MEMORY BASED PROCESSING FOR CROSS CONTEXTUAL REASONING: REMINDING AND ANALOGY USING THEMATIC STRUCTURES

Charles Dolan

March 1985
CSD-850010

UNIVERSITY OF CALIFORNIA

Los Angeles

Memory Based Processing for Cross Contextual Reasoning: Reminding and Analogy Using Thematic Structures

A thesis submitted in partial satisfaction of the requirement for the degree of Master of Science in Computer Science

by

Charles Patrick Dolan

1984

TABLE OF CONTENTS

LIST OF FIGURES

page

CHAPTER 1
Introduction

This research addresses the problem of how people understand stories about planning failures. Planning failures occur when people either use inappropriate methods to accomplish a goal or fail to take action to avoid undesirable situations (e.g. loosing a job). A planning error is a particular instance of failure to plan well (e.g. telling off one's boss). Goal failures are the undesirable consequences of planning failures.

How do we know when we are committing a planning error? When we read or hear about someone else's behavior, how do we know when they commit a planning error? These are the questions which are addressed in this thesis.

There are a large class of planning failures which are so common that metaphorical sayings have been associated with them, such as these (from Dyer's work [DYER83]):

"Cutting off your nose to spite your face",
"Burning your bridges behind you", and
"Counting your chickens before they're hatched".

Often admonitions against planning errors are given in adage form. Some examples are:

"A bird in the hand is worth two in the bush",
"Don't bite the hand the feeds you", and
"Nothing ventured nothing gained".

1.1 Recognizing Planning Failures

The problem of understanding planning errors is addressed by constructing a computer model which analyzes stories that contain planning errors. A model which can recognize planning errors in narratives, will also be useful for recognizing planning errors in plans which a planner generates. For any particular story we want the computer to find the planning failure. Also, if there is another story in memory which contains the same planning failure, we would like the program to find the similar components in the two stories. Finding the similar components in two situations is the first step to drawing an analogy. The theory presented here is implemented in a computer program CRAM (Causal Reasoning in Associative Memory). CRAM takes the conceptual representation of a story, finds causal and intentional relationships among the pieces of the story, and recognizes planning errors where they occur. This is the first pass for a system which would crtique plan created by a plan generator.

One of the stories that CRAM understands, **The Fox and the Bear**, is given below. This story contains the planning error of allowing oneself to be deceived.

The Fox and the Bear

> The Bear was at the stream catching fish to eat. He had several on the bank already. The Fox came to the Bear and told him that all the bees had left the tree and that there was honey there. The Bear left his fish and went to get the honey. The Fox took the Bear's fish back to his burrow to eat.

This story illustrates an interesting aspect of the types of planning errors considered in this thesis. A planning errors on the part of one character in a narrative is often a planning for another character. success on After processing this story, CRAM recognizes it as an instance of an ulterior motive. Since there are no other stories in memory, no comparisons are made. When CRAM processes the next story, **The Fox and the Crow**, it again finds an instance of an ulterior motive.

The Fox and the Crow

> The Crow was sitting in the tree with a piece of cheese in her mouth. The Fox came to the bottom of the tree and said to the Crow, "You have such a beautiful voice. Please sing for me."
>
> She was very flattered that the Fox liked her voice and so she started to sing. When she opened her mouth, the cheese dropped out. The Fox picked up the cheese and ran away laughing.

Because CRAM recognizes that these stories contain the same type of planning error, it uses the description of the planning failure to draw an analogy between them. Once an analogy has been drawn, techniques of analogical reasoning can be used to transfer knowledge from one context to another [CARB83, DOLA83]. The common elements it finds are given in figure 1-1.

Contrast these stories with,

*CRAM is designed to take a story's conceptual representation, as output by a conceptual analyzer, and find the planning error(s) in the story. CRAM does not take natural language as input.

-Deceiver-
Fox <--> Fox

-Deceived-
Bear <--> Crow

-The Lie-
The Fox came to the Bear and told him that all the bees had left the tree and that there was honey there.
<-->
The Fox came to the bottom of the tree and said to the Crow, "You have such a beautiful voice. Please sing for me."

-The Planning Error- both characters disable the condition which allows them to maintain contol of an object
The Bear left his fish and went to get the honey.
<-->
She was very flattered that the Fox liked her voice and she started to sing.

-The Goal Failure-
The Bear looses control of the fish.
<-->
The Crow looses control of the cheese.

Figure 1-1: Story similarities

The Fox meets the Crow again

> The Crow was walking along the ground with a piece of cheese in her mouth. The Fox came up to her and said, "Crow, that's a nice piece of cheese you have there. Give it to me or I'll eat you instead." The Crow was frightened and gave the cheese to the Fox.

Here the Fox is just invoking a particular plan for getting the cheese, THREATEN-PLAN [WILE78], and the Crow gives in. There is no planning error here, and hence CRAM would not cite this story as similar to either of the previous two stories.

The structures used in CRAM for recognizing planning errors are Thematic Abstraction Units (TAUs), refinements of the TAUs used in the BORIS program [DYER83]. A review of TAUs is given in **Chapter 2**.

1.2 Why have Knowledge about Planning Failures?

Knowledge about common planning failures has applications to several tasks in artificial intelligence. In story understanding, known planning errors can provide better expectations about what will happen in a particular situation. Expectations are used in natural language processing to more efficiently read text. It is easier to take an input and figure out if it fits an expectation than to derive its meaning bottom-up. For example, a program that knows about,

"Cutting off your nose to spite your face"

will get a better understanding of the following story from Dyer [DYER83].

Ghetto Riots

> Blacks wanted to protest the bad living conditions within the inner city, so they rioted. As a result, their own homes and businesses were destroyed, and outsiders became more afraid to invest in the area.

A program with knowledge about riots and investing will be able to understand this story and make the causal connection between the riots and the fear of the investors. However it takes knowledge about planning failures to understand the "point" of the story, that demonstation through violence is counterproductive. When a program detects the use of possibly bad plans, it should set up expectations for possible undesirable consequences.

Knowledge about planning failures is also useful when formulating plans to solve problems or achieve goals. A program which knows about,

"Don't bite the hand that feeds you"

would know to check inter-relationships with other entities before trying to counter-plan against them. Planning advice expressed in this abstract way is much more useful than specific prohibitions such as:

> don't fight with your boss,
> don't fight with your wife, and
> be careful about fighting with co-workers.

These admonitions are correct but are not general enough to be useful in other situations. The adage above is applicable in any situation where there are inter-relationships among counterplanning agents.

Planning failures allow the solution of a certain class of analogical reasoning problems dealing with planning and counter-planning [DOLA83]. The recognition of an adage-like planning error allows us to find an analogous situation in memory. The description of the planning error gives us a mapping from elements in one story to elements in another. Once we have a similar story and an analogical map we can start using techniques of analogical reasoning [CARB83, GENT83] to transfer knowledge from one situation to another.

Knowledge about planning failures can also be turned into knowledge about how to counterplan by a simple realization. Things which are bad planning on the part of one character are good planning on the part of the character who tricked him. Many people's first reaction to **The Fox and the Crow** is that the Fox is very clever.

1.3 Related Work

Natural language processing and story understanding have been studied by many researchers. No attempt is made here to review all potentially relevant literature. Instead, systems and theories which have had a significant effect on the design of CRAM are covered.

1.3.1 Conceptual Primitives - Instead of using words to represent concepts, CRAM uses conceptual primitives to express knowledge about the world. For example, the primitive for physical movement, **PTRANS**, [SCHA72] is used to represent "walking", "jumping", and "falling". Having conceptual primitives is important for a number of reasons. The foremost reason for decomposing knowledge into primitives is to allow us to express conceptual similarities and differences. For example,

"John walked to the store"
and
"John ran to the store"

both imply that John changes his location from some unspecified location to "the store". Furthermore, we know that in both cases John traveled under his own power by moving his legs. These similarities should be evident in our representation. However, John's manner of traveling is faster in the second case and this difference should also be reflected in our representation. Sentences whose meaning is similar, regardless of their grammatical structure, should have conceptual representations which are similar.

Schank's Conceptual Dependency theory (CD) [SCHA72] provides a notation for representing actions and states. CRAM uses Schank's **ACT**s to represent actions by characters. Figure 1-2 gives the 11 primitive **ACT**s. With each **ACT** there are several possible slots that may be appended. There are 7 possible slots, shown in Figure 1-3. A primitive **ACT** along with its slots comprises a conceptualization. For example, the representation for, "John walked to the store", is,

```
PTRANS actor JOHN
       obj JOHN
       to STORE
       instr (MOVE actor JOHN
                   obj (BODY-PART name LEGS
                                  owner JOHN))
```

where the objects, STORE, JOHN, and LEGS are not decomposed further in this implementation.

```
BODY-PART name LEGS
```

ATRANS	transfer of ownership
PTRANS	physical change of position
PROPEL	application of physical force
MOVE	movement of a part of the body
GRASP	"grasping" of an object by a character
INGEST	taking an object inside a character
EXPEL	expulsion of an object from a character
MTRANS	transfer of mental information between characters
MBUILD	construction of new information from old by a character
SPEAK	producing sounds
ATTEND	direction of a sense organ

Figure 1-2: Schank's primitive **ACT**s

actor	the character performing the act
obj	the object to which the act is performed
from	state before the **ACT**
to	state after the **ACT**
instr	something instrumental in the performance of the **ACT**
mode	**POS** if the **ACT** occurred, **NEG** if it didn't
manner	an optional modifier.

Figure 1-3: Slots for Schank's CD **ACT**s

owner JOHN

is the representation for "John's legs". In order to represent the sentence, "John ran to the store" we only have to append the modifier, **manner FAST**, to the representation.

A second advantage gained by using a conceptual representation is economy of inference rules. In both cases, "walk" and "run", we want to infer that John is no longer at his previous location. Without a system of conceptual primitives, we would need an inference rule for every word denoting movement. In CD, a primitive is defined by the set of inferences that can be made from that primitive.

