24509, so that when RAP goes to generate &ACT.13689 again as a sentence, it won't recursively apply RAP:INTENTIONAL-RESULTS again.

The next step is to generate the opening clause:

```
Applying rule RAP::CLAUSE-KNEW:
  (:CLAUSE &STATE.24509) ->
      (:SUBJECT &HUMAN.5369 KNEW THAT :OBJECT &STATE.24508).
```

This is accomplished by RAP:CLAUSE-KNEW, a phrase that knows how to generate states representing knowledge as a clause:

```
(rap:phrase clause-knew
  (input :clause (state nil
                        :type &know
                        :object ?what
                        :to ?who))
  (output :subject ?who knew that :object ?what))
```

RAP:CLAUSE-KNEW is applied when a state with &KNOW in the type slot is generated as a clause. RAP:CLAUSE-KNEW breaks out portions of the state (the object that is known and who is knowing it) and generates that as a simple declarative clause with the knower as the subject and the thing known as the object. The first step of generating this as a clause is to generate the subject:

```
Applying rule RAP:SIMPLE-CHAR-NAME:
```

```
  (:SUBJECT &HUMAN.5369) -> (LANCELOT).

Outputing: (LANCELOT KNEW THAT).
```

Except when first introducing a character, RAP generates a character by using the name of the character. After this has been done for the subject of the introductory clause, the name and the words "knew that" (which were produced by RAP:CLAUSE-KNEW) are output. The remaining portion of the input pattern is:

```
Remaining: (:OBJECT &STATE.24508 BECAUSE :SENTENCE &ACT.13747).
```

The next step is to generate the state representing Andrea's kiss of Frederick as the object of the introductory clause. This is accomplished by RAP:STATE-OBJECT, which simply says to generate the state as a clause:

```
Applying rule RAP:STATE-OBJECT:
  (:OBJECT &STATE.24508) -> (:CLAUSE &STATE.24508).
```

Because this state does not represent knowledge, RAP:CLAUSE-KNEW does not apply. &STATE.24508 represents Andrea's kiss directed to Frederick, so a more generic phrase is used, which knows how to speak about states that are directed towards a person:

```
Applying rule RAP:CLAUSE-STATE-TO:
  (:CLAUSE &STATE.24508) -> (:SUBJECT &HUMAN.5368
                                      &KISS WITH :OBJECT
                                      &HUMAN.5370).
```

RAP:CLAUSE-STATE-TO generates a state directed towards a person as a simple declarative clause with the person the state is directed towards as the object of the preposition "with":

```
(rap:phrase clause-state-to
  (input :clause (state nil
                        :type ?type
                        :object ?object
                        :to (*and* ?to (*value* ?to))))
  (output :subject ?object ?type with :object ?to))
```

Note that the filler of the type slot of the state is placed in the verb position in the clause being generated. The type of the state will be used to generate an appropriate verb:

```
Applying rule RAP:SIMPLE-CHAR-NAME:
  (:SUBJECT &HUMAN.5368) -> (ANDREA).

Outputing: (ANDREA).

Applying rule RAP:KISS:
```

```
  (&KISS) -> (KISSED).

Outputing: (KISSED WITH).

Applying rule RAP:SIMPLE-CHAR-NAME:
  (:OBJECT &HUMAN.5370) -> (FREDERICK).

Outputing: (FREDERICK BECAUSE).
```

Andrea's name is generated and then &KISS is used to generated the past tense of the verb "to kiss". Except for special cases, MINSTREL only knows how to generate sentences in the past tense. The verb is followed by Frederick's name and the clue word "because" (which was generated by RAP:INTENTIONAL-RESULTS). The remaining input is now the remaining sentence from the original phrase:

```
Remaining: (:SENTENCE &ACT.13747).
```

&ACT.13747 represents Lancelot's viewing of the kiss. In fact, this is a copy of the original act, with the &intends link removed so that RAP won't apply RAP:INTENTIONAL-RESULTS again. To generate this as a sentence, &ACT.13747 is generated as a clause followed by a period.

```
Applying rule RAP:SENTENCE:
  (:SENTENCE
      &ACT.13747) -> (:CLAUSE &ACT.13747 *PERIOD*).

Applying rule RAP:ACT-CLAUSE:
  (:CLAUSE &ACT.13747) -> (:SUBJECT &HUMAN.5383
                                    &ATTEND :OBJECT
                                    &STATE.24609).
```

To generate this as a clause, RAP applies RAP:ACT-CLAUSE, which is a generic phrase for generating actions. It creates a simple declarative clause with the actor of the action in the subject position, the type of the act in the verb position, and the object being acted upon in the object position:

```
(rap:phrase act-clause
  (input :clause (act nil
                      :actor ?actor
                      :type ?type
                      :object ?object))
  (output :subject ?actor ?type :object ?object))
```

As before, the name of the actor and a verb based upon the type of the action are generated:

```
Applying rule RAP:SIMPLE-CHAR-NAME:
  (:SUBJECT &HUMAN.5383) -> (LANCELOT).
```

```
Outputing: (LANCELOT).

Applying rule RAP:ATTEND:
  (&ATTEND) -> (SAW THAT).

Outputing: (SAW THAT).
```

Finally, the kiss between Andrea and Frederick must once again be generated as the object of a clause. This proceeds exactly as before:

```
Applying rule RAP:STATE-:OBJECT:
  (:OBJECT &STATE.24609) -> (:CLAUSE &STATE.24609).

Applying rule RAP:CLAUSE-STATE-TO:
  (:CLAUSE &STATE.24609) -> (:SUBJECT &HUMAN.5385
                                      &KISS NIL WITH :OBJECT
                                      &HUMAN.5384).

Applying rule RAP:SIMPLE-CHAR-NAME:
  (:SUBJECT &HUMAN.5385) -> (ANDREA).

Outputing: (ANDREA).

Applying rule RAP:KISS:
  (&KISS NIL) -> (KISSED).

Outputing: (KISSED WITH).

Applying rule RAP:SIMPLE-CHAR-NAME:
  (:OBJECT &HUMAN.5384) -> (FREDERICK).

Outputing: (FREDERICK *PERIOD*).
```

And the end of this a period is output and the sentence is complete:

```
(LANCELOT KNEW THAT ANDREA KISSED WITH FREDERICK BECAUSE
    LANCELOT SAW THAT ANDREA KISSED WITH FREDERICK *PERIOD*)
```

A post-processor later capitalizes the sentence and proper names and replaces keywords like "*PERIOD*" with the proper punctuation.

Part IV: Evaluation

The research in any area of science can be roughly divided into two types: the discovery and definition of problems, and the discovery and definition of solutions. Every field has a mix of these two types of research. New areas of science tend to concentrate on the exploration of problems; established fields tend to be more heavily weighted towards solutions.

As a relatively young discipline, cognitive science concentrates on the first type of research. The ultimate goals of cognitive science are (1) to understand human intelligence, and (2) to build intelligent computer programs. But those goals are currently so distant as to seem unattainable. The immediate goal of cognitive science is to discover, explore, and define the principles of intelligence. Not necessarily to solve those problems, but rather to form a good understanding of what the problems are.

MINSTREL is research into the problems and issues in creativity. MINSTREL is neither a solution to the problem of creativity, nor a performance model of creativity. It is instead an *exploration* of the creative process in humans and computers. It is hoped that by building a theory and computer model of creativity consistent with human creativity, MINSTREL will provide foundations and directions for further research, in both human and computer creativity.

How, then, should MINSTREL be evaluated? Were MINSTREL intended as a solution to the "creativity problem" then evaluation would be straightforward. MINSTREL could be given a standardized creativity test, or matched against humans in creativity exercises. Or MINSTREL could be asked to publish a story. However, none of these are appropriate evaluations, because although MINSTREL is an implemented system, it is not a human-level performance model.

As explorative research in creativity, MINSTREL should achieve two goals. First, it should increase our understanding of the creative process. MINSTREL should be consistent with, elucidate, and extend the current theories about the processes of creativity. Second, MINSTREL should test the theories and models it proposes to determine their strengths, limitations, and to discover new areas of research. There are several appropriate methods for evaluating how well MINSTREL has achieved these goals.

First, MINSTREL can be compared to current psychological models of human creativity. This will reveal whether MINSTREL is consistent with established knowledge about the creative process, show how MINSTREL elucidates that knowledge, and indicate how MINSTREL extends the current understanding of human creativity. This comparison is given in Chapter 12.

Second, MINSTREL can be compared to previous work in cognitive science. The issue here is both to evaluate MINSTREL's performance in comparison to similar systems and to discover how MINSTREL fits in to the larger scope of cognitive science. A comparison of MINSTREL to previous computer models of creativity is given in Chapter 13; a comparison of MINSTREL to previous computer models of storytelling is given in Chapter 14.

Finally, MINSTREL's performance as a computer model can be evaluated. Although MINSTREL is not a human-level performance model of creativity, it must still demonstrate a level of performance that supports the plausibility of its process model. This evaluation is given in Chapter 15.

Together, these four chapters should give the reader a full understanding of MINSTREL in relation to the current research in the fields of psychology and cognitive science, and a better understanding of the limits and abilities of the computer program.

CHAPTER 12
Previous Work in Psychology

Previous work in the psychology of creativity has typically been concerned with five aspects of creativity [Shouksmith 1970]:

a. Study of the creative product.

b. Study of the creative process.

c. Developing measures and tests of creativity.

d. Study of personality traits of creative people.

e. Study of the creative environment.

Figure 12.1 Five Aspects of Creativity

This research focuses on a process model of creativity, so this chapter will focus on psychological studies of the processes of creativity. These studies can be broadly divided into two categories: theoretical and practical. Practical work is concerned with developing techniques for improving creative problem solving, and is often aimed at business executives, students and management personnel ([Cooley 1984], [Isaksen 1985] [Shone 1984]). Theoretical work is concentrated on defining and describing abstract mental processes that can explain creative actions. While "how to" advice concerning creativity is often thought-provoking, it generally lacks any strong fundamental model. For that reason, this review concentrates primarily on theoretical models of the creative process.

[Wallas 1926] proposed a model of creativity which has been widely accepted and expanded by later psychologists, including [Reichenbach 1938], [Kris 1953], [Koestler 1964] and [Stein 1967]. Recently, [Weisberg 1986] has proposed an alternate theory that challenges many of the basic assumptions of Wallas-derived theories. The following sections outline the major points of both these theories.

12.1 Wallas Theory

Wallas was inspired in developing his theory of creativity by the theory and experiences of Henri Poincare. In [Poincare 1913], Poincare described an incident that occurred to him while on a vacation that interrupted fifteen days of unsuccessful work on Fuchsian functions:

This example illustrates the components of Wallas's four step model of the creative process:

The incidents of travel made me forget my mathematical work. Having reached Coutances, we entered an omnibus to go some place or other. At the moment when I put my foot on the step, the idea *(that Fuchsian functions were identical to transformations in non-Euclidean geometry)* came to me, without anything in my former thoughts seeming to have paved the way for it... I did not verify the idea; I should not have had time, as, upon taking my seat in the omnibus, I went on with a conversation already commenced, but I felt a perfect certainty. On my return to Caen, for conscience' sake, I verified the result at my leisure.

Figure 12.2 Poincare Example

a. Preparation – problem investigation from all directions.

b. Incubation – subconscious consideration of problem.

c. Illumination – occurrence of a "happy idea".

d. Verification – consideration and debugging of solution.

Figure 12.3 Wallas Model of Creativity

Poincare prepared for his solution with fifteen straight days of attempted proofs concerning Fuchsian functions (a). He then went on a vacation, during which time he did not consciously work on the problem (b). Upon entering the bus, he had a "happy idea", a flash of insight (c). Later, upon his return to home, he confirmed the insight and produced a proof based upon the insight (d).

[Reichenbach 1938], [Kris 1953], [Koestler 1964] and [Stein 1967] have all proposed variants of this model, adding features such as elaboration and communication: [1]

a. Preparation – problem investigation, background study.

b. Incubation – non-conscious consideration of problem and bisociation.

c. Inspiration – a driven state of enlightenment.

d. Elaboration – verification and debugging, filling out of details.

e. Communication – presentation of solution to an audience.

Figure 12.4 Extended Wallas Model

During the preparation period, the creator studies in the field of the problem, becomes interested

1. The outline shown here is adapted from [Stein 1967] to include features from [Koestler 1964] and [Kris 1953].

in the field and the particular problem, and builds a fertile ground of information from which to create.

In the incubation period, it is assumed that subconscious processes work on the problem. [Koestler 1964] proposed that the basic subconscious process was *bisociation*, the bringing together of two previously unrelated ideas. In the case of the Poincare example, the bisociation was between Fuschian functions and transformations in non-Euclidean geometries. One of the most famous examples of the action of the unconscious in creativity is the story of Keluke's discovery of benzene. Keluke dozed off in his chair and had a dream of atoms dancing in a ring, which led directly to the discovery of the ring structure of benzene.

In the inspiration period, the new ideas brought together by incubation combine with the problem to generate, in a flash of insight (often called the "Aha!" effect), the solution to the problem. Gestalt psychology draws a parallel between the "Aha!" effect and the perceptual restructuring that occurs in viewing illusions like the Necker cube [Weisberg 1986]. In viewing the Necker cube, a sudden restructuring occurs that allows the viewer to perceive the cube in a new way. In the creative flash of insight, a similar restructuring occurs. The problem is suddenly transformed, and the creative solution is obvious.

The inspiration period is followed by a period of elaboration and debugging, a mundane time of standard problem-solving which is used to explore the ramifications of the solution and to eliminate problems. Elaboration may discover problems or previously unrealized constraints that will force further invention or problem-solving. The story of the discovery of the structure of DNA is filled with accounts of backtracking and problem-solving caused by the discovery of flaws in early models. Crick and Watson's first model had three strands, a center backbone, and only one-tenth of the necessary water (Watson had misrecalled an important figure), quite different from their eventual, correct model [Watson 1968].

In the communication stage, the creator polishes both his problem and his solution for presentation to an audience. The communication step is the most interesting addition to Wallas's original model and has resulted from a growing awareness in the field of psychology of the importance of environment, critical audience, and societal concerns to the act of creation.

Thus, the general creative process is divided into five sub-processes: preparation, incubation, inspiration, elaboration, and communication. Several things are known about these processes:

(1) It is recognized that the creative process does not follow the steps outlined above in a linear fashion [Stein 1967]. The creative process jumps from stage to stage in an erratic fashion. In particular, the elaboration stage leads back into preparation or incubation if problems are found in a solution.

(2) There is much preparation involved in creative thinking. An extensive and deep knowledge of a field is crucial to productive creative thinking.

(3) The creative process may involve changing the original problem. Following tangents is a

well-known and important aspect of creative thinking.

(4) There is emotional response to creativity. The well-known "Aha!" is indicative of the excited emotional response that results from discovering a solution to a particularly vexing or thorny problem.

The shortcoming of Wallas theory is that it is descriptive rather than explanatory. The creative solution is found during the inspiration step, but little is said about how a restructuring of a problem might occur that would lead to a solution, or even what such a restructuring might look like. While Wallas theory does highlight some of the important early steps of the creative process (preparation, incubation), the critical creative step is left unexplained.

A more recent theory [Weisberg 1986], addresses this concern.

12.2 Weisberg's Criticisms of Wallas Theory

[Weisberg 1986] questions nearly every fundamental proposition in the Wallas theory. Weisberg's major thesis is that creativity is an extension of ordinary problem-solving processes. It is not the sole purvey of "creative geniuses" and it does not involve any processes substantially different from normal problem-solving. In particular, Weisberg attacks the "myths" of the unconscious, of flashes of insight, and of divergent (or lateral) thinking.

In analyzing the role of the unconscious in creative thinking, Weisberg dissects a number of the more famous examples of the unconscious in creative thinking, including Keluke's discovery of benzene, Poincare's discovery, and Coleridge's story of the creation of *Kubla Khan*. In each case, he finds reason to doubt that the unconscious played a major role in creation. Poincare, for instance, worked day and night for fifteen consecutive days on Fuschian functions before leaving on his (previously planned) vacation. It is difficult to believe that a man who could work nonstop on a problem for two weeks would suddenly drop it from all conscious thought. It is much more likely that Poincare toyed idly with the problem during his vacation, mulling it over occasionally. If this was the case, then Poincare's breakthrough may be attributable to conscious thought, not a subconscious process.

Weisberg also discusses various laboratory experiments designed to test the role of the unconscious in creative problem-solving. In a study by Olton [Olton 1976], two groups of chess players were set to work on a difficult chess problem. One group was given additional break time, and asked not to work on the chess problem while on break. If subconscious processes were active, the group given the break could be expected to perform better because of the progress made by the subconscious processes during the break. In fact, the group given the break performed no better than the other group. Similar studies by J. Don Read and Darryl Bruce [Read 1982] question the value of incubation in simpler memory retrieval tasks.

Studies such as these question the role of the subconscious in creativity and problem solving. However, incubation may still be important for several other reasons. First, incubation may be

used for conscious idle toying with a problem, of the sort speculated in the Poincare example. Second, sensory input during incubation may provide information useful for solving a problem (i.e., breaking a writer's block).

The second principle Weisberg questions is inspiration. Under the Wallas (and Gestalt psychology) theory, inspiration occurs because of a sudden restructuring of the problem. The most famous of these "insight" problems is the nine dot problem, shown in Figure 12.5.

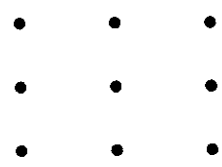


Figure 12.5 The Nine Dot Problem

The goal in the nine dot problem is to connect the nine dots using four lines and without lifting the pencil from the paper. Those readers unfamilar with this problem may wish to stop and attempt a solution.

Under the Gestalt view, the nine dot problem is difficult because people assume that the four lines must stay within the square. Once they realize that the lines can go outside the square they experience a sudden restructuring of the problem and solving the problem becomes trivially easy.

However, both laboratory studies and anecdotal evidence contradict this view. It took this author several minutes of scribbling to find the solution, even though he had solved the problem several times in the past and knew to go outside the square. In a study by [Burnham 1969] subjects were told before being given the problem to go outside the square. All the subjects used the hint, but only 20 to 25 percent were able to actually solve the problem. Both the study and the anecdotal evidence are in strong contradiction to the Gestalt view, which holds that the problem should be trivially easy after restructuring.

In general, the restructuring effect is based on the assumption that past experience interferes with creative problem solving, and it is breaking free of past experience that enables creativity. Weisberg holds the opposite view, that creative problem solving is based on past experience. This is not entirely in conflict with Wallas theory, which notes the importance of preparation and background, but it does reject inspiration due to sudden restructuring of problems.

The third myth that Weisberg attacks is the myth of divergent thinking. Divergent thinking differs from logical thinking in that it encourages free association without immediate criticism or consideration of possible problems. Divergent (or lateral) thinking is held to be of great importance to proponents of practical self-help guides to creativity such as [De Bono 1968] and [Osborn 1953].

Osborn developed a technique to foster divergent thinking called *brainstorming*. Brainstorming is a group technique used to create a flurry of ideas without being subject to criticisms that might block the production of ideas. Four basic rules guide brainstorming:

1. Criticism is discouraged.
2. Wild ideas are encouraged.
3. Quantity is encouraged.
4. Combination and improvement is encouraged.

Figure 12.6 Tenets of Brainstorming

After brainstorming has produced a large number of potential solutions, the solutions can be weighed and evaluated and a winner selected. The rules of brainstorming are intended to encourage divergent thinking in hopes of producing a solution that straightforward, logical thinking could not find.

Weisberg presents a number of laboratory studies that conclude that (a) group problem solving is no better than individual problem solving and (b) that deferring judgement produces more solutions than applying criticism, but not more *useful* ideas. While brainstorming often produces a large number of potential solutions, the number of useful solutions is no larger than in an equivalent problem-solving group that does not use brainstorming techniques. Weisberg concludes that the value of divergent thinking is largely spurious.

12.3 Weisberg's Model of Creativity

[Weisberg 1986] studiously avoids presenting a theory of creativity or a process model of creativity in people. Weisberg does, however, indicate some of the features he thinks will be necessary in a model of creativity.

Foremost, Weisberg rejects the nothion that creative thinking is fundamentally different from ordinary problem solving. Instead, he feels that creativity is an *extension* of problem solving. The same processes found in problem solving will be present in creativity, but used more flexibly and perhaps augmented by processes less used in normal problem solving situations.

Second, Weisberg believes that creativity is the result of a complex web of processes, many involving recall and modification of previous solutions. For example, Weisberg says that creative thinking requires

> *"a complex intertwining of perceptual and verbal processes concerning both the retrieval of information from memory and the modification of early solutions*

through perceptual imagination and verbal reasoning."

Finally, Weisberg believes that creativity is *incremental*, not a binary "yes/no" situation. Weisberg sees creativity in nearly everything a person does, because every situation in which a person finds himself is unique, and every response he makes novel. Thus great acts of creativity, like Picasso's *Guernica*, are only incrementally different from the acts of everyday life.

12.4 Evaluation of MINSTREL as a Psychological Model of Creativity

MINSTREL follows rather closely the guidelines [Weisberg 1986] gives for a model of creativity, but it also possesses some of the features of the Wallas model, and has differences from both theories.

The Wallas model describes creativity in four stages: (1) Preparation, (2) Incubation, (3) Illumination and (4) Verification. Of these four stages, MINSTREL most straightforwardly models preparation and verification.

In the Wallas model, preparation is a time for immersal in the problem, a time to build a wide and deep background in the problem domain. This is seen as crucial to the discovery of creative problem solutions. The MINSTREL model reflects this in its use of episodic memory. Episodic memory is used as the "fuel" for creativity. It is the source of knowledge that creativity heuristics modify to create new solutions. Thus for MINSTREL, too, it is necessary to have a wide and deep knowledge of the problem domain to facilitate creativity.

Verification is a step in which proposed problem solutions are examined for flaws and applicability to the original problem. If problems are found, the creative process jumps back to earlier steps to repair the proposed solution or invent a new solution. In the MINSTREL model, this is captured in the Assess step of problem-solving, and as was illustrated in the mechanical invention example, assessments can cause further invention.

Because MINSTREL models creativity as an integral part of the problem-solving process, MINSTREL does not have an explicit representation of the incubation and illumination steps as separate from the problem-solving process. However, MINSTREL does model one proposed feature of the illumination process.

[Koestler 1964] proposes that the basic process behind illumination is bisociation, the bringing together of two previously unrelated ideas. This mechanism is vague: there is no indication of the type of relation built between the two ideas or how these ideas are discovered. MINSTREL does, however, capture this basic notion of bringing together previously unrelated ideas. MINSTREL's creativity heuristics find and relate previously unconnected knowledge in episodic memory by transforming problems and adapting solutions. In this way, MINSTREL models a specific form of bisociation, albeit with a different mechanism. MINSTREL's bisociation is conscious and directed, but it does capture the fundamental idea of bisociation - the bringing together of previously unrelated knowledge to create a problem solution - a feature which is common

to both the Wallas and Weisberg models.

The crucial difference between the Wallas model and the MINSTREL model is the role of problem-solving in creativity. In the Wallas model, problem-solving surrounds the creative act. The first step (preparation) and the last step (elaboration) are both based in problem-solving. But the actual creative steps (incubation and inspiration) are separate from the problem-solving process. In fact, Wallas believes that inspiration is a function of subconscious (and hence presumably non-structured and possibly non-rational) thought. In the MINSTREL model, creativity is an integral part of the problem-solving process. Creativity is seen as an extension of problem-solving, and not as a separate and somewhat mysterious process.

A secondary shortcoming of the MINSTREL model is the lack of an emotional response to creativity. Wallas and others have noted that creativity is often accompanied by strong emotional responses, both in satisfaction to solving a difficult problem and in surprise at discovering an unexpected solution. MINSTREL makes no attempt to capture this feature of human creativity. The modeling of human emotional response is beyond the scope of this work. (But see [Mueller 1987] for a model of emotions as a creative drive.)

The MINSTREL model is in closer agreement with [Weisberg 1986]. Weisberg suggests three major features of creativity, all of which MINSTREL capture:

1. Creativity is an extension of problem-solving, driven by the failure of the problem-solving process.

2. Creativity is strongly memory based, involving the recall and modification of previous knowledge.

3. Creativity is incremental, a process that is active in mundane problem-solving as well as in great creative acts.

In MINSTREL, creativity is explicitly modeled as an extension of the problem-solving process that is used when normal modes of problem-solving fail. And as Weisberg suggests, these heuristics are largely involved with the recall and modification of past solutions. MINSTREL's creativity heuristics are also small, numerous and interact with one another, similar to the complex web of processes that Weisberg envisions.

Overall, MINSTREL is in general agreement with the Weisberg model and captures some of the features of the older Wallas model. More importantly, MINSTREL does not violate any of the recognized and accepted features of the creative process. MINSTREL also suggests some avenues for further psychological research. In particular, MINSTREL's incorporation of creativity into the recall process suggests psychological studies to see if in fact there is an integration of these two processes.

CHAPTER 13
Previous Work in Artificial Intelligence (Creativity)

The issues of creativity and invention have been examined in several artificial intelligence projects. This chapter discusses artificial intelligence research directly concerned with creativity (AM [Lenat 1976]), research which impinged on creativity (CHEF [Hammond 1990], DAYDREAMER [Mueller 1987],SOAR [Rosenblum 1991]), and work in the related area of scientific discovery (BACON [Langley 1987]). Previous work on storytelling (i.e., [Meehan 1976], [Lebowitz 1985] and [Dehn 1989]) is not reviewed here because it has not addressed the creativity question. Work on storytelling is reviewed in Chapter 14.

13.1 AM

[Lenat 1976] addressed the issue of creativity in the domain of mathematics. Lenat built a computer program called AM which looked at the problem of creative theory formation in mathematics: how to propose interesting new concepts and plausible hypotheses connecting them. 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.

As an invention system, AM has much in common with MINSTREL. But since AM is an exploratory system rather than a goal-directed system, and because the two systems operate in very dissimilar domains (mathematics vs. storytelling) they also have striking differences. A comparison of these systems can indicate what may be common, or different, in invention across different domains.

13.1.1 An Overview of AM

AM began with a set of 115 simple set-theoretic mathematics concepts, such as "Sets" and "Equality". These concepts were repeatedly expanded, generalized, specialized or combined in other ways to create new concepts. For example, AM might specialize the concept of "Numbers" to just those numbers with 2 as a factor, creating the new concept "Even Numbers". By repeatedly expanding old concepts and creating new concepts, AM explored the mathematical problem space.

AM's search process was controlled by a task agenda. The agenda held an ordered set of available tasks. Typical tasks were to fill in a facet (slot) of a concept or to restrict a concept to create a new concept. Tasks on the agenda were ordered by "interestingness", a rating which was determined when the task was added to the agenda, but which could later be modified if appopriate. The most interesting task on the agenda at any time was selected for execution, and after execution was finished, the cycle repeated. This was the sum of AM's control structure.

To measure interestingness and to achieve tasks, AM had two broad sets of heuristics. The first

type were interestingness heuristics. These heuristics rated tasks based on how interesting the concepts involved were, how likely the given task was to produce interesting results, how fruitful each concept had been in the past, and so on. When a new task (or concept) was created, interestingness heuristics placed reasons for pursuing that task on the task itself. Although the reasons themselves were simple English strings that AM could not manipulate, each reason had an associated numerical value, and AM could combine the weights of several reasons into one numerical representation of the interestingness of a task.

Figure 13.1 shows an example of one of AM's interestingness heuristics. This particular heuristic says that if two otherwise unrelated concepts happen to share the same boundaries, then they're both a bit more interesting. This heuristic is used in AM when AM notices that both "Primes" and "Numbers" share the same boundary cases (i.e., 0, 1). There's no reason to expect this, since Primes aren't immediately related to Numbers (AM derives Primes through the concept of divisors) and so this makes both concepts more interesting.

```
A concept is interesting if its boundary accidentally coincides with the
boundary of another, very different, interesting concept.
```

Figure 13.1 Example Interestingness Heuristic

A special set of interestingness heuristics called "Suggest" heuristics were used when the only tasks on the agenda had low interestingness. When this situation was detected, AM would execute a number of Suggest heuristics in hopes that one of the heuristics could "suggest" a more interesting line of study.

Figure 13.2 illustrates two Suggest heuristics. The first heuristic creates a new concept when it notices that two other concepts have some intersecting examples. The second heuristic adds a new task to the agenda, which will itself result in the creation of a new concept if a way can be found to generalize X.

```
IF some (but not most) examples of X are also examples of Y,
THEN create a new concept defined as the intersection of X and Y.

IF very few examples of X are found,
THEN add the following task to the agenda: Generalize X, for the reason: "X's
are quite rare; a slightly more restrictive concept might be more
interesting."
```

Figure 13.2 Example Suggest Heuristics

A second set of heuristics were used to achieve tasks. Each task heuristic applied to a particular task, such as "Fill-in the Examples slot of a concept". Task heuristics were organized according to a class hierarchy of concepts. Each task heuristic was attached to the most general concept to which it applied. When AM tried to achieve a task, it searched from the particular concept asso-

ciated with the task up through the class hierarchy for heuristics that would apply to the particular task. These heuristics were then executed from the most specific to the most general until one of the heuristics achieved the task.

Figure 13.3 shows an example of a task heuristic. This task heuristic applies to the "Fill-in the Examples slot of a concept" task, and is attached to the "Any-Concept" level of the class hierarchy (i.e., the most general concept level). This heuristic suggests finding examples of a concept by applying an operation whose range is the concept. AM uses this heuristic to fill in examples of "Even Numbers" by noticing that the operation "Double" produces even numbers.

```
To fill in examples of concept X,

Find an operation whose range is X, and find examples of that operation being
applied.
```

Figure 13.3 Example Task Heuristic

In retrospect, it might have been more clear to call task heuristics "plans". Each task heuristic is in fact a plan for achieving a specific goal in AM's mathematical problem domain, and as with most problem-solving systems, these plans are organized in a class hierarchy according to how widely applicable they are.

13.1.2 Performance of AM

With a base of 115 initial concepts and over 250 heuristics, AM was able to discover about 185 additional concepts. AM followed a line of reasoning along the path:

set theory ->
equality ->
size ->
numbers ->
multiplication ->
squares ->
factorization ->
primes

This was for AM's "best" run. Other runs produced essentially the same results, but with more side-tracking. Prime factorization was the most advanced concept AM developed. Most of AM's new concepts derived from the two base concepts of "Parallel-Join-2" (which led directly to the concept of multiplication) and "Coalesce" (which led to squares and square roots along with a large number of uninteresting concepts).

Of the 185 concepts AM discovered, Lenat rates 25 as "winning" concepts, 100 as acceptable,

and 60 as "losing". Of the created concepts, two were totally new and unexpected to both AM and Lenat. One of these involved the idea of "maximally divisible numbers" (the antithesis of prime numbers) and the other an application of Goldbach's conjecture to geometry. And while AM suggested the idea of "maximally divisible numbers", Lenat and another mathematician were responsible for developing the idea into an interesting result.

13.1.3 Comparison of AM and MINSTREL

The following sections compare AM and MINSTREL in three areas: (1) control structure, (2) memory, and (3) creativity heuristics.

13.1.3.1 Control Structure

On the surface, it appears that AM and MINSTREL have very similar control structures. Both have goal agendas in which the goals are ordered by priority. In AM, the priorities are decided by interestingness heuristics. In MINSTREL, the priorities are determined by the author-level model of storytelling. Each program has a similar execution cycle: select the highest-priority goal, solve that goal and repeat.

However, the two goal agendas actually serve very different purposes. AM is an exploratory system: it seeks outward from a set of base concepts into the search space of the mathematical domain, guiding itself into areas it finds interesting. AM doesn't know where it is going. In effect, each task on the agenda is a trailhead that will lead AM somewhere. The highest priority task on the agenda represents the trailhead AM thinks most likely to lead to an interesting place.

MINSTREL, in contrast, is a goal-directed system. MINSTREL begins processing with a specific goal (normally to tell a story about a particular story theme) and creates and achieves new subgoals in an effort to satisfy that initial goal. MINSTREL knows where it is going. Each goal on the agenda represents a single step along the path MINSTREL has charted out to reach its destination. The highest priority task on the agenda is the next step along the trail.

One reflection of this difference in design is that MINSTREL expects to execute and achieve every goal that is placed on the agenda. Every goal that reaches MINSTREL's agenda is a goal that MINSTREL has decided is necessary to tell an interesting, well-written story that illustrates a particular story theme. Consequently MINSTREL tries to achieve each of these goals, and processing does not end until the agenda is empty. AM, on the other hand, never expects to empty the agenda, or indeed, to ever finish its exploration. Ideally, AM would continue to run indefinitely, always exploring the most interesting concepts it finds, and always adding new tasks to its agenda as it expands out into the search space.

AM's interestingness heuristics are a second reflection of this difference in design. AM has a large set of interestingness heuristics that assign priorities to the goals on the agenda. In MINSTREL, goal priorities are assigned when goals are created, and reflect domain-specific problem-

solving ordering constraints. For example, when telling a theme-based story, instantiating the theme has priority over developing characterization, so theme goals are given higher priority than characterization goals. Because MINSTREL knows in advance what its ultimate goal is, it does not need heuristics to guide its selection of immediate goals.

(On the side, we note that MINSTREL's boredom heuristic sounds analogous to AM's interestingness heuristics. However, MINSTREL's boredom heuristic is used to determine whether a solution is creative or not; not to determine what goal to consider next. AM can determine whether or not a concept is interesting, but it has no notion of whether a concept is creative.)

So although MINSTREL and AM appear on the surface to have a similar control structure, they actually use it in very different ways. But this is misleading, because neither MINSTREL nor AM are creative at the top level of control. At the top level of control, both systems are planners. They find a plan to achieve the current goal, and execute that plan. Creativity actually occurs during the execution of these plans, not during the planning process. At the level of creativity, MINSTREL and AM do indeed share some similarities.

In AM, creativity occurs in the task heuristics that create new concepts. For example, the concepts of number and size are invented by a heuristic that canonizes concepts. The interesting thing here is that AM's task heuristics are independent of the agenda control structure. Their only interaction with the agenda is to sometimes add a task to the agenda; AM's control structure could be completely changed without affecting the task heuristics that create new concepts.

Similarly, in MINSTREL creativity is achieved in the Transform-Recall-Adapt Cycle, which is embedded in imaginative memory. Imaginative memory is used by the plans that achieve MINSTREL's author-level goals, but the TRAMs used to invent new scenes and problem solutions are independent of the author-level control structure. Thus MINSTREL's creativity is also independent of MINSTREL's control model.

What then is the control structure for these two systems at the creativity level? In both systems, heuristics are found by searching the concept hierarchy rooted at the current task object. For example, to invent a new method for a knight to kill a monster, MINSTREL searches for applicable TRAMs first on the general concept of an "act" (because killing something is a specific type of act) and then upwards through more general concepts. In AM, heuristics to achieve a task are found by looking first on the immediate concept, then on any concepts which are generalizations of that concept, and so on upward through the concept hierarchy. When heuristics are found, they applied from most specific to most general to the current task. If one of the discovered heuristics succeeds, both systems return the successful creation. The two systems share similar methods for organizing and applying creativity heuristics.

One way in which the two systems differ at the level of creativity is in what occurs if a creativity heuristic fails to create a solution. In MINSTREL, if the heuristics fail to invent a solution, creativity recurses (through imaginative memory) and a new set of heuristics can be applied on top of the original heuristics. By combining heuristics, MINSTREL increases the power and scope of its creativity. In contrast, AM uses only a single level of heuristics. If no single heuristic can

achieve a task in AM, then the task will fail.

Strangely, this difference is not immediately apparent in the performance of the two systems. Both are inventive, both are successful in their problem domains. How is that AM needs only a single application of heuristics, while MINSTREL must apply its heuristics in combination to achieve its goals?

Partially this is explained by AM's large number of heuristics. AM has over 250 heuristics, while MINSTREL has about 20. By applying heuristics in combination, MINSTREL achieves greater creative power in a smaller representation. But this is not the whole story. The real reason that this difference is not readily apparent in the behavior of the two systems is that AM's exploratory nature masks failed tasks. If AM had but a single task to achieve, a failure would be obvious. But AM is self-directed, and generates many, many tasks to choose from. If a particular task fails to be achieved, it simply means that AM will not explore farther in that direction. The interesting concepts AM fails to find will not be noticed, so long as AM finds *some* interesting concepts.

There are four conclusions to be drawn from this comparison of the control structures of MINSTREL and AM.

First, it appears that the difference between exploratory creativity, such as that modelled in AM and in DAYDREAMER [Mueller 87] and goal-directed creativity, such as that modelled in MINSTREL and in EDISON [Hodges 88], may be largely attributed to the creator-level goals, and not in the creative process itself. In other words, the difference between exploratory and goal-directed creativity is in the aims of the creator, and not in the way he creates.

Second, because AM is an exploratory system, it requires interestingness heuristics to restrain and direct its explosive search. Because it is a goal-directed system, MINSTREL does not have a problem with explosive search. Conversely, because MINSTREL is a goal-directed system, some care must be taken in the selection of creativity heuristics to solve a problem, because it is important to MINSTREL that each problem be solved. AM, on the other hand, can be more casual about selecting which creativity heuristic to apply, because the success or failure of any particular task is not crucial to the operation of the system.

Third, the success of AM using a broad, single level of creativity heuristics suggests that exploratory creativity may require a "weaker" level of creative effort than goal-directed creativity.

Finally, both MINSTREL and AM index creativity heuristics according to the concepts on which they act. Although not by itself sufficient evidence to show that human creativity is organized in this way, this does suggest an area for future study.

13.1.3.2 Creativity Heuristics

Both MINSTREL and AM rely on a body of creativity heuristics to invent new concepts. In MINSTREL, TRAMs are used to modify problem descriptions and adapt solutions. In AM, task heuristics create new concepts by modifying old concepts. It would be enlightening if MINSTREL and AM shared some heuristics, or if their heuristics seemed to have parallel features. But this is not generally the case.

Almost all of AM's heuristics deal with (1) judging the interest of a concept, (2) suggesting a new task, or (3) achieving a task. Because MINSTREL is a goal-directed system, it has no analogs of the first two types of tasks. MINSTREL does have analogs of the third type of heuristics - its author-level plans - but due to the wide disparity between the two problem domains, there is little overlap between MINSTREL's author-level plans and AM's task heuristics.

Typical AM tasks are filling in examples of a concept, checking boundary cases of a definition, determining the domain and range of a function and finding relationships between concepts. Typical MINSTREL author-level goals are instantiating scenes in a story that illustrate the theme, checking actions in the story to see if they're consistent, and building suspense in the climax of a story. The two domains are so different that there is no easy analogy between the two task sets. There are some similarities - checking a story for consistent character actions is somewhat similar to checking the boundary cases of a concept definition - but these analogies are too weak for any kind of meaningful comparison.

However, a small subset of AM's heuristics do have a stronger similarity to MINSTREL's creativity heuristics. These are the heuristics that create new concepts. Figure 13.4 illustrates a task heuristic from AM that creates a new concept by disjoining two previous concepts. This heuristic suggests that if AM has ever looked at two concepts which share a property, then it should create a new concept defined as the disjuction of those two concepts.

```
TO fill in generalizations of concept X,
IF some conjecture exists about "all X's and Y's"
THEN create a new concept, a generalization of both X and Y, defined
     as their disjunction.
```

Figure 13.4 Example Concept Creation Heuristic

Compare this heuristic with the heuristic from MINSTREL illustrated in Figure 13.5. TRAM-Similar-Outcomes-Implicit uses episodic memory to determine if two concepts have ever been used interchangeably. If they have, it then interchanges them in the current problem.

Both of these heuristics use a previous example of the interchangeability of two concepts to create the generalization "X and Y are functionally the same thing". The two systems differ in that AM is interested in the generalization itself, while MINSTREL is interested in the generalization

TRAM-Similar-Outcomes-Implicit

Transform Strategy

> Suppose that X is a problem constraint. Try to find two episodes in memory which are exactly the same except that in one episode Y takes the place of X. If you can find such a Y, then substitute Y for X in the original problem, and try to solve this new problem.

Adapt Strategy

> Replace Y with X in the created solution.

Figure 13.5 Example of MINSTREL TRAM

as a way to solve a specific problem.

This is typical of the two systems. MINSTREL's creativity heuristics use episodic memory to build generalizations, specializations, analogies, and so on, and use those to solve specific problems. AM's creativity heuristics build new concepts based on specialization and generalization of known concepts, and those concepts are an end in themselves. While this may seem a subtle distinction, it is in fact a major difference between the two systems. This becomes apparent when we look at the types of mistakes the two systems can make.

Suppose that MINSTREL is trying to create a scene in which a knight achieves status by impressing his king, and that MINSTREL's episodic memory contains three episodes: (1) A princess rides a horse to a fair, (2) The king rides his horse into battle, and (3) A knight impresses a princess by giving her a bunch of wildflowers. Using TRAM-Similar-Outcomes and episodes (1) and (2), MINSTREL can deduce that kings and princesses are interchangeable (because both kings and princesses ride horses). Using this knowledge and episode (3), MINSTREL will create a scene in which a knight tries to impress the king by giving him wildflowers. This unlikely scene is a mistake[1] that occurs because TRAM-Similar-Outcomes-Implicit isn't sophisticated enough to realize that similarity in riding horses doesn't imply similarity in life goals.

What would happen if AM were applied to this same problem? Using the disjunction heuristic shown above and episodes (1) and (2), AM would create a new concept "Things that are Princesses or Kings" (the intermediate concept that MINSTREL uses to guess that princesses and kings might be interchangeable). This concept would be found uninteresting (princesses and kings share very few features) and not developed further. So the same heuristic and base concepts that led MINSTREL to a mistake caused AM only to create an uninteresting concept.

This illustrates one of the basic differences between AM and MINSTREL. MINSTREL's creativity is goal-directed. MINSTREL uses creativity to solve problems. Consequently, MIN-

1. Although this is an *interesting* outcome – it might lead to the telling of a humorous story, for instance – it is still a mistake in the context of the original problem.

STREL sometimes makes errors in the solutions it invents. AM, on the other hand, uses creativity to explore a problem space, without putting that knowledge to any immediate use. The primary "mistake" that AM can make is to create an uninteresting concept.

So, while the two systems have some similar heuristics, they use them in different ways. MINSTREL uses its creativity heuristics to solve particular problems by guessing (hopefully) reasonable new solutions. At times, the new solutions will not be reasonable. AM uses its creativity tasks to create (hopefully) interesting new concepts. At times, the new concepts will not be interesting. This parallel but different operation reflects the basic differences between exploratory and goal-directed creativity.

It is also interesting to note that heuristics from the two systems which appear very different can lead to similar sorts of discoveries. Consider, for example, TRAM:Generalize-Constraint and "Argument Coincidence", shown in Figure 13.6.

MINSTREL: TRAM-Generalize-Constraint

Transform

1. Remove a problem constraint.

Adapt

2. Add the removed problem constraint to the discovered solution.

AM: Argument Coincidence

IF a function F takes two arguments of the same domain
THEN propose a function G(x) = F(x,x).

Figure 13.6 Dissimilar Creativity Heuristics from MINSTREL and AM

The MINSTREL heuristic, TRAM-Generalize-Constraint, suggests that to find a solution to a problem, you can try generalizing away one of the constraints of the problem, solve the generalized problem, and then add the constraint back to the solution you found. MINSTREL uses this TRAM when asked to invent a method for a knight to commit suicide. TRAM-Generalize-Constraint removes the constraint that the victim be the knight himself, and recalls an episode in which a knight killed a monster with his sword. When this recollection is adapted to the original problem, MINSTREL ends up with a solution in which a knight kills himself by fighting himself with his sword.

The AM heuristic, Argument Coincidence, is used by AM to discover squaring of numbers. After AM has discovered the notion of multiplication, it notices that TIMES takes two arguments of the same domain, and Argument Coincidence creates the new function definition SQUARE(x) = TIMES(x,x) by substituting the first argument for the second argument.

Although these two heuristics are very different, they arrive at very similar inventions. Both "suicide by fighting yourself" and "squaring a number" involve making an action reflexive. This illustrates a common feature of creativity: that a particular creative solution can be arrived at by a number of different methods. In fact, this is reflected within MINSTREL and AM as well. Both programs can and do discover the same new thing in several different ways.

What this says about creativity in general is interesting: That the products of creativity may not give any useful insight into the processes of creativity. Presented with an example of creativity like "The knight hits himself with a sword" we may be tempted to deduce a creativity heuristic which "makes an action reflexive". But despite the obvious appropriateness of this heuristic, the invention may have come from a completely different direction.

13.1.3.3 Memory in MINSTREL and AM

AM lacks a cognitive model of memory. In AM, all the base concepts and created concepts are stored in a single list, which is searched linearly when AM needs to find a specific concept. In contrast, MINSTREL has an organized episodic memory based on a psychological model of human memory called the "context-plus-index" model ([Schank 1982][Kolodner 1984][Reiser 1983],[Reiser, Black & Abelson 1985],[Reiser 1986]). MINSTREL's episodic memory is used to organize, store and recall character-level goals, plans and actions, author-level plans, and the story being created.

The need to search a long list of concepts no doubt slows AM's performance, but the impact is more than a simple loss of speed. The lack of an organized memory requires AM to explicitly handle tasks that would be better handled by memory.

For example, AM has heuristics that are intended to notice when concepts share surface similarities. Figure 13.7 illustrates a rule that AM uses to notice that two concepts are similar, and then to create an explicit analogy between the two concepts. AM uses this rule to create the analogy between equal collections of different objects ("Equal") and same-length collections of the same object ("Size").

```
If the current task has just created a canonical specialization
C2 of C1 with respect to F1 and F2,
Then add the following entry to the Analogies facet of C2:
    <C1     F1      Operations-on-and-into(C1)>
    <C2     F2      Those operations restricted to C2>.
```

Figure 13.7 Analogy Heuristic

Unfortunately, this rule will fail if C2 wasn't created in a single step as a canonical specialization of C1. If AM does not create the concept of collections of the same object by canonizing the concept of collections, it will never notice the similarities between the two concepts. To be as-

sured of finding similarities between concepts, AM must constantly compare all the members of the list of concepts. This is both inefficient and cognitively implausible.

Noticing similarities of this sort is precisely the role of episodic memory in cognitive models of memory [Schank 1982]. Episodes are organized according to similarities. When a new episode is indexed into memory, the problem-solver is reminded of past similar episodes. If AM had an episodic memory, then indexing a new concept - such as collections of the same object - would remind it of other similar concepts, and allow it to efficiently draw analogies and explore connections it would not otherwise discover.

A second function of episodic memory is to create generalizations based upon episodes that share similar features. MINSTREL, for example, learns that fight scenes usually involve knights fighting monsters, by creating a generalization based on the specific fight scenes it has in memory. AM has no way to automatically notice and create such generalizations and again must rely on specific heuristics to find these generalizations.

Finally, episodic memory can be used to judge the interestingness of a concept. People generally find unique situations more interesting than repetitive ones, and this notion can be captured and tested by episodic memory. MINSTREL has a boredom assessment which uses episodic memory to determine whether or not a particular story scene has grown boring. The scene is indexed into episodic memory, and if it is indexed with too many similar scenes, it is judged boring and rejected. In contrast, AM has no dynamic sense of interest, and can only judge the interestingness of a concept based on static, pre-programmed interestingness heuristics. This inability to remember its past work and to use that to judge the interestingness of a line of reasoning in fact leads AM into trouble. At one point in its invention, AM creates a large number of new concepts based on the idea of coalescing two functions. In the end, these new concepts prove largely fruitless and waste AM's time. If AM had an episodic memory and a boredom assessment, it would become bored with coalescing functions after a short time, and avoid this problem.

AM used heuristics to achieve many of the functions of episodic memory. Heuristics were used to find similar concepts, and to create generalizations between concepts with shared features. But these implementations were inefficient and fragile. As was shown in the example above, these heuristics can fail if conditions change in small ways. The lack of an episodic memory also restricts AM to static interestingness evaluations, which leads to some wasteful processing.