1.3.2 Understanding Planning - To understand planning, a program needs a representation for goals that are served by the plans. It is useful to have a taxonomy for goals, because some inferences are shared among goals of the same type. For example, when an understander encounters a character with a goal of satisfying a recurring need (food, sleep, sex) it should not be surprised when the goal occurs again later. Likewise, an understander should know that a goal of preserving the current state (health, family) is not activated unless something has happened to threaten that state. Schank and Abelson [SCHA77] have broken down character goals into several categories,

S-GOALs -	**satisfaction** of recurring needs (e.g. **S-SLEEP**, **S-HUNGER**)
E-GOALs -	**enjoyment** goals (e.g. **E-ENTERTAINMENT**, **E-SPORTS**)
A-GOALs -	**achievement** of a long term desire (e.g. **A-STATUS**, **A-WEALTH**)
P-GOALs -	**preservation** of health, possessions, etc. (e.g. **P-HEALTH**, **P-STATUS**)
C-GOALs -	**crisis** level **P-GOALs**
I-GOALs -	a goal **instrumental** in achieving another goal
D-GOALs -	**Delta** goal, similar to an **I-GOAL** but where general planning is used instead of canned script based plans (e.g. **D-PROX** - change in location)

Figure 1-4: Schank and Abelson's classification of goal types

The PAM program [WILE78, WILE83] was capable of understanding stories very similar to **The Fox meets Crow again**. PAM had knowledge about what **PLANs** are useful in achieving which **GOALs** and what **ACTs** and **STATEs** give rise to which **GOALs**. For each conceptualization in the input, PAM tried all of its inference rules to determine if the new concept related to any of the others in the story. PAM had no mechanism for remembering past stories or for detecting bad planning by characters.

1.3.3 Memory Organization - Most story understanding programs do not retain any of the stories they read. They start with a fresh memory before reading each story. Two programs, IPP [LEBO80] and CYRUS [KOLO80], have attempted to model the way that people store episodes and form generalizations from stories they have read. IPP read stories about terrorism. It was able to take stories straight from the UPI news wire and parse them into its internal representation. It could also form generalizations, finding aspects of a story that could be predicted from other aspects. For example, after reading a number of stories about bombings in Northern Ireland, IPP formed the generalization that all bombings in Northern Ireland are done by the IRA. IPP had two major limitations. It understood stories at a very shallow level, simply filling slots in frames for various terrorist activities it knew about. Also, IPP had no notion of causality, (i.e. which slots in its frames were causes and which were effects), therefore it made some false generalizations that a person would not have made. IPP decided at one point that no one was ever hurt in bombings in El Salvador.

CYRUS understood stories about the activities of then Secretary of State, Cyrus Vance. CYRUS was also able to form generalizations. In the case of diplomatic meetings, CYRUS generalized that when Vance was meeting with the Arabs or the Israelis it was about the Arab-Israeli peace. CYRUS had a much wider array of memory structures to use in understanding stories. By using the FRUMP program [DEJO79] as a front end, CYRUS was also hooked up to the UPI wire. In answering questions about Vance's activities, CYRUS was able to use temporal knowledge, when an activity occurred, and also reason about its generalization hierarchy. CYRUS, like IPP, had no causal knowledge about the domain within which it was understanding.

1.3.4 Planning Failures - The BORIS program [DYER83] implements a theory of in-depth understanding which includes a theory of planning failures. BORIS has two domains in which it understands stories: divorce and kidnapping. BORIS contains a large amount of causal knowledge. Dyer gives nine causal/intentional links, I-links, which allow conceptual structures for PLANs, **GOALs**, and **ACTs** and **STATEs** to be connected.

In the example above, **"The Fox meets Crow again"**, the Fox's **GOAL** to get the cheese, a **D-CONTrol**, is connected to other concepts with three I-links. A **'motivation'** link connects it to the Crow's possession of the cheese. An **'intention'** link goes to the instance of **THREATEN-PLAN** used by the Fox. The Crow giving up the cheese is connected to the **GOAL** with an **'achievement'** I-link. I-links declaratively represent many of the inference rules used in PAM. By representing intentionality with links instead of rules, BORIS gets many inference chains "for free". Figure 1-5 gives the I-links used in CRAM.

	GOAL	**PLAN**	**ACT** or **STATE**
intention	→**PLAN**		
achievement	→**ACT**		
realization		→**ACT** or **STATE**	
enablement		→**GOAL**	
thwarting	→**ACT** or **STATE**		
plan-blocking		→**ACT** or **STATE**	
resulting			→**ACT** or **STATE**
disablement			→**ACT** or **STATE**
motivation			→**GOAL**

Figure 1-5: I-links

There are two differences between this set and the ones used by Dyer. Another I-link has been added, "disablement", which connects a **STATE** with the **ACT**s that it disables. Omitted from the set was "suspension" for **GOAL**s which suspend other **GOAL**s. This was omitted because it does not occur in any of the stories examined here.

Dyer's Thematic Abstraction Units (TAUs) are representations of planning errors such as the ones given in the adages above. In BORIS, TAUs were activated by deviations in the expected chain of events. TAUs were used in question answering to point out important events and also in understanding for the affect information associated with planning failures.

1.4 Features of CRAM

Except for the fact that CYRUS had no knowledge about **GOAL**s and **PLAN**s, CRAM has a memory model similar to CYRUS, but with the addition that Dyer's I-links connect concepts that are causally related. TAUs are used to recognize the planning errors, but in CRAM TAUs are represented declaratively as opposed to representing each TAU with the LISP code for recognizing it. Also there is only one TAU recognition process. Because of the memory structure, TAUs are indexed off memory nodes for **PLAN** and **GOAL** failures which are likely to be produced by particular planning errors. This provides more robust recognition than activation by deviations as in BORIS.

CRAM uses the same approach as PAM in connecting an input with other story elements. Because of the memory organization employed in CRAM, expectations and inference rules can be indexed off memory nodes. This way CRAM doesn't examine every rule in the knowledge base to explain an input.

1.5 Approach

The approach taken in CRAM is to structure the memory organization and process model so that TAU recognition is easy and efficient. Situation features are used to index specializations of memory structures. Situation features include: I-links, relationships between characters and objects (e.g. possession), and relationships between characters (e.g. proximity). These same situation features are used in the representation for TAUs. TAUs are indexed off of memory structures which share some situation features.

The inference rules are grouped according to the memory structure. Rules which apply to a situation are indexed under the memory node for that situation. This organization gains two things for the program. First, it makes the application of causal knowledge more efficient because only rules that apply to a situation are examined. Second, it makes an explicit statement about the "level of generality" of a given piece of knowledge. For example, rules and expectations associated with **MTRANS** are indexed at one level of generality. After an **MTRANS** a character would be expected to know what was **MTRANS**ed. The expectations for **FLATTERY**, a specialization of **MTRANS**, are indexed at a lower level. The TAU recognition process uses both the "level" of an inference rule and the number of rules which were applied to derive a fact to decide whether or not a character should have anticipated the consequences of an **ACT**.

During the comprehension process, CRAM groups concepts according to whether or not they have been connected with I-links. Concepts are grouped according to which ones are causally connected. These groups are also used in the representation for TAUs.

Two aspects of this TAU theory make it unique. First the representation for TAUs makes it possible to encode "how" to recognize a planning error in a single procedure, while the information about the planning errors themselves are encoded declaratively. Second, the representation for TAUs allows the formulation of theory of TAU learning, which is not possible when TAUs are represented procedurally.

1.8 Guide to the Reader

This thesis is divided into two major parts. Chapters 2-3 present extensions to Dyer's TAUs [DYER83]. Chapter 2 presents an extended analysis of the two stories in **Section 1.1** and explains in detail how TAUs are represented, how they are indexed, and how TAUs index stories. Chapter 3 gives the details of how TAUs are recognized and disambiguated.

Chapters 4-6 give more detail on the implementation and process model. Chapter 4 explains the knowledge representation and memory organization. Chapter 5 presents the details of the process model, including memory incorporation, the process of finding the right memory node for a conceptualization and a review of explanation-driven reasoning, the method used for connecting the memory structures with I-links. Chapter 6 contains an annotated trace of CRAM running on **"The Fox and the Bear"**.

CHAPTER 2
Thematic Abstraction Units

The TAUs used in CRAM are refinements of the ones in BORIS [DYER83]. In BORIS, each TAU was implemented as a demon (i.e. a piece of LISP code) which contained the information necessary for recognizing the occurrence of that particular TAU. Little restriction was placed on the contents of the CONDITION and ACTION parts of these demons. This scheme does not provide a general theory of TAU recognition. Each demon embodies a portion of the TAU recognition theory. CRAM uses a declarative representation for TAUs. This has two important advantages. The first is efficiency and clarity; CRAM uses only one demon to recognize all TAUs. Changes in the theory of how TAUs are recognized are thus reflected in a single procedure. Second, a declarative representation for TAUs aids in the development of a theory of TAU acquisition. It is difficult to postulate a TAU acquisition theory if the concepts being learned do not have a uniform structure.

Dyer [DYER83] also discusses how TAUs relate to one another. He points out that some TAUs are more general than others. He also points out that different TAUs can index the same planning failure and can thereby share some components. These ideas, which were not taken any further in that research, are extended here to show how more specialized TAUs are constructed and how to determine exactly which components two TAU share.

This chapter presents two views of TAUs. The first is that TAUs are used to index episodes in memory. In **Section 2.1** we will see how stories can share multiple TAUs. The second view, presented in **Sections 2.2-3**, is that TAUs are a knowledge source for dealing with planning failures. This is the way that TAUs are used in CRAM.

2.1 Planning Errors on Multiple Levels

In this section we will see how the information contained in TAUs helps us pick out the various planning errors in a story. Any story may contain multiple planning errors. For example, **The Fox and the Crow** contains at least two on the part of the Crow:

- letting herself be deceived
- singing when she still had the cheese in her mouth

The first is called **TAU-ULTERIOR**, and the second **TAU-CONF-ENABLE** (for "confused enablement"). Later we will see that this story actually contains three distinct planning errors. It is very important that the understanding model account for multiple, related thematic structures. To see why, look at Figure 2-1; it shows what memory organization would look like if there were no more fine-grain indexing of episodes than **TAU-ULTERIOR**. Considering the number of times people encounter ulterior motives, the number of episodes would be very large. We can not even postulate a process which would pick out an appropriately similar story in a new situation given this organization.

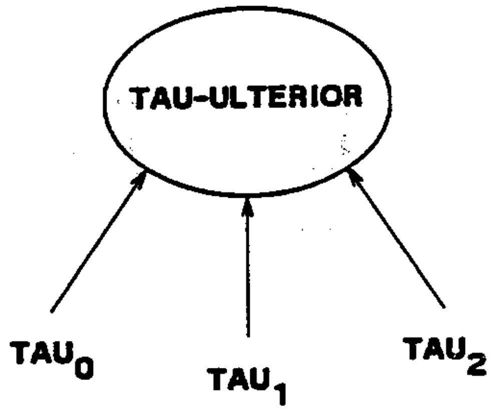

Figure 2-1: TAU instances with no indexing

Figure 2-2 shows what a memory would look like if we also indexed off of surface features, for example, the fact that the story contained a fox, a crow, and some cheese. In this case, a person would only find an appropriate episode if the current situation also had a fox, a crow, and some cheese; this would not be very likely.

A program that understands planning should be able to pick out stories that are similar on the planning level. Figure 2-3 shows an example of what some parts of such a memory might like. Such a memory, organized on the planning level, allows an understanding system to pick out the most appropriate story for use in the current situation. If we look at the two stories, **The Fox and the Crow** and **The Fox and the Bear** we see that they share two TAUs, **TAU-ULTERIOR** and **TAU-VANITY** which in turn, grouped into the composite TAU **TAU-SUCKERED**. The TAUs **TAU-SUCKERED** and **TAU-VANITY** are fully explained below. **TAU-ALLY-ENABL** and **TAU-NEGLECT** are simple specializations of **TAU-CONF-ENABL**. **TAU-ALLY-ENABL** is the planning failure as a result of alienating allies. **TAU-NEGLECT** is the planning failure of loosing control of an object by neglecting to guard it. The thick black lines indicate links of a specialization hierarchy. The elipse around **TAU-ULTERIOR** and **TAU-VANITY** in-

*When more than one story is found which shares the same planning structure, surface features are used to select the most similar story among those. This more fine-grain selection is not present in the implementation presented here.

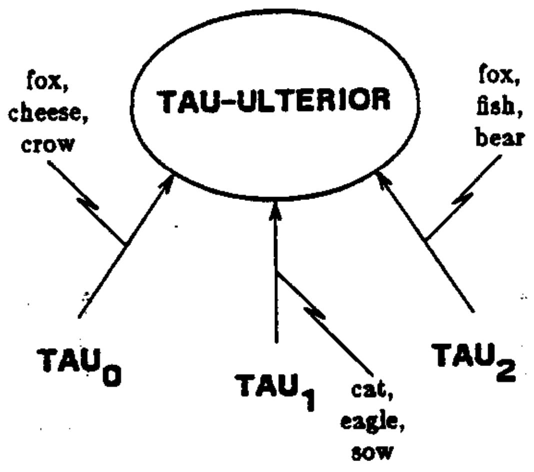

Figure 2-2: TAU instances with indexing on surface features

dicate that those TAUs are components of a larger thematic structure, **TAU-SUCKERED**.

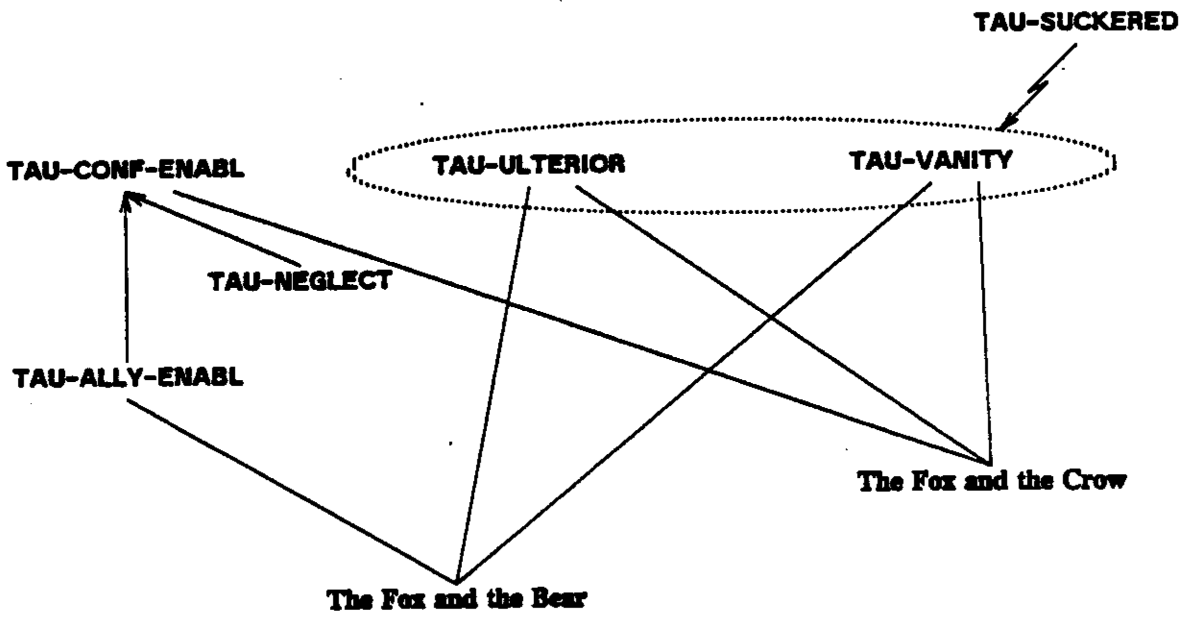

Figure 2-3: Stories indexed by multiple TAUs

The point is that by indexing off of multiple planning errors, we are more likely to find a story which is similar on the planning level.

2.1.1 Multiple reasons for a goal failure - When we refer to multiple TAUs here, we mean multiple TAUs which refer to the same **GOAL** failure. A story with more than one **GOAL** failure can easily contain multiple TAUs.

Next we will look at the planning structure of **The Fox and the Crow** as it relates to **TAU-ULTERIOR**, **TAU-VANITY**, and **TAU-CONF-ENABL**. We can see how TAUs are represented in CRAM by looking at **TAU-ULTERIOR**. Figure 2-4 shows the representation for **TAU-ULTERIOR**.

```
TAU name TAU-ULTERIOR
 binding-spec
   [?state (STATE)]
   [?goal (GOAL actor ?x
                obj ?y)]
   [?mtrans (MTRANS actor ?y
                    to ?x
                    obj ?z)]
   [?act (ACT actor ?y)]
 constraints
    achievement(?state,?goal),
    motivation(?z,?goal2),
    achievement(?goal2,?act),
    disables(?act,?state),
    goal-conflict(?goal,(GOAL actor ?y
                              obj ?w)),
    achievement(?act,(GOAL actor ?y
                          obj ?w))
    not-obvious-result(?act,?mtrans).
```

Figure 2-4: Template for **TAU-ULTERIOR**

The binding-spec gives a list of concepts which must be found in the story for this TAU, associated with variables. The constraints are semantic relationships among the concepts in binding-spec. In very simple terms, the binding-spec shows what makes up the TAU for activation purposes and the constraints provide both a further specification and a program for recognizing the TAU. The recognition process is covered fully in **Chapter 3**. This TAU can be restated in English as follows,

> There is a **STATE** which achieves a **GOAL** for a character. Another character transfers information to the first, which motivates a **GOAL** for that character. The first character acts to achieve the new goal and disables the original state. The act achieves a **GOAL** for the second character which was in conflict with the original **GOAL**.

This TAU does not specifically mention the truth or falsehood of the information transmitted between the characters. This TAU is activated by the detection of falsehoods. By far, most situations recognized as having this TAU will have a falsehood in them. It is not required, however, that the program determine the truth of a statement by a character in order to apply **TAU-ULTERIOR**. This agrees with the informal protocols taken by the author which indicate that most people think that the key element of deceit is an ulterior motive rather than lying is.

Using this TAU we get the following planning structure for **The Fox and the Crow**, shown in Figure 2-5.

TAU-ULTERIOR

?state	The Crow has the cheese
?goal1	The Crow wants the cheese
?mtrans	The Fox tells the Crow that she has a nice voice
?goal2	The Crow wants to show off
?act	The Crow sings

Figure 2-5: **TAU-ULTERIOR** for **The Fox and the Crow**

Similar to the template for **TAU-ULTERIOR**, we have a template for **TAU-VANITY** shown in Figure 2-6 which encodes the planning failure of allowing your vanity to get the better of you. The major conceptual difference between this TAU and **TAU-ULTERIOR** is that with vanity we have the specfic mention of a vain '**?belief**' which is an enablement condition on a **PLAN** for the conflicting **GOAL**. Stated in English this TAU says,

> A character appraises some part of himself as **GOOD**. This causes him to enact a **PLAN** to achieve a **GOAL**. However, the **ACT** which realizes the **PLAN** causes the failure of another **GOAL**.

From this TAU we get the same type of decomposition of **The Fox and the Crow** based on the elements of the **binding-spec** shown in Figure 2-7.

The structure for **TAU-CONF-ENABL** is much simpler than the previous two. This TAU occurs whenever a character tries to achieve one **GOAL**, and does not realize that his actions remove an enablement condition on the continued achievement of another **GOAL**.

```
TAU name TAU-VANITY
  binding-spec
   [?belief (KNOW actor ?x
                    obj (APPRAISAL obj (BODY-PART owner ?x)
                                   value GOOD))]
   [?goal1 (GOAL actor ?x)]
   [?plan (PLAN actor ?x)]
   [?goal2 (GOAL status FAILED
                 actor ?x)]
  constraints
     intention(?goal1,?plan),
     realization(?plan,?act),
     enablement((APPRAISAL obj (BODY-PART owner ?x)
                               value GOOD),
                ?act),
     thwarting(?act,?goal2).
```

Figure 2-6: Template for **TAU-VANITY**

TAU-VANITY

?belief	The Crow believes she has a nice voice
?goal1	The Crow wants to show off
?plan	The Crow plans to sing
?goal2	The Crow wants the cheese

Figure 2-7: **TAU-VANITY** for **The Fox and the Crow**

2.1.2 Learning new TAUs through combination - Now we take the planning structures for **TAU-ULTERIOR** and **TAU-VANITY** and include them with the structure for **TAU-CONF-ENABL**, and we get the structure shown in Figure 2-9. This graph shows the two different ways that stories can have multiple TAUs. The lines between concepts indicate that the two concepts are instantiated by the same concept in **The Fox and the Crow**. The first is very simple. Both **TAU-ULTERIOR** and **TAU-VANITY** have **TAU-CONF-ENABL** as a sub-component. Whenever this happens in a situation, the component TAU ends up simply adding constraints to the other TAU, creating a specialization or more specific TAU than before. This is interesting because it gives us a simple, non-trivial way of making a specializa-

```
TAU name TAU-CONF-ENABL
  binding-spec
    [?goal1 (P-GOAL actor ?x)]
    [?goal2 (GOAL actor ?x)]
    [?act (ACT actor ?x)]
  constraints
      intention(?goal2,?plan),
      realization(?act,?plan),
      resulting(?act,?state2),
      achievement(?state1,?goal1),
      disablement(?state2,?state1).
```