13.2 CHEF

MINSTREL is a case-based reasoning system. It uses a memory of past cases to solve problems and to invent new solutions. Case-based reasoning has been applied to a number of different domains and problems, including natural language understanding [Lebowitz 1980][Kolodner 1984], moral and ethical judgements [Simpson 1985][Bain 1986], and legal reasoning [Goldman ??][Rissland and Ashley 1986]. One area of research in case-based reasoning relevant to MINSTREL is plan repair.

Plan repair is the problem of taking a plan which has failed, determining the cause of the failure, and then changing the plan so that the failure will not re-occur. In many ways, creativity is similar to plan repair. Creating a new solution to a problem can be viewed as a process that "repairs" a known plan so that it can be applied to a new problem. For this reason, it is informative to compare creativity to plan repair to see what the two tasks have in common, and how they differ.

CHEF [Hammond 1990] was a system that applied case-based reasoning to the problem of plan repair. CHEF is a case-based planner that creates and debugs plans in the domain of Szechwan cooking. Given a set of input goals (desired tastes and textures of the dish), CHEF finds a suitable plan in its memory and if the plan is not perfect, repairs it to meet the input goals. CHEF was also able to learn from its mistakes, and to avoid making similar mistakes in the future, although that will not be discussed here.

13.2.1 An Overview of CHEF

CHEF is given a set of input goals consisting of the desired tastes and textures of a dish and asked to produce a suitable recipe. For example, CHEF may be told to find a recipe that produces a dish with savory beef and crisp, garlic-flavored broccoli. CHEF then searches its memory for a recipe that satisfies these goals. Usually, it cannot find a recipe which satisfies all the input goals, so it finds a recipe which best fits the input goals. To do this, CHEF uses domain knowledge about how important each goal is, and recalls the plan that meets the best combination of the input goals.

After the plan is found, it is passed to a "Modifier" that can perform simple adaptations of the plan to the current problem. For example, the Modifier knows how to substitute similar ingredients to turn the "Chicken With Snow Peas" recipe into the "Beef with Snow Peas" recipe. The modified plan is then given to a simulator, which simulates cooking the dish.

The results of cooking the dish along with the original input goals are passed to the "Repairer". The Repairer notices three types of errors. The first type occurs when some step of the plan cannot be performed because a precondition of the step was violated by an earlier step in the plan. For example, in creating a recipe for a shrimp stir-fry, CHEF's recipe fails because marinating the shrimp makes them too slippery to shell. The second type of error occurs when one of the input goals is not achieved. In creating a recipe for beef with broccoli, CHEF's recipe fails because the beef sweats liquid that makes the broccoli soggy, violating one of the input goals. The third type of error occurs when the recipe has an undesired side effect. In creating a recipe for fish stir-fry, CHEF's recipe results in an iodine taste that is undesired.

When Repairer notices an error, it characterizes the error according to the nature of the violated condition, the temporal relationship between the error and the steps that caused the error, the nature of the error, and the relationship between the error and the planner's goals. These characteristics are used to recall a Thematic Organization Packet (TOP) [Schank 1982]. In CHEF, TOPs are generalized descriptions of planning problems. For example, SIDE-

EFFECT:BLOCKED-PRECONDITION is a TOP that describes situations in which the side effect of one step of a plan violates the precondition of a later step. The "Monkey and the Cookie Jar" is the classic illustration of this problem. To get a cookie out of a jar with a narrow neck, a monkey will reach into the jar, grab the cookie, and try to withdraw his hand. But holding the cookie makes a fist too big to pass the neck of the jar. The step of possessing the cookie has a side effect that blocks the later step of removing the cookie from the jar.

In CHEF, each TOP has associated with it a number of repair heuristics. One repair that can be applied to SIDE-EFFECT:BLOCKED-PRECONDITION is REORDER, which changes the steps of a plan so that the blocking precondition does not occur until after the step it would block. In the case of the monkey and the cookie jar problem, REORDER suggests removing the cookie from the jar before trying to possess the cookie.

Once suitable repair heuristics have been found, they are tested for applicability to the current problem. The repair heuristics that are applicable are then used to suggest possible repairs to the faulty plan, and the repair which is most suitable is applied. Suitability of repairs is judged by a set of domain-specific heuristics such as "It is easier to add a preparation step than a cooking step."

Finally, the repaired plan and the problem which led to the repair are added to memory so that in future planning situations CHEF can anticipate and avoid similar planning errors.

13.2.2 Comparison of CHEF and MINSTREL

The following sections compare MINSTREL and in three areas: (1) control structure, (2) heuristics, (3) organization of memory.

13.2.2.1 Control Structure

Both MINSTREL and CHEF are planning systems. They are given an initial goal or set of goals, and attempt to find or create a plan to satisfy those goals. However, they approach this problem in different ways. If CHEF does not already know a perfect plan for a problem, it finds a plan that shares the most surface features with the current problem and repairs that plan to solve the current problem. If MINSTREL does not already know a perfect plan for a problem, it changes the problem until it does, and then uses those changes to guide the adaptation of the plan from that problem. These control structures are illustrated in Figure 13.8.

One result of the differences between the control structures of these two programs is that CHEF uses its repair heuristics more efficiently than MINSTREL uses its creativity heuristics.

CHEF	MINSTREL
1. Select a plan for the input problem using a best fit to the surface features of the problem.	1. Look for a plan that fits the input goals.
2. Notice specific failures with the selected plan and their causes.	2. If you cannot find a plan that fits, Transform the problem.
3. Use (2) to repair the plan.	3. Recursively try to solve the new problem.
	4. Use (2) to Adapt the solution to the original problem.

Figure 13.8 Comparative Control Structures in CHEF and MINSTREL

When a failure occurs in the execution of a plan, CHEF has the advantage of knowing the error that occurred and the step of the recipe that led to the error. CHEF uses this knowledge to restrict its search for a repair heuristic (by finding a TOP that describes this particular sort of error) and finds repair heuristics that apply specifically to this type of error, and which are known to correct the error. CHEF doesn't have to search amongst its repair heuristics. The failure provides the knowledge necessary to find an applicable heuristic.

In contrast, MINSTREL cannot know ahead of time what creativity heuristics it will apply to create a solution, because MINSTREL does not know what previous solution will form the basis of the new solution. The purpose of the Transform portions of MINSTREL's TRAMs is to search episodic memory for useful knowledge. But MINSTREL does not know if the knowledge needed is there until it looks for it. Consequently, MINSTREL may apply many TRAMs that fail. Contrast this with CHEF, where every repair heuristic is known to solve a particular plan fault.

However, it is easy to see that a creative system must necessarily be more inefficient than a repair system. If a creative system were to decide a priori what previous solution it would adapt to create a new solution, it could restrict its search as CHEF does. But this would greatly limit its creativity, because it would only be able to create based on the previous solution selected.

In fact, selecting the solution to be repaired prior to the repair process itself can limit CHEF as well. Suppose that CHEF knew two recipes: "Beef Casserole" and "Chicken With Snow Peas". Based on similar surface features, CHEF would select "Beef Casserole" when creating a new recipe for "Beef with Broccoli" because they share a common ingredient. Now CHEF's repair process will fail, because repairing "Beef Casserole" into an acceptable "Beef With Broccoli" is too difficult. Had CHEF known what repairs it had available, and used those to help guide its initial plan selection process, it would have chosen "Chicken With Snow Peas"

because, although it is initially a poorer fit, it can be repaired. And this is precisely the effect of MINSTREL's less efficient search: to use its knowledge of the adaptation heuristics it has available to guide the search for an initial solution. MINSTREL trades efficiency for the ability to find unobvious solutions.

A second consequence of this difference between the two systems is that MINSTREL is more robust than CHEF. To return to the above example, when CHEF selects "Beef Casserole" as the basis for a "Beef with Broccoli" recipe, it fails because it is unable to perform the necessary repairs. CHEF expends little effort selecting an initial plan because it relies on the power of its repair heuristics to correct any problems. And yet these repair heuristics will surely fail in some cases.

MINSTREL, on the other hand, uses knowledge of its adaptation heuristics to guide the search for an initial solution. Because each TRAM combines a problem transformation with a corresponding solution adaptation, if MINSTREL finds an initial solution, the adaptations necessary to apply that solution to the original problem will succeed. MINSTREL is not committed to creating a new solution from some arbitrarily selected initial solution. In MINSTREL, the search for an initial solution is an integral part of the creative process. Hence MINSTREL only fails if memory does not contain *any* solution suitable for adaptation to the current problem, or if time constraints stop MINSTREL's search before the solution is discovered.

It would appear from these two systems that the difference between plan repair and creativity can be characterized by efficiency. Plan repair is an efficient and directed process which is necessarily limited. Creativity is less efficient, but also less limited and more robust. It may be that plan repair and creativity are techniques for dealing with planning failures that lie at different ends of a scale of effort. Plan repair may be commonly used where plans are plentiful, where surface features correctly predict deep features, and where most failures require simple corrections. Creativity may be used where surface features are deceptive, where a premium is placed on discovering new knowledge, and where plan repair has failed.

13.2.2.2 Heuristics

CHEF contains seventeen general-purpose repair heuristics. These heuristics are listed in Figure 13.9. These heuristics fall into four broad categories. They can (1) replace a step with a new step, (2) add a new step that corrects a failure, (3) change the order of steps, or (4) change an object used in a step. How do these compare to the creativity heuristics used in MINSTREL?

Interestingly enough, MINSTREL's creativity heuristics and CHEF's repair heuristics have little in common. Of CHEF's seventeen heuristics, only one - ALTER-ITEM - has any analogs in MINSTREL's TRAMs. ALTER-ITEM replaces an existing item in a step with one that has the desired features but not an undesired one. MINSTREL has several TRAMs that perform similar functions, such as TRAM:GENERALIZE-ACTOR, which suggests replacing the actor of a action with a different type of character. But even this is not a close correlation.

ALTER-PLAN:SIDE-EFFECT	ALTER-PLAN:PRECONDITION	RECOVER
REORDER	ADJUST-BALANCE:UP	ADJUST-BALANCE:DOWN
ADJUNCT-PLAN:REMOVE	ADJUNCT-PLAN:PROTECT	ALTER-TIME:UP
ALTER-TIME:DOWN	ALTER-ITEM	ALTER-TOOL
SPLIT-AND-REFORM	ALTER-PLACEMENT:BEFORE	ALTER-PLACEMENT:AFTER
ALTER-FEATURE	REMOVE-FEATURE	

Figure 13.9 CHEF's Repair Heuristics

If we look again at the kinds of actions CHEF's repair heuristics take, we see that they concentrate on manipulating the steps of a many-stepped, causally-connected plan. CHEF's heuristics reorder steps, remove steps, substitute alternate steps, and so on. Little of this has applicability in the King Arthur domain. At the level at which MINSTREL creates stories, most actions are discrete units. And while some story events are causally connected, most are connected by the thematic needs of the story, by the author's intentions in telling an interesting story, and by character emotions. So it may be that MINSTREL and CHEF do not share similar heuristics because they operate in domains in which planning is very different.

This is supported by the fact that there is greater similarity between CHEF's repair heuristics and the heuristics used in MINSTREL's mechanical invention example (see Chapter 15). MINSTREL's output in the mechanical invention example can be viewed as a "recipe" for the creation of a mechanical device. Likewise, MINSTREL's consistency and efficiency assessments can be viewed as simplified "simulators" that detect errors in the device recipe. This similarity between problem domains leads to a greater similarity in heuristics.

For example, TRAM:Generalize-Power-Source is a heuristic that adds a new "step" (i.e., device component) to a mechanical device recipe. And TRAM:Power-Converter has an analog in CHEF's recipe domain: ALTER-FEATURE. TRAM:Power-Converter adds a step to the mechanical device recipe that converts one (undesirable) type of power input to another (desirable) type. ALTER-FEATURE adds a step that will change an undesired attribute of a recipe to a desired one.

The differences between CHEF's repair heuristics and MINSTREL's storytelling heuristics, along with the similarities between CHEF's repair heuristics and MINSTREL's mechanical invention heuristics suggest that creativity heuristics may fall into broad categories based on common features of problem domains. Domains such as recipe creation and mechanical device invention, which have many-stepped, causally-interconnected plans, may require different types of creativity heuristics than domains like storytelling.

13.2.2.3 Organization of Memory

In CHEF, repair heuristics are organized according to TOPs. TOPs describe particular types of plan failures and organize repair heuristics that are known to correct those failures. Consequently, when CHEF finds a planning failure, it can immediately find repairs that will apply to that failure. MINSTREL, on the other hand, organizes its creativity heuristics into broader categories. All creativity heuristics which apply to goals are indexed together, all creativity heuristics which apply to actions are indexed together, and so on. And once the creativity heuristics that apply to a situation are found, they must be tried to see if they succeed in creating a solution. Why is there a difference in memory organization between these two systems?

CHEF can organize its repair heuristics much more explicitly than MINSTREL because, at the time it looks for heuristics, CHEF has much more knowledge than MINSTREL about the problem it is trying to solve. When CHEF looks for a repair heuristic, it knows the specific error it is correcting, the plan in which the error occured and all other pertinent information. When MINSTREL looks for creativity heuristics, it knows only the problem it is trying to solve. It does not yet know the plan it intends use as a basis for creating a solution to that problem.

There are two reasons why MINSTREL does not have any knowledge about the solution it will use to create a new solution. First, MINSTREL incorporates the search for an initial solution into the process that adapts the solution. As pointed out above, this is a necessary cost for any system that intends to display creativity. Second, MINSTREL is not able to exhaustively examine the contents of episodic memory. If MINSTREL could examine all the episodes in memory, it could use that information to improve the organization and selection of creativity heuristics. But psychological research has shown that people cannot use their memory in this way [Cohen 1989]. MINSTREL uses a model of episodic memory based on work by Schank [Schank 1982] and Kolodner [Kolodner 1984], and elaborated and tested by Reiser, Black and Abelson [Reiser 1983][Reiser, Black & Abelson 1985][Reiser 1986], who term it the "context plus index" model. In this model, memories must be recalled by use of a context and a set of indices that uniquely identify the memory. Memory cannot be enumerated, so MINSTREL cannot use that information to guide the selection of a heuristic.

13.3 DAYDREAMER

One aspect that distinguishes creativity in different problem domains is the extent to which outside constraints interact with the invention process. In some domains creativity can run free with only minimal constraints. In other domains, the problem constrains very closely the form of the solution. Is the process of invention different across these types of problems?

DAYDREAMER [Mueller 1987] is a computer program which lies at one end of this scale. DAYDREAMER is a computer program that produces daydreams in reaction to being rejected for a date by a famous movie star. Daydreaming differs from storytelling largely in the number and types of problem constraints. One can daydream a ray gun that makes noisome children vanish; to use such a ray gun in a story would require some clever justifications. Similarly a day-

dream can jump from topic to topic at the daydreamer's every whim, but a story must provide a constant and recognizable framework for the reader's understanding.

By comparing the role of creativity and invention in these two programs, we hope to gain some insight into the role that problem constraints play in the invention process.

13.3.1 DAYDREAMER

Like all of the other systems discussed in this chapter, DAYDREAMER was fundamentally a planning system. DAYDREAMER had an agenda of goals, and it used its knowledge of plans and the world to achieve those goals. However, DAYDREAMER differed from previous planning systems in two ways. First, DAYDREAMER initiated its own goals. For example, if someone did something to cause DAYDREAMER to fail to achieve a goal, DAYDREAMER would entertain a goal of hurting that person in return. Second, by relaxing problem constraints and using some simple creativity heuristics, DAYDREAMER was able to discover and invent plans that a more efficient and directed planning system would not find.

Figure 13.10 (taken from [Mueller 87]) is an example of one of the daydreams DAYDREAMER has after being told that it was turned down for a date by Harrison Ford. This example illustrates how DAYDREAMER initiated its own goals, and illustrates the planning nature of DAYDREAMER.

REVENGE-1

I study to be an actor. I am a movie star even more famous than he is. I feel pleased. He is interested in me. He breaks up with his girlfriend. He wants to be going out with me. He calls me up. I turn him down. I get even with him. I feel pleased.

Figure 13.10 Example from DAYDREAMER

The creativity in DAYDREAMER is directed towards developing new plans and arises from two mechanisms: serendipity and mutation.

[Mueller 1987] defines serendipity as "the accidental juxtaposition of a recalled experience...with a problem." This new, accidentally recalled experience can then be used to help solve the problem. In the trace shown in Figure 13.11, DAYDREAMER has failed to find a way to contact Harrison Ford for a date. Then it gets the day's mail[1] and this experience jogs a possible solution. (DAYDREAMER's output is shown in bold face.)

The whimsical "What do you know!" is DAYDREAMER's method of indicating that it has just

1. Obviously, DAYDREAMER doesn't actually get mail. Events of this sort in DAYDREAMER's world are simulated.

RECOVERY-4

I want to be entertained. I feel interested in being entertained. I go outside. I grab the mail. I have the UCLA Alumni directory.

Input: *Carol Burnett went to UCLA.*
Input: *Carol's telephone number is in the UCLA Alumni directory.*

What do you know! ...I have to read Harrison Ford's telephone number in the Alumni directory. I have to know where he went to college...

Figure 13.11 Serendipity Example from DAYDREAMER

found some unexpected information that might be of use in solving a goal that was previously set aside. By being unexpectedly reminded of the fact that famous people may appear in Alumni directories, DAYDREAMER has found a solution to its goal that it wouldn't have otherwise considered.

The DAYDREAMER serendipity example begs an immediate question: Why didn't DAYDREAMER think of using the alumni directory when it was first trying to contact Harrison Ford? The answer is that while DAYDREAMER may have known about alumni directories, it hadn't correctly indexed that knowledge so that it would be recalled and applied while solving the "meeting a famous person" problem. And this is why serendipity can be a source of new problem solutions. Serendipity is a source of creativity because it alerts the problem-solver to a source of information that the problem-solver hadn't realized might be useful.

Mutation is the second source of creativity in DAYDREAMER. Mutation consists of "performing a structural transformation on an unachieved action subgoal of [a] goal" [Mueller 87]. This transformed subgoal must then be investigated to see if it leads to any new solutions to the original goal.

In the example shown in Figure 13.12 (from [Mueller 87]), a mutation transforms "Harrison Ford tells me his telephone number" into "Harrison Ford tells someone else his telephone number". DAYDREAMER can then discover a new way to meet Harrison Ford, based on getting the "someone else" to divulge Harrison Ford's telephone number.

DAYDREAMER knows of three types of mutations (permute objects, generalize object, and change action type) and three concept types to which these mutations can be applied (MTRANS, PTRANS and ATRANS). When a problem cannot be solved, DAYDREAMER begins applying these three mutations to any applicable parts of the problem. Even with a limited number of mutations, this produces a large number of mostly useless concepts.

To control this explosion of concepts, DAYDREAMER employs a spreading activation mechanism. DAYDREAMER's spreading activation performs the same function as AM's interesting-

RECOVERY-3

I have to ask him out. I have to call him. I have to know his telephone number. He has to tell me his telephone number. I have to know where he lives. Suppose he tells someone else his telephone number. What do you know! This person has to tell me his telephone number...

Figure 13.12 Mutation Example from DAYDREAMER

ness heuristics. However, in DAYDREAMER, the area of the problem space that is "interesting" is the area near the original problem, and spreading activation keeps DAYDREAMER focused in that area.

Mutation can be creative because it can, through a blind search, discover alternate plans to solve a problem. Generally mutation will be inefficient, because most of the alternate concepts it generates will not be applicable to the current problem. But DAYDREAMER is intended to use weak methods to discover solutions that a more directed planner would not.

13.3.2 The Role of Problem Constraints in Creativity

Perhaps the most intriguing generality that can be drawn from comparing DAYDREAMER and MINSTREL concerns the role of secondary problem assessments in the creative process. DAYDREAMER, the system with an unconstrained problem domain, lacks any kind of assessment mechanism. So long as they solve a problem, DAYDREAMER does not care if the solutions it creates are efficient, clever or even original. MINSTREL relies on a variety of solution assessments. For storytelling, MINSTREL relies primarily on an originality assessment. For the more constrained problem domain of mechanical invention, MINSTREL requires additional assessments to assure an efficient and correct solution. This is a surprising result: Why should the most highly constrainted problem domain also have the most secondary solution assessments? Why make an already difficult problem more difficult?

The answer may be that these systems reflect a general principle about problem-solving: The more constrained a problem is, the more important it is to get a good answer, even beyond the explicit problem constraints. In daydreaming, nothing will be lost if the solution to a problem is fanciful or inefficient. In all likelihood, the solution will never be used. And if it is, it will be reexamined before use. Mechanical invention, however, is very purposeful, and this is reflected in the tightly constrained problems. If a solution can be found, it will be put to use, and so it is in the problem-solver's best interests to find the best solution possible, even if that means applying constraints that weren't explicitly mentioned in the problem.

13.4 BACON

Another area of creativity which artificial intelligence has explored is scientific discovery. Scientific discovery is the process of creating and verifying theories that describe the real world. In contrast to activities such as storytelling, scientific discovery is data-driven and builds descriptive theories about the world. BACON [Langley 1987] is a program that explores issues in scientific discovery. Comparing MINSTREL to BACON sheds some light on the differences between scientific discovery and goal-directed creativity.

13.4.1 Overview of BACON

BACON discovers scientific laws by making inductions from sets of experimental data. Given a set of experimental data, BACON postulated new terms based on the data until it found a constant term representing an invariant relationship present in the input data, i.e., a law which describes the data. To do this, BACON had a set of three heuristics which it used to postulate new terms. These heuristics are shown in Figure 13.13.

(1) If the values of a term are constant, then infer that the term always has that value.

(2) If the values of two terms increase together, then consider their ratio.

(3) If the value of one term increases as a second term decreases, then consider the product of those terms.

Figure 13.13 BACON Heuristics

One of the scientific laws which BACON discovers using these heuristics is Kepler's Third Law of Planetary Motion: The cube of a planet's distance from the Sun is porportional to the square of its period. Figure 13.14 summarizes how BACON applied its discovery heuristics to experimental data to discover this law.

Planet	Distance	Period	(D/P)	(D^2/P)	(D^3/P^2)
A	1.0	1.0	1.0	1.0	1.0
B	4.0	8.0	0.5	2.0	1.0
C	9.0	27.0	0.33	3.0	1.0

Figure 13.14 BACON Discovers Kepler's Third Law

The first three columns of this table represent the experimental data. Planet is the independent variable. It takes on three values: A, B, and C. Distance and Period are the two dependent values. This data is supplied to BACON by the user; BACON does not suggest experiments or se-

lect data.

The second three columns represent three applications of BACON's discovery heuristics. The term (D/P) is discovered when BACON notices that period increases as distance increases and applies heuristic (2). BACON then notes that distance increases as the new term (D/P) increases, and uses heuristic (3) to create a new term which is the product of these two terms (D^2/P). Finally, BACON applies heuristic (3) again, this time to (D/P) and (D^2/P), creating the last term, (D^3/P^2). This final term is a constant, and so BACON finishes, having rediscovered Kepler's third law:

$$D^3/P^2 = k$$

By repeatedly applying simple heuristics, BACON was able to discover a hidden regularity in the input data and express this as a formula. As can be seen from this example, BACON is strongly data-driven. At each cycle of the discovery process, BACON looks for certain patterns in dependent terms and applies a heuristic which creates a new dependent term.

Later versions of BACON were extended to handle multiple independent variables, to propose intrinsic properties of observed terms, to detect symmmetry, and to operate on symbolic rather than numeric data. However, the emphasis in BACON has always been on the discovery of invariant relationships from experimental data.

13.4.2 Comparing MINSTREL and BACON

MINSTREL and BACON have little in common. In addition to the difference between scientific discovery and storytelling, there is also a wide difference in purpose between the two systems. BACON is a system to discover a hidden regularity in a set of input data. MINSTREL is a system that uses input data to creatively solve new problems.

Perhaps the most striking difference between BACON and MINSTREL is that BACON makes no reference to the semantic meaning of the data it manipulates. BACON has no in-depth understanding of its problem domain. BACON discovers Kepler's Third Law of Planetary Motion, but BACON has no prior knowledge about planets, gravity, or motion. The data it used to deduce Kepler's Third Law might just as well have described an electronic circuit.

One consequence of this is that the theories BACON develops are strictly descriptive. BACON's theories describe invariant relationships in its input data, without any causal explanation of that invariance. In this respect, BACON appears to capture a very early stage of scientific discovery: the stage at which the scientist notices an interesting feature of the world. BACON is a program that notices a particular class of interesting features of the world, but which is unable to form any explanatory theories about them.

MINSTREL, on the other hand, has an in-depth knowledge of both storytelling and the King Arthur domain, and uses this knowledge extensively to reason about story and character-level

goals. Unlike BACON's discovery heuristics, MINSTREL's creativity heuristics make use of semantic knowledge about the problems they manipulate. TRAM:Intention-Switch, for example, makes use of knowledge about how actors act intentionally to achieve their goals.

Further, MINSTREL does not develop descriptive theories about its domain except to use them as evidence to support causal theories. For example, MINSTREL may notice that "Princesses and hermits are similar things" but only as support for the theory that "Princesses and hermits share the same goals". So while MINSTREL does create some (primitive) descriptive theories about its domain, it does so only as a step towards creating a causal theory that can be used for problem-solving.

BACON and MINSTREL also differ on their level of generality. BACON is a program that solves a very specific type of goal: discovering a hidden regularity in a set of numerical input data. BACON's discovery heuristics are very specific to this task. MINSTREL is a more general-purpose problem-solver, with a model of creativity that is not tied to any particular problem type. MINSTREL has demonstrated this generality by solving a variety of problems in different domains. And while BACON was able to re-discover theories from several different scientific domains, the class of problems solved remained the same (i.e., polynomial equations describing numerical data).

A final difference between BACON and MINSTREL is that BACON is data-driven. BACON attempts to discover *the* relationship which accurately describes the experimental data. To this end, the important issue in BACON is the efficient exploration of the space of possible numerical relationships. In contrast, MINSTREL is driven not by experimental data, but by author goals. Because there are many stories which will fit MINSTREL's author goals, MINSTREL concentrates on heuristics which efficiently achieve a goal, rather than heuristics which efficiently discover a particular result.

BACON and MINSTREL are not so much different solutions to the same problem as they a re solutions to different parts of the same problem. A simple model of a scientist working in a new problem domain might have two stages. In the first stage, the scientist bootstraps himself into the new domain by observing the domain and building descriptive theories about the domain. In the second stage, the scientist uses those descriptive theories as guides for building explanatory theories about the domain, and uses these theories to solve problems in the domain. BACON is a computer program that represents the first stage of this process: using observational data about the world to build descriptive theories. MINSTREL represents the second stage: using descriptive theories and previous explanatory theories to create new explanatory theories and apply them to problem-solving.

13.5 SOAR

Our final comparison of MINSTREL to previous work in computational models of creativity is to SOAR [Rosenbloom 1991]. Unlike DAYDREAMER, AM, and BACON, SOAR is not a system that explicitly looks at creativity or a creative problem domain. Instead, SOAR is a set of functional tools intended to support the modeling of general intelligence. SOAR provides a frame-

work for the modeling of cognition, and much of the interest in SOAR lies in evaluating how well the mechanisms provided by SOAR support the cognitive tasks to which they are applied.

An interesting question to ask about SOAR is whether the mechanisms provided by SOAR could be used to implement MINSTREL. If not, the missing capabilities will point to areas where SOAR is lacking or where MINSTREL relies on different fundamental assumptions about cognition.

13.5.1 An Overview of SOAR

Functionally, the SOAR architecture provide three levels of processing and one mechanism for learning.

The lowest level of the SOAR architecture is long-term memory. All of SOAR's long-term knowledge is stored in a single production memory. Each production is a condition-action pair that performs its action when its condition is met. All of SOAR's knowledge, whether procedural, declarative or episodic, is represented and stored as one or more productions.

Memory access occurs when a predicate value in working memory matches a condition in long-term memory. When this occurs, the production in long-term memory is fired, and the actions of that production are performed. Generally, this will result in additional predicate values being added to working memory and the cycle repeating.

There are two important things to note about retrieval from SOAR's long-term memory.

First, all memory accesses are done in parallel across the entire production memory. If a predicate in working memory matches the conditions of a hundred productions in memory, then all of those productions are fired in parallel.

Second, memory access continues until working memory becomes quiescent. The cycle of matching productions in long-term memory and firing their actions repeats until no new predicates are being asserted into working memory.

The next level is termed the decision level. This level represents the ability of a general intelligence to generate or select a course of action responsive to the current situation. The decision level proceeds in a two phase elaborate-decide cycle. During elaboration, long-term memory is accessed until quiescence is reached. This results in all knowledge relevant to the current situation being brought into working memory. This includes both knowledge of applicable plans and preference knowledge, i.e., which plan to choose if a choice exists. During the decide phase an architecturally fixed decision procedure uses the knowledge found during the elaboration phase to select an action from the available alternatives.

The highest level of the SOAR architecture is termed the goal level. This level represents the ability of a general intelligence to direct its behavior towards some end. This level is based on

the decision level. Goals are set whenever a decision cannot be made, that is, when the decision procedure reaches an impasse. An impasse can occur because there are no alternatives available or when insufficient preference information exists to make a selection between alternatives. Whenever an impasse occurs, the goal level generates the goal of resolving the impasse. This new problem is then passed to the decision level (in a fresh context) and SOAR attempts to solve this new problem. This may result in another impasse, the generation of a new goal, and so on, in the kind of sub-goaling behavior typical of symbolic planning systems.

It is interesting to note that because SOAR operates essentially in parallel, it can work on several different goals at once. Goals that are at an impasse will be inactive, but if several other goals are available, SOAR will continue trying to solve these goals in parallel. If a high-level goal succeeds, it can opportunistically make lower-level goals irrelevant.

Finally, SOAR also implements a general learning mechanism termed chunking. In chunking, sequences of problem-solving that occur in subgoals are compressed into a single production. If SOAR reaches an impasse in problem-solving, generates several sub-goals and eventually resolves the impasse, that entire context is compressed and saved into a single production. Similar future impasses will never occur, because the learned production will apply to break the impasse. In effect, the learned production is a compiled shortcut. The meaning of the learned production depends upon the type of impasse. If the impasse occurred because no alternatives were available, then the learned production represents a new alternative, i.e., a new plan. If the impasse occurred because SOAR could not distinguish between available alternatives, then the production represents a selection criterion, and so on.

13.5.2 Comparison of MINSTREL and SOAR

SOAR is an general architecture of cognition that is based upon a number of assumptions about the nature of intelligence. Four critical assumptions [Norman 1991] are:

1. That there is a uniform computational architecture including a single, uniform long-term memory structure.

2. That there is a single form of learning (chunking).

3. That all intelligent operations are performed by symbol manipulation.

4. That all reasoning is done by search within a uniform problem space.

The principled use of these assumptions has led to the SOAR architecture, and where MINSTREL differs from SOAR it is because of a difference in these underlying assumptions.

For the most part, MINSTREL shares assumptions (3) and (4). Like SOAR and unlike current research in neural networks, MINSTREL is based on a model of cognition as symbol manipulation. Similarly, MINSTREL operates at a certain level as a search mechanism: looking through

the space of all problem solutions to find a solution for it's current problem. On assumptions (1) and (2), though, MINSTREL differs from SOAR.

SOAR's first assumption is of a uniform computational architecture and a single, uniform long-term memory. It is primarily on the second part of this assumption that MINSTREL differs from SOAR. SOAR assumes a uniform long-term memory with certain characteristics: accessed in parallel and repeatedly until quiescence. This differs radically from MINSTREL's model of memory.

First, in the MINSTREL model of episodic memory, individual episodes can be recalled only if the recall indices are unique or nearly so. If a set of recall indices matches more than a few episodes, then memory can only recall generalizations about those episodes. To recall a particular episode, the recall indices must be *elaborated* until they uniquely specify an episode. Compare this to the SOAR architecture, where underspecified indices recall everything, rather than nothing.

Second, the MINSTREL model of creativity focuses a great deal of effort on efficient and directed search of episodic memory for knowledge that can be used for invention. The Transform portion of MINSTREL's TRAMs are search heuristics that find areas of episodic memory likely to contain useful knowledge. The purpose of MINSTREL's TRAMs is to limit the amount of knowledge and processing a creative problem-solver faces when trying to invent a new problem solution. In contrast, SOAR accesses memory repeatedly until no more applicable knowledge can be found:

> *During elaboration, the memory is accessed repeatedly, in parallel, until quiescence is reached; that is, until no more productions can execute. This results in the retrieval into working memory of all of the accessible knowledge that is relevant to the current decision.* [Rosenbloom 1991] pp 296.

The purpose of MINSTREL's model of memory and MINSTREL's concentration on the directed, efficient search for useful knowledge is to limit the size and scope of knowledge that MINSTREL must process in order to be creative. The explicit assumption in SOAR is that parallel processing can permit a problem-solver to efficiently handle potentially large amounts of knowledge while trying to create a problem solution.

Finally, SOAR assumes is that there is a single, uniform learning mechanism: chunking. Chunking takes problem-solving sequences and compresses them into single steps, saving the chunked sequences for future use. This is similar to the way that MINSTREL saves invented problem solutions in episodic memory for future use.

But various experiments with MINSTREL have shown that chunking of this sort - even when combined with creativity - cannot adequately explain all learning (see Chapter 3, Section 13.1). Learning from outside experience is also necessary. Various psychologists have also claimed that there exist several types of learning ([Norman 1978][Anderson 1985]).

13.5.3 Conclusions

MINSTREL and SOAR differ on several fundamental points, including memory and learning. For this reason, it would be difficult to implement MINSTREL in SOAR. Not impossible, of course, because SOAR is a powerful and expressive system that could be applied to almost any problem. But any implementation of MINSTREL in SOAR would be "unnatural" in the sense that few of SOAR's primitive operations could be used directly to implement features of the MINSTREL model.

It is difficult to draw any concrete conclusions from this comparison because SOAR has never been applied to the problem of creativity. Consequently it is difficult to tell whether the SOAR model lacks the features needed to implement MINSTREL (1) because SOAR has never been applied to these types of problems and hence these shortcomings have not been noticed, or (2) because the SOAR architecture implements some fundamentally different and better ways to perform creative cognitive tasks.

CHAPTER 14
Previous Work in Artificial Intelligence (Storytelling)

14.1 Introduction

There have been several previous research projects which have looked at the issue of computer storytelling. These projects include TALESPIN ([Meehan 1976]), UNIVERSE ([Lebowitz 1985]) and STARSHIP ([Dehn 1989]). This chapter reviews this previous work and contrasts it with MINSTREL.

14.2 TALESPIN

The seminal work in computer storytelling was the TALESPIN program [Meehan 76]. TALESPIN told stories about the lives of simple woodland creatures. The thrust of TALESPIN was planning; the process of telling a story involved giving some character a goal and then watching the development of a plan to solve that goal.

In this example from [Meehan 76], John Bear has been given some initial knowledge about the world and a goal to satisfy his hunger. The resulting story is shown in Figure 14.1.

John Bear is somewhat hungry. John Bear wants to get some berries. John Bear wants to get near the blueberries. John Bear walks from a cave entrance to the bush by going through a pass through a valley through a meadow. John Bear takes the blueberries. John Bear eats the blueberries. The blueberries are gone. John Bear is not very hungry.

Figure 14.1 Example TALESPIN Story

This story illustrates both the strengths and weaknesses of TALESPIN.

First, the story is very believable and logically consistent. TALESPIN has a good grasp of how characters can go about solving their goals. This is a reflection of TALESPIN's strong planning component and illustrates (at least intuitively) that planning is an important component of storytelling.

On the other hand, the story is uninteresting and pointless. In general, hearing about the plans used to solve various goals (particularly goals like "satisfy hunger") is not particularly interesting and does not convey any particular point to the reader. This too is an artifact of TALESPIN's strong character-level planning component. TALESPIN never seeks to fulfill any story criteria above the level of character planning.

As an attempt to add a higher level of control, a component was added to TALESPIN that forced the planner to follow a pre-designated template for a story. This template reduced the story of "The Fox and the Crow" to a plan to use flattery to gain control of some object. Given two char-

acters and the template, TALESPIN was then able to use its world knowledge to set up a situation where one character would use flattery to gain control of some object the two valued in common. A story similar to "The Fox and the Crow" resulted. The resulting story *was* more interesting than TALESPIN's normal stories, underlining the importance of higher-level goals in creating an interesting story.

14.2.1 Comparison of MINSTREL and TALESPIN

The most important result to come out of TALESPIN was that character-level planning was necessary but not sufficient for storytelling. This was both a positive and a negative result. On the positive side, it identified the important role that character-level planning fills in creating a plausible story. On the negative side, it showed conclusively that character-level planning alone was not sufficient to create interesting stories.

This result is reflected in two ways in MINSTREL.

The impact of the positive aspect of this lesson is seen in MINSTREL's consistency goals. MINSTREL's author-level consistency goals have almost the same focus as TALESPIN. TALESPIN was concerned with character-level goals: how they arise, how they are achieved, and their effects on the story world and other characters; MINSTREL has consistency goals to make sure that character goals arise properly, are achieved, and that actions in the world effect the characters and the story world properly. TALESPIN identified the importance of character-level planning in creating a plausible and consistent story, and this is reflected in the important role that MINSTREL's consistency goals play in story creation.

The impact of the negative aspect of the TALESPIN lesson is seen in MINSTREL's other categories of author-level goals: theme-related goals, dramatic writing goals, and presentation goals. Each of these represents a type of author concern that TALESPIN did not address. To character-level planning, which helps create plausible stories, MINSTREL adds additional layers which help create interesting, well-written and well-presented stories.

Thus the impact of TALESPIN on MINSTREL was that it identified one important component of stories (plausibility) and the process by which that could be achieved (character-level planning). To this MINSTREL has added other important story components and the processes by which they are achieved.

On a more general level, we can identify several features of MINSTREL that TALESPIN lacked: (1) a model of creativity and episodic memory, (2) knowledge of story structure, and (3) an explicit author-level model.

MINSTREL is built upon a model of creativity which permits it to invent story scenes in ways that TALESPIN could not. TALESPIN used a limited, static set of character-level plans to create the events in the TALESPIN story world. Consequently TALESPIN's stories were limited to combinations of plans and actions already known to TALESPIN. In contrast, MINSTREL's sto-

rytelling component can make use of the underlying process of creativity to extend its knowledge of character-level goals and plans, and to combine and change that knowledge in new and unexpected ways. And because MINSTREL's model of creativity is built upon episodic memory, MINSTREL has the ability to change its behavior - to learn as it tells stories. Lacking any model of author memory, TALESPIN could not evolve or change.

A second way in which MINSTREL and TALESPIN differ is in their knowledge of story structure. TALESPIN had no knowledge of what a story is, or how a story's structure reflects its purpose. TALESPIN's stories were traces of the simulation process and nothing more. In contrast, MINSTREL has explicit knowledge about story structure and the the purpose of storytelling. MINSTREL has an explicit representation for a variety of story themes and understands storytelling as an exercise in illustrating these themes in interesting ways. Furthermore, MINSTREL has knowledge of other aspects of story structure independent of theme, such as foreshadowing and introduction scenes. MINSTREL's knowledge of story structure enables MINSTREL to consistently produce stories that have all of the expected thematic and dramatic structures we expect of stories.

Finally, MINSTREL is distinguished from TALESPIN by its explicit author model. TALESPIN had no model of the author separate from the character-level planning task. The purpose of TALESPIN as an author ("Tell a story by simulating the lives of characters in a storyworld") was represented only implicitly in the architecture and code of the system. In contrast, MINSTREL's goals and plans as an author have explicit representation. As it tells a story, MINSTREL creates and manipulates conceptual representations of its goals as an author. At the same time, it manipulates the goals and plans of the characters in the story it is telling. This distinction - the explicit representation of the author as separate from the story - enables MINSTREL to reason about its storytelling process in ways that TALESPIN could not.

14.3 UNIVERSE

TALESPIN can be characterized as a "planful" storyteller, because it is most concerned with the plans and actions of its story characters, and MINSTREL can be characterized as a "thematic" storyteller, because its primary concern is to create a story that illustrates a particular theme. Michael Lebowitz's UNIVERSE program ([Lebowitz 1984][Lebowitz 1985]) is best characterized as a "character interaction" storyteller. UNIVERSE looks at the problem of developing stories which involve complex and continuing relationships between well-developed characters. This type of serial, character-based storytelling is best exemplified by soap operas.

There are two important elements to the UNIVERSE program: character descriptions and plot fragments. Character descriptions are detailed representations of story characters involving stereotypes, individual traits, individual goals, interpersonal relationships and a history. Plot fragments are abstract descriptions of story events that will achieve particular author-level goals. To tell a story, UNIVERSE instantiates plot fragments from a library of character descriptions. Instantiating one plot fragment may generate opportunities for more plot fragments, resulting in a continuing, serial story.

An example of a plot fragment, Forced-Marriage ([Lebowitz 1985]), is shown in Figure 14.2. Forced-Marriage was inspired by the following plot synopsis from the soap opera *Days of Our Lives*:

> Liz was married to Tony. Neither loved the other, and, indeed, Liz was in love with Neil. However, unknown to either Tony or Neil, Stephano, Tony's father, who wanted Liz to produce a grandson for him, threatened Liz that if she left Tony, he would kill Neil. Convinced that he was serious by a bomb that exploded near Neil, Liz told Neil that she did not love him, that she was still in love with Tony, and that he should forget about her. Eventually, Neil was convinced and he married Marie. Later when Liz was finally free from Tony (because Stephano had died), Neil was not free to marry her and their troubles went on.

```
PLOT FRAGMENT: forced-marriage

GOALS: (churn ?him ?her) {prevent them from being happy}
TIME SCALE: months
CHARACTERS: ?him ?her ?husband ?parent
CONSTRAINTS:  (has-husband ?her)
              (has-parent ?husband)
              (< (trait-value ?parent 'niceness) -5)
              (female-adult ?her)
              (male-adult ?him)
SUB-GOALS:
  (do-threaten ?parent ?her "forget it") {threaten ?her}
  (dump-lover ?her ?him) {have ?her dump ?him}
  (worry-about ?him)     {have someone worry about ?him}
  (together * ?him)      {have ?him get involved with someone}
  (eliminate ?parent)    {get rid of parent, breaking threat}
  (do-divorce ?him ?her) {end the unhappy marriage}
  (or (churn ?him ?her)      {either keep churning, or}
      (together ?her ?him))  {try and get ?her and ?him back together}
```

Figure 14.2 Plot Fragment Forced-Marriage

Each plot fragment consists of an author-level goal or goals that it achieves, a list of characters involved in the plot fragment, a list of constraints that identify when the plot fragment can be applied, and a list of sub-goals. Forced-Marriage achieves the author-level goal of "churning" two lovers (one male and one female with a husband) by forcing the female lover to stay in her marriage. (Churning is an author-level goal to prevent characters from being happy.) This involves four characters: the two lovers being churned, the husband of the female lover, and the parent of the husband. The constraints section of the plot fragment makes sure that the characters have the necessary relationships and that the husband's parent is sufficiently nasty to make a threat

plausible[1].

The remainder of the plot fragment is a list of author-level subgoals, which will eventually translate into a sequence of story events. In Forced-Marriage, the sub-goals result in a threat from the parent of the husband which causes the wife to dump her lover, followed by the lover getting together with another character, followed by the elimination of the threat and a subsequent divorce. Finally, the plot fragment can be continued by either further churning the characters or by getting them together.

UNIVERSE's algorithm for using story fragments to create a story is straightforward, relying primarily on the richness of its story fragments and characters to create a variety of interesting stories. UNIVERSE maintains a graph of author-level goals, and how they have been achieved. At each cycle of the program, a goal is selected and expanded via a story fragment. This cycle continues repeatedly until some goals reach the "ground level", i.e., actual events, at which time they are told to the reader using a natural language generation component[1]. This algorithm is shown in Figure 14.3 ([Lebowitz 1985]).

Pick a goal with no missing pre-conditions

Pick a plot fragment for that goal,
achieving extra goals, if possible

"Execute" the plan, including
adding new goals to the goal graph and
"telling" (producing output), if appropriate

Figure 14.3 UNIVERSE Story-Telling Algorithm

One important feature of the UNIVERSE story-telling algorithm is that UNIVERSE tries to achieve multiple goals when selecting a plot fragment to achieve the current author-level goal. By this method, UNIVERSE "interleaves" plot-lines, creating the kind of intricately interwoven plot outlines typical of soap operas.

Figure 14.4 shows a partial trace of plot generation using Forced-Marriage, taken from [Lebowitz 1985]. The initial author-level goal is to churn NEIL and LIZ. At each step of the storytelling process, UNIVERSE selects an author-level goal to achieve (in left-to-right order) and a plot fragment (called "plans" in the trace) to achieve that goal. As each plot fragment is applied, the subgoals it creates are added to the story, and if they represent character-level actions they are output and expansion continues on the next sub-goal.

One of the difficulties with UNIVERSE is that the format of UNIVERSE's plot fragments limits

1. Several important constraints seem to be missing from this plot fragment: (1) There is no constraint that the two characters being churned are lovers, (2) There is no constraint to assure that the husband is married to the female lover instead of some other character, (3) There is no constraint to assure that the parent is the parent of the husband.

1. As of [Lebowitz 1985], no natural langauge component existed.

```
*(tell '(((churn neil liz))))

working on goal -- (CHURN NEIL LIZ)
Several plans to choose from FORCED-MARRIAGE LOVERS-FIGHT JOB-PROBLEM
-- using plan FORCED MARRIAGE

working on goal -- (DO-THREATEN STEPHANO LIZ "forget it")
-- using plan THREATEN

>>> STEPHANO threatens LIZ: "forget it"

working on goal -- (DUMP-LOVER LIZ NEIL)
-- using plan BREAK-UP

>>> LIZ tells NEIL she doesn't love him

working on goal -- (WORRY-ABOUT NEIL)
-- using plan BE-CONCERNED
Possible candidates -- MARLENA JULIE DOUG ROMAN DON CHRIS KAYLA
Using MARLENA for WORRIER

>>> MARLENA is worried about NEIL

working on goal -- (TOGETHER * NEIL)
Several plans to choose from SEDUCTION DRUNKEN-SNEAK-IN
-- using plan SEDUCTION
Possible candidates -- DAPHNE RENEE
Using DAPHNE for SEDUCER

>>> DAPHNE seduces NEIL

[...]

>>> RENEE tries to kill STEPHANO

[...]

>>> LIZ and TONY got divorced
```

Figure 14.4 Partial Trace from UNIVERSE [Lebowitz 1985]

the types of information that can be expressed. This is apparent in the trace shown in Figure 14.4. In this trace, MARLENA worries about NEIL, but DAPHNE seduces him. It would be more pleasing to have the worrier and the seducer be the same role (why else would the author express the worry event?) and this was probably the intent. But the limitations of UNIVERSE's knowledge representation for author-level plans, which limits communication between sub-goals to role-bindings, causes this to fail.

Another shortcoming of UNIVERSE is that each story event derives directly from a single au-

thor-level goal. Consequently, a story event cannot serve two purposes, nor be modified to achieve a second purpose. The result is that UNIVERSE's stories are one-dimensional. Comparing the story generated in the trace above:

```
>>> STEPHANO threatens LIZ: "forget it"
>>> LIZ tells NEIL she doesn't love him
>>> MARLENA is worried about NEIL
>>> DAPHNE seduces NEIL
>>> RENEE tries to kill STEPHANO
>>> LIZ and TONY got divorced
```

to the story that inspired Forced-Marriage reveals that the story UNIVERSE tells lacks depth. In the original story, the bomb explosion helps Liz believe Stephano's threat. In the generated story, there is no bomb explosion, because UNIVERSE can neither recognize nor correct the lack of evidence. Where a human author or MINSTREL would recognize that the story is inconsistent without some evidence to support Liz's belief, UNIVERSE cannot, because once created, a story event passes from UNIVERSE's attention. Consequently UNIVERSE cannot apply many author-level goals to the same story events.