Figure 2-8: Template for **TAU-CONF-ENABL**

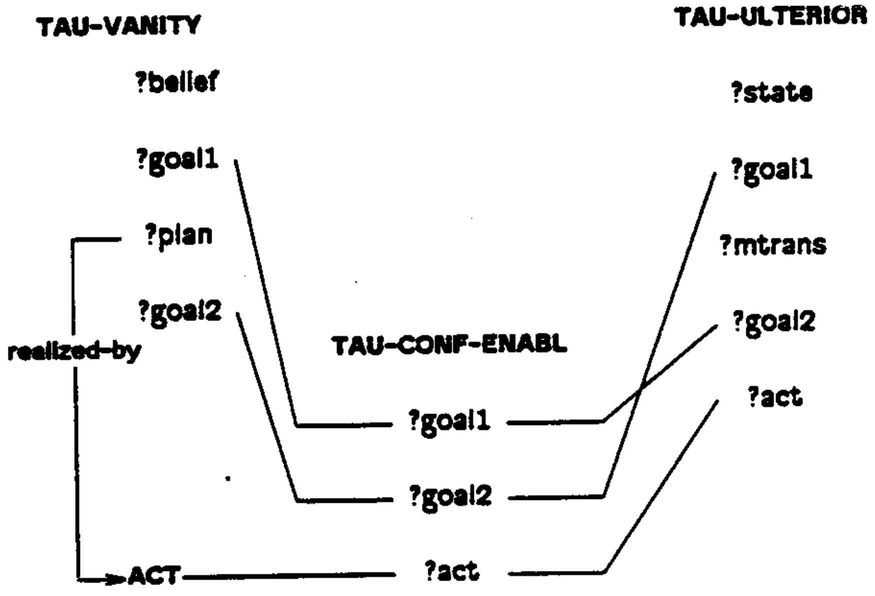

Figure 2-9: Composite planning structure of The Fox and the Crow

tion of a thematic structure.

The second type of type of sharing that goes on between **TAU-ULTERIOR** and **TAU-VANITY** is more interesting because it is more complex. It is more complex because it is not simple subsumsion, and therefore there may be other relations among the concepts in the TAU which are not already explicit. In a case such as this we can form a new TAU called **TAU-ULTERIOR-VANITY**, and specify it as having all the components and constraints of both its components. This is relatively uninteresting because in theory any two TAUs might be joined in this "conjunctive" manner. If, instead, we infer a new concept,

```
KNOW actor FOX
        obj (GOAL actor CROW
                  obj (ATTEND obj CROW))
```

that the Fox knew about the Crow's **GOAL** to have attention paid to her, then we can easily get to a much more interesting TAU by combining the two TAUs.

```
TAU name TAU-SUCKERED
  binding-spec
    [?goal2 (P-GOAL actor ?x)]
    [?goal1 (GOAL actor ?x)]
    [?know (KNOW actor ?y
                 obj ?goal1)]
    [?act (ACT actor ?y)]
    [?mistake (ACT actor ?x)]
  constraints
     intention(?goal1,?plan1),
     enablement(?plan1,?sub-goal1,?sub-goal2),
     resulting(?act,?state),
     achievement(?state,?sub-goal1),
     intention(?sub-goal2,?sub-plan),
     realization(?mistake,?sub-plan),
     resulting(?mistake,?state2),
     thwarting(?state2,?goal2).
```

Figure 2-10: **TAU-SUCKERED**

The '?mistake' which is given as an arbitrary **ACT** in the binding-spec is defined in the last three constraints of the TAU. The '?mistake' is an **ACT** that is a realization of the '?sub-plan'. It causes '?state2' which thwarts '?goal2'.

This type of learning is extremely interesting because it is driven both from the initial structures and from the data. The story of **The Fox and the Crow**, when it instantiates the two TAUs, **TAU-ULTERIOR** and **TAU-VANITY**, allows a mapping to be made between the components of the two thematic structures. It is this mapping which allows them to be combined in a non-trivial way to form **TAU-SUCKERED**. Without a specific episode, which istantiates multiple TAUs, it is not feasible to combine TAUs because there is no way to construct the mapping.

From the top level, then, the TAU relationships of **The Fox and the Crow** involves multiple containment relationships among the TAUs as depicted in Figure 2-11. It is the containment relationships which are used to cross-index planning errors among stories. There is a large difference between two TAUs occurring in a component/containment relationship and just occurring in the same situation.

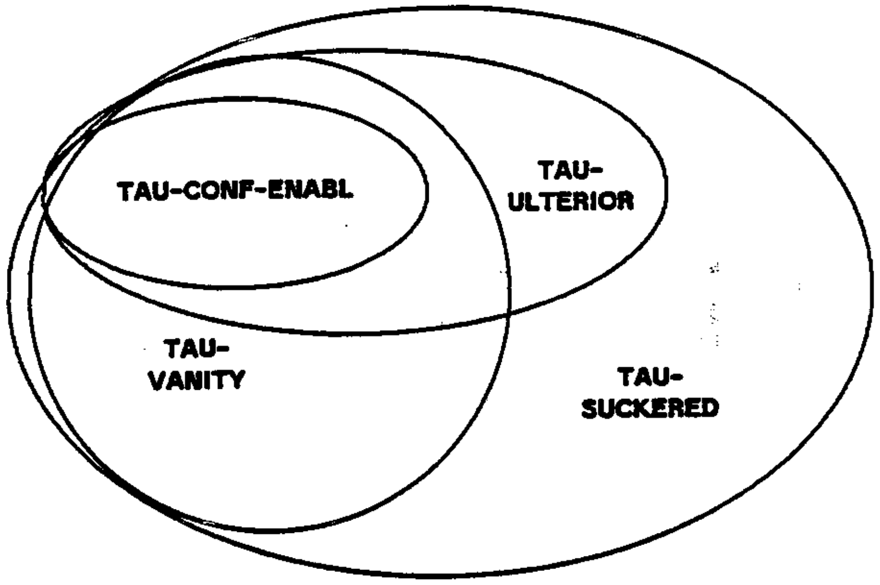

Figure 2-11: Final TAU relationships for **The Fox and the Crow**

2.2 TAUs in CRAM

TAUs are a knowledge source for dealing with understanding failures. As a result of bad planning on the part of narrative characters, some concepts do not get connected with I-links by CRAM. Without additional knowledge, the relationships among these concepts would not be found. When they are recognized as part of a TAU, they are integrated with the rest of the story. For example, CRAM has no knowledge about ulterior motives as ***a standard plan*** for achieving a **GOAL**. Instead it treats it as an exceptional case by using a TAU. As an illustration, consider Figure 2-12 which gives an abbreviated conceptual representation for **The Fox and the Bear**.

People seem to check the validity of what they hear against what they already know. It is this mechanism that detects deceit or possible deceit. However people do not seem to be constantly checking new inputs against past ones for contradictions. Deceit can be detected, post hoc, after the discovery of an ulterior motive, as mentioned in the previous section. In order to handle ulterior motives without using a TAU, CRAM would not only have to check for an ulterior motive after every occurrence of the primitive act **MTRANS**, but it would have to keep track of everything the characters say and keep checking for contradictions. This would impose a large processing overhead for a situation which is encountered quite often in read-

```
LOC actor BEAR obj STREAM
M-FISHING actor BEAR
POSS-BY actor BEAR obj FISH
PTRANS actor FOX to STREAM
MTRANS actor FOX to BEAR
          obj (LOC actor HONEY
                   obj TREE)
PTRANS actor BEAR from STREAM
GRASP actor FOX obj FISH
```

Figure 2-12: Conceptual representation for **The Fox and the Bear**

ing stories and is probably not how people do it.

Instead **TAU-ULTERIOR** is indexed off of the memory nodes **P-CONT-fail** and **LIE-ACT**. **P-CONT-fail** is the memory node for the failure of a **GOAL** to preserve control over an object. The presence of the **GOAL** failure is inferred when the Fox steals the fish. In addition to being indexed off of the **GOAL** failure, when an obviously false statement is detected, CRAM can use **TAU-ULTERIOR**[*] to set up expectations for **GOAL** conflicts between characters. Figure 2-13 shows the internal representation for **TAU-ULTERIOR** in CRAM. This is more restricted than the version presented in **Section 2.1** and can be thought of as a specialization of that TAU. The reason for the restriction is that in Aesop's fables we do not need the full generality of that representation. In the remainder of the thesis, representations will refer to the world of Aesop's fables. This is the sense of ulterior motives presented in Figure 2-13.

This TAU as instantiated for **The Fox and the Bear** is shown in Figure 2-14. In the story, **'?act2'** is the bear leaving the stream to get the honey. The concept bound to **'?act1'** is representing the fact that the bear no longer has the fish. This is inferred from the **P-CONT-fail**. The first three **'constraints'** test for I-links.

The **'not-obvious-result'** constraint is a statement about the processing state which says that we have found no causal or intentional connection between these two concepts yet. This is represented in CRAM by mantaining the story as a graph where edges are I-links. **'not-obvious-result(?x,?y)'** is true if **'?x'** and **'?y'** are in disjoint sub-graphs. This is important in distinguishing ulterior motives from more straightforward plans such as threats, as in the story **The Fox meets Crow again**, presented in **Chapter 1**. The fact that there is no I-link between these concepts means that this is an exceptional condition. Without **TAU-ULTERIOR**,

[*] CRAM does not currently check inputs against its knowledge of the world but only detects deceit either post hoc or when it is mentioned specifically as such.

```
TAU name TAU-ULTERIOR
 binding-spec
  [?mtrans (MTRANS actor ?z to ?x)]
  [?act2   (ACT head ?do)]
  [?p-cont (GOAL actor ?x
                 manner (FAIL)
                 obj (POSS-BY actor ?x
                              obj ?y))]
  [?grasp  (GRASP actor ?z
                  obj ?y)]
 constraints
  resulting(?mtrans,?act2),
  thwarting(?act1,?p-cont),
  resulting(?act2,?act1),
  not-obvious-result(?grasp,?mtrans)
```

Figure 2-13: CRAM template for **TAU-ULTERIOR**

CRAM would not be able to make any connection between the Fox grabbing the cheese and his flattery of the Crow. The constraint **'not-obvious-result'** is the opposite of **'obvious-result'** which is used in the representation for **TAU-CONF-ENABL** in **The Fox and the Apple**.