In addition to its storytelling capabilities, UNIVERSE was able to generate a cast of related story characters ([Lebowitz 1984]) and the possibility of generalizing plot fragments to create new plot fragments was being investigated ([Lebowitz 1985]). However, UNIVERSE was never able to create stories more detailed than the plot outline form shown in Figure 14.4, nor was it able to generate natural language beyond what is shown in Figure 14.4.

14.3.1 Comparison of MINSTREL and UNIVERSE

As successors to TALESPIN, MINSTREL and UNIVERSE share several common features. Both programs recognize the importance of author-level goals to storytelling. UNIVERSE, like MINSTREL, tells stories by achieving author-level goals, not by simulating character-level goals. Second, both programs recognize the importance of creativity to the storytelling process. Although UNIVERSE never implemented any methods for inventing new plot fragments, [Lebowitz 1985] indicates that explanation-based learning was being investigated as a potential source of creativity. Both of these directions can be seen as outgrowths of the lessons taught by TALESPIN. Where MINSTREL and UNIVERSE differ is in how these issues were addressed.

Architecturally, UNIVERSE can viewed as a grammar engine. UNIVERSE's plot fragments function as the rules of a story grammar, rewriting an initial author-level goal into a sequence of story events. Subgoals are accomplished in left-to-right order. The story itself consists of the leaves of the resulting grammar tree. There are two consequences of this architecture.

First, this approach constrains UNIVERSE to telling stories in temporal order, marching through the story from beginning to end. And, as pointed out above, once a leaf has been generated, it cannot be further manipulated by a second plot fragment. Second, this architecture also means

that UNIVERSE's plot fragments cannot perform "meta-level" actions which manipulate the story structure, such as rearranging the events in the story, or inserting events earlier in the story. UNIVERSE's control structure is directly reflected in the story structure.

In contrast, MINSTREL's control structure is independent of the story being created. MINSTREL's control is based upon priority-driven planner that is independent of the story structure (and, indeed, of the storytelling domain). MINSTREL creates the events of the story in an apparently haphazard way that does not correspond to the temporal order of events in the story, nor to the order in which those events are later related to the reader, but rather to the importance of the author-level goals which the story events fulfill. And because MINSTREL's control structure is not based upon the story structure, MINSTREL is able to execute "meta-level" actions which manipulate the story structure. Not only does this give MINSTREL more storytelling power than UNIVERSE, it also better reflects how human authors create stories.

A second difference between MINSTREL and UNIVERSE is the depth and complexity of the stories told. UNIVERSE tells stories in the soap opera domain, a genre that is dominated by character actions. Like TALESPIN, UNIVERSE's stories are interesting only if the character actions are intrinsically interesting. UNIVERSE has no thematic goals when telling stories, nor dramatic or artistic goals. Consequently its stories lack story structure, and only seem purposeful by happy accident. It is equally possible for UNIVERSE to tell pointless and bewildering stories. Asking UNIVERSE to "(CHURN NEIL LIZ)" could result in the following story:

```
>>> STEPHANO threatens LIZ: "forget it"
>>> LIZ tells NEIL she doesn't love him
>>> DAPHNE marries NEIL
>>> RENEE tries to kill STEPHANO
>>> LIZ and TONY got divorced

>>> JOHNNY threatens NEIL: "forget it"
>>> NEIL tells LIZ she doesn't love him
>>> TONY marries LIZ
>>> RENEE tries to kill JOHNNY
>>> NEIL and DAPHNE got divorced

>>> STEPHANO threatens LIZ: "forget it"
>>> LIZ tells NEIL she doesn't love him
>>> DAPHNE marries NEIL
>>> RENEE tries to kill STEPHANO
>>> LIZ and TONY got divorced

[ad infinitum]
```

These shortcomings are masked by (1) the soap opera domain and (2) the careful selection of interesting plot fragments, but these limitations make it doubtful whether UNIVERSE can be considered a viable general model of storytelling.

MINSTREL, on the other hand, models an author with multiple, overlapping concerns. Because MINSTREL tries to tell a story which serves a definite, overall purpose and fulfills a variety of other author-level concerns, it is much less likely to tell a pointless or boring story.

Finally, MINSTREL and UNIVERSE differ in their level of knowledge representation. In UNIVERSE, all knowledge about what a story is, all author-level goals and plans, all character-level goals and plans, and all knowledge about the story domain are captured in plot fragments. To learn new knowledge, UNIVERSE must be able to create new plot fragments, either by deducing new plot fragments from stories it reads, or by modifying the plot fragments it already knows. UNIVERSE cannot learn directly from experience; it must first generalize its experiences into abstract plot fragments.

MINSTREL, on the other hand, represents all knowledge about author-level goals, themes and character-level goals and actions as specific episodic memories. And in principal, MINSTREL could represent its author-level plans in a similar way, although for convenience sake they are represented directly as LISP code. The advantage of this is that MINSTREL can (1) reason directly upon the specific knowledge, and (2) learn directly from experience, including from its own storytelling experiences. Consequently MINSTREL is able to be creative in ways and learn from its experience in ways that UNIVERSE could not.

14.4 STARSHIP

In [Dehn 1989], Natalie Dehn examines the role of memory in storytelling. In particular, she looks at the role of reconstructive memory in achieving author-level goals, and the role of memory in opportunistic planning. [Dehn 1989] presents a model of storytelling as planning in a weakly-ordered, many-goaled domain, and explores some of the issues that arise in two computer programs called STARSHIP and AUTHOR.

[Dehn 1989] suggests that storytelling can be characterized as a task involving many weakly-prioritized goals. It is argued that traditional planning models, in which the planner rigidly pursues single goals serially, are insufficient to explain the kinds of behavior observed in human authors. Drawing upon the idea of dynamic memory ([Schank 1982]), [Dehn 1989] suggests a model of planning in the storytelling domain in which reminding plays a large role in finding opportunistic solutions to active author goals. [Dehn 1989] also looks at the role that "paper memory" plays in the human authoring process. [Dehn 1989] argues that paper memory plays an important role as a buffer and augmentation of human memory.

These ideas were partially implemented in two computer programs, AUTHOR and STARSHIP. STARSHIP was an interactive computer game which creates short, story-like descriptions as part of the play. Figure 14.5 shows a story segment produced by STARSHIP.

Like UNIVERSE and MINSTREL, STARSHIP is primarily an author-level planner. STARSHIP begins with an initial author-level goal to play a game of Starship and breaks this goal down into sub-goals, tries to achieve those sub-goals, which may create additional sub-goals to be solved,

```
The Gethestians and the Starship both want the Starship's shupergraph.  A
transportation room contains the shupergraph.  A Gethestian walked into the
transportation room.  Gethestians are strong and gullible.  The Gethestian
wants to obtain the Starship's shupergraph for the Gethestians.  At this very
moment the Gethestian is attempting to seize the Starship's shupergraph.
```

Figure 14.5 STARSHIP Example

and so on, until all goals have been solved. Figure 14.6 shows an example of STARSHIP's goal trace which illustrates this process.

```
 -- FORM AUTHOR_INTENT39
[...I should make up a goal conflict between the Gethestians
and the Starship to serve as the GOAL_INTERACTION WITH
#{Object_aspect VO:STARSHIP-SOCIETY0} of the Gethestians...]

 -- PURSUE AUTHOR_INTENT39
[...I'm going to do it...]

 -- FORM AUTHOR_INTENT41
[...so I should make up a high tech object in order to
create something to serve as an object of contention in
goal conflict GOAL_CONFLICT_TYPE:IT_IS_MINE...]

 -- PURSUE AUTHOR_INTENT41
[...I'm going to do it...]
```

Figure 14.6 STARSHIP Goal Trace

The primary shortcoming of both AUTHOR and STARSHIP is that they were never completely implemented. Although [Dehn 1989] presents some interesting ideas, such as paper memory and the opportunistic solution of multiple goals, STARSHIP and AUTHOR did not demonstrate these abilities. As Figure 14.6 makes clear, STARSHIP achieved its author-level goals in a rigid, serial fashion almost identical to TALESPIN and UNIVERSE. Similarly, although [Dehn 1989] presents some interesting speculations on the human authoring process, these speculations were never investigated in depth, and remain only interesting suggestions.

14.4.1 Comparison of MINSTREL and STARSHIP

Most of the issues discussed in [Dehn 1989] have been addressed in MINSTREL. Like STARSHIP, MINSTREL is concerned with issues of episodic and dynamic memory. Use of memory is an important part of the MINSTREL model of creativity and storytelling. Similarly, the creation and achieving of author-level goals is an important issue in both MINSTREL and STARSHIP. Conversely, there are many areas addressed in MINSTREL that were not addressed in [Dehn 1989]: creativity in problem-solving, imaginative memory, the role of themes and other high-

level structures in storytelling, the role and classification of dramatic writing goals, and consistency in storytelling, to mention a few. In this section, we'll look at how MINSTREL addresses two of the important issues suggested by [Dehn 1989]: (1) the role of memory in author-level planning, and (2) the use of paper memory to augment human memory.

[Dehn 1989] suggests that dynamic memory must play an important role in storytelling, both in the creation of ideas to achieve the storytelling task, and in the process of storytelling itself. In MINSTREL, imaginative memory is indeed fundamental both to the creation of story scenes and to author-level storytelling process. Indeed, in MINSTREL, the same processes are used to solve problems in the storytelling domain and to solve problems at the author-level, a degree of integration of problem-solving and memory beyond that suggested by [Dehn 1989].

[Dehn 1989] also suggests that the ability to opportunistically solve goals in a domain with large numbers of weakly-ordered goals is important to the storytelling problem. [Dehn 1989] argues that at any time during the storytelling process the author has a large number of provisional goals, which can and should be dynamically solved or changed by fortuitous interactions with memory. In particular, [Dehn 1989] claims that an author needs to be able to make an active goal into a dormant one when that was necessary for successful storytelling.

MINSTREL's model of the storytelling process is somewhat different from the model presented in [Dehn 1989]. In the MINSTREL model, storytelling does involve large number of author goals, but in MINSTREL, there is a much stronger ordering of goals than suggested in [Dehn 1989]. And clearly some strong ordering is necessary. To give a simple example, the author must necessarily create a story scene before it can be presented to the reader. This difference aside, MINSTREL does implement mechanisms for the opportunistic achievement of author-level goals. In the MINSTREL model, an author-level goal that cannot be achieved when it is first encountered is placed back into the author-level goal queue for later reconsideration. This permits MINSTREL to opportunistically solve the goal at some later time, when changes in the story or in MINSTREL's knowledge have made the goal soluble.

The second major issue in [Dehn 1989] is the use of "paper memory" to augment the storytelling process. [Dehn 1989] argues that the use of inviolable, unlimited, permanent memory can be used to augment episodic memory, which is reconstructive, and working memory, which is strictly limited in size.

In MINSTREL, the story being told is kept in what can be described as "paper memory". Although paper memory was not a research issue in MINSTREL, the working buffer MINSTREL uses to hold the story being created has many of the features of a paper memory. It is inviolable, has unlimited size and is not reconstructive, the main features of paper memory as described in [Dehn 1989]. The similarity between MINSTREL's working buffer and paper memory was not intentional; the working buffer was simply given traits which proved useful when implementing MINSTREL. Nonetheless, the similarity suggests that paper memory can play an important and useful role in the storytelling process.

CHAPTER 15
Evaluation of MINSTREL's Computer Model

15.1 Introduction

In this chapter we evaluate MINSTREL, the computer model of the theory of creativity and storytelling presented in this dissertation. We analyze MINSTREL's performance and present a number of experiments that have been made to determine the robustness and coverage of MINSTREL. But before we can present these results, we must first discuss the role of the computer model in this research.

In general, computer models play a number of roles in cognitive science research. First, a computer model provides an existence proof that a theory is rigorous enough to be implemented as a computer program that can perform useful and meaningful tasks. The computer acts as an idiot savant critic of the theory being implemented, constantly asking "How is this done?" When a computer model has been built for a cognitive theory, there is some assurance that the researcher has addressed details that might otherwise remain undiscovered or glossed over.

Second, the computer model gives the underlying cognitive theory *plausibility*. Theories in cognitive science explain the processes and knowledge needed to perform some particular reasoning task. The fact that the computer model can use the underlying theory to perform meaningful example tasks gives plausibility to the claim that the theory explains how that type of reasoning works. If the problem domain is well-defined stronger claims can be made about the computer model. But in general the reasoning tasks being studied in cognitive science are ill-defined and poorly understood, and consequently most computer models represent proofs of plausibility, not of correctness.

Third, the computer model is a testbed for the development and testing of a theory. The researcher can use the computer model to test the limits of his theory, to see where the theory succeeds and where it fails. Further, the researcher can use the computer model to extend and develop his theories, to test new hypotheses and possiblities: What if memory functioned differently? What happens when creativity is taken away? How do author-level goals interact with the creative process? The computer program is not only an embodiment of a theory, it is also a tool for the development of theory.

Finally, the computer model is useful for understanding the overall behavior of large systems. It is difficult to fully comprehend a cognitive model of a complicated task like storytelling. There are too many sources of knowledge and too many different processes using that knowledge. The computer program provides a tool for discovering and studying the behaviors that emerge when these many parts interact. And regardless of whether the emergent behaviors confirm or contradict the original theory, these behaviors contribute to the researcher's understanding of the prob-

lem domain and help refine and modify his theories.

The role of a computer model in cognitive research is thus four-fold:

(1) to act as a painstaking critic of a theory,
(2) to act as a plausibility proof of a theory's validity,
(3) to act as a testbed for the development and testing of a theory, and
(4) to act as a tool for discovering complicated, emergent behaviors.

How can we evaluate MINSTREL's success at filling these roles?

MINSTREL's roles as (1) a critic and as (4) a tool for studying the behavior of a complicated system must be judged by the overall quality of this research. If the theories presented in this dissertation are detailed and comprehensive, that is partly due to MINSTREL's role as a critic of the theory. And if this dissertation presents interesting and unexpected ideas (such as imaginative memory) that is partly due to MINSTREL's value as a tool for studying the complicated behavior of a creative problem-solver. In this chapter we address the roles of MINSTREL which can be evaluated more explicitly: (2) as a plausibility proof and (3) as a testbed for our theories about creativity and storytelling.

To evaluate MINSTREL as a plausibility proof, we examine MINSTREL's performance as a creator. We examine the *quality* and *quantity* of tasks that MINSTREL can perform and judge how well these support the theoretical claims of this research.

To evaluate MINSTREL as a testbed of our theories about creativity and storytelling, we present a number of experiments and studies we have performed to test the limits of MINSTREL's performance. These experiments have led to a number of interesting insights about creativity, which support MINSTREL's value as a theoretical testbed.

15.2 MINSTREL as a Plausibility Proof

The purpose of evaluating MINSTREL's performance is to show that it supports the plausibility of the underlying theory. Unfortunately, there are no objective standards for this judgement. There is no standard "storytelling problem" against which we can test MINSTREL's performance. And, as pointed out earlier, MINSTREL is not a human-level performance model of storytelling, so such a test is anyway irrelevant. Instead, we hope to show that MINSTREL's performance is *convincing*: that MINSTREL's performance on creative tasks is good enough to persuade the majority of the readers of this dissertation that it is a plausible model of human creativity. To this end we evaluate MINSTREL's performance based on two criteria: quantity and quality.

First, to show the generality of the underlying theory of creativity, MINSTREL should solve a number of different tasks. A program that solved only one example (whether simple or complex) would be unconvincing because there would be little reason to believe that the program (and con-

sequently the underlying theory) was general enough to handle a wide range of problems. Instead, a computer model should handle a variety of problems, varying in content and difficulty. In the next section we present the range of problems that MINSTREL can handle and argue that they are sufficient to establish MINSTREL's generality.

Second, MINSTREL should solve difficult tasks at a high enough level of performance so that we can reasonably expect the underlying model to account for human creativity. A one-note program that solved only a very restricted, simple type of problem, no matter in what quantity, would be unconvincing as a model of human cognition because human cognitive abilities are very rarely so restricted. Instead a computer model should handle difficult problems with an effectiveness that demonstrates: (1) the flexibility of the model, (2) the quality of the solutions, and (3) the interaction of the model with other cognitive processes. Following the next section we show how MINSTREL's performance achieves these goals.

15.2.1 Quantity

MINSTREL is primarily a storytelling program. It can tell complete stories based on four different themes. In addition, the creativity component of MINSTREL has been applied separately to the task of inventing methods of suicide, and to a simple mechanical invention task. Figure 15.1 lists the tasks MINSTREL performs and the output generated.

Task	Output
PAT-GOOD-DEEDS-REWARDED	The Hermit and the Knight (two stories)
PAT-ROMEO	The Lady's Revenge
PAT-HASTY	Richard and Lancelot
PAT-PRIDE	The Proud Knight (five stories)[1]
Suicide Invention	Three methods for a knight to commit suicide.
Mechanical Invention	Three designs for staplers to use on a thick set of papers.[2]

Figure 15.1 MINSTREL's Tasks

As this shows, MINSTREL can solve a variety of very difficult problems in creativity (telling stories) as well as a number of examples in simpler problem domains. This alone is an impressive body of examples, and may convince the reader of MINSTREL's generality. But this summary of MINSTREL's performance is misleading. MINSTREL's basic problem unit is not the story, but rather the author-level goals that are satisfied during the invention of the story. The best measure of MINSTREL's performance is not how many complete stories it can tell, but how many author-level goals it solves.

1. See 15.5, 15.6, and 15.7 for details.

2. See Section 15.12 for details.

In fact, to tell the stories listed above, MINSTREL solves 1025 author-level goals of 22 types using 44 plans. These numbers reveal with more accuracy the quantity of problems that MINSTREL is able to solve. Figure 15.2 lists the goal types, the plans used to solve each goal, and how many times each plan was used.

There are two interesting things to note about these numbers. The first is the wide variety of goals and plans MINSTREL uses to achieve its goals. This reiterates both that MINSTREL is a general model capable of solving a variety of different problems, and that MINSTREL has a level of performance that supports its claims. The second thing to note is that some of MINSTREL's author-level plans are very successful: they succeed every time they are applied. Others succeed only occasionally. This reveals the wide range in specificity of MINSTREL's author-level plans. Some are so specific that they apply only in very narrow situations in which they invariably succeed. Other plans are more general, and are tried in many situations, succeeding in only a few.

There are several reasons why MINSTREL does not tell more complete stories.

First, the focus of this research has been on creativity: the creation of new solutions with *significant* differences from past solutions. The nine stories that MINSTREL tells are all very different from one another. It would be a simple matter to have MINSTREL tell additional stories not significantly different from the stories it already tells. We could, for example, add knowledge of three new types of berries to MINSTREL's episodic memory, and have MINSTREL generate three new variants of every story it tells which involves berries. And if we also added three new types of knights, MINSTREL could generate nine new variants of each story that involved berries and knights. Adding knowledge results in a combinatorial number of new stories. But such stories are neither interesting nor enlightening.

Quantity alone is a poor measure of performance.

A more important explanation concerns MINSTREL's *in-depth* understanding of the stories it creates. MINSTREL tries to create stories which achieve a large number of specific constraints. The stories MINSTREL tells all illustrate a theme and are consistent in many different ways. Each constraint that MINSTREL must satisfy during storytelling acts as a "bottleneck" for the number of stories it can tell. If a theme involves deception, then MINSTREL can necessarily tell only as many stories about that theme as it can invent deceptions. If a theme involves a deception that later causes a goal failure for the deceiver, then MINSTREL can only tell as many stories as it can invent deceptions that *also* backfire. Similar bottlenecks occur with each of the hundreds of author-level goals MINSTREL achieves during the course of creating a story; failing any bottleneck will cause MINSTREL's storytelling to fail.

In-depth understanding leads to a combinatorial explosion of constraints.

This is in marked contrast to story grammars or more simply constrained systems like UNIVERSE [Lebowitz 1984]. These systems can generate many stories because they have only a

Goal	Plan	Success	Total
ADD-CHARACTERIZATION	ALP:ADD-CHARACTERIZATION	2	2
ADD-DENOUEMENTS	ALP:MAKE-STORY-DENOUEMENTS	14	14
ADD-STORY-INTROS	ALP:CREATE-STORY-INTROS	14	14
ADD-SUSPENSE-TO-SCENE	ALP:ADD-SUSPENSE-VIA-FAILED-ESCAPE	10	10
ADD-TRAGEDY-TO-SCENE	ALP:ADD-TRAGEDY-VIA-LOVED-ONE	1	1
CHECK-AFFECTS	ALP:CHECK-AFFECT-THWARTED-GOAL	18	54
CHECK-CONSISTENCY	ALP:ALREADY-CONSISTENT-STATE	48	69
	ALP:CHECK-CONSISTENCY-ACT	13	32
	ALP:CHECK-CONSISTENCY-BELIEF	2	3
	ALP:CHECK-CONSISTENCY-GOAL	34	68
	ALP:CHECK-CONSISTENCY-STATE	11	14
	ALP:CONSISTENCY-GOAL-OBJECT	3	10
	ALP:DEFAULT-CONSISTENT	79	85
	ALP:MAKE-COLOCATION	1	9
	ALP:MAKE-CONSISTENT-STATE-COLOCATION	3	6
CHECK-NEW-SCENE	ALP:CHECK-NEW-SCENE	294	294
CHECK-PRECONDS	ALP:CHECK-ACT-PRECONDS	35	42
CHECK-SCENE-FOR-SUSPENSE	ALP:CHECK-SCENE-FOR-SUSPENSE	10	42
CHECK-SCENE-FOR-TRAGEDY	ALP:CHECK-SCENE-FOR-TRAGEDY	9	42
CHECK-STORY-FOR-CHARACTERIZATION	ALP:CHECK-STORY-FOR-CHARACTERIZATION	14	14
CHECK-STORY-FOR-FORESHADOWING	ALP:CHECK-STORY-FOR-FORESHADOWING	14	14
CHECK-STORY-FOR-SUSPENSE	ALP:CHECK-STORY-FOR-SUSPENSE	15	15
CHECK-STORY-FOR-TRAGEDY	ALP:CHECK-STORY-FOR-TRAGEDY	15	15
CONNECT	ALP:CONNECT	15	15
DETERMINE-GOAL-TYPE	ALP:DETERMINE-GOAL-TYPE-VIA-MEMORY	1	1
GENERATE-DENOUEMENTS	ALP:GENERATE-DENOUEMENTS-PLAN	10	10
INSTANTIATE	ALP:DONT-INSTANTIATE	229	323
	ALP:GENERAL-INSTANTIATE	49	100
	ALP:INSTANTIATE-BELIEF	14	44
	ALP:INSTANTIATE-DECEPTION	1	6
	ALP:INSTANTIATE-EVIDENCE-2	1	4
	ALP:INSTANTIATE-EVIDENCE	1	5
	ALP:INSTANTIATE-REVENGE	1	5
	ALP:INSTANTIATE-SUPERSEDING-BELIEF	1	2
	ALP:INSTANTIATE-THWARTING-STATE	2	7
	ALP:INSTANTIATE-UNTHWARTS	1	3
MAKE-DENOUEMENT	ALP:BURIAL-DENOUEMENT	2	5
	ALP:TRAGIC-DENOUEMENT	9	16
MAKE-GOAL-CONSISTENT	ALP:MAKE-CONSISTENT-MOTIVATING-STATE	3	9
	ALP:MAKE-CONSISTENT-OBJECT	6	15
	ALP:MAKE-CONSISTENT-PRECOND	3	4
	ALP:MAKE-CONSISTENT-SUPERGOAL	2	10
TELL-STORY	ALP:TELL-STORY-WO-THEME	1	1
	ALP:TELL-STORY	15	16

Figure 15.2 MINSTREL's Performance

limited understanding of the stories they generate. Propp's fairy tale grammar can be used to create many fairy tales, but none of them will illustrate a moral or be consistent except by random chance, because Propp's grammar understands stories only at the level of what types of events follow other events.

The bottleneck problem is made even more difficult by MINSTREL's goal to be creative. Being creative is a constraint that eliminates most new solutions that would satisfy other constraints. Given knowledge about three new types of berries, MINSTREL would *not* tell three new variants of each berry story as suggested above. Rather, MINSTREL's goal to tell creative stories would reject these variants, correctly recognizing (by their similarity to stories already told) that the new stories were not creative. Thus the drive to be creative is an especially restrictive bottleneck, because it eliminates many solutions that would trivially satisfy other bottlenecks.

The drive to be creative eliminates trivial solutions.

In general, there is a tradeoff between quantity of performance and quality of performance. The complexity and depth of MINSTREL's understanding limits the number of complete, correct stories it can tell. In general, a system with deep understanding will require more knowledge and operate under more constraints, limiting the breadth of its performance. Human authors also face this "bottleneck" problem. Even though human authors possess knowledge and inventive problem-solving skills that exceed MINSTREL's capabilities by many orders of magnitude, they still have difficulty producing large numbers of interesting, inventive stories. Human authors who produce numerous stories tend to produce "boiler plate" stories with little that is novel or creative.

In addition to the quantity of problems that MINSTREL solves, there is other evidence to support MINSTREL's generality.

First, MINSTREL was able to invent methods of suicide using the same author-level goals and plans used in telling stories. MINSTREL was able to solve a new problem with no modification. In fact, MINSTREL is capable of solving many new problems in the King Arthur and story-telling domains without changes of any sort (see, for example, Experiment #1 in this chapter, which describes how MINSTREL was able to solve all the problems involved in telling a story about a new theme with little additional knowledge). This indicates that MINSTREL is not restricted to the particular problems it currently handles, but rather has the generality to extend to other similar problems.

Second, with the addition of appropriate episodic memories and TRAMs, MINSTREL was able to invent mechanical devices (see Section 7 of this chapter for details). Although MINSTREL required new knowledge to perform the mechanical invention task (as would a human who knew nothing about mechanical devices), MINSTREL required no changes to the underlying model of creativity. This is yet another indication that MINSTREL's model of creativity has generality.

MINSTREL's large number of solved examples and its simple extension to new problems and problem domains should be sufficient to establish the generality of the TRAM model of creativi-

ty. Of course, there is always the possibility that a new example will reveal a difficulty with the TRAM model, or that new research in psychology will establish that human creativity functions quite differently from the TRAM model. But the purpose of MINSTREL is to establish the plausibility of the TRAM model, not its correctness. And by that criteria, the quantity of examples MINSTREL solves clearly establishes the generality of the TRAM model.

15.2.2 Quality

Evaluating the quality of MINSTREL's output is even more difficult than evaluating the quantity. We can at least measure the quantity of MINSTREL's output, even if we have no accepted scale against which to judge. In evaluating the quality of the output we have neither measure or scale.

Throughout history, the standard measure of quality in creativity has been the marketplace. Getting published is the goal of the struggling writer; selling a patent the goal of the backyard inventor. Simply put, creative ideas are valuable. And so the marketplace has always been a good (albeit imperfect) judge of creativity.

If MINSTREL's stories were publishable, it would therefore be a strong argument for their quality. Alas, they are not. That's not an unexpected result. After all, MINSTREL is an infant of only a few years competing against human experts with many thousands of years of human creative history on which to draw. In fact, if MINSTREL were able to produce publishable stories it would be more supiscious than auspicious. Asking MINSTREL to compete against human authors is too strict and demanding a criteria given MINSTREL's goals and the current state of research in creativity. How, then, to judge the quality of MINSTREL's stories?

The measure we've used is based on the idea of the Turing Test. Suggested by the famous British mathematician Alfred Turing, the Turing Test is intended as a measure of computer intelligence. A person acting as a tester (T) converses with a computer (C) and with another person (P) by some means that conceals their identities (such as a teletype). T converses with C and P separately. If, after some reasonable period of time, T cannot determine which is the computer, the computer is deemed intelligent. The intent of the Turing Test is to compare computer performance against human performance in an unbiased setting.

The variant of the Turing Test presented here is much less ambitious. Rather than asking a person to compare MINSTREL directly against a human in an interactive test, we asked subjects to judge MINSTREL against their knowledge of human performance. A survey which contained one of MINSTREL's stories and MINSTREL's invented suicides was given to a selection of subjects of varying backgrounds. The subjects (who were not aware that the story had been written by a computer) were asked to read the story and answer some simple questions about the author. The answers give some insight into how unbiased adults evaluate the quality of MINSTREL's output. The questionnaire used is shown in Figure 15.3 and Figure 14.4.

This survey was given to ten subjects solicited over the Internet. One subject did not respond. A summary of the other nine responses is shown in Figure 15.5. On average, the subjects judged

Personal Information
AGE:
SEX:
EDUCATIONAL LEVEL (High School, College, or Graduate School):

A subject was asked to write a short (a few paragraphs) story about King Arthur. The subject produced the following story:

In the Spring of 1089, a knight named Lancelot returned to Camelot from elsewhere. Lancelot was hot tempered. Once, when Lancelot lost a joust, he blamed his lance and wanted to destroy it. He struck his lance and broke it.

One day, a lady of the court named Andrea wanted to pick some berries. Andrea went to the woods. Andrea picked some berries. Lancelot's horse unexpectedly carried him into the woods. Lancelot was near Andrea and fell in love with her. Lancelot wanted Andrea to love him.

Some time later, Lancelot's horse again carried him into the woods. There he saw Andrea kissing a knight named Frederick. Because of this, Lancelot believed that Andrea loved Frederick. Because Lancelot loved Andrea, he wanted her to love him. Lancelot hated Frederick. Lancelot's hot temper made him mad. He wanted to kill Frederick. Lancelot moved to Frederick and fought him. Frederick was killed.

Andrea ran to Frederick. Andrea told Lancelot that Frederick was her brother. Lancelot wanted to take back that he wanted to kill Frederick, but he could not. Lancelot hated himself.

Lancelot became a hermit to hide his shame. Frederick was buried in the woods where he died, and Andrea became a nun.

Please answer the following questions about the author of this story:

1. How old do you think the author is?
2. How much education do you think the author has had?
3. Do you think the author is male or female?
4. Do you think the author understands what a story is?
5. Would you call this story "excellent", "good", "mediocre" or "poor"?
6. Do you feel you know or can guess anything else about the author? (Free form comments)

Please rate the story from 1 to 5, where 5 indicates good or best on the following criteria:

Use of the English language:
Interesting or clever plot:
Attention to details:
Coherency:

Figure 15.3 Initial Survey, Part 1

Part 2

A subject was asked to invent ways in which one of King Arthur's knights might commit suicide. Below are listed three answers:

1. The knight could hit himself with his sword until dead.
2. The knight could drink a potion that would kill him.
3. The knight could fight a dragon and lose on purpose.

Please rate these answers according to the listed criteria on a scale of 1-5, where 5 indicates good or best.

How effective do you rate each solution? I.e., will it work?

How clever or creative is each solution?

That's it! Thank you very much for participating. When you return this questionnaire, you will receive a short explanation of this research.

Figure 15.4 Initial Survey, Part 2

MINSTREL to be about 16 years old with a high school education. They felt that MINSTREL had a good understanding of what a story is, and rated MINSTREL's story between mediocre and good. Of MINSTREL's suicide methods, they found taking a poison potion very effective and fighting a dragon with intent to lose very creative.

Subject responses to the free-form response question also indicated that they viewed MINSTREL as a human. A sampling of the free-form responses from the first survey indicate that even with MINSTREL's rough story expression, subjects still attributed human characteristics to the story's author:

> *Possibly poor. Possibly from a minority. Although there is an age implied by my guess of the amount of education I would not be suprised to learn that the author was much older and simply lacked the education... Probably reads romance novels.*
>
> *The author likes violence and the idea of sex.*
>
> *Since the "story" is rough and lacking in detail, the author either sees things in 'block-diagram' form, and is uninterested in detail, or finds detail difficult to supply.*
>
> *An intelligent female with a knowledge of cause and effect and a sense of poetic justice.*

Question	Ave. Response	Max	Min
Respondents			
Number	9		
Average age	35.7	57	20
Average education	2.6[1]	3	2
Responses			
Author age	15.8	25	12
Author education[1]	0.9	2	0
Author sex[2]	0.4		
Does the author understand what a story is?[3]	0.9		
On a scale of 1-5...			
Overall rating of story	1.5	3	1
Clever plot?	2.5	4	1
Attention to details?	2.8	5	1
Coherency?	3.6	5	2
Use of language?	2.1	1	3
Effectiveness of suicide solns (1-5)			
#1	2.6	5	1
#2	4.7	5	4
#3	3.9	5	1
Creativeness of suicide solns (1-5)			
#1	1.6	3	1
#2	3.6	5	2
#3	4.1	5	3

1. 0 = Grade School, 1 = High School, 2 = College, 3 = Grad. School
2. 0 = Female, 1 = Male
3. 0 = No, 1 = Yes

Figure 15.5 Summary of Responses to Initial Survey

I would guess that the author may have a problem with his temper.

The lack of detail in the story also makes me think the author is writing about something personal, and not describing something that might have happened many years ago. Perhaps the author has a hot temper and is in trouble for fighting someone by mistake. The problem might be recent, because Lancelot hasn't rejoined the round table, or the author might act withdrawn because he is embarassed by his temper tantrums.

To determine how much the subjects responses were affected by MINSTREL's expression of the Lancelot story (i.e., the precise wording and English of the story), a second survey was conducted using a revised version of the Lancelot story. The revised story was produced by hand, and used language more fluently without changing the basic structure or events of the story. The revised story is shown in Figure 15.6.

In the Spring of 1089, a knight named Lancelot returned to Camelot from a quest. Lancelot was known as a hot tempered knight. Once, when Lancelot lost a joust, he blamed his lance and broke it in a fit of temper.

On the day Lancelot returned, a lady of the court named Andrea was in the woods picking berries. As Lancelot rode near the woods his horse suddenly darted into the woods, and he saw Andrea. He fell in love with her immediately, and wanted Andrea to love him.

A few days later, Lancelot was again riding near the woods, thinking about Andrea. Again, his horse carried him unexpectedly into the woods. There he saw Andrea kissing a knight named Frederick. Lancelot realized that Andrea loved Frederick. His hot temper boiled over, and he charged into the clearing and struck down Frederick. Frederick was killed by the first blow.

Andrea ran to Frederick and held his body. Crying, Andrea told Lancelot that Frederick was her brother. Lancelot was filled with remorse. He hated himself. He wanted to take back his hasty action but he could not.

Lancelot became a hermit to hide his shame. Frederick was buried in the woods where he died, and Andrea became a nun.

Figure 15.6 Revised Lancelot Story

The results of the second survey are summarized in Figure 15.6. As was expected, subjects in the second survey thought the use of language in the story better. Estimates of the author's age were also higher (possibly due to the association in Western culture of language expertise with maturity) and the author was thought to be more male. But in all other categories, responses were close to those in the first survey, indicating that the subjects' judgements of MINSTREL's storytelling abilities and creativity were not overly affected by the particular expression of the sample story.

As a final test of the quality of MINSTREL's stories, the same survey was given to a third set of subjects using a story written by the 12 year old daughter of a colleague. This story is shown in Figure 15.8.

Question	Ave. Response	Max	Min
Respondents			
Number	13		
Average age	32.9	22	53
Average education[1]	2	3	2.3
Responses			
Author age	17.5	10	37
Author education[1]	0.9	0	2
Author sex[2]	0.7		
Does the author understand what a story is?[3]	0.8		
On a scale of 1-5...			
Overall rating of story	1.4	3	1
Clever plot?	2.6	5	1
Attention to details?	2.8	5	2
Coherency?	3.2	5	2
Use of language?	3.3	4	1
Effectiveness of suicide solns (1-5)			
#1	1.9	4	1
#2	4.5	5	3
#3	3.4	5	1
Creativeness of suicide solns (1-5)			
#1	1.6	3	1
#2	2.5	5	1
#3	3.9	5	1

1. 0 = Grade School, 1 = High School, 2 = College, 3 = Grad. School
2. 0 = Female, 1 = Male
3. 0 = No, 1 = Yes

Figure 15.7 Summary of Second Survey

Figure 15.9 shows the results of this survey and compares them with MINSTREL's results. As Figure 15.9 shows, respondents rated the human author as older (17.9 years vs. 16.8 years), and the story as more clever (3.3 vs. 2.5 on a scale of 1-5), but also rated the story more poorly overall (1.1 vs. 1.4 on a scale of 1-5) and found it less coherent than the MINSTREL story (2.6 vs. 3.4 on a scale of 1-5). Respondents to the Bunny story were also asked to rate MINSTREL's suicide solutions. Although the results generally agreed with the evaluations of the subjects who saw MINSTREL's story, the readers of the Bunny story found "fighting a dragon with intent to lose" less effective, "hitting himself with a sword" more creative, and "taking a potion" less creative.

More complete and careful study is needed to fully understand how people judge MINSTREL's stories, but these preliminary surveys indicate that MINSTREL's storytelling is equivalent to what people expect from a younger high-school age student. Furthermore, MINSTREL's story is not obviously written by a computer; many of the subjects made comments which reveal that they viewed the author as human.

Once upon a time, in a land over the sea and beyond the mountains, lay a kingdom and the royal family.

Well, it just so happens that the princess was a very unhappy child. All day long she would keep to herself and the animals. Princess Bunny was supplied abundantly with toys, dresses, three nannies, basically anything she asked for.

But every day, Bunny would play with her new toys for an hour or so, then go where the peasants were working with the animals.

Bunny's particular favorite animal was Henry the pig. She liked to visit Henry for hours each day. Then when it was feeding time, Bunny would go back into the castle and become dreary and lonely once more.

Bunny's daily visits to see Henry became rarer and rarer, until she stopped seeing him altogether, and the pig became very lonely. So did Bunny. The nannies tried their hardest to cheer Bunny up. Jesters came, dancers came, but nothing worked.

One morning Bunny woke up to the blast of the king's trumpets. "Someone has broken into the king's safe and stolen everything!" Upon hearing this, a smile cracked over Bunny's face.

Later that afternoon Bunny went to visit Henry. She passed a few pages and saw a peasant boy go into the barn next to Henry's pen.

Henry greeted Bunny with a pleasant oink and rubbed his head on her skirt, telling her in his way to scratch him.

"Henry," whispered Bunny. "Do you think Daddy and Mommy will love me and pay attention to me if they stop being King and Queen and loose all of their money?"

Henry oinked saying yes to this poor desperate child.

"Good, because I robbed my own father! I put the money under my bed."

Now the peasant boy who had gone in the barn heard all this and ran to the castle to confess what he'd just heard.

The king was very surprised at hearing this. He made promise to spend more time with Bunny, and the King, Queen, Bunny, the kingdom, and Henry lived happily ever after.

Figure 15.8 Third Survey Story

15.2.3 Summary of Evaluation

The basic claim of this research is that MINSTREL is a plausible explanation of aspects of human creativity. As the above sections show, MINSTREL is capable of solving a wide variety of creative tasks at an adult level. MINSTREL tells stories about four different themes, invents suicides and mechanical devices, and solves over five hundred author-level goals using a variety of plans and mechanical heuristics. The stories MINSTREL tells are comparable to those of a younger high-school student, and at least one of MINSTREL's suicide solutions is viewed as very creative by older, college-educated subjects. Both the quality and quantity of MINSTREL's outputs indicate that MINSTREL can perform at the level of a creative problem-solver, albeit on a limited set of problems.

Question	Human Author	MINSTREL
Respondents		
Number	9	22
Average age	32.8	34.1
Average education	2.4	2.4
Responses		
Author age	**17.9**	**16.8**
Author education	1.0	0.9
Author sex[2]	0.2	0.6
Does the author understand what a story is?[3]	0.8	0.8
On a scale of 1-5...		
Overall rating of story	**1.1**	**1.4**
Clever plot?	**3.3**	**2.5**
Attention to details?	**2.4**	**2.8**
Coherency?	**2.6**	**3.4**
Use of language?	2.9	2.8
Effectiveness of suicide solns (1-5)		
#1	2.1	2.1
#2	4.5	4.6
#3	2.9	3.6
Creativeness of suicide solns (1-5)		
#1	2.7	1.6
#2	2.4	3.0
#3	4.0	4.0

1. 0 = Grade School, 1 = High School, 2 = College, 3 = Grad. School
2. 0 = Female, 1 = Male
3. 0 = No, 1 = Yes

Figure 15.9 Summary of Third Survey

15.3 Experiments With MINSTREL

The second purpose of this chapter is to evaluate MINSTREL as an experimental testbed. To this end, we have performed a number of experiments and studies to test the limits of MINSTREL's performance. These experiments have led to a number of interesting insights about the processes of creativity.

15.4 Study #1: Abandoned TRAMs

During the development of MINSTREL, three particular TRAMs were implemented that were later abandoned. Examining these TRAMs and determining why they were discarded provides insight into MINSTREL's creativity process and will help evaluate the generality of the Transform-Recall-Adapt model of creativity.

15.4.1 TRAM:Ignore-Neighbors

The first of the three TRAMs that were implemented and then later abandoned is TRAM:Ignore-Neighbors. TRAM:Ignore-Neighbors suggests finding a problem solution by ignoring all the knowledge structures immediately connected to the problem description, finding a past problem that matches the modified problem description, and then using the solution to that past problem (without adaptation) to solve the current problem. Figure 15.10 shows the psuedo-code for this TRAM.

TRAM:Ignore-Neighbors

Comment:	Find a reminding by ignoring connections.
Test:	Does problem description have any connections to other knowledge structures?
Transform:	Remove all connections to other knowledge structures.
Adapt:	No adaptation.

Figure 15.10 TRAM:Ignore-Neighbors

In MINSTREL, problem descriptions are represented as goal, action and state frames connected to one another by various kinds of links. For example, the problem description "What action can a knight perform that will make him famous and make a princess love him?" is represented as an unspecified action frame connected to other frames by causal links such as "&plan-of" and "&intends". MINSTREL's representation for this problem description is shown in Figure 15.11.

TRAM:Ignore-Neighbors transforms a problem by removing all connections to other knowledge structures. Given the problem description shown in Figure 15.11, TRAM:Ignore-Neighbors removes the "&plan-of" and "&intends" links from &Act.77. The transformed problem description is "What action can a knight perform?" If MINSTREL can recall any action that a knight takes, that will be acceptable as a solution to the modified problem, and since TRAM:Ignore-Neighbors does no adaptation of any recalled solutions, will be inserted directly into the original problem. This can result in nonsensical solutions, such as a knight who becomes famous by sharpening his sword.

Interestingly enough, TRAM:Ignore-Neighbors is similar to the successful TRAM:Limited-Recall. TRAM:Limited-Recall transforms a problem description by removing all *distant* neighbors. That is, TRAM:Limited-Recall removes all of the knowledge structures connected to a problem description *except* the immediate connections. In the case of the problem shown in Figure 15.11, TRAM:Limited-Recall removes &Goal.43. The new problem description looks for actions that can make a knight famous, regardless of whether or not they were part of a plan to capture a princess's love. Consequently, the solutions discovered by TRAM:Limited-Recall will be much more usable than those discovered by TRAM:Ignore-Neighbors. TRAM:Limited-

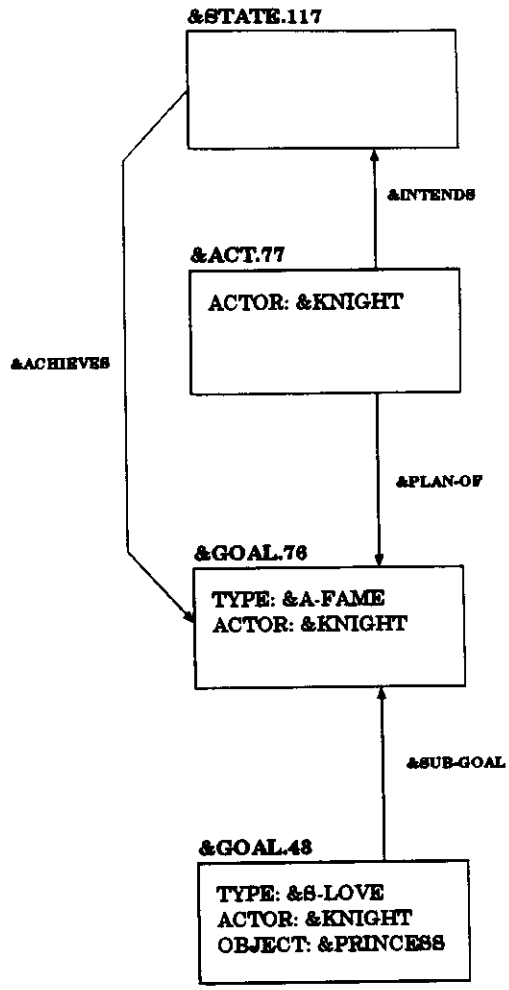


Figure 15.11 Graphical Representation of Problem Description

Recall is shown in Figure 15.12.

Why is TRAM:Limited-Recall useful and TRAM:Ignore-Neighbors not? The answer is that TRAM:Ignore-Neighbors uses a powerful problem transformation without a balancing solution adaptation. Removing all the connections to a problem description creates a new problem description that will match many more episodes in memory. But TRAM:Ignore-Neighbors cannot adapt those episodes to the original problem.

Of course, TRAM:Limited-Recall also has no adaptation process. But TRAM:Limited-Recall has a much weaker problem-transformation, so that less adaptation of discovered solutions is necessary.

TRAM:Limited-Recall

Comment:	Find a reminding by removing distant connections.
Test:	Does problem description have any second-level connections to other knowledge structures?
Transform:	Remove all connections to second-level knowledge structures.
Adapt:	No adaptation.

Figure 15.12 TRAM:Limited-Recall

It is a tenet of this research that creative solutions to problems are found by small, precise heuristics that apply specific problem transformations and corresponding solution adaptations. TRAM:Ignore-Neighbors illustrates why powerful, broad creativity heuristics must fail: because the adaptation task created by a sweeping problem transformation is likely to be more difficult than the original problem. To be sure, a heuristic like TRAM:Ignore-Neighbors will sometimes discover a novel and useful solution. But most of the time it will simply create an adaptation task more difficult than the original problem (such as explaining why sharpening a sword would make a knight famous).

Does this mean that MINSTREL cannot discover the good solutions that TRAM:Ignore-Neighbors would occasionally find? Not at all. MINSTREL must simply find those solutions using smaller and more specific steps. If a problem description has four immediate connections to other knowledge structures, MINSTREL must remove these connections singly, by using heuristics that know about these specific types of connections. (And if MINSTREL lacks a TRAM suited to removing a particular connection, it also lacks the knowledge necessary to adapt a solution that lacks that connection; and so no solution is lost.) By breaking the transformation problem into four smaller problems, MINSTREL has made the adaptation problem tractable.

15.4.2 TRAM:Switch-Focus

TRAM:Switch-Focus is the second of the abandoned TRAMs. This heuristic suggests that if MINSTREL is trying to find a state resulting from some action, it look instead for an action that intends that state. In this case, the problem description consists of two frames connected by an "&intended/&intended-by" link. This TRAM suggests considering the "other" frame to be the problem description. This transformation is shown graphically in Figure 15.13.

The psuedo-code for TRAM:Switch-Focus is given in Figure 15.14.

TRAM:Switch-Focus was intended to allow MINSTREL to apply act-specific TRAMs to problems involving states. (Since MINSTREL has 11 TRAMs that apply to acts and only 6 that apply to states, this would seem to be a worthwhile goal.) However, TRAM:Switch-Focus was never successfully applied.

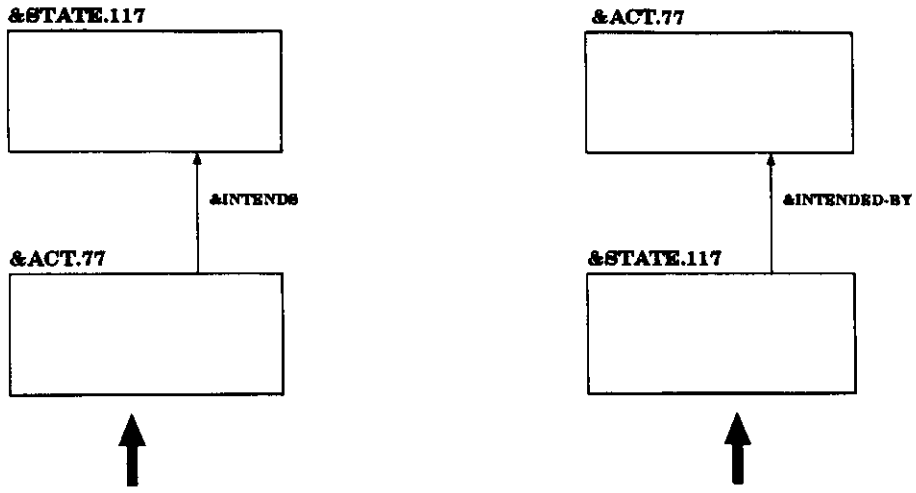


Figure 15.13 Graphical Explanation of TRAM:Switch-Focus

TRAM:Switch-Focus

Comment:	Look at this problem from the viewpoint of the act, instead of the state.
Test:	Is the problem description a state intended by an act?
Transform:	Use the intended act as the problem description.
Adapt:	The recalled episode is an act that intends a state. Return the state instead of the act.

Figure 15.14 TRAM:Switch-Focus

The reason for this is twofold. First, TRAM:Switch-Focus will never by itself lead to a discovery, because it does not change the problem description. Any episode which involves both a state and an act will be indexed in episodic memory by both, so simply shifting the focus of recall in a problem description from an act to a state will not lead to any additional solutions. So as an individual creativity heuristic, TRAM:Switch-Focus is ineffective.

Secondly, TRAM:Switch-Focus often operated at odds with author-level goals. The author-level of MINSTREL specified problems by their most important features. If a problem specification focused on a state, that was usually because the state was the most critical component of the problem description. When TRAM:Switch-Focus moved the focus to an associated action, it reduced MINSTREL's chances of finding a solution.

There are two lessons to be learned from TRAM:Switch-Focus. First, the idea of "looking at something in a new way" is often touted as a creativity technique. What TRAM:Switch-Focus emphasizes is that the "different way" must be substantively different. A simple syntactic change is unlikely to lead to a creative realization.

Second, TRAM:Switch-Focus reveals an interaction between higher level cognitive processes (called the author-level in MINSTREL) and low-level creativity. Creativity should make use of high-level knowledge (such as domain-specific knowledge on where to focus creativity) to guide the search for a new solution to a problem.

15.4.3 TRAM:Remove-Slot-Constraint

TRAM:Remove-Slot-Constraint suggests transforming a problem description by removing a single slot of the problem description frame at random, recalling a solution based on the transformed description, and then adding the removed slot back to any discovered solution. The psuedo-code for this TRAM is given in Figure 15.15.

TRAM:Remove-Slot-Constraint

Comment:	Randomly remove one slot of the description frame.
Test:	Does the frame have a slot other than :actor or :type?
Transform:	Randomly select and remove a slot other than :actor or :type.
Adapt:	Add the removed slot value to any solutions.

Figure 15.15 TRAM:Remove-Slot-Constraint

The purpose of TRAM:Remove-Slot-Constraints was to improve MINSTREL's chances of finding a solution when given a problem description that was over-constrained. When creating a large, interconnected structure like a story, filling in one part of the story will cause other parts of the story to be filled in. For instance, if the protaganist is a knight, he is a knight in every part of the story, not just in the scene where that decision is made. Consequently, when MINSTREL attempts to fill in a scene, there may be a number of possibly irrelevant constraints on that scene arising from other parts of the story. TRAM:Remove-Slot-Constraints was an attempt to write a TRAM that would remove some of these unnecessary constraints.

The difficulty with this TRAM lies in the random selection of slot constraints. There is nothing in TRAM:Remove-Slot-Constraints that attempts to decide whether a constraint is relevant or not. Consequently TRAM:Remove-Slot-Constraints removed important constraints as frequently

than it removed irrelevant constraints. (In fact, more frequently, since there are generally more relevant constraints than irrelevant ones.) An early attempt to fix this problem is evident in the Test and Transform portions of the TRAM as shown in Figure 15.15. These portions of TRAM:Remove-Slot-Constraint prevent the removal of the :actor and :type slots, two slots that were discovered by experience to be almost always relevant.[1] But this was patchwork fix that did not address the real problem.

The lesson to be learned from TRAM:Remove-Slot-Constraints is that care must be taken in removing or weakening problem constraints. Much shrift is given in popular creativity literature to removing problem constraints as a method for unlocking creativity. Yet the removal of important problem constraints will only lead to unworkable solutions or a very difficult adaptation problem. A better piece of advice is to examine a problem carefully to discover what constraints are truly necessary and what constraints only appear to be necessary, and to remove those constraints that only appear necessary.

15.5 Experiment #1: Adding a New Theme to MINSTREL

The first experiment performed with MINSTREL was to have MINSTREL tell a story about a new theme. Assuming MINSTREL had the episodic memory, representation, author-level plans and TRAMs needed to tell stories about three different themes, what additional knowledge would MINSTREL need to tell a story about a new theme? If MINSTREL required little or no additional knowledge, it would indicate that MINSTREL's author-level plans and creativity have some generality, at least for the task of storytelling within the King Arthur domain.

15.5.1 PAT:PRIDE

As the first step in this experiment, a new theme was represented and added to MINSTREL's episodic memory: PAT:PRIDE. Planning Advice Themes (PATs) are a schema-based representation of story themes which convey advice about planning. (Planning Advice Themes are based on Thematic Abstraction Units [Dyer 1982] and are described in more detail in Chapter N.) PAT:PRIDE is based on the saying: "Pride goes before a fall." In particular, PAT:PRIDE represents the goal/plan situation in which an actor is warned that if he does something, it will result in a goal failure, but because he is overly proud, attempts the action anyway. MINSTREL's representation of PAT:PRIDE is shown in Figure 15.16.

&PAT:PRIDE is a knowledge structure which captures a piece of planning advice: "Don't let your pride get in the way of accepting useful advice." The Decision portion of points to a schema-based representation of a planning decision. The Decision Point identifies the particular type of decision. In &PAT:PRIDE, the decision is plan selection, and the planning situation is an advisor telling a character to avoid a particular plan. The Consequence portion of the PAT points to the consequence of avoiding this advice: the plan has an unintended effect which causes a goal failure for the planner. Finally, the Connection describes the relation between the Decision situa-

1. But for an interesting comparison, see TRAM:Generalize-Actor, which relaxes but does not remove the :actor slot.

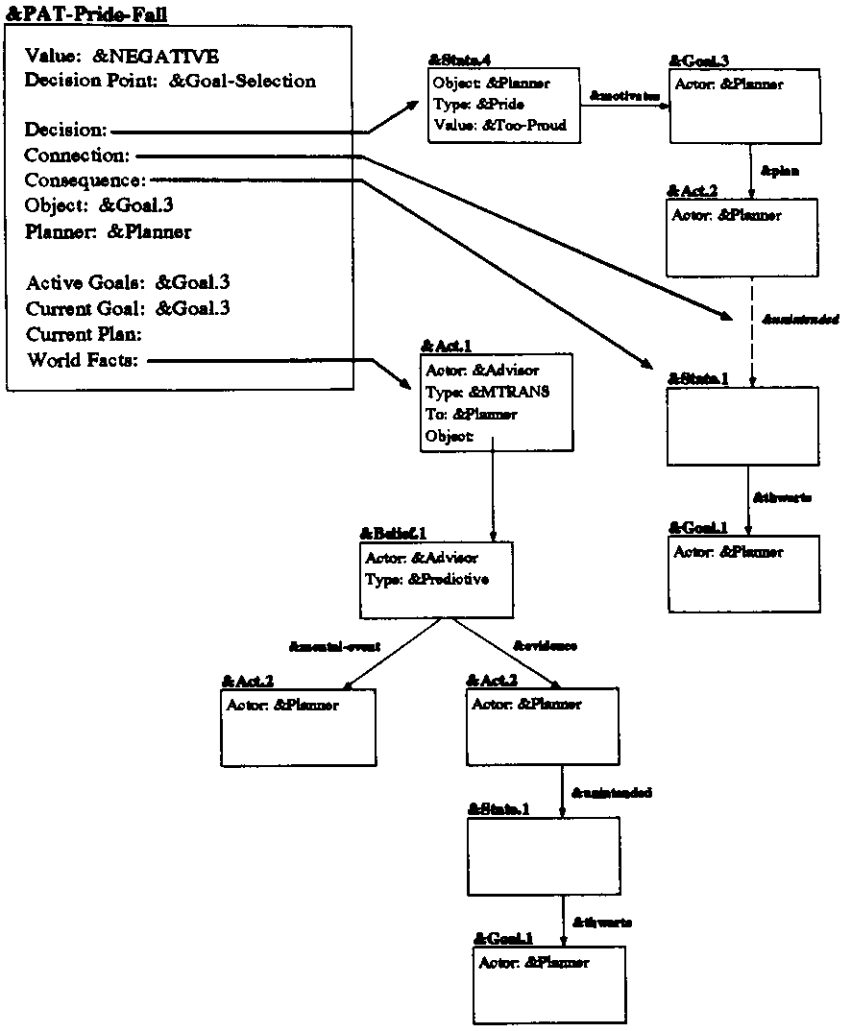


Figure 15.16 Representation of PAT:PRIDE

tion and the Consequence. In this case, the actor's pride motivates him to ignore the advice.

As with all themes in MINSTREL, this representation was hand-coded by the author and added to MINSTREL's episodic memory. No examples of PAT:PRIDE were added to memory, only the abstract representation. It would be up to MINSTREL to invent story scenes to fit this new theme, using its existing episodic memories and creativity.

15.5.2 MINSTREL's Performance

After the representation for PAT:PRIDE was added to MINSTREL, MINSTREL was given the goal of telling a story about this theme. MINSTREL immediately ran into an interesting and typical problem.

Initially, the main character of PAT:PRIDE (?Planner in Figure 15.16) is uninstantiated. MINSTREL begins storytelling by determining a role for this actor from amongst the kinds of characters it knows about. As it happens, the actor is instantiated as a hermit. MINSTREL's next step is to instantiate the prideful act (&Act.2 in Figure 15.16). To do this, MINSTREL uses episodic memory to recall a typical action by a hermit, and uses this to instantiate the prideful act. In this case, MINSTREL recalls a scene in which a hermit picks berries in the woods [1].

At this point, MINSTREL bogs down. MINSTREL cannot invent any bad consequences of picking berries. So MINSTREL is unable to fill in the "fall" portion of "Pride goes before a fall", and the story fizzles. Figure 15.17 shows an (author-supplied) English version of the failed story.

The Proud Hermit

Once upon a time there was a hermit. Someone warned him not to pick berries, but he went ahead and picked them anyway. Nothing happened.

Figure 15.17 Failed PAT:PRIDE Story

Being unable to continue a story because of a bad decision earlier in storytelling is a common problem for MINSTREL. MINSTREL has a very limited planning model, and no effort was made to give MINSTREL plans for retracting bad solutions, or for recovery from decisions that resulted in a large number of failed goals. Consequently when MINSTREL makes a decision that later causes difficulties, MINSTREL has no way to retract the earlier decision. MINSTREL's common errors and what they say about MINSTREL's model of creativity are discussed in more detail in Section 12 of this chapter.

There are several ways to address this problem without changing MINSTREL's planning model.

The first possibility is to enable MINSTREL's facility to choose randomly between equal alternatives. Choosing between random alternatives occurs when MINSTREL has several plans for a particular goal, several applicable TRAMs during creativity, or several applicable episodes when recalling from episodic memory. When this occurs, MINSTREL can either select the first alternative or select randomly between the alternatives. To permit the easy reproduction of results and to aid in the development of the program, MINSTREL normally selects the first alternative,

1. In fact, the instantiation of the actor occurs simultaneously with the instantiation of the prideful act. While this is indicative of the kinds of interactions that occur in the creation of a story, it is more easily understood if presented as two separate instantiations.

resulting in deterministic behavior.

In the case of the failed PAT:PRIDE story, MINSTREL has several alternatives for instantiating the main character's role. As it happens, the first alternative is to instantiate the character as a hermit. Other possibilities are to instantiate the character as a knight or as a princess. If random selection were enabled, MINSTREL would sometimes select one of these other alternatives, and that might result in a successful story.

An alternative solution is to give MINSTREL a "hint" that will point it a different direction. In this case, we wish to suggest an alternative role for the main character, as if MINSTREL had randomly selected one of the other alternatives when instantiating the main character. This is done by setting the role of the main character by hand to an appropriate value. Because MINSTREL knows the most about knights, we set the main character to be a knight.

MINSTREL was again given the goal to tell a story about the modified PAT:PRIDE. The result was an almost-perfect story about a knight who fought a dragon despite a warning from a friendly hermit. An author-supplied English version of the resulting story is given in Figure 15.18.

The Proud Knight

Once upon a time, a hermit named Bebe told a knight named Grunfeld that if Grunfeld fought a dragon then something bad would happen.

Grunfeld was very proud. Because he was very proud, he wanted to impress the king. Grunfeld moved to a dragon. Grunfeld fought a dragon. The dragon was destroyed, but Grunfeld was wounded. *Grunfeld was wounded because he fought a knight. Grunfeld being wounded impressed the king.*

Figure 15.18 The Proud Knight

"The Proud Knight" has two errors in it, as indicated by the italics in Figure 15.18. MINSTREL explains how Grunfeld got wounded twice, and (rather impishly) suggests that Grunfeld's injuries impress the king.

These errors occur during MINSTREL's creation of the scene in which Grunfeld is wounded. Because these errors give insight into MINSTREL's operation and the role of creativity heuristics, we provide a detailed analysis of the problem and its solution.

15.5.3 A Storytelling Error

After it has been determined that Grunfeld is wounded in the fight with the dragon, MINSTREL checks to see if it knows anything further about woundings that it should add to the scene. To do this, it tries to recall a previous scene in which a knight was wounded in a similar situation, and uses the recalled scene to further instantiate the story scene. Using two creativity heuristics (TRAM:SIMILAR-THWART-STATE and TRAM:GENERALIZE-ACTOR), MINSTREL eventually recalls the episode given in Figure 15.19.

Knight Fight

In order to impress the king, a knight killed a dragon, thwarting the dragon's goal of staying alive.

Figure 15.19 Knight Fight Episode

In fact, this is the same episode MINSTREL used to instantiate the main action of the story. In this case, however, "Knight Fight" is being used in a slightly different manner.

MINSTREL is trying to recall a scene in which some state thwarts a knight's goal of protecting his health. This specification recalls nothing, so MINSTREL applies the creativity heuristic TRAM:GENERALIZE-ACTOR, and the knight is generalized to another violent character: a monster. The new problem specification is "some state thwarts a monster's goal of protecting his health".

Because the dragon in "Knight Fight" dies rather than merely being wounded, this specification also recalls nothing. TRAM:SIMILAR-THWART-STATE is then applied to the specification. TRAM:SIMILAR-THWART-STATE is a creativity heuristic that modifies a thwarting state to a similar, related state. In this case, it modifies the knight's wounding to death. The new problem specification is "some state thwarts a monster's goal of staying alive." This specification succeeds in recalling the dragon's death in "Knight Fight".

The Adapt step of TRAM:SIMILAR-THWART-STATE adapts the recalled episode by changing the dragon's death back to a wounding. The Adapt step of TRAM:GENERALIZE-ACTOR changes the dragon back to a knight, and the modified episode is used to instantiate the knight's wounding in "The Proud Knight".

Unfortunately, the modified episode has with it the remainder of the original "Knight Fight" episode, in which the knight fights the dragon and impresses the king. These extraneous scenes are added to "The Proud Knight", resulting in the scenes in which Grunfeld fights another knight (the knight from "Knight Fight") and in which Grunfeld's injury impresses the king (the same way the dragon's death impressed the king in "Knight Fight").

This problem arises because TRAM:SIMILAR-TWHART-STATE did not correctly Adapt the re-

called episode. TRAM:SIMILAR-THWART-STATE knows specifically about states that thwart goals. While this TRAM was able to correctly adapt the thwarting state, it was unable to correctly adapt the extraneous scenes. In cases where a TRAM does not have the knowledge to correctly adapt some portion of a recalled episode, the TRAM should remove the unadaptable portion. Although this may lose some valuable information, it assures that the TRAM will not produce an incorrectly adapted solution, and is in keeping with MINSTREL's philosophy of TRAMs that are specific and limited in scope.

When TRAM:SIMILAR-THWART-STATE was modified to remove extraneous links, MINSTREL was able to tell "The Proud Knight" without error.

15.5.4 English Language Generation

The final step in MINSTREL's storytelling is to generate an English version of the story. This dissertation does not address the problem of language generation, but for completeness sake (and because it is interesting), we review here what was necessary to make MINSTREL generate an English version of "The Proud Knight".

MINSTREL uses a phrasal generator ([Reeves 1989][Zernik 1987]) as well as a small number of author-level plans that determine the order in which to generate the elements of a story. MINSTREL's first attempt to generate "The Proud Knight" was disappointing. This attempt is shown in Figure 15.20.

The Proud Knight

A knight named Grunfeld was very proud. Because he was very proud, Grunfeld wanted to impress the king. He fought a dragon. The dragon was destroyed but Grunfeld was hurt. Grunfeld wanted to protect his health. But he could not because he was hurt. Because he was hurt, Grunfeld wanted to be healed. Grunfeld hated himself. Grunfeld was hurt. Because he was hurt, Grunfeld wanted to be healed. Because Grunfeld was hurt. Because he was hurt, Grunfeld wanted to be healed. Grunfeld wanted to be healed.

Figure 15.20 Abortive Attempt to Generate "The Proud Knight"

MINSTREL's repetitive output and general confusion was traced to the pattern that was being used to generate the state in which Grunfeld was wounded. This state both thwarts a goal and motivates a goal: it thwarts a goal to stay healthy and motivates a goal to be healed. MINSTREL's phrasal pattern to talk about this situation generates the state without mentioning the thwarted and motivated goals, and then generates the thwarted goal, and then the motivated goal. This pattern is shown in Figure 15.21.

The notation shown in Figure 15.21 is complex, but the important features are easily explained[1].

```
(rap:phrase motivating-and-thwarting-state
    (input :sentence
           (*and* ?state
                  (state nil
                         &thwarts ?thwarted-goal
                         &motivates ?motivated-goal)))
    (output :sentence (:wo ?state &thwarts &motivates)
            :sentence ?thwarted-goal
            :sentence ?motivated-goal))
```

Figure 15.21 Phrasal Pattern

The pattern takes as input a state being generated as a sentence. The state being generated must have a thwarted goal and a motivated goal, as shown by the pattern in the second position of the *and* statement on the input side of the phrase. The first position of the *and* statement is a logical variable, ?state, to which the input state gets bound if it matches the second component of the *and*.

On the output side, the input state is stripped of its &thwarts and &motivates links (by the :wo macro) and generated as a sentence. It is important to strip the input state of its links if they are being processed separately, so that the phrasal generator does not get caught in an endless recursion. If the thwarted and motivated goals were not removed, this same pattern could match again, creating infinite recursion. After the stripped state is generated, the thwarted goal and the motivated goal are generated separately. This phrase is a simple application of conquer-and-divide. The input state is broken into parts that are smaller and easier to generate.

The difficulty with this phrase arises because it fails to strip the thwarted and motivated goals. The :wo macro is used to strip the thwarted and motivated goals from the original state, but is not used to strip the state away from the motivated and thwarted goals. This is necessary because in MINSTREL, links between schemas are bi-directional, and must be explicitly removed from each end. Consequently, when MINSTREL tried to generate English from the thwarted and motivated goals, it sometimes found itself back at the original state. This caused the repetition in the earlier output.

The solution is simple: the :wo keyword is used to remove the links between the thwarted and motivated goals and the original state. The corrected pattern is shown in Figure 15.22.

With this phrasal pattern corrected, MINSTREL produces a much better text for the story. This text is given in Figure 15.23. This story deserves two additional comments. First, the moral of this story is a canned phrase associated with PAT:PRIDE. MINSTREL cannot examine a themat-

1. For a more complete discussion of MINSTREL's representation, see Chapter 2. For a description of MINSTREL's phrasal generator, see Appendix A.

```
(rap:phrase motivating-and-thwarting-state
    (input :sentence
          (*and* ?state
                (state nil
                      &thwarts ?thwarted-goal
                      &motivates ?motivated-goal)))
    (output :simple-sentence (:wo ?state &thwarts &motivates)
            :sentence (:wo ?thwarted-goal &thwarted-by)
            :sentence (:wo ?motivated-goal &motivated-by)))
```

Figure 15.22 Phrasal Pattern

The Proud Knight

It was the Spring of 1089, and a knight named Grunfeld returned to Camelot.

A hermit named Bebe told Grunfeld that Bebe believed that if Grunfeld fought with the dragon then something bad would happen.

Grunfeld was very proud. Because he was very proud, Grunfeld wanted to impress the King. Grunfeld wanted to be near a dragon. Grunfeld moved to a dragon. Grunfeld was near a dragon. The dragon was destroyed because Grunfeld fought with the dragon. The dragon was destroyed but Grunfeld was hurt. Grunfeld wanted to protect his health. Grunfeld wanted to be healed. Grunfeld hated himself. Grunfeld became a hermit.

Moral: Pride goes before a fall.

Figure 15.23 The Proud Knight

ic structure and invent a pithy saying summarizing the abstraction. Second, the reader will note several features of this story similar to previous stories, such as Grunfeld's change of roles and Grunfeld's emotional reaction to his bad decision. As with the other stories, MINSTREL tries to achieve various "good writing" goals in telling PAT:PRIDE, but MINSTREL's limited repetoire of plans leads to scenes similar to those in the other stories.

15.6 Experiment #2: Adding a New Role to MINSTREL

In the first experiment, a new theme was added to MINSTREL. Although the structure of goals, actions and states used in the theme was new, the theme did not require any extension of MINSTREL's basic representation. As a second experiment, we decided to extend MINSTREL's representation slightly, to see how MINSTREL could handle storytelling involving entirely new

concepts.

15.6.1 The King

Until this point, MINSTREL knew of three roles that a character could have: a character could be a knight, a princess, or a hermit. We extended this representation by adding a new character role: the king.

Three things were necessary to add the king to MINSTREL's knowledge of character roles. First, a new symbol was assigned to represent this role: "&king". Second, the new symbol was added to MINSTREL's class hierarchy of roles. Kings were described as violent (as are knights and monsters) and social (as are princesses). Finally, phrases were added to the phrasal generator to tell MINSTREL how to talk about this new concept.

MINSTREL was given no other information about kings. In particular, MINSTREL was not given any episodes or story fragments involving kings. MINSTREL would be asked to tell a story about a king knowing nothing more than that a king was a kind of "sociable knight".

15.6.2 The King in Storytelling

MINSTREL was asked to tell a story about PAT-PRIDE with the main character being a king instead of a knight. Figure 15.24 shows the resulting story.

The Proud King

It was the Spring of 1089, and King Arthur returned to Camelot from elsewhere.

A hermit named Bebe told Arthur that Bebe believed that if Arthur fought with the dragon then something bad would happen.

Arthur was very proud. Because he was very proud, Arthur wanted to impress his subjects. Arthur wanted to be near a dragon. Arthur moved to a dragon. Arthur was near a dragon. The dragon was destroyed because Arthur fought with the dragon. The dragon was destroyed but Arthur was hurt. Arthur wanted to protect his health. Arthur wanted to be healed. Arthur hated himself. Arthur became a hermit.

Moral: Pride goes before a fall.

Figure 15.24 The Proud King

This story is exactly similar to "The Proud Knight". An examination of the trace shows that MINSTREL created this story in the same way it created "The Proud Knight", with one excep-

tion. In this story, MINSTREL used TRAM:GENERALIZE-ACTOR and its knowledge that kings and knights were both violent to generalize the king to a knight (and subsequently to a dragon).

This story shows how MINSTREL's creativity heuristics can, with a very minimal amount of information about a new concept, use it in a problem solution. Not unsurprisingly, this story is not particularly novel. MINSTREL does not have enough knowledge about kings to invent anything strikingly original. But MINSTREL is able to make use of what it does know to create a competent solution.

15.7 Experiment #3: Learning in MINSTREL

The previous two experiments looked at how MINSTREL's knowledge could be extended from outside the program. But MINSTREL also has the ability to extend its own knowledge through creativity. How can MINSTREL use creativity to extend its knowledge and change its behavior?

In MINSTREL, the basic mechanism for learning is episodic memory. As MINSTREL tells stories and invents solutions to problems, these problems and their solutions are added to episodic memory. These episodes can be used in turn to solve future problems. In this way, MINSTREL uses creativity to permanently extend its knowledge.

Novelty is judged against experiences.

MINSTREL also uses episodic memory to judge whether a problem solution is novel or creative. When MINSTREL encounters a problem solution - by recalling it from episodic memory, inventing it, or (hypothetically) reading about it, MINSTREL can determine whether the solution is novel and interesting by comparing it to the solutions in episodic memory. If the new solution has significant differences from all past solutions, MINSTREL recognizes the new solution as creative. And because MINSTREL's episodic memory changes as it solves problems and adds their solutions to memory, MINSTREL's definition of creativity also changes.

What is creative changes with experience.

In creative problem-solving domains like storytelling, this can be used to implement an "artistic drive". When telling a story, MINSTREL can compare the scenes it is writing with past scenes in episodic memory. If a scene is too similar to a past scene, the scene can be discarded on the grounds that it is too "boring". And as MINSTREL tells stories about particular subjects and adds those stories to memory, it's definition of what is interesting will change. Like a human artist, MINSTREL will be driven to be creative.

In this experiment, we look at this issue. The first part of this experiment shows how MINSTREL remembers solutions as they are invented and can then use them immediately - without re-inventing them - when asked to solve a similar problem. The second part of this experiment examines how MINSTREL's storytelling behavior changes as it tells stories and is driven to invent new stories.

15.7.1 Learning in MINSTREL

Chapter 3 showed how MINSTREL uses creativity to invent ways for a knight to commit suicide. Using two episodes from memory and three creativity heuristics, MINSTREL invents three methods for a knight to commit suicide. Figure 15.25 shows the reasoning MINSTREL used to invent one of these solutions.

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

Problem-Solving Cycle: &ACT.105.
 Executing TRAM:EXAGGERATE-SCALED-VALUE.
   Recalling: NIL.
[...]
 Executing TRAM:SIMILAR-OUTCOMES-PARTIAL-CHANGE.
   Recalling: NIL.
    [TRAM Recursion: &ACT.136.]
       Executing TRAM:GENERALIZE-CONSTRAINT.
       Generalizing :ACTOR on &ACT.136.
         Recalling: &PRINCESS-POTION.

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

Added back to memory as &ACT.206.
[...]
```

Figure 15.25 Initial Reasoning for Poisoning Example

As Figure 15.25 shows, when MINSTREL is first asked to invent a method for a knight to commit suicide, it uses two TRAMS (TRAM:SIMILAR-OUTCOMES-PARTIAL-CHANGE and TRAM:GENERALIZE-CONSTRAINT) to invent an episode in which a knight poisons himself. But once this new solution has been discovered and added to episodic memory, it can be used again with invoking any creativity heuristics.

Figure 15.27 shows a trace of MINSTREL being asked a second time for a method by which a knight can commit suicide. This time, MINSTREL recalls directly the method it had previously invented (&ACT.206). Once MINSTREL has invented a solution and indexed it in episodic memory, it can find and apply that solution quickly, without using creativity. By remembering created solutions, MINSTREL is able to improve its efficiency as a problem-solver.

```
================================================
            MINSTREL Invention
================================================
Initial specification is &ACT.354:
 (A KNIGHT NAMED JOHN DID SOMETHING *PERIOD*
  JOHN DIED *PERIOD*)

Problem-Solving Cycle: &ACT.354.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:  (&ACT.206).
Problem-Solving Cycle succeeds: (&ACT.358).

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

Figure 15.26 Reasoning After Learning

Knowledge extends creativity.

Remembering invented solutions not only extends MINSTREL's knowledge and improves its efficiency as a problem-solver, it also extends MINSTREL's creativity. Each remembered solution is the endpoint of a successful search that used creativity heuristics to find a new solution to a problem. In the case above, the poison solution is the endpoint of a creative process that invented the idea of poison and the idea of a knight intentionally ingesting poison to kill himself, both concepts that were previously unknown to MINSTREL. But now that these concepts have been discovered and remembered, they provide a new starting point for creativity. By starting at this remembered endpoint, MINSTREL can search further into the potential solution space to find new solutions that it would not have been able to discover previously because of search constraints.

For example, now that MINSTREL has invented the notion of intentional poisoning, it can invent yet another solution to the problem of killing yourself: having an agent poison you. MINSTREL was unable to discover this solution originally because it requires applying four creativity heuristics, and MINSTREL applies at most three creativity heuristics when searching for a problem solution[1]. But now that the compiled result of previous creativity is available as a new starting point for creative problem-solving, MINSTREL can discover this new solution (Figure 15.27).

Remembering invented solutions therefore provides MINSTREL with a mechanism for extending its knowledge, a way to improve its efficiency as a problem-solver, and a method for increas-

1. MINSTREL supposes some limit to the creative search process. This may be a time limit, an effort limit, or some combination of the two. Limiting MINSTREL to a search depth of three implements this type of a processing limit, but is not intended as a theoretical claim.

```
==============================================
            MINSTREL Invention
==============================================
Initial specification is &ACT.457:
 (A KNIGHT NAMED JOHN DID SOMETHING *PERIOD*
  JOHN DIED *PERIOD*)

[...]
 Executing TRAM:ACHIEVE-VIA-AGENT
   Recalling: (&ACT.206).
Problem-Solving Cycle succeeds: (&ACT.206).

Minstrel invented this solution:
 (A PERSON FED A KNIGHT NAMED JOHN A POTION
  IN ORDER TO KILL JOHN *PERIOD*
  JOHN DIED *PERIOD*)
```

Figure 15.27 Additional Suicide Solution

ing the scope of its creativity.

15.7.2 Learning and the Artistic Drive

In art, a premium is placed on significant new ideas. This drive to create is implemented in MINSTREL as a domain assessment that rejects problem solutions that are non-creative, i.e., problem solutions that are similar to known solutions. Consequently, MINSTREL's storytelling behavior changes as it tells and remembers stories. After creating a few stories in which a knight fights a dragon, MINSTREL becomes "bored" with knights fighting dragons and turns its storytelling attentions elsewhere. (For this reason, the domain assessment that drives MINSTREL to create new stories is called the "boredom assessment".)

This section looks at how the boredom assessment drives MINSTREL to change its storytelling behavior.

15.7.2.1 Methodology

The boredom assessment operates during the Assess step of problem-solving (see Chapter 3, Section 11). The boredom assessment tests each solution generated by creative problem-solving to see if the solution is "boring", i.e., non-creative. To do this, the boredom assessment uses the generated solution as an index to the problem-solver's episodic memory. If the generated solution recalls more than a small number of similar past solutions, it is judged boring and rejected.

The threshold at which the boredom assessment rejects a solution is a simple measure of how quickly a problem-solver becomes bored with reusing past solutions. In non-artistic domains, this threshold is high, indicating that the problem-solver is willing to reuse solutions many times. In artistic domains, this threshold is low, indicating that the problem-solver is motivated to discover new solutions. For this experiment the threshold was set to two. With this setting MINSTREL is willing to use a solution twice before beginning to reject it as boring.

To study how this affects MINSTREL's behavior, MINSTREL was repeatedly asked to tell a story about the theme "Pride goes before a fall" (PAT:Pride), in which the main character was a knight. After each story was told, the scenes in the story were added to episodic memory. In combination with the boredom assessment, this causes MINSTREL to tell a sequence of different stories.

15.7.2.2 The First Story

When MINSTREL is initially asked to tell a story about the theme "Pride goes before a fall" it tells the story "The Proud Knight", as detailed in Section 5 of this chapter. In this story, a knight tries to impress his king by killing a dragon, even though he is forewarned that fighting a dragon will lead to trouble.

After this story is told, it is added to episodic memory. MINSTREL adds a story to episodic memory by adding each schema which makes up the story to episodic memory. Although each schema is added individually to memory, the connections between the schemas are retained, so that MINSTREL will recall the entire story whenever a set of recall indices match any part of the story. (For more on MINSTREL's episodic memory, see Chapter 2.)

This story turns on MINSTREL's recall of an episode in which a knight fights a dragon. This episode is used to create the scene in "The Proud Knight" in which the knight fights a dragon and is injured. After this story has been added to episodic memory, there are two scenes in episodic memory in which a knight fights a dragon: the original scene, and the scene which was used in "The Proud Knight". Consequently, MINSTREL no longer creates scenes in which a knight fights a dragon, because those scenes have now become "boring".

15.7.2.3 The Second Story

In telling the original story about PAT:Pride, MINSTREL recalled an episode in which a knight fought a dragon, and used that to create the story scene in which the knight suffers his fall. To tell another story about PAT:Pride, MINSTREL must recall or invent a new story scene in which a knight suffers a failure. MINSTREL does this by using a recalled episode in which a knight competes at a joust. The resulting story is shown in Figure 15.28.

In this story, the knight's fall is a failure at a joust. This is created in two steps. First, MINSTREL creates the action that leads to the knights fall:

The Proud Knight II

It was the Spring of 1089, and a knight named Godwin returned to Camelot from elsewhere.

A hermit named Bebe told Godwin that Bebe believed that if Godwin jousted then something bad would happen. Godwin was very proud. Because Godwin was very proud, Godwin wanted to impress his king. Godwin jousted. Godwin lost the joust. Godwin hated himself.

Moral: Pride goes before a fall.

Figure 15.28 The Proud Knight II

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &ACT.4896.
Trying author-level plan ALP:DONT-INSTANTIATE.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
TRAM Cycle: &ACT.4896.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:
     Specializing TYPE to &JOUST.
   ...TRAM succeeds: (&ACT.4963).
TRAM Cycle succeeds: (&ACT.4963).
Found a reminding in ALP:GENERAL-INSTANTIATE.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

To create this scene, MINSTREL recalls an episode in which a knight fights another knight (a joust). This episode was not recalled in creating the original "The Proud Knight" story because the episode in which a knight fights a dragon was recalled instead. But now that scene is considered boring and MINSTREL can recall instead an alternate scene. This reveals one aspect of MINSTREL's behavior when using the boredom assessment: As solutions become boring, MINSTREL can discover other, usable solutions which it has known all along, but never previously used because other solutions were preferred.

The second part of the knight's fall is the failure he suffers at the joust. This is created using ALP:INSTANTIATE-FAILURE, which suggests that one way an action can cause a goal failure is if the action itself fails. The overall result is a story in which a knight suffers an unintended loss at a joust. Again, this is added to memory.

15.7.2.4 The Third Story

MINSTREL has now told stories about a knight fighting a monster and a knight fighting a joust. Both these episodes are now considered boring because they have been used twice. (Once in the original episode, and once in the new stories.) There remains one episode in memory which involves a knight which has not become boring. In this episode, a knight rides his horse to the woods. MINSTREL uses this as a basis for a new story, as shown in Figure 15.29.

The Proud Knight III

It was the Spring of 1089, and a knight named Cedric returned to Camelot from elsewhere.

A hermit named Bebe told Cedric that Bebe believed that if Cedric moved to the woods then something bad would happen. Cedric was very proud. Because he was very proud, Cedric wanted to have some berries. Cedric wanted to be near the woods. He was at the woods because Cedric moved to the woods. He was at the woods but saw that a knight named Arthur kissed with a princess named Andrea. Cedric loved Andrea. Because Cedric loved Andrea, Cedric wanted to be the love of Andrea. But he could not because Arthur kissed with Andrea. Cedric hated himself. Cedric had some berries because Cedric picked some berries.

Moral: Pride goes before a fall.

Figure 15.29 The Proud Knight III

MINSTREL invents the important elements of this story in two steps. First, MINSTREL uses TRAM:RECALL-ACT to invent the action which will lead to the knight's fall. Because the other scenes in memory which involve a knight have become boring, TRAM:RECALL-ACT recalls the episode in which a knight rides to the woods to fight a dragon. TRAM:RECALL-ACT adapts this for use in the current story:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &ACT.4891.
Trying author-level plan ALP:DONT-INSTANTIATE.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
TRAM Cycle: &ACT.4891.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    Recall failed.
   ...TRAM failed.
   Executing TRAM:RECALL-ACT.
     Recalling:    (&CEDRIC-RIDE).
   ...TRAM succeeds: (&ACT.4964).
TAM Cycle succeeds: (&ACT.4964).
Found a reminding in ALP:GENERAL-INSTANTIATE.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

The second step is to invent the knight's fall, i.e., the events that lead to the knight having a failed goal. This is accomplished by using three TRAMs in conjunction (TRAM:OPPOSITE-STATE-ACHIEVES, TRAM:GENERALIZE-ACTOR, and TRAM:LIMITED-RECALL) to invent a scene in which the knight suffers a goal failure by seeing another knight kissing a princess he loves:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &GOAL.8323.
Trying author-level plan ALP:DONT-INSTANTIATE.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
TRAM Cycle: &GOAL.8323.
[...]
   Executing TRAM:OPPOSITE-STATE-ACHIEVES.
     Recalling:    Recall failed.
   [TRAM Recursion: &GOAL.8715.]
[...]
      Executing TRAM:GENERALIZE-ACTOR.
        Recalling:       Recall failed.
      [TRAM Recursion: &GOAL.8748.]
[...]
         Executing TRAM:LIMITED-RECALL.
           Recalling:           (&AR-AFFECTION).
             ...TRAM succeeds: (&GOAL.8757).
TRAM Cycle succeeds: (&GOAL.8757).
Found a reminding in ALP:GENERAL-INSTANTIATE.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

The ultimate source of this scene is an episode named &AR-AFFECTION in which a princess achieves her goal of loving a knight by giving him a kiss. How is this used to create a scene in which a knight suffers a goal failure?

First, TRAM:LIMITED-RECALL and TRAM:GENERALIZE-ACTOR combine to substitute a knight for the princess, so that this recalled episode can be applied to the knight in the story.

Second, TRAM:OPPOSITE-STATE-ACHIEVES turns the princess's successful goal into the knight's failed goal by "reversing" the state that achieves the princess's goal. In this case, that results in a different character kissing the princess. (See Chapter 4 for more details about how TRAM:OPPOSITE-STATE-ACHIEVES works.)

The end result is a scene in which a knight suffers a goal failure when another knight kisses the princess he loves.

There is one more interesting bit of creativity involved in telling this story. After MINSTREL creates the scenes shown above, it notices that the first scene – in which a knight rides to the woods – is inconsistent, because there is no reason for the knight to do that action. MINSTREL

knows of some actions that knights take without specific reason – such as fighting monsters – but riding to the woods is not one of them. Consequently, MINSTREL gives itself the goal of finding a purpose for the knight's trip to the woods.

One reason a knight might travel to the woods is to fight a dragon, but MINSTREL's previous use of that type of scene and the boredom assessment prevent it from being reused in this context. So MINSTREL must invent a new reason for a knight to travel to the woods. It does this using TRAM:GENERALIZE-ACTOR:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &MAKE-GOAL-CONSISTENT applied: &GOAL.8322.
Trying author-level plan ALP:MAKE-CONSISTENT-PRECOND.
Trying author-level plan ALP:MAKE-CONSISTENT-ALOVE.
Trying author-level plan ALP:MAKE-CONSISTENT-SUPERGOAL.
TRAM Cycle: &GOAL.9096.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    Recall failed.
   ...TRAM failed.
   [...]
   Executing TRAM:GENERALIZE-ACTOR.
     Recalling:    (&GOAL-BERRIES).
     Adapting &GOAL.9256.
   ...TRAM succeeds: (&GOAL.9256).
TRAM Cycle succeeds: (&GOAL.9256).
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

TRAM:GENERALIZE-ACTOR suggests that knights might be similar in some ways to princesses (because they are both social characters), recalls that princesses sometimes go to the woods to pick berries, and consequently creates a scene in which a knight goes to the woods to pick berries.

As it happens, this too is inconsistent (knights do not normally pick berries). Again, MINSTREL notices this and creates a goal to add story scenes to explain why a knight would be picking berries. But MINSTREL cannot recall or invent any reason for a knight to be picking berries, so this scene has no further explanation in the final story.

In this case, MINSTREL's initial reasoning (that knights might be picking berries because they share characteristics with princesses, who do pick berries) is sound, and the resulting scene reasonable. In general, MINSTREL does not currently try to judge when a failed author-level goal results in a "bad" story. How this might be done is discussed in Chapter 16, Future Work.

15.7.2.5 The Fourth Story

MINSTREL has now used all the episodes in memory which involve a knight at least twice. To continue to tell stories about proud knights, MINSTREL must begin adapting knowledge about different characters (i.e., monsters, princesses, and hermits). The first result of this is shown in Figure 15.30.

The Proud Knight IV

It was the Spring of 1089, and a knight named Cedric returned to Camelot from elsewhere.

A hermit named Bebe told Cedric that Bebe believed that if Cedric fought Frederick then something bad would happen. Cedric was very proud. Because he was very proud, Cedric wanted to kill Frederick. Cedric wanted to be near Frederick. Cedric moved to Frederick. Frederick was killed because Cedric fought Frederick. Frederick was killed but Cedric was killed. Cedric wanted to protect his health. Cedric wanted to be healed. Cedric hated himself.

Moral: Pride goes before a fall.

Figure 15.30 The Proud Knight IV

In the first story MINSTREL told about PAT:Pride, a knight fought a monster and was wounded. This story is similar to the first story, but differs in several significant ways: the knight fights another knight instead of a monster, and he is killed instead of just being wounded. And unlike the scenes in the previous stories, these scenes are not based upon episodes in which a knight is the main actor.

Before discussing how MINSTREL creates this story, it is interesting to look at the final scenes of the story. At the end of this story, Cedric is killed, and then wants to protect his health, be healed, and hates himself. Inasmuch as doing anything while dead is difficult, these are story-telling mistakes.

These mistakes arise because MINSTREL's English language generation doesn't make clear the timing of character reactions. In most cases, the timing is irrelevant. But in the case of a character dying, language should be used which makes it clear that character reactions occuring during the dying process:

> ...Because he was very proud, Cedric wanted to kill Frederick. Cedric wanted to be near Frederick. Cedric moved to Frederick. Frederick was killed because Cedric fought Frederick. Frederick was killed but Cedric was killed. *As Cedric was dying,* he wanted to protect his health. Cedric wanted to be healed. Cedric hated himself. Cedric died.

The story representation is the same; but the way in which the story is expressed has been

changed. This reinforces the complex nature of storytelling: to tell a story successfully requires the careful interaction and cooperation of a variety of complex processes.

To create the scene in which Cedric fights Frederick, MINSTREL uses TRAM:GENERALIZE-ACTOR to adapt an episode involving a monster to the current story:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &ACT.4875.
Trying author-level plan ALP:DONT-INSTANTIATE.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
TRAM Cycle: &ACT.4875.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    Recall failed.
   ...TRAM failed.
[...]
   Executing TRAM:RECALL-ACT.
     Recalling:    Recall failed.
   [TRAM Recursion: &ACT.4947.]
[...]
      Executing TRAM:GENERALIZE-ACTOR.
        Recalling:      (&KILLING-ACT).
            Adapting &ACT.4997.
      ...TRAM succeeds: (&ACT.4997).
   ...TRAM succeeds: (&ACT.4998).
TRAM Cycle succeeds: (&ACT.4998).
Found a reminding in ALP:GENERAL-INSTANTIATE.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

TRAM:GENERALIZE-ACTOR suggests that a knight might be like a monster in some ways (they are both violent characters), and this recalls the episode &KILLING-ACT, which is an episode in which a monster fights a knight. TRAM:GENERALIZE-ACTOR adapts this episode to the current story by substituting Cedric for the monster, resulting in a scene in which Cedric fights another knight (Frederick).

To create the goal failure resulting from this action, MINSTREL again uses TRAM:GENERALIZE-ACTOR to suggest that knights are like monsters. This time, TRAM:GENERALIZE-ACTOR helps MINSTREL recall the scene from the first PAT:Pride story in which the monster was killed by the knight. MINSTREL then adapts this to the current story by replacing the monster with the knight, resulting in a scene in which a knight is killed by a knight:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &GOAL.8303.
Trying author-level plan ALP:DONT-INSTANTIATE.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
TRAM Cycle: &GOAL.8303.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    Recall failed.
   ...TRAM failed.
[...]
   Executing TRAM:GENERALIZE-ACTOR.
     Recalling:    (&FIGHT2).
     Recalling:    Recall failed.
     Recalling:    (&KILLING-ACT).
     Recalling:    (&GOAL.8947).
   ...TRAM succeeds: (&GOAL.8947).
TRAM Cycle succeeds: (&GOAL.8947).
Found a reminding in ALP:GENERAL-INSTANTIATE.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

There are two revealing points to this story.

Creativity uses old experiences in new ways.

First, this story illustrates an interesting feature of how MINSTREL's creative drive and episodic memory interact. Although the boredom assessment prevents MINSTREL from re-using a scene in a way similar to previously told stories, it does not prevent MINSTREL from making use of a previously used scene in a new way. This is evident in the creation of the second part of this story, in which MINSTREL reuses a scene from the first story (a knight killing a dragon) in a new way to create a scene which passes the boredom assessment (a knight dying as a result of fighting another knight).

Creativity creates new experiences.

The second interesting feature concerns how episodic memory changes during storytelling. When MINSTREL began telling this story, all of the episodes in memory involving knights as actors were considered boring, forcing MINSTREL to invent scenes by analogizing knights to monsters. But after telling this story, episodic memory contains a new scene involving a knight as an actor which is *not* boring: the scene in which a knight fights a knight. Since this scene is completely new to MINSTREL, and the boredom heuristic only considers an episode boring after it has been used twice, this scene is still considered interesting.

The important point here is not the precise setting of the boredom heuristic, but rather the general behavior of MINSTREL. As MINSTREL creates new story scenes in response to story needs and the drive of the boredom assessment, episodic memory grows and changes. New episodes are added; old episodes are found to be boring. Memory and behavior evolve in response to

problem-solving.

15.7.2.6 The Fifth Story

In telling the next story, MINSTREL makes use of the scene it invented for the fourth story - a knight fighting a knight - and invents a new goal failure for the second part of the theme. This story is shown in Figure 15.31.

The Proud Knight V

It was the Spring of 1089, and a knight named Cedric returned to Camelot from elsewhere.

A hermit named Bebe told Cedric that Bebe believed that if Cedric fought Frederick then something bad would happen. Cedric was very proud. Because he was very proud, Cedric wanted to kill Frederick. Cedric wanted to be near Frederick. Cedric moved to Frederick. Frederick was killed because Cedric fought Frederick. Frederick was killed but Cedric's sword was destroyed. Cedric wanted to possess a sword. Cedric wanted Cedric's sword to be healed. Cedric hated himself.

Moral: Pride goes before a fall.