After recognizing **TAU-ULTERIOR**, CRAM creates an instantiation of that TAU. It contains the variable bindings for conceptual objects which match the patterns and satisfy the constraints. Figure 2-14 gives the instantiated TAU for **The Fox and the Bear**. The slot **'BINDINGS'** gives the correspondence between variables in the template of Figure 2-13 and the elements of the story.

Another story which is processed and indexed by the CRAM program is an adaptation of **The Dog and the Shadow**:

The Fox and the Apple

> The Fox was walking along with an apple in his mouth. He was on the bridge when he saw a bunch of grapes in the tree above him. He jumped up to try to grab the grapes. When he opened his mouth he dropped his apple in the river. Also the grapes in the tree were too far up for him to reach, so he ended up with nothing.

The conceptual representation for this story is given in Figure 2-15.

```
(TAU INSTANTIATION-OF &TAU-ULTERIOR
  BINDINGS
    [?DO2 . PTRANS]
    [?Z . (FOX)]
    [?GRASP . (GRASP ACTOR (FOX)
                     OBJ (FISH))]
    [?MTRANS . (MTRANS ACTOR (FOX)
                       TO (BEAR)
                       OBJ (LOC
                             ACTOR (HONEY)
                             OBJ (PLACE
                                   NAME (TREE))))]
    [?ACT2 . (PTRANS ACTOR (BEAR)
                     FROM (PLACE NAME (STREAM))
                     TO (PLACE NAME (TREE))
                     OBJ (BEAR))]
    [?ACT1 . (LOC MODE (NEG)
                  ACTOR (BEAR)
                  OBJ (PLACE NAME (STREAM)))]
    [?P-CONT . (GOAL MANNER (FAIL)
                     OBJ (POSS-BY ACTOR (BEAR)
                                  OBJ (FISH)))
    [?Y . (FISH)]
    [?X . (BEAR))]
```

Figure 2-14: **TAU-ULTERIOR** for **The Fox and the Bear**

```
PTRANS actor FOX
LOC actor APPLE obj mouth(FOX)
LOC actor FOX obj BRIDGE
ATTEND actor FOX obj GRAPES
LOC actor GRAPE obj TREE
PTRANS actor FOX to TREE manner JUMP
GOAL actor FOX
       obj (POSS-BY actor FOX obj GRAPES)
PTRANS obj APPLE from mouth(FOX)
LOC mode NEG actor FOX obj TREE
```

Figure 2-15: Conceptual representation for **The Fox and the Apple**

This story can be processed in terms of a specialization of **TAU-CONF-ENABL** which is possessed by CRAM. This TAU represents the planning failure for

"A bird in the hand is worth two in the bush".

The essential elements of this TAU are given graphically in Figure 2-16. The arcs labeled with lowercase letters indicate I-links.

```
                POSS-BY
                    |
                    m
                    |
D-CONT           P-CONT-fail
  |                     |
   +-i-+                +-t-+
       |                    |
       |                    |
      ACT--result-chain---ACT
```

With the additional constraints
1 - the objects in the **GOALs** are of similar type.
2 - the result link is obvious

Key
m - motivation
t - thwarting
i - intention
r - resulting

Figure 2-16: Template for a specialization of **TAU-CONF-ENABL**

The second constraint in Figure 2-16 is the most important. It states that the result link should have been obvious. In CRAM this is tested by looking at how many rule applications were required to construct the I-link. If an I-link took more than one rule application to find, it is not considered obvious.

The two constraints **'not-obvious-result'** and **'obvious-**

result' are attempts at representing *what is obvious**. The common aspect of these two constraints is that they do not refer to pieces of the representation, but to the internal state of CRAM, the working memory and the semantic memory organization.

In **TAU-ULTERIOR** we examined the state of the working memory to see if two concepts have been connected with I-links. In **TAU-ULTERIOR** the key constraint was **'not-obvious-result'**, a condition based on the processing state of the program. In **TAU-CONF-ENABL** the key constraint was not one on the processing state but on the state of the memory organization. That is, if the understander did not think that jumping up to get the grapes would obviously make the apple drop out of the Fox's mouth, it would not get reminded of any stories through **TAU-CONF-ENABL**. This is because TAU are used to encode planning failures, not **GOAL** failures which are beyond the planners control.

2.3 TAUs Index Memory in CRAM

TAUs serve to index memory. Stories which contain similar planning errors will be indexed under the same TAU. Figure 2-17 shows part of the CRAM memory after processing the following three stories:

The Fox and the Crow,
The Fox and the Bear, and
The Fox and the Apple.

The portion of the memory shown is associated with the node **P-CONT-fail**. The **'-x-'** lines indicate indexed entries. Indexing means that there is a D-net [CHAR80] attached to the entity, which uses the concept pointed to as an index. Lowercase entries stand for concepts from the stories.

There are two TAUs indexed under **P-CONT-fail**, **TAU-ULTERIOR** and **TAU-CONF-ENABL**. Indexed under the generic TAUs are specific instantiations such as the one in Figure 2-14. The concepts which were found in the instantiation of **TAU-ULTERIOR** are encircled by the dotted line.

2.4 Conclusions

TAUs differ from other knowledge sources because they characterize exceptional situations. The exceptional situations are recognized primarily with the constraints

not-obvious-result and
obvious-result.

The **'not-obvious-result'** constraint tests for exceptions to what CRAM knows about planning. It does this is by looking at which concepts CRAM has connected with I-links. The **'obvious-result'** constraint tests for exceptions in a

*Knowing what is, or is not, obvious in planning is important in assessing the planning of others. While a very interesting research issue, the problem of representing obviousness is beyond the scope of this thesis.

```
                  P-CONT-fail
                  !    !    !             crow-has-cheese
              +-X-+    X    +-X+          !             !
                !      !      !           m             d
                !      !      !      +----+             !
      fox-loose-apple  !    crow-looses-                !
       !               !       cheese---t----cheese-drops
       !         ......!......................          !
     +t+         .bear-looses-fish           .          !
      !          .              !            .          !
apple-drop       ......         !            .          !
  !                   .         +---t-+      .          !
  !                   .               !      .          !
  !     bear-has-fish----------d---fox-theft .          X
  X   ................                   !   .          !
  !   .                                  !  .           !
  !   .fox-mtrans--r--bear-departure     X .            !
  !   ...................................!.             !
  !                                      !              !
  !                                      !              !
TAU-CONF-ENABL                      TAU-ULTERIOR--------+
            !                        !
           +--X-+                +X-+
              !                   !
              (back to P-CONT-fail)
```

Key
X - indexing
m - motivation
t - thwarting
r - resulting
d - disablement

Figure 2-17: CRAM episodic memory

character's planning behavior; that is, doing things which obviously wouldn't work. It does this by looking at its own memory organization to see if the consequences of an action "look" obvious.

As CRAM acquires more causal knowledge in the form of processing rules, more things will "look" obvious. The same thing happens with people. No causal relation is easy to deduce but once it is learned it is considered obvious.

TAUs are useful because they organize information about exceptions at a higher level than **ACT**s, **STATE**s, **GOAL**s, and **PLAN**s. If this information were indexed at these lower levels, it would be stored many times. In addition it would be checked each time one of these lower level structures was encountered. Recall the example for **MTRANS** from **"The Fox and the Bear"** in Figures 2-12 and 2-13. Instead of including large amounts of detailed planning information at every memory node, we use TAUs to represent common planning errors and index them off the **GOAL** and **PLAN** failures that they cause.

In addition TAUs can index stories that are analogically similar at the **GOAL/PLAN** level because they represent information at the **GOAL/PLAN** level and not at the **ACT/STATE** level. TAUs make statements about the **PLAN**s and **GOAL**s active for characters. Because TAUs represent exceptional situations, the representation of a TAU has to make assertions about the processing and memory states in order to distinguish TAUs from normal planning and counter-planning situations.

In comparison with the BORIS [DYER83] program, the major extensions which CRAM makes are the indexing of multiple stories by the TAUs which they contain, and the movement of knowledge about how to recognize TAUs out of the representation of the TAUs and into a single procedure.

CHAPTER 3
TAU Recognition

The previous chapter described the representation for TAUs. If we had a completely processed story with all the I-links in place, we could, in principle, be able do TAU recognition with the following algorithm:

```
for all TAUs in the database

    match the 'binding-spec' against the
    concepts in the story

    check the 'constraints'

until a match is found
```

However, using this algorithm, we must compare each TAU with each conceptual entity in the representation of a story in order to recognize that a TAU should be instantiated. This nullifies the advantages we gain by providing a compact declarative representation for planning errors. For that reason we instead index TAUs off of memory structures and provide an efficient method for recognizing TAUs from their representation. In addition, the algorithm presented in this chapter, starts recognizing TAUs before the actual planning error occurs, and therefore could also be adapted for plan critiquing in a plan generator.

3.1 TAU indexing - a review

In CRAM TAUs are indexed off of memory structures as shown in Figure 2-5. The methods used in CRAM for activating TAUs are those given by Dyer [DYER83]. TAUs are activated by three things:

GOAL failures
PLAN choices
expectation failures

There are memory nodes for **GOAL** failures, so activation of a TAU may occur when we access a **GOAL** failure memory node during processing. For example, this allows CRAM to recognize an ulterior motive, even when no expectation for deceit has been generated through detecting a lie. An example of this was presented in **Chapter 2**. **P-CONT-fail** had two TAUs, **TAU-ULTERIOR** and **TAU-CONF-ENABL** indexed under it. CRAM models an understander who does not know that crows have awful voices but is still able to recognize deception by detecting the ulterior motive. This agrees with one informal protocol in which a listener claimed not to know that crows do not sing, but was still able to detect the planning error.

The reason for indexing TAUs under **GOAL** failures is that a character who experienced a goal failure may have committed a planning error. One area which still needs exploration is the indexing of many TAUs under very general memory nodes such as **P-CONT-fail**. It is clear that people do not activate this TAU for every planning error they know about whenever someone looses control of something.

There are two ways to activate TAUs by **PLAN** choices. The first is to index the TAU under the memory node for the **PLAN**. This will result in a search for a TAU every time that **PLAN** is executed. This method is used when executing **PLANs** for very important **GOALs**. For example if a student had a **GOAL** failure as a result of turning in a poorly typed paper, he would have a TAU indexed under the **PLAN** for turning in a paper. The TAU would embody the adage,

"An ounce of prevention is worth a pound of cure".

This TAU would remind him to proof-read the paper before turning it in. The TAU is indexed under the **PLAN** itself because the price for committing the error is high. The trade-off [DYER83] is the cost of extra processing during planing vs. the cost of recovery in case of failure.

TAUs may also be activated with **PLAN** choices after the **PLAN** has failed. An example of this is found in the common planning error of waiting to wash the dishes until the morning after the party. In executing the **PLAN** for getting dishes clean, a person might note that it would have been much easier to wash them right after the meal, before the food had dried and stuck to the plates. This would bring up the above adage. In this case, the TAU is indexed under the actual failure of the **PLAN** because the price for bad planning is relatively low [DYER83, p. 67] compared to the extra effort it takes to think about the dishes after a party.

Expectation violations are another means of activating TAUs. In this situation, we index the TAU off of the memory node for an event which does not occur, but which a reader would expect in normal circumstances. Dyer [DYER83] gives the following story as an example:

Car Aflame

> Henry came upon an over-turned automobile with a driver unconscious at the wheel. The car was on fire and Henry barely managed to drag the driver from the wreck before the car burst into flames. A month later, Henry was sued for having injured the driver's back when pulling him out of the wreck.

After Henry rescues the driver, the reader expects some show of gratitude from the driver. The lawsuit is a violation of this expectation. This story is understood with a TAU, **TAU-UNJUST-RETALIATION**.

The question of exactly which structures to use for indexing requires collecting protocols from people. The limited number of protocols collected for this research (5) indicate that for **The Fox and the Crow** and **The Fox and the Bear**, good places to activate **TAU-ULTERIOR** are,

> 1) when the deceived character actually looses control of the object or
>
> 2) when the deceived character does something to allow the deceiver character to take the object, i.e. when the loss of control can be predicted.

3.2 TAU processing

TAU recognition involves two distinct activities, (1) finding the concepts in the story that match the **'binding-spec'** and (2) testing the **'constraints'**. In Figure 3-1, the → points to the conceptualization from which the TAU is activated. Where a TAU is indexed affects when it is recognized.

```
     TAU
       binding-spec
         [?mtrans (MTRANS actor ?z to ?x)]
         [?act2   (ACT head ?do)]
 →       [?p-cont (GOAL actor ?x
                        manner FAIL
                        obj (POSS-BY actor ?x
                                     obj ?y))]
         [?grasp  (GRASP actor ?z
                         obj ?y)]
       constraints
         resulting(?mtrans,?act2),
         thwarting(?act1,?p-cont)
         resulting(?act2,?act1),
         not-obvious-result(?grasp,?mtrans)
```

Figure 3-1: **TAU-ULTERIOR** revisited

If the → is moved higher, the TAU is triggered ealier in processing the story, lower -- later. The rules for deciding where to index a TAU are given in **Section 3.3 TAU re-indexing.**

As an example, let's look at how **TAU-ULTERIOR** gets applied to **The Fox and the Crow.** Figure 3-2 shows the conceptual representation for the story.

```
    LOC actor CROW obj TREE
    LOC actor CHEESE obj mouth(CROW)
1.  PTRANS actor FOX to TREE
    MTRANS actor FOX to CROW
            obj "you have a nice voice"
    MTRANS actor FOX to CROW
            obj "sing for me"
    SPEAK actor CROW manner SONG
2.  PTRANS actor GRAVITY obj CHEESE from mouth(CROW)
    GRASP actor FOX obj CHEESE
```

Figure 3-2: Conceptual representation for **The Fox and the Crow**

We will start the analysis at the second **PTRANS**. It is recognized as an instance of dropping something by the fact that GRAVITY is the **actor**. A causal link to the Crow's singing is inferred, i.e. that the **SPEAK** resulted in the **PTRANS**. One of the checks made at that memory node is to see if any character possessed the object which was dropped. We check for this condition by looking for either a **POSS-BY** or a **LOC**. A **P-CONT-fail** is inferred from this loss of possession because the Crow had the cheese before she dropped it. The demon **'TAU-RECOGNITION'** is spawned to check for the occurrence of all TAUs indexed under the **P-CONT-fail** node. One instance of the demon is spawned for each TAU and the checking is done in parallel. In this case two demons are spawned, one for each of **TAU-ULTERIOR** and **TAU-CONF-ENABL**. This discussion will center on the recognition of **TAU-ULTERIOR**. Since the TAU is spawned on **P-CONT-fail**, the recognition process immediately gets a match for '?p-cont' with

```
GOAL actor CROW
     obj (POSS-BY actor CROW
                  obj CHEESE)
```

After matching this concept, **'TAU-RECOGNITION'** searches for concepts in working memory to match **'?mtrans'** and **'?act2'** which will also satisfy the first three constraints in Figure 3-1. The concepts it finds are

```
MTRANS actor FOX
       to CROW
       manner REQUEST
       obj (SPEAK actor CROW
                  manner SONG)
```

and

```
SPEAK actor CROW
      manner SONG
```

If the concepts had not been found, **'TAU-RECOGNITION'** for this TAU would have stopped.

Since it did succeed in finding the concepts, it sets up an expectation for the **'?grasp'** pattern[*]. The last concept is processed. It matches with **'?grasp'** and satisfies the last of the **'constraints'**. At this point CRAM decides that this is an occurrence of **TAU-ULTERIOR** and indexes the new instantiation under that TAU using the concept that matched **'?p-cont'** as the index. CRAM produces the mapping of story elements to TAU pattern variables shown in Figure 3-3.

```
(TAU INSTANTIATION-OF &TAU-ULTERIOR
  BINDINGS (
    [?DO2 . SPEAK]
    [?MTRANS . (MTRANS ACTOR (FOX)
                       TO (CROW)
                       OBJ (SPEAK ACTOR (CROW)
                                  MANNER (SONG))
                       MANNER (REQUEST))]
    [?ACT2 . (SPEAK ACTOR (CROW)
                    MANNER (SONG))]
    [?ACT1 . (PTRANS ACTOR (GRAVITY)
                     FROM (BODY-PART NAME (MOUTH)
                                     OWNER (CROW))
                     OBJ (CHEESE))]
    [?GRASP . (GRASP ACTOR (FOX)
                     OBJ (CHEESE))]
    [?Z . (FOX)]
    [?P-CONT . (GOAL MANNER (FAIL)
                     OBJ (POSS-BY ACTOR (CROW)
                                  OBJ (CHEESE)))]
    [?Y. (CHEESE)]
    [?X . (CROW)])
```

Figure 3-3: **TAU-ULTERIOR** for **The Fox and the Crow**

[*] The **TAU-ULTERIOR** used in the current implementation of CRAM is a specialization of the one presented in **Section 2.1**. The only **GOALs** are possession **GOALs** and the only way to get something is to **GRASP** it.

3.3 TAU re-indexing

The fact that we can point to the place in a TAU where it is indexed is important for re-indexing TAUs. Dyer [DYER83] gives two strategies for deciding whether to index on a memory structure before or after the goal failure occurs, based on a trade-off of processing time vs. good planning performance. Figure 3-4 shows Dyer's indexing rules.

```
if the goal to be achieved has IMPORTANCE < NORM
  and the situation arises infrequently
then don't build access link to the associated TAU
   until most of the pattern has been recognized
   (possibly after a failure has occurred)

if the goal to be achieved has IMPORTANCE > NORM
  or the situation occurs frequently
then build access links to the associated TAU
   as early as possible (i.e. before a failure
   has occurred)
```

Figure 3-4: Dyer's TAU indexing rules

In our representation for TAUs, we simply have to move the TAU index to a concept *above* the → in the **'binding-spec'** to cause recognition to occur earlier. The index is moved *below* the current location of the → to make recognition occur later.

3.4 TAU Disambiguation

We need to deal with the case of multiple TAUs indexed off of the same structure. This happens in CRAM where **TAU-ULTERIOR** and **TAU-CONF-ENABL** are both indexed because both share the memory node **P-CONT-fail**. Since there are cases where a TAU indexed off a very general memory node such as **P-CONT-fail** will not be instantiated, we need to know when to stop looking for a particular TAU.

Since the processing is very heavily demon-based [RIES79, DYER83] the natural solution to this problem is to spawn a demon for each TAU indexed under the memory structure. The memory structure for **P-CONT-fail** is shown in Figure 3-5. Thus two demons will be spawned when **P-CONT-fail** is encountered .

*However this is not the only place where **TAU-ULTERIOR** is indexed. It is also indexed off of the memory node for lying as was mentioned in **Chapter 2**. The code for detecting lies is not implemented in CRAM and so it only finds them if they are mentioned as such.

```
       P-CONT
         ||
         ||  isa
         ||
    P-CONT-fail
         |
     +---+---+
     |       |
     |       |
     |       |
   TAU-    TAU-
   CONF   DECEIT
   ENABL
```

Figure 3-5 **P-CONT-fail**

Now we have to deal with the question of when to kill the demon for the TAU which is not present. The method used in CRAM is to have each instantiation of **'TAU-RECOGNITION'** count the number of concepts it finds after it is spawned (i.e. the ones below the → in Figure 3-1). The demons can then be queried to find out how many patterns they have recognized. Demons which fall behind in the number of concepts they recognize are killed. In the current implementation this is a numberic threshold.

3.5 Conclusions

For the recognition process, the representation of TAUs used by CRAM has several advantages over that used in BORIS [DYER83]. First, the indexing point in the TAU allows a quick decision to be made whether or not to continue processing a particular TAU. If the concepts above the → are not found, recognition stops. This means that a large number of TAUs can be indexed off of memory structures without imposing a large processing burden on the program.

An additional advantage is that if we want to re-index a TAU, we can use the indexing point, indicated by the → in the examples here, to make TAU recognition occur earlier or later for any particular TAU. The pattern pointed to by the → is used to locate the memory node at which to index the TAU.

Another advantage is that the representation gives us a metric to measure the "goodness" of the match during the recognition process. The metric is the number of concepts *recognized* by the **TAU-RECOGNITION** demon after it was spawned on that particular TAU. This number is com-

** A numeric threshold is probably not the best to decide when a demon should be killed, but it provides an approximation.

pared among TAUs spawned to explain the same **GOAL** failure. Again, this means we can index a number of TAUs off the same memory node without worrying about possible conflicts. For example, we saw that a story can have both **TAU-ULTERIOR** and **TAU-VANITY**, but we would not want a story to have the two adages,

"Pride goeth before a fall" and
"Nothing ventured, nothing gained"

because they are contradictory.* This is also important in using TAUs as planning critics.