Figure 15.31 The Proud Knight V

The development of this story is straightforward. MINSTREL creates the scene in which Cedric fights Frederick by recalling the similar scene from the previous story. The goal failure associated with this scene is created by an author-level plan which suggests that an unintended result of an action which involves an object could be the destruction of that object:

```
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Author-level goal &INSTANTIATE applied to &STATE.7242.
Trying author-level plan ALP:DONT-INSTANTIATE.
Trying author-level plan ALP:GENERAL-INSTANTIATE.
Trying author-level plan ALP:INSTANTIATE-EVIDENCE-2.
Trying author-level plan ALP:INSTANTIATE-UNTHWARTS.
Trying author-level plan ALP:INSTANTIATE-SUPERSEDING-BELIEF.
Trying author-level plan ALP:INSTANTIATE-LOSS-OF-OBJ.
Loss of &PHYSOBJ.549.
Author-level planning succeeded.
++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Again, this story illustrates how creativity has opened a new area for MINSTREL to explore. In the previous story, MINSTREL invented a scene in which a knight fights a knight, and discovered one possible bad consequence of that action. In this story, MINSTREL continues to explore

this new area, inventing another possible bad consequence. To do this, MINSTREL used an author-level plan that wasn't applicable before creativity had invented the scene of two knights fighting. As this demonstrates, MINSTREL's creativity and domain-specific knowledge can interact to extend MINSTREL's knowledge.

15.7.2.7 The Sixth Story

At this point, MINSTREL's storytelling fails. MINSTREL has exhausted its ability to invent story scenes which meet the constraints of the theme and pass the boredom assessment. Although there are many possible combinations of the episodes in memory, and many more combinations that can be invented using creativity heuristics, only a very small percentage of that space can be used for stories based on PAT:Pride. PAT:Pride requires scenes in which a character is motivated by pride to perform a hasty action that he later regrets. This combination of specific constraints limits the range of MINSTREL's creativity.

This is not unexpected. MINSTREL is a finite system. When asked repeatedly to find new solutions to a problem, it will eventually run out of ideas.

An interesting question which can be posed at this juncture is whether the knowledge that MINSTREL acquired in telling five stories about the theme "Pride goes before a fall" carries over into other tasks. This sequence of five stories has demonstrated how MINSTREL's behavior changes as it repeatedly solved the same task. Does this learning carry over into other similar problems? To examine this issues, MINSTREL was asked to tell a story about another theme, PAT:Good-Deeds-Rewarded.

PAT:Good-Deeds-Rewarded is based on a theme from the movie *It's a Wonderful Life*. *It's a Wonderful Life* stars Jimmy Stewart as George Bailey, a good, unselfish man who constantly turns away from his dreams in order to do good deeds for others. When it appears that the town tyrant is going to take the Bailey Savings and Loan away from George he despairs and wishes that he had never been born. An angel shows George what the lives of his friends and family would have been like if he had never been born and that experience restores his spirit. At the end of the film, the people he has helped all his life band together to help him save the bank. One of the central themes of this movie is that one should be kindly and help others when possible, because someday that kindness may be returned. PAT:Good-Deeds-Rewarded represents this theme. (For more details on PAT:Good-Deeds-Rewarded, see Chapter 6.)

When MINSTREL is initially asked to tell a story about PAT:Good-Deeds-Rewarded, it tells the story shown in Figure 15.32.

The first part of this story is based upon the scene in episodic memory in which a knight fights a dragon. However, when MINSTREL is asked to tell a story about this theme after telling the previous five stories about PAT:Pride, this episode is not available. It has been used in several previous stories and is rejected by the boredom assessment. Consequently, MINSTREL tells a completely different story about PAT:Good-Deeds-Rewarded. This new story is shown in Figure

The Knight and the Hermit

Once upon a time, there was a hermit named Bebe and a knight named Cedric. One day, Cedric was wounded when he killed a dragon in order to impress the King. Bebe, who was in the woods picking berries, healed Cedric. Cedric was grateful and vowed to return the favor.

Later, Bebe believed he would die because he saw a dragon moving towards him and believed that it would eat him. Bebe tried to run away but failed. Cedric, who was in the woods, killed a dragon and saved Bebe.

Moral: "A favor earned is a favor returned."

Figure 15.32 Initial PAT:Good-Deeds-Rewarded Story

15.33.

The Knight and the Hermit

Once upon a time, there was a hermit named Bebe and a knight named Cedric. One day, Cedric moved to a princess named Andrea. Cedric was with Andrea. Cedric loved Andrea. Because Cedric loved Andrea, Cedric wanted to be the love of Andrea. Bebe kissed Andrea. Cedric was grateful and vowed to return the favor.

Later, Bebe wanted to scare a hermit named Edwina. Bebe wanted Edwina to believe she would die. Cedric fed Edwina a magic potion that made her appear dead.

Moral: "A favor earned is a favor returned."

Figure 15.33 Initial PAT:Good-Deeds-Rewarded Story

There are several interesting points about this story.

First, MINSTREL makes an "error of knowledge" in telling this story. MINSTREL does not realize that the goal of achieving love cannot be satisfied by an agent. Consequently, MINSTREL creates the questionable scene in which Cedric's goal to have Andrea love him is achieved by Bebe kissing Andrea. Section 12 of this Chapter discusses in more detail the types of errors MINSTREL makes, and what they have to say about MINSTREL's model of creativity. What is important to note here, though, is that MINSTREL's experiences in telling stories about PAT:Pride force it to create a new solution to the first part of this story. Although the solution MINSTREL invents is not quite correct, it is still indicative of how MINSTREL behavior changes as it learns.

Second, this example illustrates how MINSTREL's learning is not limited to the context in which the learning occurred. As MINSTREL told stories about PAT:Pride, its behavior changed. But that change of behavior has also carried over into a different context. What MINSTREL learned telling stories about PAT:Pride affects the story it tells about PAT:Good-Deeds-Rewarded.

This becomes more evident when one examines *how* MINSTREL creates this story. To create the scene in the second part of this story in which Cedric feeds Edwina a potion that makes her appear dead, MINSTREL makes use of the idea of poison, a concept which was invented during MINSTREL's creation of suicide methods for a knight. MINSTREL thus makes use of what it has previously learned to solve a later problem.

15.7.3 Boredom Assessment

Earlier we presented boredom assessment as a simple heuristic which rejected any problem solution that was similar to two other solutions in MINSTREL's episodic memory. As implemented for the storytelling domain, however, boredom assessment is more complicated.

Not everything in a story needs to be novel. Familiar events give the reader a context by which to understand a story. Further, to make every aspect of a story new and interesting would require a great deal of effort by the author. Even a human author would find this difficult, and it would add little to the reader's enjoyment of the story. Generally, authors seek to make the important parts of their stories novel and interesting, and are content to use familiar solutions in the supporting parts of the story.

In the case of theme-based stories such as MINSTREL tells, the important parts of the story are the scenes that illustrate the theme (the plot). Consequently, MINSTREL applies the boredom assessment only to author-levels goals that create the scenes that illustrate the theme. Further, MINSTREL only requires that some part of the plot be creative; it does not require that every aspect of the plot be new.

Each of the five stories MINSTREL tells about PAT:Pride has the opening scene:

> It was the Spring of 1089, and a knight named Cedric returned to Camelot from elsewhere.

This scene results from an author-level goal to create an introduction to the story that establishes the time of the story and presents the main character to the reader. It is used repeatedly because it is not part of the plot.

However, even scenes in the plot can be repeated. For example, the following scene appears in all five of MINSTREL's stories about PAT:Pride:

> A hermit named Bebe told Cedric that Bebe believed that if Cedric did ... then something bad would happen.

The structure of this scene derives directly from the theme and therefore must be the same in all of the stories. But the choice to make the Bebe character a hermit does not derive from theme, and could change from story to story. It does not because other parts of the plot change from story to story, and MINSTREL only requires that some part of the plot change to find a story acceptable.

MINSTREL has implemented a particular drive for creativity - one that looks for novelty and interest in the parts of a story that illustrate the theme of the story. For MINSTREL, this is a reasonable heuristic that produces realistic storytelling behavior. But each artistic problem-solving domain and each artist will have different requirements about novelty, and these will change over time. The importance of the boredom assessment is not that it defines precisely when an artist gets bored, but rather that it provides a mechanism for implementing many different creative drives.

By using episodic memory to determine whether a problem solution is creative, MINSTREL provides a dynamic definition of creativity that changes in response to the problem-solver's goals, understanding of the problem, and his problem-solving history. This understanding of how to judge the creativity of a problem solution provides a basis for implementing many different types of creative drives. Furthermore, by basing judgement of creativity upon episodic memory and not upon static standards of interestingness MINSTREL model has a mechanism by which an author can learn new standards of what is creative.

15.8 Experiment #4: Mechanical Invention

The first two experiments described here looked at how adaptable MINSTREL's creative problem-solving was within the storytelling domain. The third experiment looked at MINSTREL's learning behavior in the domain of storytelling. At a higher level, we'd also like to examine how general MINSTREL's creativity model is across problem domains. If MINSTREL can be easily adapted to a problem domain different from storytelling, it is some assurance that MINSTREL's model of creativity is not merely an ad-hoc solution to the storytelling problem.

To that end, MINSTREL was applied to the problem of simple mechanical invention. In this experiment, MINSTREL invents staplers that can handle a thick set of papers. MINSTREL knows about ordinary staplers and some simple mechanical devices, and uses this knowledge to invent several heavy-duty staplers. This experiment illustrates both how MINSTREL's creativity process can be applied to problems outside of the storytelling domain, and how domain assessments can be used to drive the creative process.

15.8.1 Representation

MINSTREL knows about three types of mechanical devices. "Sources" are devices such as motors and hands that generate power. "Sinks" are devices such as brakes and the head of a stapler that use power. "Converters" are devices that transfer and translate power between devices. Levers and pistons are examples of this kind of device.

All of these devices are represented by the &DEVICE schema. The &DEVICE schema has slots for the type of the device, the input power of the device, the input power type (i.e., rotary, vector), and the output power and type. Each instance of a &DEVICE can also have links to an input device and an output device. Figure 15.34 illustrates the representation for a lever that transforms a weak vector input into a normal vector output.

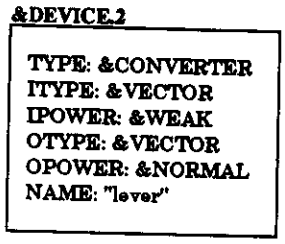


Figure 15.34 Representation of a Lever

MINSTREL's representation of mechanical devices is obviously very simplified. The purpose of MINSTREL's mechanical device invention is to demonstrate creativity in a second problem domain, not to accurately model human understanding of devices. For more detailed models of mechanical device understanding, see [Hodges 90][Forbus][Barchardt][].

15.8.2 Episodic Memory

MINSTREL's episodic memory initially contains four devices: a hand-powered stapler, a weak-to-normal strength lever, a rotary motor, and a piston that converts normal-strength vector motion to normal-strength rotary motion (i.e., a car piston). The representations of these devices are shown in Figure 15.35.

15.8.3 Domain Assessments

MINSTREL uses two domain assessments during mechanical invention. The *consistency assessment* examines each device MINSTREL invents to determine if it has a proper power source. If it doesn't, invention is used recursively to invent a power source. The *efficiency assessment* rejects devices that have superfluous components, such as a device that converts rotary motion into vector motion only to convert it back again.

These assessments are examples of how domain-specific requirements are added to the creative process. Assessing a domain constraint after creation is complete and then repairing any faults, as the consistency assessment does, frees the creative process from having to exhaustively understand the problem domain and take into account every domain constraint during problem-solving. Delaying criticism of inventions, as the efficiency assessment does, gives the creator the freedom to explore solutions that a more strict interpretation of the domain constraints might

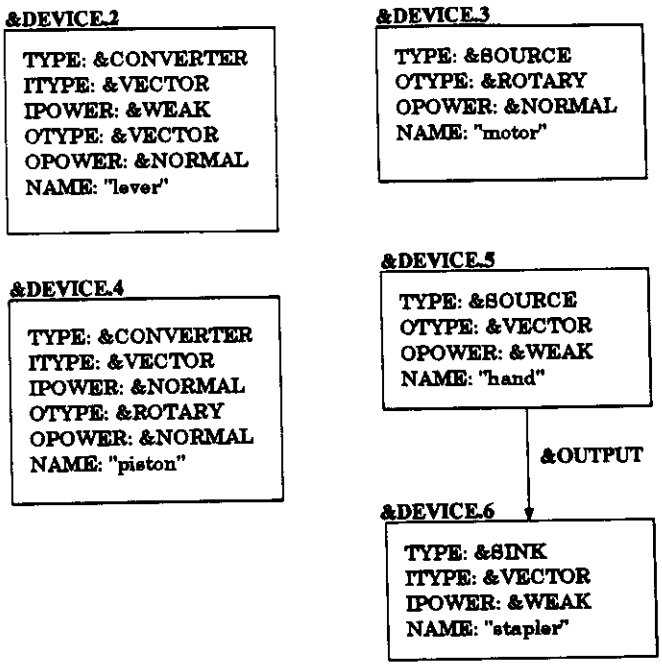


Figure 15.35 Episodic Memory

miss. Delaying criticism is the main feature of brainstorming, a suggested human creativity technique [Osborn 1953].

15.8.4 Creativity Heuristics for Mechanical Invention

For this new problem domain, MINSTREL was given a small set of Transform-Recall-Adapt Methods: TRAM:Generalize-Power-Source, TRAM:Generalize-Converter, TRAM:Power-Converter, and TRAM:Reversible-Devices. These TRAMs capture simple, commonplace understandings about how mechanical devices can be modified.

TRAM:Generalize-Power-Source suggests that a device that can provide power at one level might also be able to provide power at another level. If weak electrical motors exist, more powerful electrical motors might exist. To prevent over-generalizations, MINSTREL assumes only small changes in power output, i.e., a weak motor generalizes to only a normal motor, not a strong motor.

TRAM:Generalize-Converter makes a similar suggestion about devices which convert power from one form to another. TRAM:Generalize-Converter suggests that a converter that steps up or down the power of a particular input might also be able to step up or down the same amount a different input, i.e., a device that converts weak vector power to normal vector power might be

also be used to convert normal vector power to strong vector power.

TRAM:Power-Converter suggests substituting a power converter for a power source, and then looking for a power source for the converter.

Finally, TRAM:Reversible-Devices suggests that converters can be run backwards. That is, if MINSTREL knows of a converter that transforms weak power to strong power, then it can guess that the same converter can be used backwards to transform strong power to weak power.

15.8.5 The Problem

In this experiment, MINSTREL is asked to invent a heavy-duty stapler - a stapler head which has a strong vector input. Figure 15.36 shows MINSTREL's representation of this problem. &DEVICE.7 is the stapler head; &DEVICE.6 is an unknown device that provides strong vector input power to the stapler head. MINSTREL's task is to invent a device or series of devices that can provide strong vector power to the stapler head.

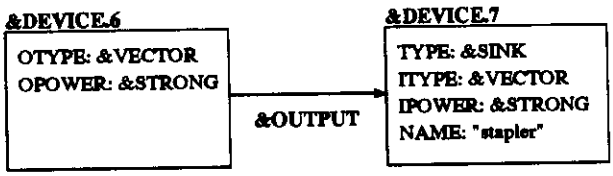


Figure 15.36 Representation of a Lever

15.8.6 Trace

In this experiment, MINSTREL exhaustively invents solutions to the heavy-duty stapler problem. Because invention is used recursively by the consistency assessment, and because MINSTREL follows several reasoning paths concurrently in this experiment, an unedited trace of MINSTREL inventing solutions to the stapler problem is difficult to follow. For the reader's convenience, the trace shown in this section has been edited for clarity. Portions of the trace irrelevant to the solution have been deleted (and marked with "[...]"), and concurrent lines of reasoning have been separated and presented in a serial fashion. MINSTREL's reasoning for one of the four stapler inventions is shown in its entirety. MINSTREL's reasoning for the remaining three inventions is summarized briefly at the end of this section.

MINSTREL begins invention by trying to find a device with a strong vector power output to drive the stapler head (&DEVICE.6 in Figure 15.36). Normal problem-solving (TRAM:Standard-Problem-Solving) fails, because there is no such device in memory. MINSTREL then applies TRAM:Power-Converter to the problem. TRAM:Power-Converter suggests looking for a converter with a strong vector output instead of a power source with strong vector

output:

```
==============================================
            MINSTREL Invention
==============================================
Initial specification is &DEVICE.6:
   (AN UNKNOWN DEVICE *COMMA* FEEDING INTO A
      STRONG STAPLER *PERIOD*)

TRAM Cycle: &DEVICE.6.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    NIL.
   ...TRAM failed.
   Executing TRAM:POWER-CONVERTER.
     Recalling:    NIL.
   ...TRAM failed.
```

The new problem description (&DEVICE.11) also fails to recall a device from episodic memory (as indicated by the line `Recalling... NIL`), because there are no devices in memory which convert power to a strong vector output. (Recall that the lever in MINSTREL's memory converts to *normal* vector output.) Since problem-solving has failed to find an appropriate converter, MINSTREL recursively applies another TRAM, this time to the converter suggested by TRAM:Power-Converter. TRAM:Generalize-Converter is selected and applied. TRAM:Generalize-Converter suggests that a converter that steps up or down the power of a particular input might also be able to step up or down the same amount a different input. In this case, it suggests looking for a device that can convert weak power to normal power, and then use that device to convert normal power to strong power. This time problem-solving succeeds, because the new problem description recalls the lever in MINSTREL's memory:

```
   [TRAM Recursion: &DEVICE.11.]
      Executing TRAM:GENERALIZE-CONVERTER.
        Recalling:    (&EXAMPLE-LEVER).
   ...TRAM succeeds: (&DEVICE.61).
Minstrel invented &DEVICE.101.

(DEVICE &DEVICE.101
   :IPOWER &NORMAL
   :ITYPE  &VECTOR
   :TYPE   &LEVER
   :OTYPE  &VECTOR
   :OPOWER &STRONG
   &OUTPUT <==>  (&DEVICE.102))
```

The initial problem has now been solved. MINSTREL has found a device that outputs strong vector power: a type of lever. &DEVICE.101 is illustrated in Figure 15.37.

However, examining Figure 15.37 will reveal a new problem: there is no input power to &DE-

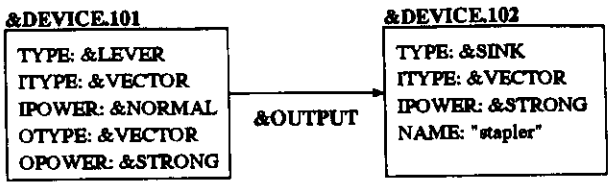


Figure 15.37 Initial Problem Solution

VICE.101. Solving the original problem has given rise to a new problem: the lever needs a normal vector power source. MINSTREL notices this problem using a domain assessment[1] and calls creative problem-solving recursively to correct it:

```
&DEVICE.101 is inconsistent: it lacks a power source.
Inventing a power source for &DEVICE.101.

TRAM Cycle: &DEVICE.105.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:    NIL.
   ...TRAM failed.
   Executing TRAM:POWER-CONVERTER.
     Recalling:    NIL.
   ...TRAM failed.
   [TRAM Recursion: &DEVICE.241.]
      Executing TRAM:REVERSIBLE-DEVICES.
        Recalling:       (&EXAMPLE-PISTON).
    ...TRAM succeeds: (&DEVICE.248).
[...]

Minstrel invented &DEVICE.253.

(DEVICE &DEVICE.253
   :IPOWER &NORMAL
   :ITYPE  &ROTARY
   :TYPE   &PISTON
   :OPOWER &NORMAL
   :OTYPE  &VECTOR
   &OUTPUT <==>  (&DEVICE.254))
```

There is no normal vector power source in MINSTREL's memory, so TRAM:Power-Converter is applied to the problem description. This TRAM suggests looking for a device that converts nor-

1. Domain assessments are domain-dependent measures of the acceptability of a solution. They act during the Assess step of problem-solving. In mechanical invention, one domain assessment checks inventions to be sure that they have a power source. Another checks for redundant sub-parts of a device.

mal rotary power to normal vector power. This also fails, because the piston in MINSTREL's memory converts normal vector power to normal rotary power (not vice versa), so TRAM:Reversible-Devices is applied. This TRAM suggests that a converter device can be reversed. When this reversal is applied to the problem description, the piston is recalled, and MINSTREL solves the problem of providing normal vector power to the lever by using a piston "backwards".

Once again the domain assessment module notices that the device lacks a power source, so problem-solving is called again to find a normal rotary power source. This succeeds immediately when MINSTREL recalls the motor from memory:

```
&DEVICE.253 is inconsistent: it lacks a power source.
Inventing a power source for &DEVICE.253.

TRAM Cycle: &DEVICE.268.
   Executing TRAM:STANDARD-PROBLEM-SOLVING.
     Recalling:     (&EXAMPLE-MOTOR).
```

The final device uses a normal rotary motor along with a piston and a lever to power the heavy-duty stapler head:

```
Minstrel invented &DEVICE.273:
   (A DEVICE POWERED BY A NORMAL ROTARY MOTOR
      *COMMA* FEEDING INTO A PISTON INTO A LEVER
      INTO A STRONG STAPLER *PERIOD*)
```

MINSTREL has invented the electric stapler. The final invention is illustrated in Figure 15.38.

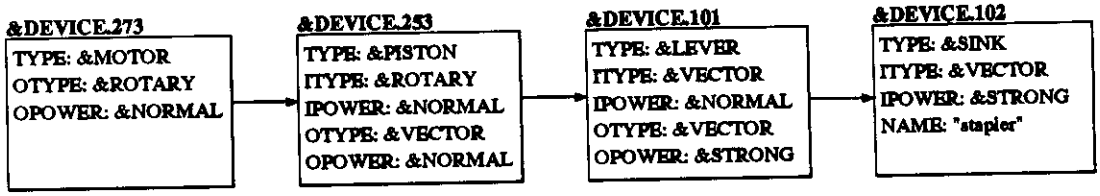


Figure 15.38 Final Solution

15.8.7 Additional Invention

As mentioned earlier, MINSTREL invents three other staplers. These are shown in Figure 15.39.

The first of these other solutions suggests finding a "strong vector hand" to power the stapler, i.e., finding someone stronger than yourself to do the stapling.