*Contraditory themes is a literary device which is not accounted for by this theory.

CHAPTER 4
Representation and Memory Organization

This chapter describes the way that **ACT**s, **STATE**s, **GOAL**s, and **PLAN**s are represented in CRAM. It also covers the representation and organization of memory nodes. These are the same memory nodes used for TAU indexing in the previous chapters.

The representation format used in CRAM is slot/filler, sometimes referred to as frame representation. A conceptual object is represented with a 'head', indicating the type of object, and several 'slot/filler' pairs indicating properties of the object. A 'slot' is a property name and a 'filler' is the value of that property. Objects are grouped into classes which share the same set of possible 'slots'. For example, **ACT** and **STATE** are representation classes. The fillers for slots can be other concepts, with slots and fillers, or atomic concepts. Atomic concepts are those which have no slots. For example, the representation for

"The fox walked to the tree"

is

```
PTRANS actor FOX
       obj FOX
       to (PLACE name TREE)
       instr (MOVE actor FOX
                   obj (BODY-PART name LEGS
                                  owner FOX))
```

where the words in capital letters are either heads or atomic concepts and slots are indicated with lower case letters. In this example, the heads are: **PTRANS**, **MOVE**, **PLACE**, and **BODY-PART**. The atomic concepts are: FOX, TREE, and LEGS. Frame 'heads' are also called primitives when referring to the set of concepts which have that 'head'. In the above example **PTRANS** is a primitive **ACT**.

4.1 Memory Representation

The frames for memory nodes are called MFRAMEs. These are loosely related to Schank's MOPs (Memory Organization Packets) [SCHA82a] in that each MFRAME has an index to similar but more specialized MFRAMEs. Figure 4-1 shows the slots for a MFRAME.

MFRAMEs are organized into a specialization hierarchy [LEBO80, KOLO80] by the isa links and the spec-MFRAME discriminator.

```
MFRAME isa -parent-MFRAME-
       instantiation -conceptualization-
       spec-MFRAMEs -discriminator-
       demons -expectations-
       rules -processing-rules-
  +
 optional I-links
```

where the structure of I-links is

```
       link-name MFRAME
```

Figure 4-1: MFRAME structure

In addition, the 'instantiation' slot holds the concept that the MFRAME organizes. For MFRAMEs that hold concepts encountered in a story, this is an **ACT**, **STATE**, **PLAN**, or **GOAL** from the story. MFRAMEs that represent uninstantiated concepts have patterns, with free variables, which are matched against story concepts during memory incorporation. If an MFRAME has a filled in concept in its instantiation slot then it is part of an episode. If it has a pattern, then it is a concept template.

Concept template MFRAMEs also contain knowledge. This knowledge defines the meaning of the concept. The 'demons' slot tells which demons should be spawned if this memory structure is activated. In CRAM, demons are used for

-memory search
-inference
-expectation
-TAU recognition

Demons have three major parts: **TEST**, **+ACT**, and **KILL** [DYER83]. When a demon is spawned it is placed on the agenda. If its **TEST** part (a LISP expression) becomes true, the **+ACT** part (another LISP expression) is executed. If the **KILL** part becomes true before the **TEST** part does, the spawned demon is removed from the agenda. The 'rules' slot holds the set of all the processing rules which can be applied when connecting this MFRAME to another MFRAME.

MFRAMEs have 18 optional slots, one for each direction of the 9 possible I-links of **Section 1.3.4.** The I-links with the names for forward and backward directions are given below.

LINK	Forward	Backward
intention	intends	intended-by
achievement	achieves	achieved-by
realization	realizes	realized-by
enablement	thwarts	thwarted-by
plan-blocking	blocks	blocked-by
resulting	resulting-in	result-of
disablement	disables	disabled-by
motivation	motivates	motivated-by

There are also two links for temporal relations: 'follows' and 'followed-by'.

4.2 **ACT**, **STATE**, **GOAL**, and **PLAN** Representation

The ACTs used in this implementation are those given in **Section 1.3.1**, Schank's CD **ACTs**. The 'heads' for the **ACTs** are: **ATRANS**, **PTRANS**, **PROPEL**, **MOVE**, **GRASP**, **INGEST**, **EXPEL**, **MTRANS**, **MBUILD**, **SPEAK**, **ATTEND**. There are 7 possible slots an **ACT** can have: actor, obj, from, to, instr, mode, manner.

More complex conceptual activities, such as "fishing", are represented with pointers to the memory node for that activity and a slot for the 'actor'. The representation for "The Fox is fishing" is

```
MFRAME isa &FISHING
       actor FOX
```

Words in capital letters prefixed by a '&' are pointers to concept templates.

There are 4 primitives for **STATE**s in the forest micro-world.

APPRAISAL - a subjective judgement on the quality of something
POSS-BY - a character possessing an object
LOC - the location of an object or character
HEALTH - the physical health of a character

These primitives take 4 slots:

mode - can be filled by either POS or NEG to
indicate the truth or falsity of the **STATE**.
actor - the character or object to which the **STATE** refers
obj - the value of the **STATE**
manner - a modifier

For example, the representation for "The Fox says the Crow has a good voice" is

```
MTRANS actor FOX
       obj (APPRAISAL actor (BODY-PART name MOUTH
                                       owner CROW)
                          obj GOOD
                          manner SONG)
```

The tag GOOD is used as part of a three place ordering: GOOD, NEUTRAL, and BAD. This ordering is used in CRAM any place a metric is required.

There are two types of objects in CRAM: primitive objects, which are represented as atomic concepts, and complex objects. Examples of primitive objects are: FOX, BEAR, and HONEY. These are primitive because for this domain, the decision was made not to provide a more complex representation for these objects. In the original conceptual dependency theory [SCHA72] these were referred to as picture producers. There is some information associated with primitive objects. For example, FOX and BEAR are tagged as animates, and HONEY is tagged as a food. Besides the primitive objects in the forest two types of complex objects are required. The complex object types are

BODY-PART and
PLACE

The slots for complex objects are

owner - the owner of the **BODY-PART** or the character who lives at the **PLACE**.
name - a distinguishing name for the object
spec - for objects with sub-parts, a specification of a sub-part (e.g. BRANCHES of the TREE)
relation - for **PLACE**s, another **PLACE** with a specific relationship to the first.

The representation for the complex objects is designed to have the same characteristics as the primitive **ACT**s, namely, that conceptually related objects have similar representations but that differences are also reflected. For example, when representing the location of characters who are "at the stream" we want to make sure that the representation reflects the close proximity of the characters. Hence we may infer that these characters can talk to each other. However, we don't want to say that all the characters who are near the stream are in it, or CRAM would infer that they were all engaged in an activity associated with the stream (e.g. bathing, fishing, drowning, etc.). For example, the representation for "near the bank of the stream" is

* This is obviously not an adequate representation of "good voice", but the emphasis in this research was not placed on finding a correct representation for "talent". Instead, the emphasis was placed on using a representation in detecting planning errors, given that we have *some* adequate representation for "talent".

```
PLACE name STREAM
      spec SHORE
      relation NEAR
```

This specifies a **PLACE** which is at the STREAM, more specifically, the SHORE, and in particular not exactly at the SHORE, only NEAR it.

GOALs and **PLANs** are represented with simply an **'actor'** for the desirer and an **'obj'** for the objective of the desire. A **GOAL** represents the desire of a character; a **PLAN** is a decision by a character to take some action to bring about a **GOAL**. CRAM uses the **GOAL** types of Schank and Ableson [SCHA77] described in **Section 1.3.2**. The **GOAL** types required for the stories presented here are **D-GOALs** and **P-GOALs**. The delta **GOALs** used are **D-KNOW**, for the goal of a character knowing something, and **D-CONT**, for the goal of a character gaining possession of an object. The **P-GOALs** used are **P-CONT**, for the preservation of control of an object, and **P-HEALTH** for preservation of physical well-being. In order to denote **GOAL** and **PLAN** types, we use pointers to memory nodes for the generic **GOAL** and **PLAN** types. For example there are memory nodes for **P-CONT**, **D-CONT**, and **P-CONT-fail**. A representation for the Fox having a **D-CONT** goal for the cheese is

```
MFRAME isa &D-CONT
   instantiation
      (GOAL actor FOX
            obj (POSS-BY actor FOX
                         obj CHEESE))
```

Along the same lines,

```
MFRAME isa &GRAB-PLAN
      (PLAN actor FOX
            obj (POSS-BY actor FOX
                         obj CHEESE))
```

represents an instantiation of **GRAB-PLAN** by the Fox to get the cheese.

This representation is more flexible than primitives for specific **GOALs** and **PLANs** because we don't need to specify the **GOAL** and **PLAN** types at parse time. For example, the sentence, "The Crow was disappointed"*, can be represented by

```
GOAL actor CROW
     manner FAIL
```

Later, if a story mentions the reason for the failure, the **'obj'** slot and the pointer to the **GOAL** type can be filled in. Likewise, we can represent "The Fox tried to get the cheese" with the **PLAN** above, without the pointer, i.e.,

* This representation does not deal with issues of affect. For a good treatment of this subject, see Dyer [DYER83].

```
PLAN actor FOX
     manner FAIL
     obj (POSS-BY actor FOX
                  obj CHEESE)
```

since we don't know the actual plan that the Fox employed.

4.3 Memory Organization

Two of the slots in the MFRAME frame are used for implementing hierarchical memory: '**isa**' and '**spec-mops**'. The '**isa**' slot is used to point to the generalization of a memory node. For example, **P-CONT** is a generalization of both **P-CONT-fail** and **P-CONT-succeed** because it does not specify the status of the **GOAL**.