```
[...]
Minstrel invented this device:
   (A DEVICE POWERED BY A STRONG VECTOR HAND *COMMA* FEEDING
      INTO A STRONG STAPLER *PERIOD*)
[...]
Minstrel invented &DEVICE.716:
   (A DEVICE POWERED BY A NORMAL VECTOR HAND *COMMA*
      FEEDING INTO A LEVER INTO A STRONG STAPLER *PERIOD*)
[...]
Minstrel invented &DEVICE.938:
   (A DEVICE POWERED BY A WEAK VECTOR HAND *COMMA*
      FEEDING INTO A LEVER INTO A LEVER INTO A STRONG
      STAPLER *PERIOD*)
```

Figure 15.39 Mechanical Invention Trace Part 3

The second and third solutions both use levers to generate the force needed to power the heavy-duty stapler. A single lever suffices for a normal strength hand. Two stacked together are needed for weak hands. These are the familiar long-handled extension staplers available in most offices.

In fact, MINSTREL will stack as many levers as needed to generate a required force. In a sense, MINSTREL learns the general rule "The longer the lever, the greater the leverage", although it never represents this knowledge explicitly. Also, MINSTREL does not realize that two stacked short levers can be replaced by a single long lever, because MINSTREL's simple representation of this domain does not include concepts like attachment or length.

15.8.8 Domain-Independent Creativity Heuristics

It was not an original goal of this research to identify universal creativity heuristics. MINSTREL's initial set of TRAMs were developed specifically for the storytelling domain. Indeed, MINSTREL's model of creativity grew out of the desire to build a storytelling model that was capable of telling novel stories. Only later did we note the general nature of many of MINSTREL's creativity heuristics.

TRAM:Limited-Recall, for example, finds new solutions to problems by examining and changing the problem descriptions schema structure, without any reference to the content of the schemas. Similarly, heuristics like TRAM:Recall-Act can apply to any Act schema, not just Act schemas in the storytelling domain. Indeed, TRAM:General-Constraint, TRAM:Similar-Outcomes, and TRAM:Intention-Switch were used to invent methods of suicide as well as to tell stories.

This experience suggested that there might exist a set of creativity heuristics that could apply to any, or at least many, problem domains. To investigate this idea, MINSTREL was modified to

invent mechanical devices. Inventing mechanical devices is a problem domain very different from telling stories. The hope was that a comparison of creativity heuristics from these two differing domains would reveal some general features of creativity. And certain similarities were found.

In the storytelling domain, TRAM:Similar-Outcomes looked for actions that had effects similar to the desired effect, and guessed that recalled actions might also have the desired effect. TRAM:Generalize-Converter looked for devices that did the right type of conversion at a different power level, and guessed that the converter could also be used at the desired power level.

TRAM:Intention-Switch substituted an unintentional action for an intentional action, and then repaired the recalled episode by making the action intentional. TRAM:Power-Converter substituted a converter for a power source and then repaired the recalled converter by adding a power source to it.

The comparison of the TRAMs from these two domains has led to an initial classification of some general types of creativity heuristics: (1) Relaxation, (2) Generalization, (3) Substitution of Similar Sub-Parts, and (4) Planning Knowledge. Section 15.10.3 of this chapter discusses these categories in more detail and classifies MINSTREL's TRAMs according to these categories.

A topic for future research is to explore this classification, and, for a consistent problem representation, attempt to develop creativity heuristics that can apply to a wide range of problem domains.

It is interesting to note, though, that two of the creativity heuristics which were developed and then later abandoned (see Section 15.4) were domain-independent: TRAM:Ignore-Neighbors and TRAM:Remove-Slot-Constraint. Both of these heuristics led to problems in adaptation, suggesting that the problem of solution adaptation may be an important issue in developing domain-independent creativity heuristics.

15.8.9 Summary

There are two conclusions to be drawn from this example.

First, the same creativity architecture was able to invent solutions in two very disparate problem-solving domains: storytelling and mechanical invention. This indicates that the MINSTREL model of creativity is not an ad-hoc solution to a single problem, but a general model that is applicable across different types of problem domains.

Second, similarities between the creativity heuristics for storytelling and the creativity heuristics for mechanical invention suggest that there might be a class of "generic" creativity heuristics. Both storytelling and mechanical invention used TRAMs that generalized constraints and TRAMs that substituted different but similar problems. The parallels between creativity in storytelling and mechanical invention suggest that developing universal creativity heuristics might be

a useful and productive task.

15.9 Experiment #5: Modifying MINSTREL's Planning Algorithm

MINSTREL uses a queue-based planning model. Each author-level goal is given a priority at the time it is created. Goals are kept on a queue, and at every cycle, MINSTREL takes the highest priority goal off the queue and attempts to solve the goal by applying author-level plans. If one of the plans succeeds, the cycle repeats. If none of the plans succeeds, the priority of the failed goal is reduced, it is added back to the queue, and the cycle repeats. This allows MINSTREL to try goals repeatedly, in the hope that some change in the story will allow a goal that failed initially to later succeed.

In this experiment, we modified MINSTREL so that failed goals were never re-queued. This allowed us to study the role that re-queueing goals played in MINSTREL's performance.

15.9.1 Results

After modifying MINSTREL to never re-queue failed goals, MINSTREL was asked to tell a story based on PAT-PRIDE. Surprisingly, MINSTREL told "The Proud Knight" exactly as before. Apparently, re-queueing of goals was not used in the telling of this story. This was a surprising result. Earlier versions of MINSTREL had often solved goals on the second or third attempt.

In light of this result, the traces for stories based on PAT-GOOD-DEEDS-REWARDED, PAT-ROMEO, PAT-PRIDE and PAT-HASTY were analyzed to determine how often MINSTREL had solved re-queued goals. That analysis is shown in Figure 15.40. Figure 15.40 shows the total number of goals that MINSTREL attempted in telling ten different stories, the number of those goals that failed (i.e., were never achieved) and the number of goals that were solved on the first, second, or third attempt.

		Successful Goals		
Total	Failed	1st	2nd	3rd
2466	1262	1191	12	1

Figure 15.40 MINSTREL's Use of Re-Queued Goals

As Figure 15.40 shows, MINSTREL does occasionally solve a goal on the second or third attempt. This occurs when, between the time the failed goal is first tried and the time it is re-tried, an intervening goal succeeds and as a side-effect changes the failed goal. For example, the failed goal might be "instantiate a scene in which a knight does a favor for some other character." A goal operating on different part of the story might decide that the other character is a hermit. And while MINSTREL could not solve the original goal specification, it can solve the modified specification "instantiate a scene in which a knight does a favor for a hermit."

However, as these statistics show, this happens rarely (about 1% of the time). In this respect, MINSTREL is a very efficient planner: of the goals that it can solve, it solves almost all of them immediately. This efficiency is a direct consequence of MINSTREL's creative ability.

MINSTREL's creativity heuristics permit it to make full use of the domain knowledge in episodic memory. MINSTREL's creativity heuristics make many plausible guesses about how the knowledge in episodic memory can be generalized and applied to different situations. If there is any way to apply an episode to a particular goal, MINSTREL's creativity heuristics are likely to discover it. Re-queueing a goal succeeds only when some intervening goal adds information to the original goal that the creativity heuristics could not discover. And the more powerful MINSTREL's creativity heuristics, the less likely this is to happen.

One interesting conclusion we can draw from this is that *a creative problem-solver is more flexible than a non-creative problem solver.* Even if goals are poorly stated or attempted in the wrong order, a creative problem-solver may still be able to find solutions. In turn, makes the creative problem-solver more efficient and successful. Creativity obviously cannot shoulder all of the planning burden. But creativity does make planning a less critical activity.

This also suggests a way to learn creativity. When re-queueing of a goal does result in success, the creative problem-solver should take note of the happening and analyze it. If the problem-solver can determine what information was added that permitted the solution of the previously insoluble goal, that knowledge can be used to invent a new creativity heuristic. Consider the example given above, in which a problem is solvable once a secondary character is given a role. This suggests a new creativity heuristic which modifies a problem specification by adding a role for a secondary character.

So an analysis of MINSTREL's planning activity reveals a number of interesting things. First, MINSTREL is an efficient planner; it solves almost all author-level goals on the first try. Second, creative problem-solvers may expend less effort planning than non-creative problem-solvers. And finally, when planning succeeds after an initial failure, the problem-solver has an opportunity to use this situation to learn a new creativity heuristic.

15.10 Study #2: Analysis of MINSTREL's TRAMs

MINSTREL has a library of twenty-four TRAMs. These TRAMs arose in a number of ways. Some were developed independently of MINSTREL's particular tasks, often by analogy to "creativity techniques" expounded for humans, such as generalization, relaxation of problem constraints, and so on. Other TRAMs were developed in response to particular difficulties in MINSTREL's problem domain. In this section, we characterize MINSTREL's TRAMS by usage, applicability, and function.

15.10.1 Usage of TRAMs

Statistics on how frequently MINSTREL used each TRAM are shown in Figure 15.41. These statistics were gathered on MINSTREL's performance telling stories based on PAT-GOOD-DEEDS-REWARDED, PAT-ROMEO, PAT-HASTY, PAT-PRIDE, and on MINSTREL's performance inventing novel methods of suicide.

TRAM	Number of Uses
TRAM:IGNORE-MOTIVATIONS	62
TRAM:STANDARD-PROBLEM-SOLVING	33
TRAM:GENERALIZE-ACTOR	18
TRAM:SIMILAR-STATES	10
TRAM:RECALL-ACT	9
TRAM:INTENTION-SWITCH-1	6
TRAM:LIMITED-RECALL	5
TRAM:FAVOR-GOAL	4
TRAM:GENERALIZE-ROLE	4
TRAM:SIMILAR-STATE-OUTCOME	4
TRAM:ACHIEVE-B-MOTIVATED-P-GOAL	3
TRAM:GENERALIZE-OBJECT	3
TRAM:SIMILAR-THWART-STATE	3
TRAM:USE-MAGIC	3
TRAM:GENERALIZE-CONSTRAINT	2
TRAM:OPPOSITE-STATE-ACHIEVES	2
TRAM:SIMILAR-OUTCOMES-PARTIAL-CHANGE	2
TRAM:THWART-VIA-ESCAPE	2
TRAM:ACT-OF-A-FAVOR	1
TRAM:CREATE-FAILURE	1
TRAM:CROSS-DOMAIN-REMINDING	1
TRAM:EXIST-AS-PRECOND	1
TRAM:IGNORE-SUBGOAL	1
TRAM:RECALL-WO-PRECOND	1
TRAM:THWART-VIA-DEATH	1

Figure 15.41 Usage of TRAMs

The table in Figure 15.42 has been divided into three rough categories corresponding to frequency of use. TRAMs in the first section were used quite frequently, TRAMs in the second section occasionally, and TRAMs in the last section only in specialized situations. This table reveals three interesting traits of MINSTREL's TRAM usage.

First, TRAM:STANDARD-PROBLEM-SOLVING is one of the most frequently used TRAMs. This is no surprise. Most of the problems encountered by a problem-solver should be very simi-

lar to past problems, and require no creativity. What is more surprising is that creativity heuristics as a whole are used four times *more* frequently than TRAM:STANDARD-PROBLEM-SOLVING. It seems unexpected that creativity should be so important to MINSTREL.

In part this is because MINSTREL was purposely given a limited episodic memory. With a broader range of experience, MINSTREL would be able to solve more problems without creativity. Nonetheless, the surprisingly high usage of creativity heuristics may indicate that low levels of creativity are more in use in human cognition than previously recognized.

A second interesting feature of the table in Figure 15.42 is the extensive usage of TRAM:IGNORE-MOTIVATIONS. TRAM:IGNORE-MOTIVATIONS is a heuristic which ignores motivating states when trying to find a plan to achieve a goal. In more common terms, TRAM:IGNORE-MOTIVATIONS says that when you are trying to solve a goal, it doesn't matter why you have that goal. This is "common sense" to a human problem-solver. The fact that it is applied so frequently and successfully as a creativity technique indicates: (a) that commonly known, frequently used problem-solving techniques may have their origin in the creative process, and (b) that creativity may be involved in day-to-day problem-solving to a greater degree than normally suspected.

Finally, Figure 15.42 shows that most of MINSTREL's creativity heuristics get used infrequently. About 80% of MINSTREL's TRAMs were used 6 times or less in all the stories MINSTREL told, and some were used only once. Why is this?

Creativity by its nature is a rare occurence. The need to be creative arises only when a problem-solver cannot find any other solution to a problem. Successful creativity is rarer; and the use of any particular creativity heuristic rarer still. MINSTREL, which has a purposely sparse episodic memory and heuristics to drive it be creative, uses creativity in only 12% of the goals it solved (150 of 1200), and a human problem-solver is undoubtedly creative much, much less frequently. Given this, it is no surprise that most of MINSTREL's creativity heuristics are used infrequently; that should be expected of any creative problem-solver.

Of course, the infrequency of creativity causes difficulties in the development and testing of a model of creative problem-solving. When a creativity heuristic is used only a few times, it is difficult to determine whether the heuristic is general and useful, or an ad-hoc solution to a particular, narrow problem. Indeed, one of MINSTREL's creativity heuristics - TRAM:CROSS-DOMAIN-REMINDING - was added to MINSTREL to specifically look at a particular kind of creativity, and consequently doesn't have any wide applicability. In general, though, MINSTREL's creativity heuristics appear reasonable. Most have been used in several different problem-solving situations, and have been used in combination with other creativity heuristics and with different author-level plans, all of which give some assurance that these heuristics are not reasonable initial attempts to codify some of the mechanisms of creativity.

Figure 15.42 shows another analysis of MINSTREL's TRAM usage. The first table in Figure 15.42 summarizes each successful use of a TRAM in stories based on PAT-GOOD-DEEDS-REWARDED, PAT-ROMEO, PAT-HASTY and PAT-PRIDE. Unlike the previous table, this table

shows how often two or more TRAMs were used in conjunction to discover a solution. The second table in Figure 15.42 summarizes total successes using one, two, or three TRAMs. (Notice that TRAM:STANDARD-PROBLEM-SOLVING is not included in these tables.)

First Level	Second Level	Third Level	Uses
TRAM:ACHIEVE-B-MOT-P-GOAL	TRAM:THWART-VIA-ESCAPE		1
TRAM:ACHIEVE-B-MOT-P-GOAL	TRAM:THWART-VIA-ESCAPE	TRAM:GENERALIZE-ACTOR	1
TRAM:ACT-OF-A-FAVOR	TRAM:ACHIEVE-B-MOT-P-GOAL	TRAM:THWART-VIA-DEATH	1
TRAM:CREATE-FAILURE			1
TRAM:CROSS-DOMAIN-REMINDING			1
TRAM:EXIST-AS-PRECOND			1
TRAM:FAVOR-GOAL			3
TRAM:FAVOR-GOAL	TRAM:GENERALIZE-ACTOR		1
TRAM:GENERALIZE-ACTOR			5
TRAM:GENERALIZE-ACTOR	TRAM:LIMITED-RECALL		1
TRAM:GENERALIZE-CONSTRAINT	TRAM:SMLR-OUTCOMES	TRAM:INTENT-SWITCH-1	2
TRAM:GENERALIZE-OBJECT			1
TRAM:GENERALIZE-ROLE			2
TRAM:IGNORE-MOTIVATIONS			62
TRAM:IGNORE-SUBGOAL			1
TRAM:INTENT-SWITCH-1	TRAM:USE-MAGIC		1
TRAM:LIMITED-RECALL			3
TRAM:OPPOSITE-STATE-ACHIEVES			1
TRAM:OPPOSITE-STATE-ACHIEVES	TRAM:GENERALIZE-ACTOR	TRAM:LIMITED-RECALL	1
TRAM:RECALL-ACT			2
TRAM:RECALL-ACT	TRAM:GENERALIZE-ACTOR		5
TRAM:RECALL-ACT	TRAM:GENERALIZE-ROLE	TRAM:GENERALIZE-ACTOR	1
TRAM:RECALL-ACT	TRAM:SMLR-STATE-OUTCOME	TRAM:INTENT-SWITCH-1	1
TRAM:RECALL-WO-PRECOND			1
TRAM:SMLR-STATE-OUTCOME	TRAM:INTENT-SWITCH-1		1
TRAM:SMLR-STATE-OUTCOME	TRAM:SMLR-STATE-OUTCOME	TRAM:INTENT-SWITCH-1	1
TRAM:SMLR-STATES			8
TRAM:SMLR-STATES	TRAM:GENERALIZE-OBJECT		2
TRAM:SMLR-THWART-STATE	TRAM:GENERALIZE-ACTOR		2
TRAM:SMLR-THWART-STATE	TRAM:GENERALIZE-ROLE	TRAM:GENERALIZE-ACTOR	1
TRAM:STANDARD-PROBLEM-SOLVING			31
TRAM:USE-MAGIC			2

	Single TRAM	Two TRAMs	Three TRAMs
Successes	124	13	10
Attempts	5345	1448	953
Percentage	2.3	0.9	1.0

Figure 15.42 Nested TRAM Usage

The statistics shown in Figure 15.42 were gathered on 1025 goals MINSTREL achieved telling stories about PAT-PRIDE, PAT-HASTY, PAT-ROMEO and PAT-GOOD-DEEDS-REWARDED (see Section 15.2), indicating that on the average 2.8 TRAMs applied to each goal. With three levels of recursion, MINSTREL applies (on average) 22 TRAMs to each goal that cannot be solved with standard problem-solving. With this level of effort, MINSTREL solves a little over 12% of the goals that were not solvable by standard problem-solving. Timings indicate that creativity adds less than 20% to the run-time of a typical storytelling session. Based on these numbers, is MINSTREL's creativity productive? Is a 12% increase in performance worth a 20% increase in run-time? We argue that it is, for three reasons.

First, creativity is expected to be less efficient than standard problem-solving. Case-based problem-solving is simple and efficient: a solution is recalled and applied to the current problem with little or no adaptation. Creativity, on the other hand, is an extension of problem-solving that is more powerful at the cost of being less efficient. The creativity problem-solver must search memory in inefficient ways, and adapt the solutions it finds. So it is not surprising that a 12% increase in performance has a cost of 20%.

Second, there is no reason to believe that there exists any more efficient method for MINSTREL to gain that additional performance. MINSTREL is very efficient in both (1) searching memory for potential solutions and (2) adapting those solutions. In fact, MINSTREL is surprisingly efficient. MINSTREL applies only 22 TRAMs on average while searching for a problem solution. Were MINSTREL to apply TRAMs by exhaustively choosing from its pool of 24 TRAMs for three levels of recursion, it would apply 24*23*22 = 12,144 TRAM sequences to each problem! [1] MINSTREL's adaptation of discovered solutions is also efficient, because MINSTREL associates specific appropriate adaptation with each search heuristic.

Finally, the 12% success figure is somewhat deceptive. Many of these successes are key to the stories in which they occur. For example, MINSTREL uses creativity to solve only two goals in telling "The Proud King". But without creativity, MINSTREL cannot solve a single goal in telling that story. The two goals that MINSTREL solves creatively contain knowledge that MINSTREL uses to solve every other goal. So while creativity may directly account for only a small percentage of MINSTREL's success, it contributes indirectly to many other successes.

As with human creativity, MINSTREL uses creativity to solve key portions of the problems it is given. Solving the rest of the problem is often straightforward problem-solving. But without the creativity to solve the key portion of the problem, solving the rest of the problem is useless even if possible.

15.10.2 Applicability of TRAMs

In MINSTREL, each TRAM applies to a particular type of representation object, such as an act, a goal, or a state. Figure 15.43 shows how many TRAMs MINSTREL has which apply to each type of representation object.

15.10.3 Functional Distribution of TRAMs

Figure 15.44 shows the functional distribution of MINSTREL's TRAMs into Relaxation, Generalization, Substitution of Similar Sub-Part, Planning Knowledge and Other categories. Each of these categories represents a particular type of creativity heuristic. RELAXATION - Relaxation heuristics attempt to find a new solution to a problem by relaxing or removing some of the origi-

1. This assumes that each TRAM can be applied to a problem only once. This is true for most of MINSTREL's TRAMs, but some can be used repeatedly. If all TRAMs could be used repeatedly, the number of TRAMs applicable to a search depth of 3 would be 24*24*24 = 13,824.

Representation	TRAMs	Uses	TRAMs (Uses)
Any object	4	58	TRAM:STANDARD-PROBLEM-SOLVING (33), TRAM:GENERALIZE-ACTOR (18), TRAM:LIMITED-RECALL (5), TRAM:GENERALIZE-ROLE (4)
Goals	6	74	TRAM:IGNORE-MOTIVATIONS (62), TRAM:FAVOR-GOAL (4), TRAM:ACHIEVE-B-MOTIVATED-P-GOAL (3), TRAM:SIMILAR-THWART-STATE (3), TRAM:OPPOSITE-STATE-ACHIEVES (2).
Acts	9	27	TRAM:RECALL-ACT (9), TRAM:INTENTION-SWITCH-1 (6), TRAM:SIMILAR-STATE-OUTCOME (4), TRAM:USE-MAGIC (3), TRAM:RECALL-WO-PRECOND (1), TRAM:EXIST-AS-PRECOND (1), TRAM:CROSS-DOMAIN-REMINDING (1), TRAM:CREATE-FAILURE (1), TRAM:ACT-OF-A-FAVOR (1).
States	5	16	TRAM:SIMILAR-STATES (10), TRAM:THWART-VIA-ESCAPE (2), TRAM:GENERALIZE-OBJECT (3), TRAM:THWART-VIA-ESCAPE (2), TRAM:THWART-VIA-DEATH (1).

Figure 15.43 Applicability of TRAMs

nal problem constraints. TRAM:IGNORE-MOTIVATIONS is an example of a relaxation heuristic. When MINSTREL is attempting to fill in a character goal, TRAM:IGNORE-MOTIVATIONS removes from the problem description any motivating states. This permits MINSTREL to use information from a similar past goal, even if that goal was motivated by a different situation.

GENERALIZATION - Generalization heuristics attempt to find a new solution to a problem by changing the problem into a more general problem, and then adapting any general solutions found to the specific original problem. TRAM:GENERALIZE-ACTOR does this by generalizing the actor in an action, goal or state. This permits MINSTREL to generalize knowledge across character types: what MINSTREL knows about monsters can be used to augment what MINSTREL knows about knights, and so on.

SUBSTITUTION OF A SIMILAR SUB-PART - These type of heuristics attempt to find a solution by replacing a sub-part of the original problem with a similar but different part. This is a specific type of analogical reasoning which guesses that because two things share some similarities they may also share other similarities. For example, TRAM:SIMILAR-STATES modifies

Functional Category	TRAMS
Relaxation	TRAM:IGNORE-MOTIVATIONS TRAM:RECALL-WO-PRECOND TRAM:LIMITED-RECALL TRAM:RECALL-ACT
Generalization	TRAM:GENERALIZE-ACTOR TRAM:GENERALIZE-OBJECT TRAM:GENERALIZE-CONSTRAINT TRAM:GENERALIZE-ROLE TRAM:IGNORE-SUBGOAL
Substitution of a Similar Sub-Part	TRAM:SIMILAR-STATES TRAM:SIMILAR-STATE-OUTCOME TRAM:INTENTION-SWITCH-1 TRAM:SIMILAR-THWART-STATE TRAM:OPPOSITE-STATE-ACHIEVES TRAM:CREATE-FAILURE TRAM:SIMILAR-OUTCOMES
Planning Knowledge	TRAM:ACHIEVE-B-MOTIVATED-P-GOAL TRAM:THWART-VIA-ESCAPE TRAM:THWART-VIA-DEATH TRAM:EXIST-AS-PRECOND TRAM:ACT-OF-A-FAVOR
Other	TRAM:FAVOR-GOAL TRAM:USE-MAGIC TRAM:CROSS-DOMAIN-REMINDING

Figure 15.44 Functional Distribution of TRAMs

problem descriptions which contain states by replacing the states with other, similar states. This permits MINSTREL to generalize about the results of actions. If MINSTREL is trying to find an action which results in a knight moving to a castle, TRAM:SIMILAR-STATES lets MINSTREL guess that an action which moves a knight to the woods might also move him to a castle.

PLANNING KNOWLEDGE - Planning knowledge heuristics use specific knowledge about planning to discover new problem solutions. For instance, TRAM:THWART-VIA-DEATH knows that a goal will be thwarted if the actor of that goal is killed. This specific knowledge about planning can be used to find problem solutions that would otherwise be undiscovered.

OTHER - Three of MINSTREL's creativity heuristics do not easily fit into the previous categories. TRAM:FAVOR-GOAL finds a solution to the goal to do someone a favor by finding a solution to the goal that the favor goal is subsuming. This is a syntactic modification similar to the

one used by the (failed) TRAM:SWITCH-FOCUS. TRAM:USE-MAGIC is a creativity heuristic specific to the King Arthur domain which suggests that magic artifacts can be used to achieve almost any goal. And finally, TRAM:CROSS-DOMAIN-REMINDING is an analogy heuristic that looks for a problem solution by translating a problem description from the King Arthur domain to another domain. Currently, MINSTREL knows only of one other domain - modern life - and of only one episode in that domain, so consequently TRAM:CROSS-DOMAIN-REMINDING is usable in very few situations.

As Figure 15.44 shows, many of MINSTREL's creativity heuristics fall into categories often described in creativity literature: generalization, relaxation and analogy. The surprise here is the number of heuristics which use planning knowledge. Why is a technique of obvious use to MINSTREL not broadly acknowledged as a creativity technique?

The lack of mention of creativity techniques based on planning knowledge may be because creativity literature is written both for and about an accomplished, adult audience. As the high usage of TRAM:IGNORE-MOTIVATIONS suggests, human problem-solvers may be unaware of their extensive use of some creativity heuristics, or such techniques may be so widely and frequently used that they are no longer considered "creative". In particular we would expect this to be true for planning knowledge heuristics. By the time of adulthood, humans are such accomplished planners that simple heuristics based on planning knowledge should be so practiced as to be unnoticeable. Where human problem-solvers can be expected to have difficulty is in using infrequent creativity heuristics, such as those that fall into the "specialized situation" category of MINSTREL's TRAM usage chart. In addition, adult human problem-solvers have such a wide variety of problem-solving situations in memory that many problem knowledge heuristics may be unnecessary. This suggests that some of MINSTREL's creativity heuristics may be primarily useful to a planner working in a new problem domain.

Secondly, most creativity work to date has focused on acts of outstanding creativity, such as great artistic works and major scientific discoveries. The model presented in this work explains creativity as a logical extension to problem-solving, spanning the spectrum from simple, everyday problem-solving to works of outstanding creativity. MINSTREL's planning knowledge heuristics are used primarily to discover slightly different solutions near the problem-solving end of this spectrum. For example, TRAM:THWART-VIA-DEATH is used to discover that being killed can thwart a knight's goal to love someone. This is not outstandingly creative, but because MINSTREL knows nothing of such a concept, neither is it mere rote recall. Because creativity research has previously focused on acts of outstanding creativity, it has overlooked the role of such "day-to-day" creativity in problem-solving.

15.11 Experiment #6: TRAMs and Creativity

This experiment looks at the robustness of MINSTREL's creativity model. In particular, we look at how MINSTREL's performance is affected by the loss of TRAMs and of creativity in general.

15.11.1 Loss of TRAMs

As a first experiment, random TRAMs were removed from MINSTREL's library of creativity heuristics. Between 10% and 50% of MINSTREL's TRAMs were removed, and then MINSTREL was given the goal of telling a story about one of themes it knew, chosen at random. This experiment was performed a number of times in order to develop a general understanding of MINSTREL's performance in degraded modes.

Figure 15.45 shows a typical story told in one of these experiments. This story is based on PAT-PRIDE and was told by MINSTREL with the following TRAMs removed from MINSTREL's pool of creativity heuristics: TRAM:SIMILAR-STATES and TRAM:GENERALIZE-OBJECT. A comparison of this story with the original story (given in Section 2 of this chapter) will reveal that this story is complete except for a missing scene in which Grunfeld travels to the dragon.

The Proud Knight II

It was the Spring of 1089, and a knight named Grunfeld returned to Camelot.

A hermit named Bebe told Grunfeld that Bebe believed that if Grunfeld fought with the dragon then he would be hurt.

Grunfeld was very proud. Because he was very proud, Grunfeld wanted to impress the King. Grunfeld was near a dragon. The dragon was destroyed because Grunfeld fought with the dragon. The dragon was destroyed but Grunfeld was hurt. Grunfeld wanted to protect his health. Grunfeld wanted to be healed. Grunfeld hated himself. Grunfeld became a hermit.

Moral: Pride goes before a fall.

Figure 15.45 The Proud Knight II

This example is typical of the results of this experiment. Removing TRAMs from MINSTREL generally caused MINSTREL to fail to complete one or more small parts of the stories it told. Because MINSTREL's stories have many sub-parts, because the stories are created by many interacting goals, and because any particular task involves only a few TRAMs, the loss of particular TRAMs usually caused only minor, localized failures in storytelling.

One exception to this did occur. Because MINSTREL has no episodic memories concerning kings, MINSTREL's ability to tell any story about a king turns on its use of TRAM:GENERALIZE-ACTOR to guess that kings are like knights. Figure 15.46 shows the result of telling the PAT-PRIDE story about a king if TRAM:GENERALIZE-ACTOR is not available.

"The Proud King II" is a critical failure of MINSTREL's creativity, but there were also situa-

The Proud King II

It was the Spring of 1089, and a King Arthur returned to Camelot from elsewhere.

A hermit named Bebe warned Arthur.

Arthur became a hermit.

Figure 15.46 The Proud King II

tions where MINSTREL achieved a critical success by substituting for a missing TRAM. In the story "The Knight and the Hermit" based upon PAT-GOOD-DEEDS-REWARDED, MINSTREL uses TRAM:THWART-VIA-ESCAPE to create a scene where a hermit tries to thwart a dragon's goal of eating him by escaping (the attempt fails):

> *...Bebe believed he would die because he saw a dragon moving towards him and believed that it would eat him. Bebe tried to run away but failed.*

If TRAM:THWART-VIA-ESCAPE is removed from MINSTREL's available creativity heuristics, MINSTREL will invent a different method of thwarting the dragon's goal by using TRAM:THWART-VIA-DEATH:

> *...Bebe believed he would die because he saw a dragon moving towards him and believed that it would eat him. Bebe tried to kill the dragon with his hands but failed.*

In general we conclude that because MINSTREL has a large library of TRAMs, because problem-solving goals are typically localized, and because MINSTREL can sometimes discover alternate solutions, MINSTREL's performance degrades gracefully as TRAMs are lost. Sometimes, a single TRAM is critical to an important decision, and loss of that TRAM will consequently cause a major failure.

15.12 Study #3: MINSTREL's Common Failure Modes

In describing MINSTREL's attempts to tell a story about PAT:PRIDE, we noted one of MINSTREL's common failure modes. This section further discusses that and other common failure modes. By looking at how MINSTREL fails, we hope to cast some light on the creative process and discover new areas for research.

15.12.1 Recovery From Bad Decisions

In its standard configuration, if MINSTREL is asked to tell a story about PAT:PRIDE, it chooses to make the main character a hermit. As it happens, this is a bad decision. PAT:PRIDE requires the main character to perform an action that has unintended bad consequences, but MINSTREL cannot recall or invent such an action for a hermit. Consequently storytelling fails. The incomplete story is shown in Figure 15.47.

The Proud Hermit

Once upon a time there was a hermit. Someone warned him not to pick berries, but he went ahead and picked them anyway. Nothing happened.

Figure 15.47 Failed PAT:PRIDE Story

Being unable to continue a story because of a bad decision earlier in storytelling is a common problem for MINSTREL. MINSTREL has a very simple planning model, and no effort was made to give MINSTREL the ability to recover from bad planning decisions. So when MINSTREL makes a bad decision that prevents it from successfully completing a story, it cannot recover from that decision. To recover from bad decisions, three facilities would be needed.

First, MINSTREL needs the ability to detect when storytelling fails. This is easily implemented. By checking the status of its important storytelling goals (primarily whether the theme has been instantiated) MINSTREL can determine how successful storytelling has been. MINSTREL can even make simple judgements about whether a story has been told well or in a minimal fashion, by checking the status of its author-level writing goals.

Second, MINSTREL also needs to be able to trace storytelling failure to a bad decision. This is a more difficult task for several reasons. The first difficulty is that MINSTREL does not maintain a history or memory of the storytelling process itself, and so has no way to examine what it has done in the past. The second difficulty is in determining which of the many decisions involved in the storytelling process is the critical bad decision. This is very much an open research question. One can imagine various simple heuristics (such as finding the last successful goal previous to a long string of goal failures) but the efficacy of such heuristics is unknown.

Finally, MINSTREL needs to be able to retract a bad decision. The simplest method for achieving this is to throw out the entire unsuccessful story and start anew, avoiding the bad decision. But that's potentially very wasteful. A better method would be to recover from a bad decision by repairing the failed story. MINSTREL does not currently have any plan repair abilities, but for a comparison of MINSTREL's creative process to the plan repair process, see Chapter 7.

MINSTREL's inability to recover from bad planning decisions is a consequence of MINSTREL's simple planning model. Were MINSTREL able to backtrack away from failed solutions, explore multiple solutions in parallel, or detect and repair bad decisions as outlined above, it would avoid

most failures of this type. This type of failure reflects more upon MINSTREL's planning model than its model of creativity.

15.12.2 Errors of Knowledge

A second type of failure that MINSTREL encounters is due to inadequacies in MINSTREL's knowledge of the problem domain. An interesting example of this occurs when MINSTREL is asked to tell a story about PAT:PRIDE in which the main character is a princess. As with the hermit story above, MINSTREL is unable to complete its storytelling. The failed story is given in Figure 15.48.

The Proud Princess

Jennifer was a lady of the court. One day a hermit named Bebe told Jennifer that Bebe believed if Jennifer drank a magic potion something bad would happen.

Jennifer was very proud. Jennifer drank the magic potion and appeared to be dead. Jennifer was buried in the woods.

Figure 15.48 Failed PAT:PRIDE II

This story is particularly interesting because it *appears* to fulfill the theme. The hermit predicts that something bad will happen if Jennifer drinks the potion and it does: She gets buried in the woods. But this is misleading.

The bad consequence of drinking the magic potion – being buried in the woods – is actually created by MINSTREL to resolve Jennifer's fate, not to resolve the theme. One of MINSTREL's author-level writing goals is to resolve loose ends at the end of a story by determining the fate of the characters in the story. Dead characters are buried and mourned, characters who have brought tragedy on themselves or others may change their roles, and so on. In this case, MINSTREL mistakenly believes that Jennifer is dead at the conclusion of the story, and buries her as a way of resolving her fate. This error occurs because MINSTREL's author-level plan that achieves the goal of resolving character fates lacks the knowledge to correctly interpret MINSTREL's representation of deceptive states.

MINSTREL represents deception by two states of the world. The true state of the world is "shadowed" by the deceptive state. In this case, Jennifer's good health is shadowed by her death. A graphic representation of this is shown in Figure 15.49.

However, MINSTREL's author-level plan for determining a character's fate does not have the knowledge to correctly interpret shadowed states. When this plan examines the story, it notices that Jennifer is dead, without noticing that Jennifer's death is in fact a deception state. Consequently, the author-level plan assumes that Jennifer is truly dead and decides her fate on that ba-

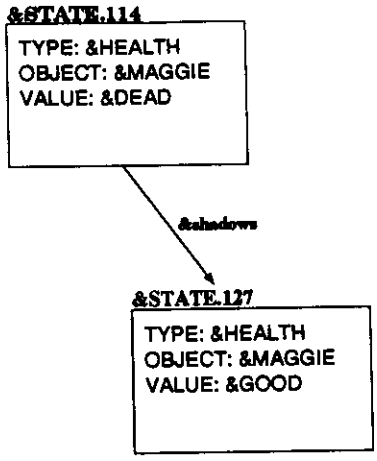


Figure 15.49 Representation of Deception

sis.

Of course, it would be interesting if MINSTREL had intentionally reasoned that Jennifer's deception might result in a premature burial. But in this case, Jennifer's burial is an error that occurred because MINSTREL's knowledge of the problem domain was incomplete.

This type of error does not indicate any shortcoming of MINSTREL's planning or creativity models. Errors of knowledge can be made by the most sophisticated planners and problem-solvers when their knowledge of a problem domain is incomplete. MINSTREL has a very rudimentary knowledge of the King Arthur and storytelling problem domains, and can be expected to make errors of knowledge.

15.12.3 Creativity Errors

A final type of error that can occur in MINSTREL is a creativity error. The philosophy of MINSTREL's creativity heuristics is to make small, incremental changes. Many of these heuristics will never make an error. For example, TRAM:INTENTION-SWITCH-1 changes intended actions into unintended actions. That's a problem modification that is always reasonable. But many of MINSTREL's creativity heuristics are guesses about what might be "reasonable" about the world. And sometimes these guesses are wrong.

One error of this type occurs when MINSTREL tries to solve a knight's goal of feeding himself. Because MINSTREL does not have any episodes in memory in which a knight feeds himself, TRAM:GENERALIZE-ACTOR is applied. This TRAM generalizes the knight to a monster, because both knights and monsters are violent characters. This recalls a scene in which a dragon captures a princess and eats her to assuage its hunger. When applied to this problem, the result is

a scene in which a knight kills and eats a princess. Oops!

This particular error occurs because MINSTREL has a simple-minded understanding of the similarities between knights and monsters, and modifying MINSTREL to avoid it would be trivial. But it is indicative of the types of errors that creativity heuristics can introduce.

The importance of these types of errors are that they validate MINSTREL's creativity model. A creative problem-solver *should* make errors, because being creative requires the problem-solver to extend the limits of his knowledge. By making good use of what he already knows, a problem-solver can often make accurate guesses about what he doesn't know. But sometimes he will err. One of the challenges of creativity research is to find a model of creativity which limits errors while still being powerful enough to meaningfully extend the limits of a problem-solver's knowledge. MINSTREL's creativity model, which incrementally extends the problem-solver's knowledge in small steps, has this characteristic. By making small changes near the border of its knowledge, MINSTREL is assured that what it invents is likely to be reasonable. But because MINSTREL can apply many creativity heuristics to a single problem, it also has the power to make bigger leaps when necessary.

CHAPTER 16
Future Work and Conclusions

16.1 Future Work

MINSTREL demonstrates many of the talents of a human author: creativity, use of themes, literary skill and use of language. But MINSTREL's competence in these areas is not equal to that of an accomplished human author. In this section we examine how MINSTREL's abilities differ from those of a human author and speculate on how MINSTREL could be extended and improved.

MINSTREL's shortcomings can be broadly characterized as *quantitative* or *qualitative*.

Quantitative shortcomings arise because MINSTREL is a prototype and necessarily lacks the depth of knowledge that a human author possesses. For example, MINSTREL cannot tell stories set in modern-day New York City because it knows only the King Arthur domain. But extending MINSTREL to tell stories in the New York City domain would not require any fundamental changes in MINSTREL's structure or processes, only the addition of knowledge about the modern-day domain to MINSTREL's episodic memory. In this sense, quantitative shortcomings can be remedied by adding "more of the same".

More interesting are qualititative shortcomings. Qualititative shortcomings cannot be remedied without fundamental changes to the MINSTREL's knowledge and process structure. For example, extending MINSTREL to create humorous stories would require adding new types of knowledge (about what makes something funny) and possibly new cognitive processes (i.e., a joke creator). Qualitative shortcomings identify where MINSTREL's model of the storytelling process is incomplete, and suggest areas of future work.

In the following sections we will examine some of MINSTREL's qualititative shortcomings, and where possible, suggest how these problems might be addressed. To help identify MINSTREL's shortcomings we find it useful to compare MINSTREL to a human author, Saki. Saki is well known for his finely crafted and engaging stories. Although Saki's stories are not set in the King Arthur domain, they are in many ways like MINSTREL's stories. They are short, focus on character actions, and often have "morals".

The particular story we will focus on is "Tobermory" [Saki 1982]. In this story, an inventive scientist teaches a cat to speak. He discloses his achievement to a diverse group of people spending a weekend in the country at the mansion where the educated cat resides. At first the others are incredulous, but soon the cat appears and demonstrates not only the ability of speech, but a sharp wit as well. Astonishment soon turns to worry, however, as the cat coldly repeats several embarassing conversations. Before anyone can take action, Tobermory dashes out the window in pursuit of a stray tom.

The guest spend an uneasy night awaiting the return of Tobermory, having decided to poison the

cat. The mood is strained as all ponder what the cat might reveal. In the morning Tobermory is discovered dead of wounds suffered in the battle with the tom. The guest sigh relief and, denouncing the scientist on their way, leave for the city.

16.1.1 Theme and Analogy in "Tobermory"

Like many authors, Saki weaves several themes into "Tobermory." The unexpected problems that arise from Tobermory's gift of speech illustrate that an initially appealing idea may have unwanted consequences. The conversations that Tobermory reveals illustrate that people can be cruel and deceptive. And like many authors who give animals human characteristics, Saki uses "Tobermory" to reveal the hypocrisy of man's "civilized" nature.

Saki's treatment of theme in "Tobermory" is more sophisticated than MINSTREL's in several ways.

First, Saki is able to use the events of "Tobermory" to illustrate simultaneously several different but related themes. MINSTREL lacks the knowledge needed to recognize compatible themes, and does not have the author-level plans required to create a story which illustrates more than one theme.

Second, Saki employs subtle themes that are not directly concerned with planning errors in the manner of MINSTREL's themes. Saki's theme about the hypocrisy of man's "civilized" behavior cannot be expressed as a Planning Advice Theme, nor would MINSTREL be able to create events to illustrate this theme.

Third, Saki uses analogy to illustrate his themes, by contrasting and comparing the behaviors of the cat and the guests. MINSTREL has no knowledge of analogy or of how it can be used to illustrate a point.

Finally, we may speculate on how Saki's development of a theme differs from MINSTREL's. It seems reasonable to suppose that "Tobermory" developed out of the author's speculations on the consequences of a talking cat ("If the walls could talk..."). This process:

what if → consequences → theme → story

is a common developmental process in science fiction ("[Sprague De Camp] has often conceived a story first in the form of a problem: What if something happened to cause all mankind to grow long hair all over, like monkeys?", pp. 98, [De Camp 1975]), where many stories are overtly concerned with exploring the consequences of some technological development. As with Tobermory, a technological development is cited ("Animals can be taught to speak"), consquences are deduced ("They may reveal what they've seen and heard"), and then a theme is selected which will fit these already-determined story elements. In contrast, MINSTREL first selects a theme, and then develops all other story elements to fit the theme. It is likely that an accomplished author needs the flexibility to adapt his theme to his story and vice versa, but MINSTREL does not

have this sophistication.

To equal human competency in the use of themes, MINSTREL would require much more knowledge about the nature and use of themes, including:

- New types of themes.
- How themes inter-relate.
- How themes can be combined.
- How themes can be invented.
- New methods for illustrating themes (i.e., analogy).
- New methods for selecting themes.

In general, MINSTREL's knowledge of themes is limited to themes based on planning advice. To achieve human competency, MINSTREL would have to understand many types of themes and different methods of selecting and illustrating those themes.

16.1.2 Humor

An area of storytelling in which MINSTREL has no ability is humor.[1] And humor is a particularly interesting topic because it is often a conundrum even to the people who create it. How might MINSTREL be extended to create humorous stories?

Humor comes in many different forms. It is, to a large extent, a mystery even to those who practice it. And humor plays a large part in authoring, even in stories that are not expressly comic.

The final paragraphs of "Tobermory" deal with the scientist's eventual fate (Saki, like MINSTREL, resolves the fates of his important characters before ending a story) and reveals Saki's sense of humor:

> *Tobermory had been Appin's one successful pupil, and he was destined to have no successor. A few weeks later an elephant in the Dresden Zoological Garden, which had shown no previous signs of irritability, broke loose and killed an Englishman who had apparently been teasing it. The victim's name was variously reported in the papers as Oppin and Eppelin, but his first name was faithfully rendered Cornelius.*
>
> *"If he was trying German irregular verbs on the poor beast," said Clovis, "he deserved*

1. Unintentional humor is a separate problem. Since we are most interested in the idea of an author who understands the humor he creates, we do not address this issue.

what he got." (pp. 115, [Saki 1982])

The humor in this passage arises from incongruity. To the reader, conjugating German verbs is a difficult task. But compared to the difficulty of teaching an elephant to speak, it seems trivial. Saki plays on this incongruity when he portrays an elephant as being so peeved at the task of learning German verbs that it kills its trainer. The reader chuckles at the miracle of teaching an elephant to speak being overshadowed by the difficulty of German verbs.

Incongruity is but one element of humor. Another type of humor is *wit*. Wit employs language in sophisticated and clever ways to emphasize the speaker's intelligence and by contrast, to demean the person to whom or of whom he speaks. The best wit employs a wry humor to make its point. In "Tobermory," the cat employs a sharp and laconical wit when addressing its "superiors":

"What do you think of human intelligence?" asked Mavis Pellington lamely.

"Of whose intelligence in particular?" asked Tobermory coldly.

Saki's use of wit in "Tobermory" is particularly clever, because he uses it to support the theme of the story. The contrast between Tobermory's clever repartee and the guests' simple and awkward questions is a device that Saki uses to contrast the civility and superiority of the cat and the guests.

Related to humor is irony. Irony uses incongruity to express an idea in a clever and forceful way by stating its opposite, as when a person exclaims "Beautiful weather, isn't it?" in the midst of a sudden downpour. Irony also refers to the feeling of incongruity the reader experiences when events have an outcome opposite to what was, or might have been, expected. Understanding irony requires knowledge of reader beliefs and reasoning about how those beliefs conflict with the beliefs implicitly represented in a story [Reeves 1991].

Humor is complicated, poorly-understood phenomenon. As these simple examples have indicated, creating or understanding humor requires a wide variety of knowledge including:

- Beliefs
- Incongruity
- Contradictions
- Language

and much more. MINSTREL has no model of humor, and none of this knowledge, and consequently cannot intentionally create a humorous story.

16.1.3 Creativity

MINSTREL's model of creativity explains several of the important issues in creativity: how appropriate old knowledge is found, how it is combined and changed to form new solutions, and how great leaps of creativity can be achieved in many small steps. However, there are areas of human creativity that MINSTREL does not try to explain.

One area is non-directed creativity, or speculation. Earlier we suggested an alternative approach to developing a story theme in which the author posited some development ("Cats can talk") and then extrapolated from that idea until he found or invented a theme. In this type of speculation, creativity interacts with problem-solving in a non-goal-directed manner. In the MINSTREL model, creativity is at the service of problem-solving. It is always directed towards solving a particular goal (or in the case of imaginative memory, recalling something to fit a particular specification). Consequently, MINSTREL is not capable of this type of open-ended speculation.

MINSTREL also lacks the ability to let creativity direct goals. During the course of creativity, a human may discover or invent something that causes him to change his goals or adopt new goals. For example, we might imagine an author who begins telling a story and invents the notion of a talking cat to fulfill some minor storytelling need. This in turn suggests to the author a new theme attacking man's view of himself as superior to the animals, and a new basis for a story. Creativity has changed the author's goals and his evaluations of the importance of those goals. What was a minor storytelling goal has become an important goal for a new story.

DAYDREAMER [Mueller 1989] and AM [Lenat 1976] both examined the problem of exploratory or non-goal-directed creativity. Both of these systems employed a control structure which was directed by *interestingness*. In each system, when an interesting idea arose, the system could react by pursuing the new idea. In contrast, MINSTREL has no notion of whether or not the goals it is pursuing or the story scenes it is creating are interesting. Consequently MINSTREL cannot speculate or change its goals in response to its creativity.

The comparison between these systems and MINSTREL suggests a hybrid control structure in which creativity is normally at the service of problem-solving, but which can be "interrupted" when an unexpectedly interesting idea is invented. How this might be implemented and whether or not MINSTREL's TRAMs would be suitable for open-ended creativity are open questions.

16.2 Conclusions

Storytelling and creativity are broad topics. Creating a computer model that can tell stories has required addressing many issues from a variety of disciplines: cognitive science, psychology, computer science and art. In the preceding pages we have examined a large number of disparate and seemingly unconnected topics: memory, creativity, themes, the artistic drive, foreshadowing, and so on. And we have looked at each issue from a variety of angles: as a cognitive model, as a computational model, as a psychological model, as it interrelates with other issues, as a part in a complex system, and as part of a storytelling machine.

Because of this wide variety of issues, this research is of interest to many different types of scientists:

- **Cognitive Scientists**: MINSTREL defines a concrete, operative model of creativity and storytelling that can be studied for insights into how these processes can be modelled.

- **Psychologists**: MINSTREL is based upon current psychological understanding of human creativity. It provides a possible explanation for human creativity that can be used as a tool for exploring human psychology.

- **Computer Scientists**: The MINSTREL creativity model suggests a basis for implementing computer systems with new levels of flexibility and robustness.

- **Artists**: MINSTREL defines certain artistic concepts - the artistic drive, creativity - in precise, process-oriented ways. The understanding of these processes could lead artists to new self-awareness and increased productivity, as well as new ways to teach and execute art.

Neither scientists nor artists will find a complete answer to their questions about creativity and storytelling in this dissertation. There is still a long way to go before computers will be built that can compete with human authors. But MINSTREL does make many important contributions to understanding these topics. In this section, we will try to pull together the issues that MINSTREL addresses and summarize MINSTREL's most important conclusions.

16.2.1 Creativity

In the introduction to this dissertation we identified a number of challenges that a model of creativity faced. We return now to those challenges and examine how MINSTREL has addressed them.

The Challenge	*The Answer*
Explain the relation of creativity to ordinary problem-solving.	**Creativity is an integral part of the problem-solving process.**

One of the principal questions a researcher must face about creativity is: Is it understandable? Great works of creativity do seem almost miraculous, and it is tempting to deduce from these that creativity is a mysterious process beyond our comprehension. But this ignores the day-to-day creativity that each of us exhibit in our own lives. MINSTREL provides a model of creativity not as something entirely different from problem-solving, but as an extension to problem-solving that can explain both mundane, non-creative problem-solving, day-to-day creativity, artistic creativity, and great works of creativity. MINSTREL provides an explanation of creativity as a proces that can be understood, analyzed, and studied.

The Challenge	*The Answer*
Explain how a problem-solver can find and adapt old knowledge to create new solutions.	**The fundamental process of creativity is integrated search and adaptation.**

One the challenges a model of creativity faces is to explain how a creative problem-solver can find useful knowledge to apply towards creating a new problem solution while still adapting the knowledge he finds in an efficient way. How does one know what old knowledge can form new solutions? And how can one know what combinations will lead to something useful? By combining the search for knowledge with process of adapting that knowledge once found, the MINSTREL model of creativity explains how the creative problem-solver can find useful knowledge and apply it without being overwhelmed by an intractable adaptation problem. MINSTREL's Transform-Recall-Adapt heuristics discover useful knowledge by transforming the problem description, and eliminate the adaptation problem by bundling a specific solution adaptation with each problem transformation.

The Challenge	*The Answer*
Explain great leaps of creativity.	**Leaps of creativity are composed of many small steps.**

One of the fundamental concerns of creativity is whether a continuum exists from non-creative problem-solving to great leaps of creativity. Is it necessary to posit a different process to explain great leaps of creativity? MINSTREL's claim is that great leaps of creativity can be explained as many small steps. Although each of MINSTREL's creativity heuristics makes only a small change in discovering a new problem solution, MINSTREL can combine many of these heuristics to make great leaps of creativity.

The Challenge	*The Answer*
Explain how people can be creative in a variety of problem domains and cognitive processes.	**The integration of creativity with memory makes creativity transparently available to all cognitive processes.**

MINSTREL's integration of creativity with the process of recall to create an *imaginative memory* explains how creativity can be a domain-independent cognitive process. All cognitive process must rely upon memory to store and recall the knowledge they use to understand and solve problems. If the memory itself is capable of inventing new knowledge, then creativity becomes transparently available to all cognitive processes.

16.2.1.1 Storytelling

Creativity is a fundamental cognitive process. Consequently, MINSTREL's important contributions about creativity concern the process model of creativity: how it achieves creativity, how it integrates with other cognitive processes, and how it explains human creative behavior.

In contrast, storytelling is a problem task which makes use of cognitive processes to achieve a goal. More so than with creativity, MINSTREL's contributions to the area of storytelling focus on the specifics of this problem task: what knowledge is needed, what processes are used, and what the specific difficulties are in solving the storytelling problem.

Storytelling is problem-solving.

A major thesis of this research is that there exists no difference at the process level between artistic problem domains and non-artistic problem domains. Because creativity is integrated into the problem-solving process and because the artistic drive can be modelled as a domain assessment, MINSTREL can use the same processes to solve problems in the artistic domain of storytelling that it uses to solve problems in the domains of planning and mechanical invention. Art can be understood in terms of the same cognitive processes that explain problem-solving in domains like science and day-to-day living.

Stories are created through the interaction of many simple author-level goals.

As MINSTREL demonstrates, interesting, complex and cohesive stories can emerge from the interactions of many, small, simple author-level goals. No particular author-level goal or plan in MINSTREL understands a story in its entirety. Each author-level plan is a simple, easily understood method for achieving a simple and narrow problem. Yet together these goals solve a very broad and complex problem: creating a story. Both by this behavior and by the use of many simple creativity heuristics to make great leaps of creativity, MINSTREL demonstrates that complex, difficult-to-understand phenomenon can often be understood as emergent behaviors of many interacting simpler processes.

Important goals for storytelling include: theme, consistency, drama and presentation.

For the types of stories which MINSTREL tells, four classes of author-level goals emerged as essential to the storytelling problem. Whether the particular author-level goals and plans MINSTREL uses have applicability to other types of storytelling is an open question. But the classes of author-level goals MINSTREL identifies and their interactions with one another to create a story are the basis for an understanding of the storytelling process.

Instantiation is a fundamental storytelling process.

A final, unexpected result of MINSTREL was the importance of instantiation to the storytelling process. Instantiation – the process of taking an incomplete description and filling out its details – surprised us by emerging as an important process in nearly every step of the storytelling process. A further surprise was that instantiation could often be achieved by imaginative memory. The importance of instantiation to storytelling and its relationship to imaginative memory argue that instantiation may be a fundamental process to artistic problem domains or perhaps all types of problem-solving.

16.2.2 Final Retrospection

What stands out most in my experiences creating MINSTREL?

The first impression is of overwhelming complexity. To tell even a simple story, MINSTREL must make intelligent, creative use of an enormous amount of knowledge. MINSTREL knows about knights, horses, princesses, places in the world, swordfights, jousts, monsters, love, revenge, anger, death, magic and a host of other facts about King Arthur's knights. On top of that, MINSTREL knows about stories: themes, foreshadowing, characterization, story structure, paragraphing, consistency, language. And to make use of all this MINSTREL implements a variety of cognitive processes: memory, planning, problem-solving, creativity. The complexity of the storytelling task is staggering.

And yet paradoxically the second impression is of elegant simplicity. Case-based problem-solving is a simple idea that answers a number of difficult questions about how people solve problems. TRAMs encapsulate creativity in an understandable way and answer the challenge of inventing new knowledge without making endless mistakes or being overwhelmed by complexity. Imaginative memory provides an elegant integration of creativity into the larger cognitive model. The problems MINSTREL addresses are complex, but the answers it finds are simple.

To me this paradox is oddly comforting. I believe that intelligence must, in the final analysis, be based upon simple but powerful mechanisms. Although the mechanisms that MINSTREL proposes may someday be shown to be totally incorrect, they are comforting because they show that complex behaviors can arise from simple mechanisms, and therefore give hope that someday we can understand intelligence, and hence ourselves, a little bit better.

References

Adams, J.L., *Conceptual Blockbusting: A Guide to Better Ideas.* Reading, MA: Addison-Wesley, 1986.

Alvarado, Sergio, *Understanding Editorial Text: A Computer Model of Argument Comprehension.* (Technical Report UCLA-AI-89-07) Artificial Intelligence Laboratory, Computer Science Department, University of California, Los Angeles, June 1989.

Anderson, J.R., *Cognitive Psychology and Its Implications*, Freeman, New York, 1985.

Arens, Yigal, *Cluster: An Approach to Contextual Language Understanding*, University of California Berkeley, Report No. UCB/CSD 86/293 (Ph.D. dissertation) 1986.

Bartlett, F.C., *Remembering: A study in experimental and social psychology.* Cambridge: Cambridge University Press, 1932.

Becker, J.D., The phrasal lexicon. In *Proceedings of the Interdisciplinary Workshop on Theoretical Issues in Natural Language Processing (TINLAP-1)*, Cambridge, MA., 1975.

Bickham, Jack, "Step 6: Revising Your Story", *The Writer's Digest*, June 1992.

Bresnan, J. and Kaplan, R.M. Lecical-functional grammar: A formal system for grammatical representation. In Bresnan, J., editor, *The Mental Representation of Grammatical Relations.* Cambridge University Press, Cambridge, England, 1982.

Burnham, C.A., and Davis, K.G. (1969). *The 9-dot problem: beyond perceptual organization.* Psychonomic Science, 17, 321-323.

Callahan, Steven, *Adrift, Seventy-Six Days Lost at Sea*, Ballantine, 1986.

Cooley, L.F. *Imaginology*. Englewood Cliffs, NJ: Prentice-Hall 1984.

Dauw, Dean C. and Fredian, Alan J., *Creativity and Innovation in Organizations.* Kendall/Hunt Publishing, Dubuque, Iowa, 1971.

De Bono, E. (1968). *New think.* New York: Basic Books.

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.

Dehn, Natalie, "Story Generation After TALE-SPIN", in Proceedings of the 7th International Joint Conference on Artificial Intelligence, University of British Columbia, Vancouver, Canada, 1981, pp. 16-18.

Dehn, Natalie, "The Significance of Dynamic Memory to Creativity", in *Proceedings of the First Annual Workshop on Theoretical Issues in Conceptual Information Processing*, Atlanta, Georgia, 1984.

Dolan, Charles P., *Tensor Manipulation Networks: Connectionist and Symbolic Approaches to Comprehension, Learning and Planning*, Ph.D. Dissertation, UCLA 1989.

Dyer, Michael G., *In Depth Understanding*, The MIT Press, Cambridge, MA, 1983.

Dyer, Michael G., *Emotions and their Computations: Three Computer Models*, Cognition and Emotion, 1987, I (3), pp 323-347.

Ebbinghaus, H.E., *Memory: A contribution to experimental psychology.* New York: Dover 1964. (Originally published 1885; translated 1913.)

Edwards, B., *Drawing on the Artist Within: A Guide to Innovation, Invention, Imagination and Creativity.* New York: Simon and Schuster, 1986.

Fillmore, C., The case for case. In Bach, E. and Harms, R.T., editors, *Universals in Linguistic Theory*, pages 1-90. Holt, Reinhart and Winston, Chicago, IL., 1968.

Fikes, R.E., and Nilsson, N.J. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, Vol. 2, No. 3-4, pp. 189-208, 1971.

Flower, Linda S., and Hayes, John R., "The Dynamics of Composing: Making Plans and Juggling Constraints", in Gregg, Lee W., Steinberg, Erwin R., eds., *Cognitive Processes in Writing*, Lawrence Erlbaum Associates, Hillsdale, NJ 1980.

Gardner, Martin, *Aha! Insight.*, New York: Scientific American, 1978.

Ghiselin, B., Rompel R. & Taylor, C.W., "A creative process check list: Its development and validation." In C.W. Taylor (editor) *Widening horizons in creativity.* Wiley, New York, 1964.

Guilford, J.P., *The Nature of Human Intelligence.* McGraw-Hill, New York, 1967.

Hammond, Kristian, *Case-Based Planning*, Proceedings of the Case-Based Reasoning Workshop, May 1988.

Hayes, John R., and Flower, Linda S., "Identifying the Organization of the Writing Process", in Gregg, Lee W., Steinberg, Erwin R., eds., *Cognitive Processes in Writing*, Lawrence Erlbaum Associates, Hillsdale, NJ 1980.

Isaksen, S.G. and Treffinger, D. *Creative problem solving: the basic course.* Buffalo: Bearly Limited, 1985.

Jacobs, Paul S., *A Knowledge Based Approach to Language Production.* University of California Berkeley, Report No. UCB/CSD 86/254 (Ph.D. dissertation) 1985.

Kay, M., Functional grammar. In *Proceedings of the Fifth Annual Meeting of the Berkeley Linguistics Society*, pages 142-158, 1979.

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

Kolodner, Janet L., *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*, Lawrence Earlbaum Associates, Hillsdale, New Jersey, 1984.

Kolodner, Janet L., *Extending Problem Solver Capabilities Through Case-Based Inference*, Proceedings of the 4th Annual International Machine Learning Workshop, 1987.

Lebowitz, Michael, *Creating Characters in a Story-Telling Universe*, Poetics 13 (1984), pp. 171-194.

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

Lehnert, Wendy G., *The Process of Question Answering*, LEA, 1978, Hillside, New Jersey.

Lehnert, Wendy G., Plot Units: A Narrative Summarization Strategy, in W. Lehnert and M. Ringle, eds., *Strategies for Natural Language Processing*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1982.

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.

Maguire, J. *What Does Childhood Taste Like? Mental Workouts That Will Stretch, Bend and Energize the Way You Think.* New York: Quill/Morrow, 1986.

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

Miyamoto, Musashi, *A book of five rings*, translated from the Japanese by Victor Harris. Woodstock, N.Y. : Overlook Press, 1982.

Muller, Herbert J., *The Spirit of Tragedy*, ALfred A. Knopf, New York, 1956.

Norman, D.A., *Approaches to the study of intelligence*, Journal of Artificial Intelligence, Vol. 47, Nos. 1-3, January 1991, pp 327-346.

Norman, D.A. & Bobrow, D. G. (1979). "Descriptions: An intermediate stage in memory retrieval," in *Cognitive Psychology*, 11, 293-331.

Norman, D.A., & Rumelhart, D.E., *Acretion, tuning and restructuring: three modes of learning*, in J.W. Cotton & R. Kaltzky, eds., *Semantic Factors in Cognition*, Lawrence Erlbaum, Hillsdale, NJ, 1978.

Oatley, Keith, and Johnson-Laird, P.N., *Towards a Cognitive Theory of Emotions*, Cognition and Emotion, 1987, I (1), pp 29-50.

Osborn, A. (1953). *Applied imagination.* New York: Charles Scribner's Sons.

Olton, R.M. and Johnson, D.M. (1976) "Mechanisms of incubation in creative problem solving." *American Journal of Psychology*, 89, 617-630.

Pereira, F.C.N., and Warren, D.H., Definite clause grammars for language analysis - a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence*, 13(3):231-278, 1980.

Poincare, H., *Mathematical creation.* In The Foundations of Science Series. (Translated by G.B. Halsted.) New York: The Science Press, 1913. Reprinted in [Weisberg 1986].

Price, Roger, and Stern Leonard, *The Original Mad Libs*, Price Stern Sloan, Inc., Los Angeles 1958.

Propp, Vladmir, *Morphology of the Folktale*. Translated by Laurence Scott. University of Texas Press, 1968.

Read, J.D., and Bruce, D. (1982). "Longitudinal tracking of difficult memory retrievals." *Cognitive Psychology*, 14, 280-300.

Rees, Ennis, *Fables from Aesop*, Oxford University Press, NY, 1966

Rees, Jonathan A., Adams, Norman I., and Meehan, James R., *The T Manual*, Yale Computer Science Department, 1984.

Reeves, John, *Computational Morality: A Process Model of Belief Conflict and Resolution for Story Understanding*, University of California Los Angeles, (Ph.D. dissertation), May 1991.

Reeves, John, *The Rhapsody Phrasal Parser and Generator.* (Technical Note UCLA-AI-N-89-14). Artificial Intelligence Laboratory, Computer Science Department, University of California, Los Angeles, November 1989.

Roberts, Edgar V., *Writing Themes About Literature*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1977.

Rosenbloom, P.S., Laird, J.E., Newell, A., and McCarl, R., *A preliminary analysis of the Soar architecture as a basis for general intelligence*, Journal of Artificial Intelligence, Vol. 47, Nos. 1-3, January 1991, pp 289-326.

Reisbeck, Christopher K., and Schank, Roger C. (1989), eds., *Inside Case-Based Reasoning*, Lawrence Erlbaum Associates, Hillsdale, New Jersey.

Reiser, B.J. (1983). *Contexts and indices in autobiographical memory* (Tech. Rep. 24). Cognitive Science Program, Yale University.

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.

Reiser, B.J., and Black, J.B. (1982). Processing the structual models of comprehension. Text, 2, 225-252.

Reiser, B.J., and Black, J.B. (1983). "The roles of interference and inference in the retrieval of autobiographical memories." *Proceedings of the Fifth Annual Conference of the Cognitive Science Society.* Rochester, NY, 1983.

Reiser, B.J., & Black, J.B., & Abelson, R.P. (1985) "Knowledge structures in the organization and retrieval of autobiographical memories." *Cognitive Psychology*, 17, 89-137.

Saki, *The Complete Saki*, Penguin Books, London, England, 1982.

Schank, Roger C., "Identification of the conceptualizations underlying natural language." In Schank, Roger C., and Colby, K.M., editors, *COmputer Models of Thought and Language.* W.H. Freeman, San Francisco, CA, 1973.

Schank, Roger C., editor, *Conceptual Information Processing.* American Elsevier, New York, 1975.

Schank, Roger C., Abelson, Robert, *Scripts, Plans, Goals, and Understanding*, Lawrence Earlbaum Associates, Publishers, Hillsdale, New Jersey, 1977.

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

Schank, Roger C., *Interestingness: Controlling Inferences*, Artificial Intelligence 12, pp. 273-297.

Schank, Roger C., and Leake, D.B., *Creativity and Learning in a Case-Based Explainer*, Artificial Intelligence 40, pp 353-385.

Shone, R. *Creative visualization.* New York: Thorsons Pub., 1984.

Shouksmith, George, *Intelligence, Creativity and Cognitive Style.* B.T. Batsford, Ltd., London, 1970.

Slade, S., *The T Programming Language: A Dialect of Lisp.* Prentice-Hall, Inc, Englewood Cliffs, NJ.

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

Smith, Frank, *Writing and the Writer*, Heinemann Educational Books, 1982.

Stein, Morris I., *Stimulating Creativity, Vol. 1.* Academic Press, New York, 1974.

Tchudi, Stephen, and Tchudi, Susan, *The Young Writer's Handbook*, Charles Scribner's Sons, New York, 1984.

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

Turner, Scott, *MINSTREL, A Story Invention System*, M.S. Thesis, University of California, Los Angeles, 1985.

Turner, Scott and Reeves, John, *The RHAPSODY Manual* (Technical Note UCLA-AI-87-3). Artificial Intelligence Laboratory, Computer Science Department, University of California, Los Angeles, 1987.

Warren, David H. D., *WARPLAN: A System for Generating Plans*, Memo No. 76, Edinburgh University, July 1978.

Watson, J., *The double helix.* New York: Signet, 1968.

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

Wilensky, R. and Arens, Y., PHRAN: A knowledge-based natural language understander. In *Proceedings of the 18th Annual Meeting of the Association for Computational Linguistics (ACL-80)*, Philadelphia, PA, 1980.

Wilensky, Robert, Points: A Theory of the Structure of Stories in Memory, in W. Lehnert and M. Ringle, eds., *Strategies for Natural Language Processing*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1982.

Wilensky, Robert, *Planning and Understanding: A Computational Approach to Human Reasoning*, Addison-Wesley Publishing Company, Read, MA, 1983.

Willimans, M.D. & Hollan, J.D. "The process of retrieval from very-long term memory." *Cognitive Science*, 5, 87-119, 1981.

Wingfield, Arthur & Byrnes, Dennis. *The Psychology of Human Memory*. New York: Academic Press, 1981.

Winograd, Terry. *Understanding Natural Language*, NY: Academic Press, 1972.

Zernik, Uri. *Strategies in Language Acquisitions: Learning Phrases from Examples in Context.* University of California Los Angeles, Technical Report UCLA-AI-87-1 (Ph.D. dissertation) 1987.

APPENDIX A
The Rhapsody Knowledge Representation System

Rhapsody[1] is a tools package for AI program development written at UCLA. Rhapsody provides the user with ways to declare and manipulate simple frame-style representations, and a number of tools for building programs that use these representations.

Rhapsody was designed to meet the following goals and constraints:

1. Usability in multiple domains. Rhapsody is used in a number of different projects in the UCLA AI Lab using conceptual symbolic modeling techniques.

2. A class-based frame representation system without undue built-in semantics. To be usable in a wide variety of AI projects, the representation system had to leave semantic issues up to the user, and not build them in to the package.

3. Efficient implementation. While AI projects are typically developed on powerful machines, it is incorrect to rely on the brute force of the computation engine to save the user from poor design choices. A tools package should be implemented as efficiently as possible in order to be usable in large applications.

Rhapsody was originally written in T ([Rees 1984][Slade 1987]) and ported to Common Lisp ([Steele 1984]). T is an object-oriented language, so the system was influenced by the data/object-oriented style of programming. The consistency of the object-oriented approach sped code development as code could often be re-used.

The following naming conventions were adopted in Rhapsody. Functions are named as *<package>:<action>*. For instance, some of the functions in the hash table package (HT) are named HT:ENTRY, HT:COMBINE, etc. Where the *action* part is specific to some object within the package, we've tried to maintain names of the form *<action>-<object>*. Following the T convention, functions which modify their arguments have names that end in "!". For example, the function HT:COMBINE! is a destructive version of HT:COMBINE.

The following sections describe the individual tools packages packages that make up Rhapsody. The packages are presented in building block order: (1) the hash table package implements efficient storage and access data structures, (2) the representation package implements frame structures and organization, and (3) the pattern matchers implement comparison operations between representation objects. Each section begins with an introduction to the problem that the tool addresses, and then presents the package functions.

1. The material in this chapter is adapted from [Turner 1987], and updated for the most recent version of Rhapsody.

A.1 Hash Tables

The problem with using lists as the sole representation construct (as LISP encourages) is that searching a list for an item takes time dependent on the length of the list. It is preferable to have to have a representation that allows indexing directly to the needed items. For example, to store the following associations between keys and values, there must be a way to quickly find a given key in the representation:

Key		**Value**
john	→	10
chair	→	(a b c)
11	→	"trademark"

Arrays do this for items indexed by number; given the position in the array, we can quickly retrieve the item stored there. To use an array when the indexes aren't all numbers, we must first convert the key we are given to a new, numerical key and then use that. The process of converting an arbitrary key to a number is called hashing, and an array indexed by hashing is called a hash table. Most of the data structures Rhapsody uses are represented by hash tables. Representation objects (frames), for instance, consist of a number of slots and links whose indices are usually symbols.

Normally the user of Rhapsody need never worry about the hash table functions presented in this section. Hash tables are a low-level building block of Rhapsody, and the following tools packages define higher-level functions built on hash tables.

The following functions are used to create and manipulate hash tables in Rhapsody.

(HT:CREATE <*name*> <*predicate*> &rest <*initial-list*>)

HT:CREATE is used to create a new hash table. If *name* is given as nil, then a random name will be created. (The name is used only when the hash table is printed out. The hash table will not be bound to the name -- you should catch the return value (the hash table) in whatever variable you desire.) *Predicate* is used to test for equality; usually this will be #'EQL. *nitial-list* is an optional argument which consists of sequence of `key . value` pairs used to initialize the table. An example call would look like `(ht:create 'foo (HT:ENTRY` <*ht*> <*key*>) *(Settable)*

HT:ENTRY is a settable function used to retrieve or store a value in a hash table. The first argument is the hash table and the second the key. Indices that haven't been used before return a value of nil. HT:ENTRY can be set in order to add a value to the hash table:

```
> (ht:entry example 'foo)
nil
> (set (ht:entry example 'foo) 'bar)
bar
> (ht:entry example 'foo)
```

```
bar
```

(HT:PAIR <*ht*> <*key*>)

HT:PAIR is identical to HT:ENTRY except that it returns a list consisting of the key and the value. (HT:ENTRY returns only the value.)

(HT:WALK <*ht*> <*func*>)
(HT:MAP <*ht*> <*func*>)

HT:WALK and HT:MAP walk or map a procedure over the contents of a hash table. The procedure should be a lambda of two arguments (the key and value). For instance:

```
> (ht:walk example #'(lambda (key val)
                       (format t "~a -> ~a~%" key val)))
foo -> bar
#S{HT.27}
```

(HT:FIND-ENTRY <*ht*> <*predicate*>)

HT:FIND-ENTRY walks a predicate over each key in the hash table until it finds a key for which the predicate returns true. The keys are not searched in any particular order.

(HT? <*object*>) *(Predicate)*

HT? is a predicate that returns true if its argument is a hash table.

(HT:KEY? <*ht*> <*key*>) *(Predicate)*

HT:KEY? returns true if the given key (key) is defined in the hash table. (Note that the key's value in the hash table might still be nil.)

(HT:REMOVE <*ht*> <*key*>)

HT:REMOVE removes the given key from the hash table. It returns the (key val) pair removed from the table (or nil if the key wasn't present in the table). This is not quite the same as setting HT:ENTRY to nil. In both cases subsequent uses of HT:ENTRY will return nil, but after using HT:REMOVE, the key will actually be gone from the table, and so will answer false to HT:KEY? and the implied tests in HT:UPDATE and HT:ADD.

(HT:KEYS <*ht*>)
(HT:RECORDS <*ht*>)
(HT:COUNT <*ht*>)
(HT:PAIRS <*ht*>)
(HT:COPY <*ht*>)

HT:KEYS returns the list of keys defined for the hash table. HT:RECORDS returns the list of records defined for the hash table. HT:COUNT returns the number of (key record) pairs in the hash table. HT:PAIRS returns the hash table in association list form. That is, it returns a list of (key . value) pairs. HT:COPY returns a new hash table whose keys and values are the same as the original hash table.

(HT:COMBINE <*dest-ht*> <*src-ht*>)

HT:COMBINE returns a new hash table whose entries are the same as the entries in *dest-ht* plus the entries in *src-ht*. Entries in *src-ht* do not overwrite entries in *dest-ht*, so that if the key "a" is defined in both hash tables, the value in the returned, combined hash table will be the value from *dest-ht*. HT:COMBINE! is similar, but it modifies and returns *dest-ht*.

A.2 The Representation Package

Rhapsody is built around a frame-style representation package. Frames contain *slots*, which are sub-parts of the frame, and *links*, which are relationships between frames. Thus, the "human" frame might have a slot for "name" and a link for "brother."

Frames are divided into *classes* and *instances*. A class is a general frame that serves as a template and information holder for instances of that class. For example, a general class called "human" might have many instances: "joe", "bob", "pete", etc. Generally speaking, computation is done on instances, and information is attached to classes.

Slots in a frame represent a sub-part of the frame. For instance, if we have a frame for "family" it might have slots for "father", "mother" and "children", those being the sub-parts of a family.

Links are distinguished from slots because they represent relationships between instances. Each link has a back-link that is automatically set to point in the other direction. That is, if we set the "sister" of "bob" to be "sue", then the "brother" link of "sue" will be automatically set to be "bob."

The user begins by declaring links and classes. When declaring a link, the user defines what classes the link is on and what the back-link is. When declaring a class, the user defines what the class name is and what slots and links are on that class. The user then can create and manipulate instances.

The representation package keeps its own name table. This is accessed through the use of the "&" prefix. If the user wants to refer the "human" class object, he types "&human." Similarly, if he has created an instance of &human and named it "bob" he then refers to it as "&bob."

A.3 Class Functions

(CLASS:DEFINE <*name*> <*slots*>) *(Macro)*

CLASS:DEFINE is a macro that creates a new class and a number of useful auxiliary functions. Because CLASS:DEFINE is a macro, you do not need to quote the arguments. To declare a class of human, the user might type:

```
(class:define human (name age))
```

building a class `&human` with slots `name` and `age`.

In addition to creating a class object `&human`, this call binds `human` to a function that is used to create instances of the `&human` class and `human?` to a function that is used to test whether or not something is of the `&human` class.

(CLASS:SLOTS <*class*>)
(CLASS:LINKS <*class*>)
(CLASS:SLOT? <*class*> <*slot*>) *(Predicate)*
(CLASS:LINK? <*class*> <*link*>) *(Predicate)*

CLASS:SLOTS returns a list of the legal slots for the class as declared in the initial call to CLASS:DEFINE. CLASS:LINKS returns a list of the legal links for the class as declared in the initial call to CLASS:DEFINE. CLASS:SLOT? returns true if *slot* is a legal slot for *class*. CLASS:LINK? returns true if *link* is a legal link for *class*.

(CLASS:PROP <*class*> <*property-name*>) *(Settable)*

CLASS:PROP is a settable function that allows the user to attach information to a class object as if it were a hash table. Properties are used to hold information that isn't strictly part of the representation. For instance, the natural language generator can store information on how to generate a particular type of representation object on a property on the class object:

```
> (class:prop &human 'generation)
#{Generation Proc 2334}
```

(CLASS:WALK <*class*> <*func*>)

(CLASS:MAP <*class*> <*func*>)

CLASS:WALK and CLASS:MAP walk or map over all the instances of a class object. This is useful largely for debugging and resetting purposes.

A.4 Instance Functions

(INST:CREATE <*class*> <*name*> &rest <*slot-value-pairs*>)

INST:CREATE is a way to create instances of a class. It takes the class, a name for the instance, and a list of initial slot-value pairs:

```
(inst:create &human 'bob 'name 'bob)
```

would create an instance of `&human` named `&bob` and with the slot `name` set to the value `bob`.

INST:CREATE is rarely used because the call to CLASS:DEFINE defines an easier instance creation function:

```
(human 'bob 'name 'age)
```

would have the same effect as the previous call. If a nil name is given to either instance creation function, a name is generated from the class name (i.e., `&human.1`, `&human.2`, etc.)

(INST:CLASS <*instance*>)

INST:CLASS returns the class of an instance.

(INST:SLOT <*instance*> <*slot*>) *(Settable)*

INST:SLOT is a settable function that accesses a slot value on an instance. This can be set to give a slot a value, for example:

```
(set (inst:slot &bob 'name) 'robert)
```

would change Bob's name. To delete a slot, set its value to nil. Note that this means that you cannot have a slot with value nil (except as indistinguishable from a null slot).

(INST:LEGAL-SLOTS <*instance*>)
(INST:SLOT? <*instance*> <*slot*>) *(Predicate)*
(INST:IS-SLOT? <*instance*> <*slot*>) *(Predicate)*

INST:SLOT? returns true if *slot is a legal slot for instance.* INST:LEGAL-SLOTS returns the list of legal slots. INST:IS-SLOT? returns true if *slot* is defined on the *instance* (i.e., has a non-nil value).

(INST:WALK-SLOTS <*instance*> <*func*>)
(INST:MAP-SLOTS <*instance*> <*func*>)

INST:WALK-SLOTS walks a function over every slot of an instance. The function should take two arguments, a slot name and a slot value. INST:MAP-SLOTS maps a function over all the slots in an instance.

(INST:LINK <*instance*> <*link*>)

INST:LINK returns the list of links under a particular link name. Unlike INST:SLOT, INST:LINK is not settable. For instance:

```
> (inst:link &human.12 &sibling)
(&HUMAN.13)
```

shows that `&human.12` has a `&sibling &human.13`.

(INST:ADD-LINK <*instance*> <*link*> <*value*>)

INST:ADD-LINK is used to add a value to a link. Generally speaking the value is expected to be another Rhapsody instance. For example:

```
> (inst:link &human.12 &sibling)
(&HUMAN.13)
> (inst:add-link &human.12 &sibling &human.4)
(&HUMAN.4 &HUMAN.13)
```

adds another sibling to `&human.12`.

Note that there can be multiple values under a link and that the values aren't in any particular order. The same value can also be on the link several different times.

(INST:DELETE-LINK <*instance*> <*link*> <*value*>)

INST:DELETE-LINK deletes the first occurrence of *value* under *link.* Note that if *value* appears multiple times on the link only the first occurrence will be deleted.

(INST:LINK? <*instance*> <*link*>) *(Predicate)*
(INST:LEGAL-LINKS <*instance*>)

INST:LINK? returns true if *link* is a legal link for *instance*. INST:LEGAL-LINKS returns the list of legal links.

(INST:WALK-LINKS <*instance*> <*func*>)
(INST:MAP-LINKS <*instance* <*func*>)

INST:WALK-LINKS walks a function over every link of an instance. The function should take two arguments, a link name and a link value. INST:MAP-LINKS maps a function over all the links in an instance.

(INST:SLOTS <*instance*>)
(INST:LINKS <*instance*>)

INST:SLOTS and INST:LINKS return the defined slots and links for a particular instance (i.e., the links and slots that have non-nil values).

(INST:PROP <*instance*> <*prop*>) *(Settable)*

Like CLASS:PROP, INST:PROP uses *instance* like a hash table, and is used to hang non-representation information on an instance object.

(INST:COPY <*instance*>)

INST:COPY creates a copy of the given instance. If an instance appears in a slot or a link, it is also copied. If lists appear in a slot or a link they are copied. Everything else remains unchanged. So, for instance:

```
> (pp &jane)
(HUMAN JANE
   NAME JANE
   &LOVER <==> &BOB)

> (set xx (inst:copy &jane))
{HUMAN.12}

> (pp &human.12)
(HUMAN HUMAN.12
  NAME JANE
```

```
  &LOVER <==> &HUMAN.13)
```

Note that `&BOB` was also copied (becoming `&HUMAN.13`).

(INST:WALK-TREE <*instance*> <*proc*>)
(INST:MAP-TREE <*instance*> <*proc*>)

INST:WALK-TREE and INST:MAP-TREE take a procedure and apply it to every instance reachable (via links) from *instance*. The procedure should be a lambda of one argument, which will be an instance. For example:

```
> (pp &jane)
(HUMAN JANE
   NAME JANE
   &LOVER ==> &BOB)
> (inst:walk-tree &jane
      #'(lambda (inst) (print inst)))
JANE
BOB
```

A.5 Link Functions

(LINK:DEFINE <*name*> <*name-class*> <*back-name*> <*back-name-class*>) *(Macro)*

LINK:DEFINE is used to define the name of a link, the classes it can be used on, its back-link name, and the classes the back-link can be used on. LINK:DEFINE must be used after CLASS:DEFINE. For example:

```
> (class:define human (name age))
#{HUMAN}
> (class:define animal (species name))
#{ANIMAL}
> (link:define pet &human pet-of &animal)
&PET
```

declares a link `&pet` which will point from `&human` to `&animal` and a link called `&pet-of` which will point from `&animal` to `&human`. *name-class* and *back-name-class* can be lists, in which case the link points to or from more than one class of objects.

(LINK:BACK <*link*>)

LINK:BACK returns the back-pointer for a link, i.e, `(link:back &pet)` would return

`&pet-of` and `(link:back &pet-of)` would return `&pet`.

A.6 The Pattern Matching Package

Matching consists of taking two representation objects and deciding whether or not they represent the same thing. For instance, we'd like a matcher to tell us that `&HUMAN.1` and `&HUMAN.2` represent the same thing, even though they don't have the same name:

```
(HUMAN &HUMAN.1
   NAME SCOTT
   AGE 24)

(HUMAN &HUMAN.2
     NAME SCOTT
     AGE 24)
```

#'EQ is clearly not useful in this case, since &HUMAN.1 and &HUMAN.2 are different representation objects and will result in #'EQ returning false.

The above example gives an intuitive idea of when two things match: they are of the same class and their slots have the same values. Two things complicate this intuitive notion: variables and links.

Variables come into play because often we'd like one of our comparison objects to be a pattern. A pattern could be used, for instance, to check to see if a person has the same last name as first name:

```
(HUMAN &HUMAN.1
   LAST-NAME ?X
   FIRST-NAME ?X)
```

If fact, patterns get far more complex than this. We'd like to be able to embed tests and boolean conditionals in our patterns as well. If our matcher handles patterns, it should return a table that contains that values assigned to the variables during the match.

Links confuse the matching problem in a different way. Unlike a slot, one link may point off to several different representation objects. For instance, we might have two events that caused a number of state changes (i.e., the event of knocking the milk glass over resulted in the glass being empty and the table having milk on it and the mother being upset):

```
(EVENT &MILK-GLASS
  &CAUSES <==> (&STATE-CHANGE.1 &STATE-CHANGE.2))

(EVENT &UNKNOWN
```

```
&CAUSES <==> (&STATE-CHANGE.3 &STATE-CHANGE.4))
```

How are we to compare these two events? Should we compare &STATE-CHANGE.1 against &STATE-CHANGE.3? Or against &STATE-CHANGE.4? Should we return the first match we find, or somehow return all the matches we find? This is a particularly difficult issue, and in the matchers we present here we will skirt this issue, leaving it up the user what is to be done when links complicate the matching issue.

The kind of matching we have talked about so far is close to unification. There are, however, other kinds of matching. Another type of matcher that is often useful is one that returns some kind of "similarity index." That is, it returns a number that is indicative of how close a match two things are, and that has the property that closer matches generate higher numbers. Another type of matcher returns the differences between two objects.

The following sections discuss the issues of variables and instantiation, and then present the three matchers that are implemented in Rhapsody: a simple matcher, a similarity matcher, and a difference matcher.

A.7 Variables and Patterns

The user creates a variable by prefacing a name with a question mark, for example `?actor`. This creates a variable whose name is "actor." Two variables that have the same name are the same variable as far as Rhapsody is concerned. Thus if you were to create an instance of a goal:

```
(goal nil
  'actor ?actor
  'to ?actor)
```

conceptually, the same "thing" would be filling both the actor and the to slots.

During matching, variables can match anything. After the match, the pattern matcher returns a binding form. The binding form consists of variable names and the values they would have to take in order for the match to work. So, if the above goal is matched with:

```
(goal nil
  'actor &human.21
  'to &human.21
```

the binding form returned by the matcher would show ?actor bound to &human.21.

The following functions are defined for variables:

(VAR? *<object>*) *(Predicate)*

Predicate that returns true if <*object*> is a variable.

(VAR:NAME <*var*>)

Returns the name of a variable.

(VAR:VALUE <*binding-form*> <*variable*>)

Returns the value of the variable in the binding form.

A.8 Instantiation

Instantiation is the process of taking a hash table which contains variable bindings and an instance which contains variables and replacing the variables in the instance with the values they have in the binding table. For instance, given the following binding table and instance:

```
?X ==> SCOTT

(HUMAN &RON
       NAME ?X)
```

If &RON is instantiated from the binding form, the result would be:

```
(HUMAN &RON
  NAME SCOTT
```

where ?X is replaced with its binding value.

In order to instantiate an instance in Rhapsody, the user calls one of the following explicit instantiation functions:

(VAR:INSTAN <*pattern*> <*binding-form*>)
(VAR:INSTAN! <*pattern*> <*binding-form*>)
(VAR:INSTAN-TREE <*pattern*> <*binding-form*>)
(VAR:INSTAN-TREE! <*pattern*> <*binding-form*>)

VAR:INSTAN returns an instance identical to *pattern* except that all occurrences of variables will have been replaced by their bindings in *binding-form*. VAR:INSTAN! is similar, but makes its replacements destructively on *pattern*.

The tree versions of INSTAN follow links from the initial pattern, and instantiated variables in

all instances that can be reached from the input *pattern*.

A.9 Matching Functions

Rhapsody provides three matchers. The first is called the simple link matcher (INST:SLMATCH) and is a "unification" style matcher. It is called a simple link matcher because its treatment of links is simplistic. The second matcher is the similarity matcher (INST:SIMILAR) and it returns a numerical index indicating how "alike" two objects are. The third matcher is a difference matcher (INST:DIFF) and it returns a list of the differences between two objects.

(INST:SLMATCH <*inst1*> <*inst2*> &rest <*keywords*>)
(INST:SLMATCH-TREE <*inst1*> <*inst2*> &rest <*keywords*>)

INST:SLMATCH performs a unification-style match between instances. The matcher walks through the instances, comparing slots and links. INST:SLMATCH does not follow links (i.e., ignores them entirely) while INST:SLMATCH-TREE follows links in a rather simplistic manner -- by assuming there is only one link by each name. If there is more than one link by a name, then INST:SLMATCH-TREE matches the links in order (clearly the wrong behavior).

In addition to allowing pattern-matching variables as discussed above, INST:SLMATCH and INST:SLMATCH-TREE allow other special matching constructs, namely *AND* and *PROC*.

If a list that starts with *AND* is found in a slot, then the matcher tries to match the slot against everything that appears in the list. So, for instance, if we attempted to match these two instances:

```
(HUMAN &HUMAN.1
   NAME (*AND* SCOTT ?X))

(HUMAN &HUMAN.2
       NAME SCOTT)
```

The NAME slot from &HUMAN.2 would be matched against both SCOTT and ?X, each match would succeed, and the result would be a binding table with ?X bound to SCOTT. The *AND* construct can have more than two conditions.

If a list that starts with *PROC* is found in a slot, then the matcher assumes that the second element in the list is a procedure of two arguments, the first of which is the object being matched against and the second of which is the current binding list. The match succeeds if the procedure returns true.

As an example, suppose we were matching the following two instances:

```
(HUMAN &HUMAN.1
  FRIEND (*PROC* #'(LAMBDA (INST BINDS)
           (AND (HUMAN? INST)
           (EQ (INST:SLOT INST 'SEX) 'MALE)))))

(HUMAN &HUMAN.2
  FRIEND (HUMAN &HUMAN.3
     NAME 'JOHN
     SEX 'MALE))
```

The *PROC* checks to see if the FRIEND is a HUMAN and a male. When the procedure is called, INST will be bound to &HUMAN.3 and BINDS to the current binding table (in this case, an empty table since no variables have been bound). The AND within the procedure will return true since &HUMAN.3 is a HUMAN? and has SEX MALE.

Rhapsody provides a couple of read macros to facilitate use of *AND* and *PROC*. These macros remove the need to quote the list, so that the user can write:

```
(human nil
  'name 'scott
  'friend (*proc* #'(lambda (inst binds)
      (and (human? inst)
       (eq (inst:slot inst 'sex) 'male)))))
```

Rather than the more cumbersome:

```
(human nil
  'name 'scott
  'friend (list '*proc* #'(lambda (inst binds)
       (and (human? inst)
       (eq (inst:slot inst 'sex) 'male)))))
```

AND and *PROC* can be nested and are useful in conjunction. The user can combine an *AND* and *PROC* to make a test and bind a variable if the test succeeds. For instance:

```
(HUMAN &HUMAN.1
  FRIEND (*AND* ?FRIEND
      (*PROC* #'(LAMBDA (INST BINDS)
           (AND (HUMAN? INST)
           (EQ (INST:SLOT INST 'SEX) 'MALE)))))

(HUMAN &HUMAN.2
  FRIEND (HUMAN &HUMAN.3
    NAME 'JOHN
    SEX 'MALE))
```

binds ?FRIEND to the friend if the friend passes the *PROC* test. Other such combinations are also useful.

INST:SLMATCH and INST:SLMATCH-TREE also take a number of optional keywords that affect the behavior of the matcher:

NONIL Keyword

Normally a missing or nil slot (there is no distinction in Rhapsody) in the pattern matches anything. This allows the user to specify the minimal matching pattern; things he doesn't mention are assumed to be unimportant. With this behavior, the following two things match:

```
(HUMAN &HUMAN.1
  NAME SCOTT)

(HUMAN &HUMAN.2
  NAME SCOTT
  SEX MALE)
```

The nil value for SEX on &HUMAN.1 matches the MALE value on &HUMAN.2.

If you want to make nil slots active (i.e., matching only other nil slots), then you can give the NONIL keyword after the other arguments in INST:SLMATCH and INST:SLMATCH-TREE. The call would be:

```
> (inst:slmatch &human.1 &human.2 'nonil)
```

IGNORE Keyword

The IGNORE keyword is used to specify slots and links that the matcher should ignore during the matching process. Ignored slots and links are invisible to the matcher. They do not affect the matching process.

If the IGNORE keyword is given alone, as in this call:

```
> (inst:slmatch &human.1 &human.2 'ignore)
```

then the matcher checks a PROP (see INST:PROP, CLASS:PROP) called IGNORE-SLOTS on both the instance being matched (&HUMAN.1 in this case) and on the class of the instance being matched (&HUMAN in this case). This prop should contain a list of slot and/or links to be ignored by the matching process.

For instance, suppose we were attempting to match the following:

```
> (pp &human.1)
(HUMAN &HUMAN.1
  NAME SCOTT
  AGE 25)
> (pp &human.2)
(HUMAN &HUMAN.2
   NAME SCOTT
   AGE 21)
> (inst:slmatch &human.1 &human.2)
nil
```

The match fails because the AGE slot on the two instances does not match. We can tell the matcher to ignore the AGE slot by putting the slot on the IGNORE-SLOTS property of the class and giving the keyword IGNORE:

```
> (set (class:prop &human 'ignore-slots) '(age))
(AGE)
> (inst:slmatch &human.1 &human.2 'ignore)
#S{HT.101}
```

Now the match was successful (it returned the hash table containing variable bindings; since there were no variables this table is empty, but it is still a non-nil value) because the AGE slot was ignored.

The IGNORE keyword can be followed by a property name to be used instead of IGNORE-SLOTS. For instance:

```
> (inst:slmatch &human.1 &human.2 'ignore 'my-ignore)
```

This is useful if you want to ignore different slots in different circumstances; you use a different property for each circumstance.

Finally, the IGNORE keyword can also be a lambda of two arguments (the instance being matched and the current slot) and if the lambda returns true, then the slot is ignored. This is a versatile method that can be used, for instance, to ignore all slots on a particular list:

```
> (inst:slmatch &human.1 &human.2
  'ignore #'(lambda (inst slot)
    (memq? slot *ignore-list*)))
```

This would ignore all slots that appeared on the global `*IGNORE-LIST*`.

Initial Binding Table

The user may specify an initial variable binding table by including it as an optional final argu-

ment. Most often this is the binding returned by a previous match. For instance, suppose we match:

```
> (pp &human.1)
(HUMAN &HUMAN.1
   NAME SCOTT)
> (pp &human.2)
(HUMAN &HUMAN.2
   NAME ?NAME)
> (set result (inst:slmatch &human.1 &human.2))
#{HT.101
```

RESULT now contains the binding table from the first match, whose only entry binds ?NAME to SCOTT. We can now use this binding table in a second match:

```
> (pp &human.3)
(HUMAN &HUMAN.3
  FRIEND (HUMAN HUMAN.5
     NAME ?NAME))
> (pp &human.4)
(HUMAN &HUMAN.4
   FRIEND (HUMAN HUMAN.6
    NAME FRED))
> (inst:slmatch &human.3 &human.4 result)
nil
```

This match fails because RESULT already has ?NAME bound to SCOTT, and so the match of ?NAME to FRED fails.

The keywords for the matchers have been presented independently, but they can of course be combined:

```
> (inst:slmatch &human.1 &human.2 'ignore 'my-ignore 'nonil)
```

(INST:SIMILAR *<inst1>* *<inst2>* &rest *<keywords>*)
(INST:SIMILAR-TREE *<inst1>* *<inst2>* &rest *<keywords>*)

INST:SIMILAR and INST:SIMILAR-TREE take two instances and return a (floating point) number between -1 and 1 indicating how similar the two instances are. The number has no intrinsic meaning and is only useful to rank pairs of instances. (The number is actually calculated by subtracting the number of slots that didn't match from the number of slots that did match and dividing by the total number of slots. Thus 0 indicates that there were as many slots that matched as didn't, and 1 indicates that all the slots matched.)

INST:SIMILAR and INST:SIMILAR-TREE take the same keywords as INST:SLMATCH.

(INST:DIFF <*inst1*> <*inst2*> &rest <*keywords*>)
(INST:DIFF-TREE <*inst1*> <*inst2*> &rest <*keywords*>)

INST:DIFF takes two instances and returns a list of slots that did not match. INST:DIFF-TREE takes two instances and returns a list of (INST1 INST2 SLOT) lists, whose meaning was that INST1 was matched against INST2 and SLOT did not match. INST:DIFF and INST:DIFF-TREE also take the same keywords as INST:SLMATCH.

APPENDIX B
MINSTREL Implementation

This appendix contains implementation statistics and samples of the Common Lisp source code from the current version of MINSTREL. This material is intended primarily to give the reader a feeling for MINSTREL's implementation at the Lisp level. There is no attempt to cover all of MINSTREL's functions. Instead we have chosen to present illustrative examples from the most important parts of MINSTREL.

B.1 Implementation Details

MINSTREL is written in Common Lisp [Steele 1984] and was developed on a Sun 3/50 workstation. MINSTREL is about 8,000 lines of code (not counting comments or blank lines), and is built upon a tools package called Rhapsody [Turner 1987] that is itself about 10,000 lines of code. The division of code into MINSTREL's major components is shown in the following table:

Component	Lines
Control	1016
Representation	784
Episodic Memory	1485
Semantic Memory	138
TRAMs	1185
Author-Level Plans	2182
Language	1014
Utilities	322

MINSTREL uses about 1400 seconds of CPU time to tell *The Mistaken Knight*; other stories take similar amounts of time.

B.2 Control Structure

MINSTREL has a hierachical control structure with three levels:

1. Storytelling
2. Creativity
3. Episodic Memory

At the top level (storytelling) MINSTREL is an agenda-based planner. At each cycle of the storytelling process, MINSTREL pops the highest priority goal off of the planning agenda, finds plans to achieve that goal, and applies those plans until one succeeds. This cycle repeats until the

planning agenda is empty.

Each author-level plan can make use of creativity to achieve its goals. MINSTREL's creativity is embedded in imaginative memory. To make use of creativity, an author-level plan uses a recall specification to attempt recall from imaginative memory. Initially, imaginative memory attempts recall from episodic memory. If this fails, imaginative memory applies a TRAM to the recall specification and recursively calls imaginative memory. This cycle repeats until some combination of TRAMs invents a memory that matches the original recall specification or until the recursion depth has been exceeded.

Episodic memory is at the lowest level of control. Episodic memory is called by imaginative memory when recall is first attempted, and after every application of a TRAM. Episodic memory searches a logical tree structure to see if memory contains any episodes which match the recall specification.

The following sections describe the control code at each level of control.

B.3 Storytelling

MINSTREL's storytelling algorithm is based upon a priority queue. MINSTREL's initial storytelling goal is placed upon the queue. Thereafter, at each iteration of the planning cycle, the highest priority goal is popped off the queue and processed. This continues until the priority queue is empty:

```
(defun alp:run (initial-goal)
  (queue:add-task ru:*alq* initial-goal)
  (do ()
      ((not (alp:execute-queue ru:*alq*)) t)))
```

ALP:EXECUTE-QUEUE pops the highest priority goal off of the author-level queue and attempts to solve the goal. If problem-solving fails, the goal's priority is lowered and the goal is returned to the queue. This permits MINSTREL to attempt to solve goals several times.

```
(defun alp:execute-queue (queue)
  ;;
  ;;  Pop a goal off the queue and pass it to alp:solve-goal.
  ;;
  (let ((alg (queue:get-task queue)))
    (if (null alg) (return-from alp:execute-queue))
    (if (and (not (alp:solve-goal alg))
             (> (inst:slot alg :priority)
                *alp-min-priority*))
        ;;
        ;;  Requeue failed goals.
        ;;
        (progn
          (setf (inst:slot alg :priority)
                (- (inst:slot alg :priority) *alp-priority-decrement*))
```

```
        (trace:cformat *alp-trace*
                       "Re-queuing at priority %d.\n"
                       (inst:slot alg :priority))
        (queue:add-task queue alg))
      ;;
      ;;  Free all others.
      ;;
      (ru:free alg))
  alg))
```

ALP:SOLVE-GOAL uses imaginative memory (TRAM:DO-RECALL) to find author-level plans that apply to the top priority goal. (This permits MINSTREL to use creativity at the author level.) Whatever plans are found are tried sequentially until a plan succeeds:

```
(defun alp::solve-goal (alg)
  ;;
  ;; Get the next author-level goal and announce it.
  ;;
  (trace:cformat *alp-trace*
     "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n")
  (trace:cformat *alp-trace*
                 "Author-level goal %a applied to %a.\n"
                 (inst:slot alg :type)
                 (inst:slot alg :object))
  ;;
  ;;  If this goal has already been solved, note that and exit.
  ;;
  (if (inst:prop (inst:slot alg :object) (inst:slot alg :type))
    (progn
      (trace:cformat *alp-trace* "***  This goal already achieved.  ***\n")
      t)
    (let ((plns (tram:do-recall alg :recurse 0 :tram-list alp::*tram-list*
                                :eval-list alp::*eval-list* :memory *alm*))
          (solved nil))
      ;;
      ;;  tram:do-recall finds plans to achieve this author-level
      ;;  goal.  We try the plans one at a time until one succeeds
      ;;  or we run out of plans.
      ;;
      (do ((plans plns (cdr plans)))

          ((or (null plans) solved)
           (if solved
               (trace:cformat *alp-trace*
                              "Author-level planning succeeded.\n")
               (trace:cformat *alp-trace*
                              "Author-level planning failed.\n")))


          (setq solved (alp:execute-plan
                         (inst:slot (car plans) :plan) alg)))
      ;;
      ;;  And out... it's that simple.
      ;;
      (trace:cformat *alp-trace*
       "++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++\n")
```

```
      solved))
  )
```

MINSTREL's author-level plans (ALPs) are pieces of structured Lisp code. ALP:EXECUTE-PLAN simply executes the parts of the author-level plan structure in the correct order:

```
(defun alp:execute-plan (alp alg)
  (trace:cformat alp:*alp-trace*
        "Trying author-level plan %a.\n" (alp:alp-structure-name alp))
  ;;
  ;;  First try the test part if present.  If the test fails, all fails.
  ;;
  (if (alp:alp-structure-test alp)
      (let ((test-result (apply (alp:alp-structure-test alp) alg
                                (inst:slot alg :object) nil)))
        (trace:cformat alp:*alp-verbose* "Test result is %a.\n" test-result)
        (if (null test-result) (return-from alp:execute-plan nil))))
  ;;
  ;;  Test succeeded or not present, so try executing the body.
  ;;
  (trace:cformat alp:*alp-verbose* "Applying body of alp-eval.\n")
  (if (apply (alp:alp-structure-body alp) alg (inst:slot alg :object) nil)
      ;;
      ;;  Body was successful, so mark the representation object of the
      ;;  author-level goal as having achieved the author-level goal.
      ;;  This keeps us from repeating a goal where it has already been
      ;;  achieved.
      ;;
      (setf (inst:prop (inst:slot alg :object) (inst:slot alg :type)) t)))
```

Note that ALP:EXECUTE-PLAN records when a goal has succeeded. This is used in ALP:SOLVE-GOAL to prevent MINSTREL from attempting to solve again a goal that has already been solved.

Author-level plans (ALPs) are structured Lisp code. ALPs have several components:

Name	Identifies the ALP.
Comment	Comment strings that can be printed out in the trace when the author-level plan is applied.
Type	Identifies the sub-type of goal the author-level plan applies to.
Test	A predicate that returns true if an ALP applies to a particular author-level goal. Usually the test looks at the object of the goal, which is bound to *al-obj*.
Body	Lisp code to achieve the author-level plan.

ALP:Tell-Story is an example of an author-level plan. It applies to goals of type &tell-story applied to Planning Advice Themes (PAT). It achieves the goal of telling a story by creating a variety of author-level sub-goals:

```
(alp:plan tell-story
  (comment "To tell a story about a theme, instantiate all the"
```

```
        "parts of theme and then achieve other author-level goals.")
(type &tell-story)
(test (pat? *al-obj*))
(body
  ;;
  ;;  To tell a story about a PAT, instantiate the first half, then
  ;;  the second half.  The priority of 89 in the second half is 1
  ;;  below the priority of the &check-scene goals, assuring that
  ;;  the entire decision will be filled in before we move on to
  ;;  the consequence.
  ;;
  (let* ((pat (inst:slot *al-goal* :object))
         (p1 (inst:slot pat :decision))
         (p2 (inst:slot pat :consequence)))

    ;;
    ;;  Instantiate first half followed by 2nd half.
    ;;
    (inst:walk-tree p1
         #'(lambda (inst)
           (alp:make-goal &instantiate inst 100)))
    (inst:walk-tree p2
         #'(lambda (inst)
           (alp:make-goal &instantiate inst 88)))
    ;;
    ;;  Make a goal to connect the two halves of the Pat.
    ;;
    (alp:make-goal &connect pat 89)
    ;;
    ;;  Check the story for suspense && tragedy.
    ;;
    (alp:make-goal &check-story-for-suspense p1 40)
    (alp:make-goal &check-story-for-tragedy p1 40)
    (alp:make-goal &check-story-for-foreshadowing p1 40)
    (alp:make-goal &check-story-for-characterization pat 40)
    ;;
    ;;  Story Presentation goals
    (alp:make-goal &add-story-intros *story* 30)
    (alp:make-goal &add-denouements *story* 30)
    ;;
    ;;  Make sure we succeed.
    ;;
    t)))
```

ALP:TELL-STORY achieves its goal by creating sub-goals. An example of an ALP that achieves its goals by creating story scenes is ALP:General-Instantiate. ALP:General-Instantiate tries to flesh out a scene by using imaginative memory to recall a similar scene. If it can recall or invent a similar scene, ALP:General-Instantiate copies information from the recalled scene to the original scene to flesh it out:

```
(alp:plan general-instantiate
  (comment "To instantiate a portion of a story, use the TRAM cycle.")
  (type &instantiate)
  (body
    ;;
```

```
;;  *al-obj* is the object we are trying to instantiate.
;;
(let ((remindings (tram:do-recall *al-obj*
                                  :recurse *minstrel-tram-recurse-level*
                                  :eval-list eval::*evaluation-list*)))
  ;;
  ;; If we got any remindings, use them.  Otherwise we fail.
  ;;
  (if remindings
      ;;
      ;;  Use the first of the remindings.
      ;;
      (progn
        (trace:cformat alp::*alp-trace*
                       "Found a reminding in alp:general-instantiate.\n")
        (jammer:apply-reminding *al-obj* (car remindings))
        ;;
        ;;  Now that we've done the jamming we can free all of the
        ;;  remindings.
        ;;
        (mapc #'(lambda (rem) (free-tree rem)) remindings)
        ;;
        ;;  Since we've just jammed in some new stuff, check all
        ;;  the new scenes.
        ;;
        (inst:walk-tree *al-obj*
               #'(lambda (inst)
                   (alp:make-goal &check-new-scene inst 90)))
        t)
      ;;
      ;;  Otherwise we failed.
      ;;
      (progn
        (trace:cformat alp::*alp-trace*
                       "ALP:GENERAL-INSTANTIATE failed.\n")
        nil)))))
```

The call to TRAM:DO-RECALL in the body of ALP:General-Instantiate invokes imaginative memory to find a reminding that matches the recall specification *AL-OBJ*. This transfers control to the creativity level, where TRAMs will be applied to invent an appropriate reminding if one does not already exist in memory.

The "jammer" is a component that merges two representation objects (i.e., jams them together). JAMMER:APPLY-REMINDING merges the recalled episode with the original object.

B.4 Creativity

MINSTREL's creative process is centered around Transform-Recall-Adapt Methods (TRAMs). TRAMs are implemented as structured Lisp code. A macro takes the definition of a TRAM and translates it into a Lisp structure with the parts of the structures containing Lisp code. The structure is then compiled for greater efficiency and stored in the TRAM pool.

TRAM:Standard-Problem-Solving is the simplest of MINSTREL's TRAMs. It neither transforms the original problem nor adapts the recalled solution; it merely passes through both unchanged, thereby implementing standard case-based reasoning:

```
(tram:rule standard-problem-solving
   (options :never-recurse)
   (comment "Standard problem-solving recalls past episodes that are exactly"
            "similar to the current episode, and which can be used without any"
            "special adaptation.")
   (transform *rspec*)
   (adapt *episode*))
```

This TRAM has four parts. The options field controls the behavior and usage of the TRAM. In this case, TRAM:Standard-Problem-Solving is marked "never recurse" which means that no further TRAMs will be applied after TRAM:Standard-Problem-Solving. (This ensures that MINSTREL will not get caught in an endless recursion.) The comment field contains any number of comment strings that can be printed out in the trace when the TRAM is used. The transform field takes the original recall specification (*rspec*) and transforms it; in this case, the recall specification is returned unchanged. The adapt field takes any solution recalled and adapts it for the orginal problem. In this case, the recall specification was passed through unchanged, so the recall solution can be as well.

TRAM:Recall-Act is slightly more complicated. TRAM:Recall-Act tries to recall an action by stripping off all links except links to related goals and states:

```
(tram:rule recall-act
   (comment "An act that has an intended state can recall anything that"
            "matches that act, the goal and the intended state.  Other"
            "links are superfluous.")
   (options :dont-repeat)
   (test (act? *spec*))
   (transform
     ;;
     ;;  Make a list of the links to save.
     ;;
     (let ((save (list *rspec*
                       (car (inst:link *rspec* &intends))
                       (car (inst:link *rspec* &plan-of)))))
       (do ((s save (cdr s)))
           ((null s)
               *rspec*)
           ;;
           ;;  If not on the list, remove it.
           ;;
           (if (inst? (car s))
               (inst:walk-links (car s)
                   #'(lambda (link val)
                       (if (not (member val save))
                           (inst:delete-link (car s) link val)))))
           )))

   ;;
   ;;  No adaptation is necessary.
   ;;
   (adapt *episode*))
```

There are several differences between TRAM:Recall-Act and TRAM:Standard-Problem-Solving. First, TRAM:Recall-Act has a test field. The test field is used to determine whether a TRAM is applicable to a particular problem. In this case, TRAM:Recall-Act is applicable only to act schemas. Second, TRAM:Recall-Act has a new option, :dont-repeat. This option ensures that MINSTREL won't apply this TRAM twice to the same problem. In this case, there is no point in applying TRAM:Recall-Act twice, because links can only be removed once. Finally, TRAM:Recall-Act has a more complicated transform field. The code in the transform field strips the recall specification of links to all other schemas except any pointed to by &Intends or &Plan-Of links.

Now let's examine a TRAM that has an adaptation component. TRAM:Intention-Switch is a simple heuristic which applies to actions which have intentional effects. It suggests that if the effect of an action was intentional it might just as well have been unintentional:

```
(tram:rule intention-switch
  (comment "To create a scene with an intentional outcome, look for a scene"
           "with the appropriate unintentional outcome and adjust it"
           "accordingly.")
  (options :dont-repeat)
  ;;
  ;;  This TRAM applies to acts with intended outcomes.
  ;;
```

```
(test (and (act? *spec*)
           (inst:link *spec* &intends)))
;;
;;  Modify the *rspec* by switching the intended act to an unintended
;;  one.
;;
(recall
  (let ((st (car (inst:link *rspec* &intends))))

    (inst:delete-link *rspec* &intends st)
    (inst:add-link *rspec* &unintended st))
  *rspec*)

;;
;;  Adapt the recalled episodes by flipping the intention back.
;;
(adapt
  (let ((st (car (inst:link *episode* &unintended))))

    (inst:delete-link *episode* &unintended st)
    (inst:add-link *episode* &intends st)
    *episode*)))
```

TRAM:Intention-Switch is very straightforward. In the transform field, the &intends link in the recall specification is replaced with an &unintended link. In the adapt field, the recalled &unintended link is replaced with an &intends link. Using TRAM:Intention-Switch, MINSTREL can recall unintended act and use it to invent an intentional act.

Of course, some of MINSTREL's creativity heuristics are more complex. For example, TRAM:Achieve-B-Motivated-P-Goal involves both a complicated transformation and a complicated adaptation:

```
(tram:rule achieve-b-motivated-p-goal
           (comment "To achieve a belief-motivated protection goal,"
                    "thwart the goal in the belief.")
           ;;
           ;;  PG is the goal of the act.
           ;;
           (local (pg (car (inst:link *spec* &plan-of)))
                  (belief-goal nil))
           ;;
           ;;  Applies to acts that are the plan-of a belief-motivated p-goal.
           ;;
           (test
             (and pg
                  (isa:son? &p-goals (inst:slot pg :type))
                  (inst:link pg &b-motivated)))
  ;;
  ;;  Now try to find an act which intends a state which thwarts the belief
  ;;  goal.
  ;;
  (transform
    (let ((new-act (act nil))
          (new-goal (goal nil))
```

```
      (new-state (state nil))
      (belief-goal
        (car (inst:link
               (car (inst:link
                      (car (inst:link pg &b-motivated))
                      &mental-event))
               &plan-of))))
    ;;
    ;;  Copy the important parts of the act, and connect it to the
    ;;  new-state.
    ;;
    (setf (inst:slot new-act :type) (inst:slot *spec* :type))
    (setf (inst:slot new-act :actor) (inst:slot *spec* :actor))
    (inst:add-link new-state &intended-by new-act)
    ;;
    ;;  Copy the important parts of the belief goal over to the
    ;;  new-goal.
    ;;
    (inst:add-link new-state &thwarts new-goal)
    ;;
    ;;  Return the new-state.
    ;;
    new-state))
;;
;;  The episode we return has to match up against the original *all
;;  the way down to the goal of the mental event* or else the
;;  various actors won't get matched up correctly and we'll have
;;  extra characters wandering around the story.
;;
;;  Above, we copied the important parts of the belief-goal to the
;;  new goal that was thwarted.  In the adapt we reverse this, and
;;  copy the important parts of the thwarted goal (in the reminding)
;;  to a made-up belief goal.
;;
(adapt
  (let* ((nphg (goal nil))
         (nbg (goal nil &plan
                    (act nil &event-in
                         (belief nil &b-motivates nphg))))
         (tg (car (inst:link *episode* &thwarts)))
         (ta (car (inst:link *episode* &intended-by))))
    ;;
    ;;  Remove &thwarts
    ;;
    (mapc #'(lambda (tg) (inst:delete-link *episode* &thwarts tg))
          (inst:link *episode* &thwarts))
    ;;
    ;;  Remove plans of the act.
    ;;
    (mapc #'(lambda (tg) (inst:delete-link ta &plan-of tg))
          (inst:link ta &plan-of))
    ;;
    ;;  Copy important stuff from tg to nbg.
    ;;
    (setf (inst:slot nbg :type) (inst:slot tg :type))
    (setf (inst:slot nbg :actor) (inst:slot tg :actor))
    (setf (inst:slot nbg :object) (inst:slot tg :object))
```

```
    ;;
    ;;  Connect the nphg to the *episode*
    ;;
    (inst:add-link ta &plan-of nphg)
    ;;
    ;;  Return the act, not the state.
    ;;
    ta)))
```

TRAMs are applied during storytelling by two functions.

The top-level function, TRAM:DO-RECALL, invokes imaginative memory. TRAM:DO-RECALL uses KSM:RECALL to try to recall an episode from episodic memory using elaboration. But if KSM:RECALL fails, TRAM:DO-RECALL tries to invent an episode by calling a lower-level function, TRAM:REALLY-EXECUTE, which applies TRAMs to the problem:

```
(defun tram:do-recall (*rspec* memory no-evaluate random-selection
                       random-tram recurse tram-list eval-list
                       exhaustive dont-repeat self)
  ;;
  ;;  First try normal recall.
  ;;
  (let ((recall (ksm:recall memory *rspec* nil nil nil)))
    ;;
    ;;  If recalled episode was too general, or nothing was recalled,
    ;;  then try applying creativity.
    ;;
    (if (eq recall :too-general) (setq recall nil))
    (if (and (null recall)
             (integerp recurse)
             (> recurse 0)
             (not (member :never-recurse
                          (tram:tram-structure-options self))))
        (progn
          (trace:cformat *tram-trace* "[TRAM Recursion: %a.]\n" *rspec*)
          (setq recall (tram:really-execute *rspec* memory
                                    no-evaluate
                                    random-selection
                                    random-tram
                                    (- recurse 1)
                                    tram-list
                                    nil            ;; no evals during recursion
                                    exhaustive
                                    dont-repeat))))
    recall))
```

Both TRAM:DO-RECALL and TRAM:REALLY-EXECUTE take a large number of arguments which control the behavior of imaginative memory. The use of these arguments isn't important; more important is the general flow of control. TRAM:DO-RECALL first tries to recall something from episodic memory (KSM:RECALL). If that fails, TRAM:DO-RECALL tries to invent something useful (TRAM:REALLY-EXECUTE).

The actual application of TRAMs to the problem is done by TRAM:REALLY-EXECUTE.

TRAM:REALLY-EXECUTE is a long but straightforward function. It collects the applicable TRAMS from the TRAM pool and then applies them one at a time to the current problem. If a TRAM succeeds in recalling something, then the function returns the recalled episodes:

```
(defun really-execute
    (*spec* memory no-evaluate random-selection random-tram
            recurse tram-list eval-list exhaustive dont-repeat)
  ;;
  ;;  If we weren't given a specific tram-list, then use the default one.
  ;;
  (if (null tram-list)
      (setq tram-list tram:*tram-list*))
  ;;
  ;;  If we weren't given a specific eval-list, then use the default one.
  ;;
  (if (null eval-list)
      (setq eval-list eval:*evaluation-list*))
  ;;
  ;;  The loop
  ;;
  (do ((tram-pool (if random-tram
                      (randomize-tram tram-list)
                      tram-list))
       (current-tram nil)
       (results nil nil)
       (accepted nil))

    ;;
    ;;  Execution is done if results have been obtained.  If
    ;;  exhaustive is set, the execution isn't done until we run out
    ;;  of TRAMs to try.  Other possible completions return out with
    ;;  return-from.
    ;;

    ((or (and exhaustive (null tram-pool))
         (and (not exhaustive) accepted))
     (if random-selection (setq accepted (util:randomize-list accepted)))
     accepted)
    ;;
    ;;  Find the next TRAM that isn't in the dont-repeat list and put
    ;;  it into the current-tram.
    ;;
    (setq current-tram
          (do ((ntram (pop tram-pool) (pop tram-pool)))
              ((not (member ntram dont-repeat)) ntram)))
    ;;
    ;;  If we've run out of TRAMs w/o a result, we're done.
    ;;
    (if (null current-tram)
        (progn
          (trace:pop)
          (return-from tram:really-execute nil)))
    ;;
    ;;  If the current-tram has the :dont-repeat option, then add it to
    ;;  the dont-repeat list.
    ;;
```

```
(if (member :dont-repeat (tram:tram-structure-options current-tram))
    (push current-tram dont-repeat))
;;
;;  Announce the current TRAM.
;;
(trace:cformat *tram-verbose* "Trying Recall-Adapt-Transformation %a.\n"
               (tram:tram-structure-name current-tram))
;;
;;  Print out the comment.
;;
(dolist (c (tram:tram-structure-comment current-tram))
  (trace:cformat *tram-verbose* "  %s\n" c))
;;
;;  Execute Step
;;
;;  The transform, recall and adapt portions of the TRAM are all
;;  encapsulated in the TRAM body, so execute that:
;;
;;
(setq results (apply (tram:tram-structure-body current-tram)
                     *spec* memory no-evaluate random-selection
                     random-tram recurse tram-list eval-list
                     exhaustive
                     dont-repeat current-tram nil))
;;
;;  Evaluation Step
;;
;;
;;  If evaluations aren't turned on, then skip the evaluation step
;;  and simply accept the results.
;;
(if (or no-evaluate (null eval-list))
    (progn
      (setq accepted (append accepted results))
      (go continue)))

;;
;;  Otherwise, take each episode in result and apply evaluation
;;  rules to it until you find one that returns true (i.e., passes
;;  the episode).  If you don't find an evaluation that accepts
;;  the episode, leave it out.
;;
;;  For each episode in result
;;
(do ((try results (cdr try))
     (current nil))

    ((null try))

  (setq current (car try))

  (trace:cformat *tram-verbose* "Evaluating %a.\n" current)

  ;;
  ;;  For each evaluation
  ;;
  (do ((evls eval-list (cdr evls))
```

```
          (evl nil)
          (evl-result nil))

         ((or evl-result (null evls))
          (if evl-result
              ;;
              ;;  If it was accepted, save it as accepted.
              ;;
              (progn
                (push current accepted)
                (trace:cformat *tram-verbose* "%a accepted by %a.\n"
                               current (eval:evaluation-structure-name evl)))
              ;;
              ;;  If it wasn't accepted, you can free it.
              ;;
              (rhapsody:free current)))

        (setq evl (car evls))

        ;;
        ;;  Announce the evaluation.
        ;;
        (trace:cformat *tram-verbose* "Applying evaluation rule %a.\n"
                       (eval:evaluation-structure-name evl))

        ;;
        ;;  Try this evaluation on this result.
        ;;
        (setq evl-result
              (apply (eval:evaluation-structure-test evl) *spec* current nil))

        ;;
        ;;  Announce the result.
        ;;
        (if evl-result
            (trace:cformat *tram-verbose* "Evaluation rule succeeded.\n")
            (trace:cformat *tram-verbose* "Evaluation rule failed.\n"))
        )
      )

    ;;
    ;;  This is where we go to continue the loop.
    ;;
    continue
))
```

Note that the Execute step of TRAM:REALLY-EXECUTE applies the body of the selected TRAM. The body of each TRAM is a lambda expression containing the test, transform, recall, and adapt portions of the TRAM. A pertinent detail here is that the recall portion of each TRAM expands into a call to TRAM:DO-RECALL:

```
(tram:do-recall *rspec* ...)
```

This is the source of MINSTREL's ability to apply multiple TRAMs to a single problem. Each time a TRAM is applied, there is a recursive call to TRAM:DO-RECALL, which may result in

applying another TRAM. This process halts when a solution is found or when a TRAM marked with :dont-recurse (i.e., TRAM:Standard-Problem-Solving) is encountered.

B.5 Episodic Memory

In the context-plus-index model of episodic memory, an episode is recalled by specifying a context and a combination of feature/value pairs that uniquely identify the episode. [Kolodner 1982] modelled this behavior by organizing episodic memory as a difference tree. Contexts were represented as frames, and each episode in memory was indexed in a tree of features associated with the frame of the episode. For example, a "Knightly Fight" frame which had the features "weapon is a sword", "opponent is a troll" and "opponent is killed" would be indexed by these features under the "Knightly Fight" frame. An example of this organization is shown in Figure B.1.

In the Kolodner organization, an episode is indexed according to all the combinations of its features which uniquely specify the episode. In the above example, EV1 is indexed under "opponent is a troll", "outcome is victory", "weapon is a sword and opponent is a troll" and "weapon is a sword and outcome is victory". Each of these sets of features uniquely specifies EV1, distinguishing it from all the other episodes which share the same context (in this example, EV2).

To recall an episode from this difference tree, target feature/value pairs are used to traverse the tree downwards from the root until an episode is encountered. For example, the features "weapon is a sword" and "opponent is a troll" can be used to follow a branch downward from the root to recall episode EV1.

Although the difference tree organization reflects in a straightforward manner the organization and behavior of episodic memory, it has the disadvantage of requiring a great deal of space. To avoid this problem, MINSTREL organizes episodes using two-level hash tables.

In this organization, there are two levels of hash tables associated with each frame. The top level hash table is called the feature table, and contains the features associated with the frame. Each feature in the feature table indexes a second hash table, called the feature value table. The feature value table contains values for the corresponding feature, and each value in the feature value table indexes a list of episodes containing that feature/value pair.

For example, the feature table associated with the "Knightly Fight" frame contains the features for that frame, such as "type of weapon", "type of opponent" and "outcome of the fight". Each of these features indexes a feature value table containing values for these features, such as "sword" or "lance" for the "type of weapon" feature. (These are not all the possible values for that feature, but rather the values for that feature that have been encountered by episodic memory.) Each of these feature values indexes the episodes containing that value. For example, indexed under the "type of weapon"/"sword" pair are all the episodes in memory in which a knight fought someone using a sword.

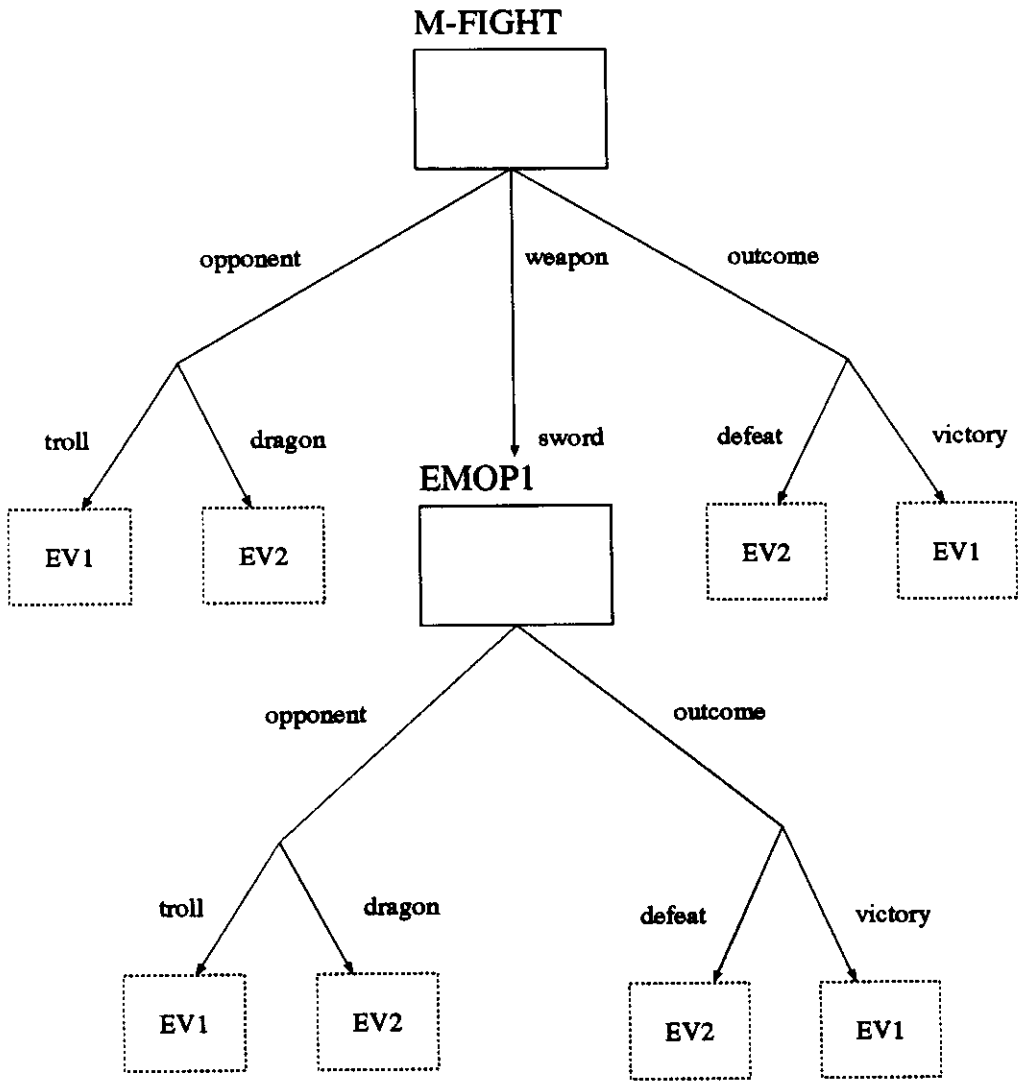


Figure B.1 Tree Structure of Episodic Memory

Figure B.2 shows how the difference tree illustrated in Figure B.1 would be represented using MINSTREL's two-layer hash tables.

In the difference tree representation, an episode is recalled by using the target feature values to traverse the memory tree, stopping when a specific episode is found. In MINSTREL's representation, episodic memory is traversed by using all of the target feature/value pairs as indices simultaneously and then intersecting the resulting lists of episodes. The result of this intersection is the set of episodes containing all of the target feature/value pairs.

For example, to recall an episode using the features "weapon is a sword" and "opponent is a

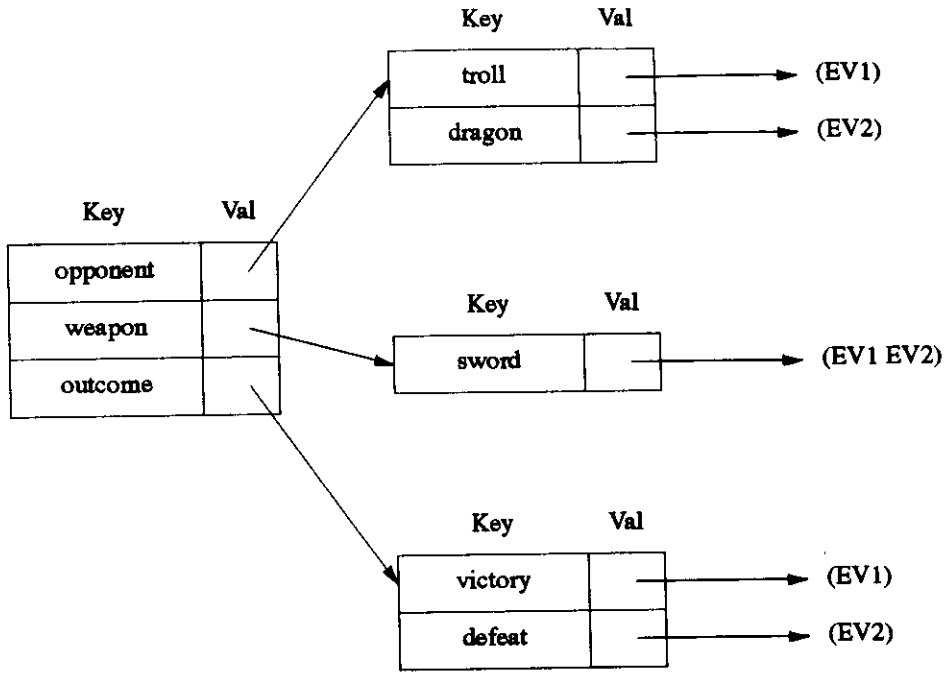


Figure B.2 Example MINSTREL Organization

troll", MINSTREL uses the first feature to find the episode list (EV1 EV2) and then intersects this list with the list found using the second feature, (EV1).

The MINSTREL implementation of episodic memory has several advantages over the more obvious tree structure representation.

First, the MINSTREL implementation is more space efficient than a tree representation. A straightforward implementation of difference trees can result in explosive space requirements as new episodes are indexed, primarily because every permutation of the feature set must be indexed separately. MINSTREL's implementation requires space porportional to the number of feature values.

Second, MINSTREL's implementation of episodic memory is more time efficient than a tree representation because it can be implemented using parallel processing. During memory search, each feature/value pair from a set of target features can be treated independently, resulting in a speed-up roughly porportional to the number of target features. (However, the current version of MINSTREL does not use parallel processing in this way.)

MINSTREL's recall process is implemented primarily in two functions: KSM:FIND-PAT and

KSM:RECALL. KSM:FIND-PAT implements a normal search of episodic memory without elaboration. KSM:FIND-PAT succeeds if there is an episode that matches the recall index exactly. KSM:RECALL implements elaboration. In elaboration, a recall index that is too general to recall any specific episode has new features added to it until a specific episode is recalled (see Chapter 2 for a more detailed explanation of elaboration).

KSM:FIND-PAT takes two arguments:

ksm The root of the episodic memory tree from which to recall. Having this as a parameter permits MINSTREL to have several separate episodic memory trees, but currently MINSTREL uses only one.

pat The schema representing the recall indices.

KSM:FIND-PAT collects all the feature/value pairs from the recall pattern, and then uses each pair to index memory. As the episodes index by each pair are recalled they are intersected to get only the episodes index by all the feature/value pairs. The result is the recalled episode(s):

```
(defun find-pat (ksm pat)
    ;;
    ;;  LOFS is a list of the slots and links in pat which have values
    ;;  (i.e., the features of the recall index).
    ;;
      (let* ((x (class:prop (inst:class pat) ksm))
             (lofs (intersection
                     (nconc (inst:map-slots pat
                                  #'(lambda (slot val)
                                      (declare (ignore val))
                                      slot))
                            (inst:map-links pat
                                  #'(lambda (link val)
                                      (declare (ignore val))
                                      link)))
                     (car x)))
             ;;
             ;;  HT is the top-level feature hash table of episodic
             ;;  memory.
             ;;
             (ht (cdr x))
             (first t)
             (current nil))
        ;;
        ;;  Now for each slot/feature pair in LOFS, find all the
        ;;  episodes in memory indexed by that pair.  As you do this,
        ;;  keep intersecting the lists of episodes, so that you find
        ;;  only the episodes that are indexed by every slot/feature
        ;;  pair in LOFS.
        ;;
        (block early
          (mapc #'(lambda (l)
                    (cond
                      ((and (not first) (null current))
                       (return-from find-pat :too-specific))
```

```
                ((inst:slot pat 1)
                 (let ((kgv (ksm-get-vals (gethash 1 ht)
                                          (inst:slot pat 1) ksm)))
                   (if (null kgv)
                       (trace:cformat *recall-verbose*
                           "Null return on %a.\n" 1))
                   (if first
                       (block nil
                         (setq current kgv)
                         (setq first nil))
                       (setq current (intersection current kgv)))))
                ((inst:link pat 1)
                 (let ((kgv (ksm-get-vals (gethash 1 ht)
                                          (car (inst:link pat 1)) ksm)))
                   (if (null kgv)
                       (trace:cformat *recall-verbose*
                           "Null return on %a.\n" 1))
                   (if first
                       (block nil
                         (setq current kgv)
                         (setq first nil))
                       (setq current (intersection current kgv)))))))
        lofs))
(cond
  ;;
  ;;  If you used the whole list and found nothing, the index
  ;;  is too general.
  ;;
  ((null lofs) :too-general)
  ;;
  ;;  If the intersections are null, the index is too specific.
  ;;
  ((null current) :too-specific)
  ;;
  ;;  If there are too many episodes indexed by pat, then the
  ;;  index is also too general.
  ;;
  ((> (length current) (ksm-len ksm)) :too-general)
  ;;
  ;;  Otherwise, return the recalled episodes.
  ;;
  (t current)))))
```

KSM:RECALL implements elaboration. Elaboration in CYRUS [Kolodner 1984] used domain-specific, knowledge-based heuristics, such as map-walking and knowledge about state dinners, that did not directly examine the structure of episodic memory. This reflected human protocols which indicated that people cannot directly elaborate the feature values of a context. However, elaboration is not a primary concern in MINSTREL, so MINSTREL uses an algorithm for elaboration which directly inspects episodic memory to generate new feature/value pairs for elaboration.

The recall process is implemented in a function call KSM:RECALL. KSM:RECALL takes three arguments:

ksm The root of the episodic memory tree from which to recall. Having this as a parameter permits MINSTREL to have several separate episodic memory trees, but currently MINSTREL uses only one.

pat The schema representing the recall indices.

choose-random Flag to indicate whether or not to choose randomly during recall. If NIL, MINSTREL always returns the first episode when the recall process finds more than one episode that matches **pat**. When T, MINSTREL selects randomly from the recalled episodes.

KSM:RECALL takes the result of a call to KSM:FIND-PAT and, if it is too general, repeatedly elaborates it until an episode is recalled or elaboration fails. To elaborate the recall pattern, KSM:RECALL gathers a list of all the possible elaborations by examining memory. This list is then used in a straightforward manner to elaborate the recall index.

```
(defun recall (ksm pat choose-random)
  ;;
  ;;  Print out debugging info if *recall-trace* is true.
  ;;
  (trace:cformat *recall-trace* "Recalling %a: " pat)
  ;;
  ;;  Get the list of possible elaborations for this schema class.
  ;;
  (let ((sl (ksm:specifics ksm (inst:class pat))))
    ;;
    ;;  If we are choosing at random, randomize the elaboration list.
    ;;
    (if choose-random (setq sl (util:randomize-list sl)))
    (labels
      ;;
      ;;  DOIT is an internal function used to handle each new
      ;;  elaboration.
      ;;
      ((doit (val sl)
             (cond
               ;;
               ;;  If the recall index is too specific, return nil.
               ;;
               ((eq val :too-specific) nil)
               ;;
               ;;  If the recall index is too general, we can try
               ;;  specializing.
               ;;
               ((eq val :too-general)
                (if (null sl)
                  ;;
                  ;;  There's nothing left in the elaboration list, so
                  ;;  this index really is too general.
                  ;;
                  :too-general
                  ;;
                  ;;  Otherwise, try using the top elaboration.
                  ;;
                  (let ((first-vals (list (pop sl))))
```

```
                ;;
                ;;  Collect all the possible specializations for
                ;;  this feature.
                ;;
                (do ()
                    ((not (eq (caar sl) (caar first-vals))))
                    (push (pop sl) first-vals))
                (setq first-vals (nreverse first-vals))
                ;;
                ;;  Now try each specialization in turn, until
                ;;  we find one that works:
                ;;
                (do ((fv first-vals (cdr fv)))
                    ((null fv) :too-general)
                    ;;
                    ;;  FV is the current specialization.  Add
                    ;;  it to the recall indices.
                    ;;
                    (if (link? (caar fv))
                        (inst:add-link pat (caar fv)
                                       (ksm::slot-copy (cdar fv)))
                      (setf (inst:slot pat (caar fv)) (cdar fv)))
                    (trace:cformat *recall-trace*
                                   "Specializing ~a to ~a.\n"
                                   (caar fv) (cdar fv))
                    ;;
                    ;;  Now recurse and see if the new,
                    ;;  specialized recall indices recall
                    ;;  anything.
                    ;;
                    (doit (ksm:find-pat ksm pat) sl)
                    ;;
                    ;;  If DOIT returns, recall failed, so take
                    ;;  out the specialization and try the next
                    ;;  one.
                    ;;
                    (trace:cformat *recall-trace*
                                   "Generalizing ~a.\n"
                                   (caar fv))
                    (if (link? (caar fv))
                        (inst:delete-links pat (caar fv))
                      (setf (inst:slot pat (caar fv)) nil))))))
            ;;
            ;;  If val is neither :too-specific or :too-general,
            ;;  then it is a recalled episode, and we can return
            ;;  out of the high-level function call, KSM:RECALL
            ;;  with the recalled episodes.
            ;;
            (t
              (trace:cformat *recall-trace* "%a.\n" val)
              (return-from recall val)))))
;;
;;  Start things off by using KSM:FIND-PAT to look as far down
;;  in memory as the original pattern allows.  Then use DOIT to
;;  specialize that as necessary.
;;
(progl
```

```
    (doit (ksm:find-pat ksm pat) sl)
    (trace:cformat *recall-trace* "Recall failed.\n"))
))))
```

B.6 Initial Knowledge Structures

The previous sections showed MINSTREL's control structure, illustrating how control flows from the application of author-level plans through creativity to episodic memory. We'll now look at the initial knowledge structures that MINSTREL has when storytelling begins.

MINSTREL's initial knowledge structures can be divided into three classes:

1. Storytelling Knowledge (Author-Level Plans, or ALPs)

2. Creativity Knowledge (Transform-Recall-Adapt Methods, or TRAMs)

3. Domain Knowledge (King Arthur domain knowledge)

All of these knowledge structures are organized in MINSTREL's episodic memory. Examples of ALPs and TRAMs were given in the previous sections. In this section we'll examine MINSTREL's initial knowledge of the King Arthur domain.

When MINSTREL begins storytelling, it has indexed in episodic memory twelve simple story scenes. These scenes are set in the King Arthur domain, as if MINSTREL had previously read some King Arthur stories (or story fragments). In English, these episodes are:

A man had a goal to scare another man.

A dragon moved toward a princess to eat her, and the princess was scared.

Juliet drank a magic potion that made her appear dead.

A princess picked berries in the woods.

To impress the king, a knight fought and killed a dragon. The knight was wounded.

A knight fought a dragon and broke his sword.

A hermit healed a knight.

A knight rode a horse to the woods.

A troll killed a knight.

A knight has his love for a princess thwarted.

A knight kisses a princess to win her love.

A knight kisses his brother the hermit to express affection.

These story scenes capture all that MINSTREL knows about the King Arthur domain. Everything that MINSTREL puts in its stories about the King Arthur domain is invented from this knowledge.

These story scenes are represented by schemas as detailed in Chapter 2. In MINSTREL's initialization files, these schemas are expressed in list notation. The following shows the representation of the scene "A knight fought a dragon and broke his sword" as it appears in the file that initializes MINSTREL's episodic memory:

```
;;;
;;;  A scene where a knight breaks a sword in a battle.
;;;

;;;
;;;  The Knight
;;;
(human 'hans
       :name 'hans
       :sex 'male
       :type '&knight)

;;;
;;;  The Monster
;;;
(monster 'hans-monster)

;;;
;;;  The Sword
;;;
(physobj 'hans-sword
       :type &sword)

;;;
;;;  Where
;;;
(setting 'hans-setting)

;;;
;;;  The knight's goal of possessing his sword.
;;;
(goal 'hans-goal
      :actor &hans
      :type &possess
      :new-state (state nil
                        :type &possess
                        :object '&hans-sword))
;;;
;;;  The sword is broken, thwarting the knight's goal.
;;;
```

```
(state 'lost-sword
       :value &dead
       :type '&health
       :object '&hans-sword
       '&thwarts '&hans-goal)

;;;
;;;  The fight that unintentionally results in the broken sword.
;;;
(act 'fight2
       :type '&m-fight
       :actor '&hans
       :to '&hans-monster
       :setting '&hans-setting
       '&unintended '&lost-sword)
```

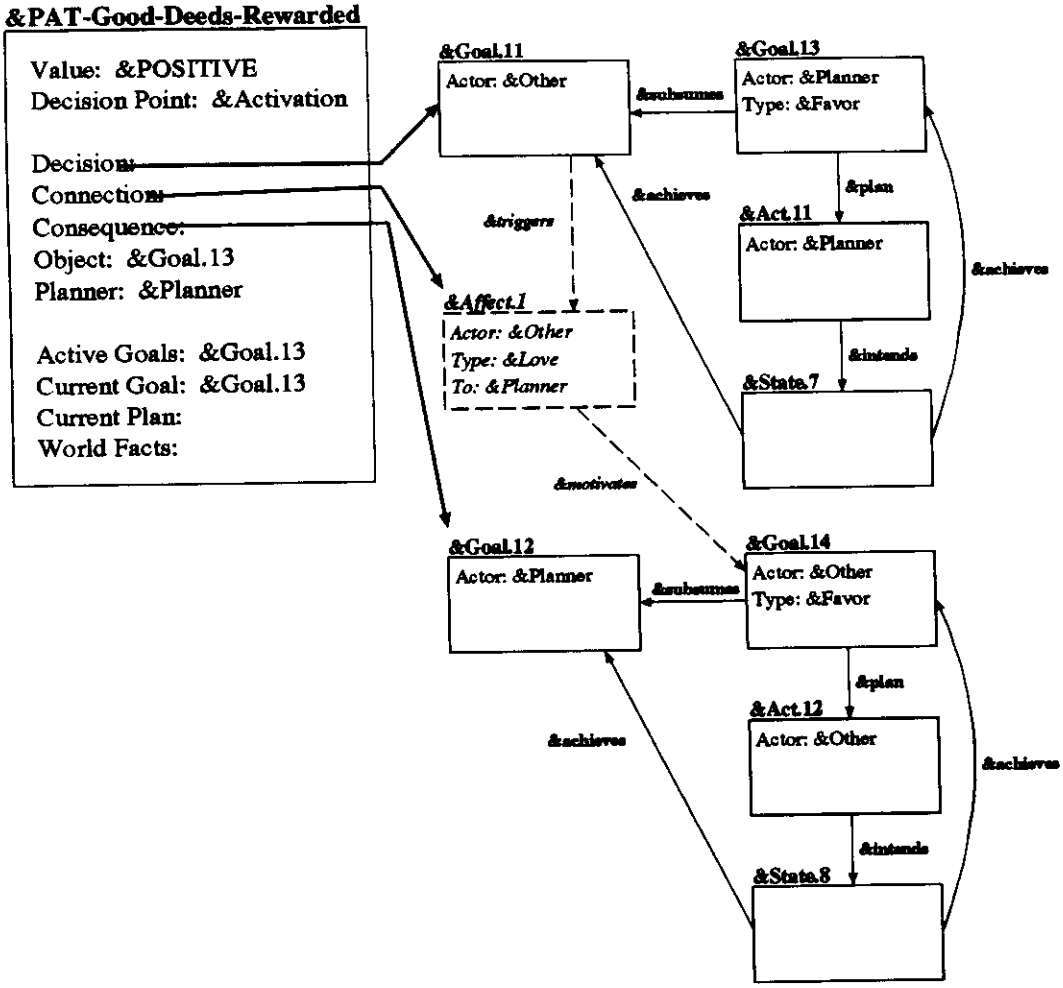
In addition to this knowledge about the King Arthur domain, MINSTREL begins storytelling with some knowledge about six themes: PAT:Good-Deeds-Rewarded, PAT:Bird-In-Hand, PAT:Spite-Face, PAT:Juliet, PAT:Hasty-Impulse-Regretted, and PAT:Pride. . Themes are story structures that organize a story and provide a "point" or moral for the story.

Like MINSTREL's representation of knowledge from the King Arthur domain, MINSTREL's themes are represented as schemas. These schemas are called Planning Advice Themes, or PAT. Each PAT represents a stereotypical planning decision and advice concerning that decision. The planning situation is represented by goal, plan and state schemas. Figure B.3 shows the representation for PAT:Good-Deeds-Rewarded.

Each of the themes that MINSTREL knows are represented in list notation in the file that initializes MINSTREL's episodic memory. (Like all other knowledge structures in MINSTREL, themes are organized and indexed by episodic memory.) The following shows the list notation for PAT:Good-Deeds-Rewarded[1]:

```
;;;
;;; PAT:GOOD-DEEDS-REWARDED
;;;
;;; "Do good deeds for others because someday they may return the
;;;  favor."
;;;

;;;
;;;  The Planner
;;;
(human 'planner nil)

;;;
;;;  Other
;;;
(human 'other nil)

;;;
```

1. The schemas in this example have been renamed to correspond with the schema names shown in Figure B.3 .



Summaries	
Decision:	&Planner does a favor for &Other, helping him achieve a goal.
Connection:	&Other loves &Planner for his good deed.
Consequence:	&Other returns &Planner's good deed.

Figure B.3 PAT:Good-Deeds-Rewarded

```
;;;  The Decision
;;;
;;;  Planner does a favor for other.
;;;
(goal 'goal.11
      :actor &other)

(goal 'goal.13
      :actor &planner
```

```
      :type &opportunistic-favor
      &subsumes &goal.11)

(act 'act.11
      :actor &planner
      &intends (state 'state.7 &achieves &goal.11)
      &plan-of &goal.13)

;;;
;;;  The Consequence
;;;
;;;  Other does a favor for planner.
;;;
(goal 'goal.12
      :actor &planner)

(goal 'goal.14
      :actor &other
      :type &opportunistic-favor
      &subsumes &goal.11)

(act 'act.12
      :actor &planner
      &intends (state 'state.8 &achieves &goal.11)
      &plan-of &goal.12)

;;;
;;;  The Connection
;;;
(state 'affect.1
       :value &love
       :object &other
       :to &planner)

;;;
;;;  The Planning Advice Theme
;;;
(pat 'good-deeds-rewarded
     :value &positive
     :decision-point &goal-activation
     :planner &planner
     :decision &goal.11
     :consequence &goal.12
     :connection (list (list &goal.11 &trigger &affect.1)
                       (list &affect.1 &motivates &goal.14))
     :active-goals &goal.13
     :current-goal &goal.13)
```

B.7 Final Knowledge Structures

The stories MINSTREL creates are represented as complex webs of schemas. After creating each story, each individual schema in the story is added to episodic memory, increasing MINSTREL's knowledge.

To give the reader an idea of the scope and complexity of MINSTREL's final stories, we reproduce here the schema representation of *The Mistaken Knight*:

```
(STORY 'STORY.1
  :ENDING-SCENES  (list &STATE.24494 &STATE.24463 &STATE.24462)
  :ESTABLISHING-SCENES  (list &STATE.24407 &STATE.24406 &GOAL.10608)
  :THEME  '&TAU.8 )

(STATE 'STATE.24494
  :VALUE  &HERMIT
  :TYPE  &ROLE-CHANGE
  :OBJECT  '&HUMAN.138 )

(HUMAN 'HUMAN.138
  :SEX  'MALE
  :NAME  'LANCELOT
  :TYPE  &KNIGHT)

(STATE 'STATE.24463
  :VALUE  '&SETTING.3724
  :OBJECT  '&HUMAN.274
  :TYPE  &BURIED)

(SETTING 'SETTING.3724
  :TYPE  &WOODS)

(HUMAN 'HUMAN.274
  :NAME  'FREDERICK
  :TYPE  &KNIGHT)

(STATE 'STATE.24462
  :VALUE  &HERMIT
  :TYPE  &ROLE-CHANGE
  :OBJECT  '&HUMAN.137 )

(HUMAN 'HUMAN.137
  :TYPE  &PRINCESS
  :NAME  'ANDREA)

(STATE 'STATE.24407
  :VALUE  &SPRING
  :OBJECT  '1089
  :TYPE  &DATE)

(STATE 'STATE.24406
  :VALUE  '&SETTING.3712
  :OBJECT  '&HUMAN.138
  :TYPE  &LOC
```

```
  &SUPERSEDES  '&STATE.24405 )

(STATE 'STATE.24405
  :VALUE  '&SETTING.3713
  :OBJECT  '&HUMAN.138
  :TYPE  &LOC
  &SUPERSEDED-BY  '&STATE.24406 )

(SETTING 'SETTING.3713
  :TYPE  &ELSEWHERE)

(SETTING 'SETTING.3712
  :TYPE  &CAMELOT)

(GOAL 'GOAL.10608
  :TYPE  &DESTROY
  :ACTOR  '&HUMAN.138
  :OBJECT  '&PHYSOBJ.736
  &ACHIEVED-BY  '&STATE.17721
  &PLAN  '&ACT.9818
  &MOTIVATED-BY  '&STATE.17722
  &MOTIVATED-BY  '&STATE.17716 )

(STATE 'STATE.17721
  :TYPE  &HEALTH
  :VALUE  &DEAD
  :OBJECT  '&PHYSOBJ.736
  &ACHIEVES  '&GOAL.10608
  &INTENDED-BY  '&ACT.9818 )

(ACT 'ACT.9818
  :TYPE  &HIT
  :ACTOR  '&HUMAN.138
  :OBJECT  '&PHYSOBJ.736
  &INTENDS  '&STATE.17721
  &PLAN-OF  '&GOAL.10608 )

(PHYSOBJ 'PHYSOBJ.736
  :TYPE  &SWORD)

(STATE 'STATE.17722
  :TYPE  &FAILURE
  :OBJECT  '&HUMAN.138
  :VALUE  &JOUST
  &MOTIVATES  '&GOAL.10608 )

(STATE 'STATE.17716
  :TYPE  &TEMPER
  :OBJECT  '&HUMAN.138
  :VALUE  &HOT
  &MOTIVATES  '&GOAL.10608 )

(PAT 'PAT.8
  :CONNECTION  (list &GOAL.226 '&THWARTED-BY &STATE.204
                     &STATE.206 '&SUPERSEDES &STATE.202)
  :WORLD  (list &STATE.203)
  :DECISION  '&BELIEF.28
```

```
  :CONSEQUENCE  '&BELIEF.29
  :PLANNER  '&HUMAN.138
  :DECISION-POINT  &PLAN-SELECTION
  :VALUE  &NEGATIVE)

(GOAL 'GOAL.226
  :OBJECT  '&GOAL.225
  :TYPE  &RETRACT
  :ACTOR  '&HUMAN.138
  &REACTION  '&STATE.4991
  &THWARTED-BY  '&STATE.204
  &B-MOTIVATED  '&BELIEF.29 )

(STATE 'STATE.4991
  :VALUE  &DISLIKE
  :TO  '&HUMAN.138
  :OBJECT  '&HUMAN.138
  :TYPE  &AFFECT
  &TRIGGER  '&GOAL.226 )

(STATE 'STATE.204
  :VALUE  &DEAD
  :TYPE  &HEALTH
  :OBJECT  '&HUMAN.274
  &THWARTS  '&GOAL.226
  &ACHIEVES  '&GOAL.225
  &INTENDED-BY  '&ACT.118 )

(GOAL 'GOAL.225
  :OBJECT  '&HUMAN.274
  :TYPE  &DESTROY
  :NEW-STATE  '&STATE.204
  :ACTOR  '&HUMAN.138
  &PLAN  '&ACT.118
  &ACHIEVED-BY  '&STATE.204
  &SUBGOAL-OF  '&GOAL.224
  &MOTIVATED-BY  '&STATE.202
  &MOTIVATED-BY  '&STATE.203 )

(ACT 'ACT.118
  :AT  '&SETTING.43
  :TO  '&HUMAN.274
  :TYPE  &M-FIGHT
  :ACTOR  '&HUMAN.138
  &PRECOND  '&STATE.6782
  &PLAN-OF  '&GOAL.225
  &INTENDS  '&STATE.204 )

(STATE 'STATE.6782
  :TYPE  &LOC
  :OBJECT  '&HUMAN.138
  :VALUE  '&HUMAN.274
  &ACHIEVES  '&GOAL.4286
  &INTENDED-BY  '&ACT.3262
  &PRECOND-OF  '&ACT.118 )

(GOAL 'GOAL.4286
```

```
  :OBJECT  '&HUMAN.274
  :TYPE  &D-LOC
  :ACTOR  '&HUMAN.138
  :NEW-STATE  '&STATE.6782
  &ACHIEVED-BY  '&STATE.6782
  &PLAN  '&ACT.3262 )

(ACT '&ACT.3262
  :TYPE  &PTRANS
  :ACTOR  '&HUMAN.138
  :OBJECT  '&HUMAN.138
  :TO  '&HUMAN.274
  &PLAN-OF  '&GOAL.4286
  &INTENDS  '&STATE.6782 )

(SETTING '&SETTING.43
  :TYPE  &LAIR)

(GOAL '&GOAL.224
  :NEW-STATE  '&STATE.207
  :ACTOR  '&HUMAN.138
  :TYPE  &A-LOVE
  :OBJECT  '&HUMAN.137
  &REACTION  '&STATE.1478
  &MOTIVATED-BY  '&STATE.1470
  &SUBGOAL  '&GOAL.225
  &THWARTED-BY  '&STATE.202 )

(STATE '&STATE.1478
  :VALUE  &HATE
  :TO  '&HUMAN.274
  :OBJECT  '&HUMAN.138
  :TYPE  &AFFECT
  &TRIGGER  '&GOAL.224 )

(STATE '&STATE.1470
  :OBJECT  '&HUMAN.138
  :TO  '&HUMAN.137
  :TYPE  &AFFECT
  :VALUE  &LOVE
  &CAUSE  '&ACT.673
  &MOTIVATES  '&GOAL.224 )

(ACT '&ACT.673
  :AT  '&SETTING.215
  :TO  '&SETTING.215
  :TYPE  &PTRANS
  :ACTOR  '&ANIMAL.5
  :OBJECT  '&HUMAN.138
  &PRECOND  '&STATE.2479
  &UNINTENDED  '&STATE.1470
  &UNINTENDED  '&STATE.1800 )

(STATE '&STATE.2479
  :VALUE  '&SETTING.215
  :OBJECT  '&HUMAN.137
  :TYPE  &LOC
```

```
  &ACHIEVES  '&GOAL.1786
  &INTENDED-BY  '&ACT.1036
  &PRECOND-OF  '&ACT.673 )

(GOAL 'GOAL.1786
  :TYPE  &D-LOC
  :ACTOR  '&HUMAN.137
  :OBJECT  '&SETTING.215
  :NEW-STATE  '&STATE.2479
  &SUBGOAL-OF  '&GOAL.3125
  &ACHIEVED-BY  '&STATE.2479
  &PLAN  '&ACT.1036 )

(GOAL 'GOAL.3125
  :TYPE  &D-CONT
  :OBJECT  '&PHYSOBJ.48
  :ACTOR  '&HUMAN.137
  &ACHIEVED-BY  '&STATE.5074
  &PLAN  '&ACT.2301
  &SUBGOAL  '&GOAL.1786 )

(STATE 'STATE.5074
  :TYPE  &POSSESS
  :OBJECT  '&HUMAN.137
  :VALUE  '&PHYSOBJ.48
  &ACHIEVES  '&GOAL.3125
  &INTENDED-BY  '&ACT.2301 )

(ACT 'ACT.2301
  :AT  '&SETTING.215
  :TYPE  &ATRANS
  :ACTOR  '&HUMAN.137
  :OBJECT  '&PHYSOBJ.48
  :FROM  '&SETTING.215
  :TO  '&HUMAN.137
  &INTENDS  '&STATE.5074
  &PLAN-OF  '&GOAL.3125 )

(SETTING 'SETTING.215
  :TYPE  &WOODS)

(PHYSOBJ 'PHYSOBJ.48
  :TYPE  &BERRIES)

(ACT 'ACT.1036
  :AT  '&SETTING.215
  :TYPE  &PTRANS
  :ACTOR  '&HUMAN.137
  :OBJECT  '&HUMAN.137
  :TO  '&SETTING.215
  &PLAN-OF  '&GOAL.1786
  &INTENDS  '&STATE.2479 )

(STATE 'STATE.1800
  :VALUE  '&HUMAN.137
  :OBJECT  '&HUMAN.138
  :TYPE  &LOC
```

```
  &SUPERSEDES  '&STATE.1819
  &CAUSE  '&ACT.673 )

(STATE 'STATE.1819
  :TYPE  &LOC
  :VALUE  '&SETTING.194
  :OBJECT  '&HUMAN.138
  &SUPERSEDED-BY  '&STATE.1800
  &ACHIEVES  '&GOAL.1394
  &INTENDED-BY  '&ACT.692 )

(GOAL 'GOAL.1394
  :OBJECT  '&SETTING.194
  :TYPE  &D-LOC
  :NEW-STATE  '&STATE.1819
  :ACTOR  '&HUMAN.138
  &ACHIEVED-BY  '&STATE.1819
  &PLAN  '&ACT.692 )

(ACT 'ACT.692
  :AT  '&SETTING.194
  :TYPE  &PTRANS
  :OBJECT  '&HUMAN.138
  :TO  '&SETTING.194
  :ACTOR  '&HUMAN.138
  &PLAN-OF  '&GOAL.1394
  &INTENDS  '&STATE.1819 )

(SETTING 'SETTING.194
  :TYPE  &WOODS)

(ANIMAL 'ANIMAL.5
  :TYPE  &HORSE)

(STATE 'STATE.202
  :VALUE  &LOVE
  :TO  '&HUMAN.274
  :OBJECT  '&HUMAN.137
  :TYPE  &AFFECT
  &SUPERSEDED-BY  '&STATE.206
  &MOTIVATES  '&GOAL.225
  &EVENT-IN  '&BELIEF.28
  &THWARTS  '&GOAL.224 )

(STATE 'STATE.206
  :TYPE  &RELATION
  :OBJECT  '&HUMAN.137
  :TO  '&HUMAN.274
  :VALUE  &SIBLINGS
  &MOTIVATES  '&GOAL.2808
  &SUPERSEDES  '&STATE.202
  &EVENT-IN  '&BELIEF.29 )

(GOAL 'GOAL.2808
  :TYPE  &A-AFFECTION
  :ACTOR  '&HUMAN.137
  :OBJECT  '&HUMAN.274
```

```
  &ACHIEVED-BY  '&STATE.4426
  &MOTIVATED-BY  '&STATE.206
  &PLAN  '&ACT.1969 )

(STATE 'STATE.4426
  :TYPE  &KISS
  :OBJECT  '&HUMAN.137
  :TO  '&HUMAN.274
  &ACHIEVES  '&GOAL.2808
  &INTENDED-BY  '&ACT.1969 )

(ACT 'ACT.1969
  :ACTOR  '&HUMAN.137
  :TYPE  &KISS
  :OBJECT  '&HUMAN.274
  &INTENDS  '&STATE.4426
  &PLAN-OF  '&GOAL.2808 )

(BELIEF 'BELIEF.29
  :MODE  &DEDUCTIVE
  :ACTOR  '&HUMAN.138
  &B-MOTIVATES  '&GOAL.226
  &MENTAL-EVENT  '&STATE.206
  &EVIDENCE  '&STATE.205 )

(STATE 'STATE.205
  :TO  '&HUMAN.138
  :OBJECT  '&STATE.4915
  :TYPE  &KNOW
  &INTENDED-BY  '&ACT.2218
  &EVIDENCE-FOR  '&BELIEF.29 )

(ACT 'ACT.2218
  :AT  '&HUMAN.137
  :TO  '&HUMAN.138
  :OBJECT  '&STATE.4915
  :ACTOR  '&HUMAN.137
  :TYPE  &MTRANS
  &PRECOND  '&STATE.7279
  &INTENDS  '&STATE.205 )

(STATE 'STATE.7279
  :TYPE  &LOC
  :OBJECT  '&HUMAN.137
  :VALUE  '&HUMAN.274
  &ACHIEVES  '&GOAL.4596
  &INTENDED-BY  '&ACT.3567
  &PRECOND-OF  '&ACT.2218 )

(GOAL 'GOAL.4596
  :TYPE  &D-LOC
  :ACTOR  '&HUMAN.137
  :OBJECT  '&HUMAN.274
  :NEW-STATE  '&STATE.7279
  &ACHIEVED-BY  '&STATE.7279
  &PLAN  '&ACT.3567 )
```

```
(ACT 'ACT.3567
  :TYPE  &PTRANS
  :ACTOR  '&HUMAN.137
  :OBJECT  '&HUMAN.137
  :TO  '&HUMAN.274
  &PLAN-OF  '&GOAL.4596
  &INTENDS  '&STATE.7279 )

(STATE 'STATE.4915
  :VALUE  &SIBLINGS
  :TO  '&HUMAN.274
  :OBJECT  '&HUMAN.137
  :TYPE  &RELATION)

(BELIEF 'BELIEF.28
  :MODE  &DEDUCTIVE
  :ACTOR  '&HUMAN.138
  &B-PRECOND  '&STATE.1163
  &MENTAL-EVENT  '&STATE.202
  &EVIDENCE  '&STATE.201 )

(STATE 'STATE.1163
  :OBJECT  '&STATE.201
  :TO  '&HUMAN.138
  :TYPE  &KNOW
  &B-PRECOND-OF  '&BELIEF.28
  &INTENDED-BY  '&ACT.392 )

(ACT 'ACT.392
  :AT  '&HUMAN.137
  :TO  '&HUMAN.138
  :OBJECT  '&STATE.201
  :ACTOR  '&HUMAN.138
  :TYPE  &ATTEND
  &PRECOND  '&STATE.17630
  &INTENDS  '&STATE.1163 )

(STATE 'STATE.17630
  :TYPE  &LOC
  :OBJECT  '&HUMAN.138
  :VALUE  '&HUMAN.137
  &ACHIEVES  '&GOAL.4388
  &PRECOND-OF  '&ACT.392
  &CAUSE  '&ACT.9768 )

(GOAL 'GOAL.4388
  :OBJECT  '&HUMAN.137
  :TYPE  &D-LOC
  :ACTOR  '&HUMAN.138
  :NEW-STATE  '&STATE.6950
  &PLAN  '&ACT.9768
  &ACHIEVED-BY  '&STATE.17630 )

(ACT 'ACT.9768
  :OBJECT  '&HUMAN.138
  :ACTOR  '&ANIMAL.5
  :TYPE  &PTRANS
```

```
  :TO  '&SETTING.215
  :AT  '&SETTING.215
  &PLAN-OF  '&GOAL.4388
  &UNINTENDED  '&STATE.17630 )

(STATE 'STATE.6950
  :TYPE  &LOC
  :OBJECT  '&HUMAN.138
  :VALUE  '&HUMAN.137 )

(STATE 'STATE.201
  :OBJECT  '&HUMAN.137
  :TO  '&HUMAN.274
  :TYPE  &KISS
  &EVIDENCE-FOR  '&BELIEF.28 )

(STATE 'STATE.207
  :TYPE  &AFFECT
  :OBJECT  '&HUMAN.137
  :TO  '&HUMAN.138
  :VALUE  &LOVE
  &INTENDED-BY  '&ACT.492 )

(ACT 'ACT.492
  :ACTOR  '&HUMAN.138
  :TYPE  &KISS
  :OBJECT  '&HUMAN.137
  &INTENDS  '&STATE.207 )

(STATE 'STATE.203
  :VALUE  &HOT
  :OBJECT  '&HUMAN.138
  :TYPE  &TEMPER
  &MOTIVATES  '&GOAL.225 )
```

B.8 Language

MINSTREL uses a phrasal generator called RAP to express its stories in natural language. The details of phrasal generation and RAP are discussed in some detail in Chapter 10. To refresh the reader's memory, phrases are combinations of input/output patterns, such as:

```
(rap:phrase clause-state-to
  (input :clause (state nil
                        :type ?type
                        :object ?object
                        :value ?value
                        :to (*and* ?to (*value* ?to))))
  (output :subject ?object ?type ?value with :object ?to))
```

Concepts are generated by repeatedly matching them against the input patterns for phrases and replacing them with the corresponding output patterns. This process is implemented in a function called RAP:GEN. Although very long, the function is straightforward.

The initial input to RAP:GEN is a list of concepts to generate. This list is called the input list.

RAP stores phrases in "bins" sorted by the number of elements in the input pattern of the phrase. This permits RAP to try the longest phrases first when generating. The generation process consists of repetitively trying to match the input patterns of phrases against the input list. When a phrase is found whose input pattern matches the first part of the input list, the matched concepts are removed from the input list and replaced by the output pattern of the phrase. If a symbol is encountered in the input list, it is output. This process continues until the input list has been emptied:

```
(defun rap:gen (initial-input)
  ;;
  ;;  Process the input.
  ;;
  (do ((input initial-input)
       (output nil))
      ((null input) output)

      ;;
      ;;  Move all the symbols at the front of the input off to the
      ;;  (end of the) output.
      ;;
      (do ((il input)
           (ol nil))
          ((or (null il)
               (keywordp (car il))
               (not (symbolp (car il))))
           (when ol
                 (setq input il)
                 (trace:cformat rap:*rap-verbose*
                                "Outputing: %s.\n" (reverse ol))
                 (setq output (nconc output (reverse ol)))))
          (push (pop il) ol))

      ;;
      ;;  Try the rules from longest to shortest against the input.
      ;;
      (do ((i (min 10 (length input)) (1- i))
           (success nil))
          ((or success
               (< i 0))
           ;;
           ;;  If no success, kick out the left-most unit in the input
           ;;  and report to the user.
           ;;
           (unless success
                (when input
                      (trace:cformat rap:*rap-trace*
                                     "No rule for %a.\n" (car input))
                      (setq output
                            (nconc output (list (pop input))))))))
          (declare (fixnum i))
          ;;
          ;;  Go through the rules in this bin.
```

```
;;
(do* ((candidates (svref rap:*rap-array* i) (cdr candidates))
      (current (car candidates) (car candidates)))
     ((or success
          (null candidates)) nil)
     ;;
     ;;  Clear the binding-table and then try the current
     ;;  rule.  Attempt to match the input of the rule to
     ;;  the left side of the input.
     ;;
     (ht:clear rap:*rap-bindings*)
     (do* ((rule-input (rap:rap-structure-input current)
                       (cdr rule-input))
           (real-input input (cdr real-input))
           (matched nil))
          ((or (null rule-input)
               (not (inst:slmatch-tree
                      (car rule-input) (car real-input)
                      :bindings rap:*rap-bindings*)))
           ;;
           ;;  If null input, we matched all the input.  Now
           ;;  check the test slot.
           ;;
           (when (and (null rule-input)
                      (funcall (rap:rap-structure-test current)
                               (rap:rap-structure-input current)
                               (rap:rap-structure-output current)
                               rap:*rap-bindings*
                               (reverse matched)))

                 (setq success t)
                 (setq input real-input)
                 (funcall (rap:rap-structure-proc current)
                          (rap:rap-structure-input current)
                          (rap:rap-structure-output current)
                          rap:*rap-bindings*
                          (reverse matched))
                 (do ((rule-output (rap:rap-structure-output current)
                                   (cdr rule-output))
                      (tmp nil)
                      (real-output nil))
                     ((null rule-output)
                      (trace:cformat rap:*rap-verbose*
                            "Applying rule %s:\n  %s --> %s.\n"
                            (rap:rap-structure-name current)
                            (reverse matched)
                            (reverse real-output))
                      (setq input
                            (nconc (reverse real-output) input)))
                     ;;
                     ;;  Process the first thing in the rule
                     ;;  output (i.e., (car rule-output)).  If
                     ;;  it is a :wo clause, then make call
                     ;;  copy-wo appropriately.  If it is a
                     ;;  variable, instan it.  Otherwise,
                     ;;  pass it along.
                     (cond
```

```
                ((and (listp (car rule-output))
                      (eq (caar rule-output)
                          :wo))
                 (setq tmp
                       (copy-wo
                         (var:instan-tree
                           (cadar rule-output)
                           rap:*rap-bindings*)
                         (cddar rule-output))))
                ((var? (car rule-output))
                 (setq tmp
                       (var:instan-tree (car rule-output)
                                        rap:*rap-bindings*)))
                (t (setq tmp (car rule-output))))
              ;;
              ;;  If there's a problem, note it.
              ;;
              (if (var? tmp)
                  (progn
                    (trace:cformat rap:*rap-trace*
                       "Uninstan var in output: %s.\n"
                       tmp)
                    (setq tmp (var:name tmp))))
              ;;
              ;;  Add it to real-output
              ;;
              (push tmp real-output))))
;;
;;  If we got here, we matched the top of real-input
;;  to the top of rule-input, so push it into the
;;  matched.
;;
(push (car real-input) matched))))))
```