The '**spec-mops**' slot points to a discriminator which determines if a given concept is an occurrence of a more specialized memory node. Figure 4-2 gives the discrimination net for **PTRANS**. Figure 4-3 gives the internal representation for **&PTRANS**.

```
              PTRANS
                 |
     actor=      |
     GRAVITY     |    actor=obj
      +----------+---------+
      |          |         |  from=
  &FALLING       |         |(PLACE ...)
                 |         +---------+
     manner=     |                   |
      JUMP       |                   |
      +----------+               &LEAVING
      |          |
  &JUMPING       |
                 |
              instr= MOVE obj (BODY-PART name LEGS)
                 |
      +----------+
      |
  &WALKING
```

Figure 4-2: Discrimination net for **PTRANS**

The elements of the **NET** in figure 4-3 are pairs,

[pattern memory-node]

where '**pattern**' specifies the additional conceptual features which discrim-

```
MFRAME isa &ACT
  actor ?x
  instantiation
    PTRANS actor ?x from ?z
              to ?y obj ?w
  spec-mops
    NET
      [(PTRANS instr (MOVE obj (BODY-PART name (LEGS))))
       &WALKING]
      [(PTRANS manner (JUMP)) &JUMPING]
      [(PTRANS actor (GRAVITY)) &FALLING]
      [(PTRANS actor ?x
               obj ?x
               from (PLACE))

  rules
    RULE-SET
      &movement-rule
      &getting-there-rule
```

Figure 4-3: The MFRAME for **PTRANS**

inate **'memory-node'** from its generalization. So, for example, if a **PTRANS** has **'actor'** GRAVITY, then it is an occurrence of &FALLING. The last [ ]'ed pair shows the discriminator for a character departing from a location. The discriminating features are that the **'from'** slot is filled and the **'actor'** and **'obj'** slots are the same. Phrases or words in lowercase preceded by an '**&**' are processing rules. Figure 4-4 shows the memory nodes for &FALLING and &LEAVING.

Consider the **RULE-SET**s used by these three MFRAMEs. There are two rules applicable to all **PTRANS**es. The **'&movement-rule'** expresses the knowledge that a **PTRANS** changes the **LOC** of an object. The **'&getting-there-rule'** states that a **PTRANS** to a place is connected with an **'achievement'** link to a **GOAL** of being at that place. These rules are general and apply to all physical movement of objects. For that reason they are indexed under &PTRANS and not more specific nodes. If we did not use an MFRAME hierarchy, we would have to repeat these rules for each of the more specific forms **PTRANS** can take. This would be quite inefficient from a processing point of view and also would not reflect the kinds of generalizations that humans make.

The more specific nodes, **&FALLING** and **&LEAVING**, have more specific rules. The **'&drop-rule'**, indexed off the node **&FALLING**, states that jumping up can cause a character to drop what it is carrying. These nodes also have demons associated with them. **&FALLING** has a demon to check whether a **P-CONT-fail** should be instantiated. **&LEAVING** spawns a demon

```
               +-----+
               | ACT |
               +-----+
                  |
                  |isa
             +----------+
             | PTRANS   |
             |    rules |
             +----------+
                  |
          +-------+---------+
          |                 |
          |isa              |isa
  +-----------+    +------------+
  | FALLING   |    | LEAVING    |
  |    demons |    |     rules  |
  +-----------+    |     demons |
                   +------------+
```

Figure 4-4: Memory nodes for **&FALLING** and **&LEAVING**

to update the database and assert that the character who left is no longer at the **'from'** slot.

4.4 Where Hierarchical Organization Fails

Even in the forest micro-world we have situations where the discrimination hierarchy does not work. There is not always a primitive to use as a conceptual feature. One place where this happens is in the story **"The Fox and the Bear"**. Here there is mention of an activity, **FISHING**, which cannot be classified based on a conceptual **ACT**. There is no single **ACT** associated with fishing. However there is a distinct memory structure for this activity. In this model, memory structures without specific instantiating **ACT**s serve three purposes.

The first is to provide memory connections to other structures. Some **PLAN**s and **GOAL**s are **'achieved-by'** and **'realized-by'** larger activities. For example, the **S-HUNGER** goal and the **CATCH-FOOD-PLAN**, intended by this goal, are both connected to the **FISHING** structure by **'achieves'** and **'realizes'** links respectively. Another way to think of higher level memory nodes is as scripts that only get mentioned, not fully instantiated. That is because in CRAM there is no more detailed representation of what it means to fish, only the abstract concept connected to other concepts.

The memory nodes for higher level activities also organize inference rules and expectations associated with that activity. For example, there are props and settings associated with an activity. The mention of these things needs to be connected to another node for the instantiation of that activity.

In **The Fox and the Bear**, the mention of the fish on the bank is linked to the instantiation of the **FISHING** structure with a **'resulting'** link. There is an expectation from **FISHING** to see fish and also to see characters eating fish. This allows the mention of the prop to be connected to the concept for the activity. The organization of memory around **&FISHING** is shown in Figure 4-5.

```
GOAL                                                 PLAN
  !                                                    !
  !isa                                                 !isa
S-HUNGER<----------FISHING---------->CATCH-FOOD-PLAN
  ^         achieves              realizes             ^
  !                                                    !
  +----------------------------------------------------+
                        intention
```

Figure 4-5: Memory organization around **&FISHING**

Memory nodes also provide a higher level representation than simple **ACT**s and **STATE**s. For example when a parser comes across the sentence,

"The Bear was fishing in the stream"

it needs a structure with which to represent this. In the CRAM model, this sentence is represented with an instantiation of **FISHING** and a **LOC**. The representation is shown in Figure 4-6.

```
MFRAME isa &FISHING
       actor BEAR

LOC actor BEAR
    obj (PLACE name STREAM)
```

Figure 4-6: Representation **FISHING**

4.5 Bottom-Up Recognition of Structures

There are two ways that structures such as **FISHING** can be instantiated. The first was alluded to above; we can determine that a particular structure should be instantiated from some lexical item in the input. This lexical item may be a word or an idiomatic phrase. Another method is through expectation. In CRAM, whenever a scene (**LOC**) is mentioned, and does not get connected to any other structure, expectations are set up for all the activities which usually take place in that setting. This is the case with the STREAM and **FISHING**. Mention of the STREAM causes an expectation for other elements of the **FISHING** activity, specifically FISH. These higher level activities have no **ACT** associated with them, but, as in Figure 4-5, they are linked to other **ACT**s, **PLAN**s, and **GOAL**s. The concepts linked to the activity are used as expectations. Subsequent inputs are checked to see if they satisfy expectations for a particular activity. In this manner, the mention of the Bear at the stream and the fish on the bank are sufficient to instantiate the **FISHING** structure.

CHAPTER 5
Process Model

The process model is the description of how the knowledge structures, described in the last chapter, are used to understand a story. The parts of the process model that are covered in this chapter are:

- Memory Incorporation - finding the right memory node under which to index a concept.
- Expectation - generating expected events from the memory node found for a concept.
- Causal Reasoning - the application of processing rules for forming I-links.

The top level processing loop for CRAM is given in Figure 5-1. The words in **CAPITALs** will be explained in the following sections. The '*'ed entries are where inputs can get **EXPLAINED?**. A concept is considered explained if it is connect with an I-link to some other concept in the story.

```
for each concept in the story
    MEMORY INCORPORATION(concept)
    CHECK EXPECTATIONS
    if EXPLAINED?(concept)* then
        SPAWN DEMONS
    else
        RUN ACTIVE CAUSAL RULES on(concept)
        if EXPLAINED?(concept)* then
            SPAWN DEMONS
        else
            ADD concept TO *WORKING-MEMORY*
            SPAWN DEMONS
```

Figure 5-1: CRAM top level understanding loop

5.1 Memory Incorporation

The placement of a concept in the memory hierarchy is very important. Placing a concept under too general a memory structure will inhibit the application of knowledge potentially applicable to the situation. For example, when classifying a **PTRANS**, it is important to be able to tell "falling" from "departing" as in **Chapter 4**. The node for "falling" has knowledge about characters possibly losing control of the object which fell. The node for "departing" has information about **GOALs** associated with departing and expectations about arrivals later in the story.

There are two ways that an input gets placed in memory. One is bottom-up, by specific reference. For example, the representation of the sentence,

"The bear was fishing in the stream"

is immediately placed at the memory structure **FISHING**.

The other type of incorporation, the type referred to by **MEMORY CLASSIFICATION** in Figure 5-1, is top-down. For each memory node, there is an index of specializations of the concept it holds. For example, in the sentence, "John killed his brother", we can get bottom-up to the **MURDER** node, but it takes a top-down process using the **'spec-mop'** index of **MURDER** to check the relation between the killer and victim to get to the **FRATRICIDE** memory structure.

During the memory incorporation phase, a concept is compared to the **'spec-mop'** index of the MFRAME where it is initially classified. This may result in the concept being placed at a more specific memory structure. This process is repeated, classifying the concept at successively more specific MFRAMEs, until no further specializations can be found.

As an example, take the sentence, "The Fox told the Bear there was honey at the tree", which is represented as,

```
MTRANS actor FOX
       to BEAR
       obj (LOC actor HONEY
                obj TREE)
```

The first step is to recognize that this is an **MTRANS**. Then CRAM looks at the **'spec-mops'** discriminator for the generic **MTRANS** and sees that this is an occurrence of the structure for telling someone the location of something, **MTRANS-loc**. This classification makes knowledge associated with **MTRANS-loc** available for explaining this and subsequent inputs. Some knowledge structures indexed under **MTRANS-loc** are expectations for **D-CONT**rol **GOALs** motivated by the **MTRANS**.

Figure 5-2 shows a portion of CRAM's memory before any stories are processed. The tree structures indicate **'isa'** links. The diamond under **P-CONTfail** indicates a choice between two possible TAUs. For clarity, the I-links which also connect these memory nodes are not shown.

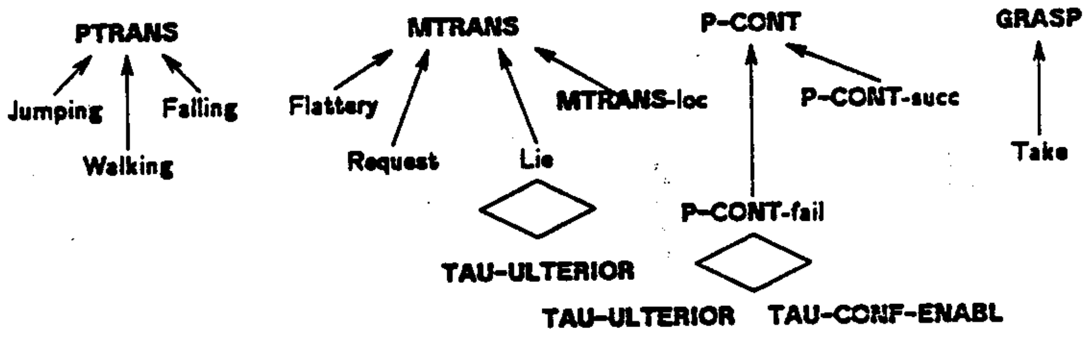

Figure 5-2: Empty CRAM memory

After CRAM has processed some stories it has concepts from those stories indexed in its memory. Figure 5-3 shows CRAM's memory after processing **The Fox and the Apple** and **The Fox and the Bear**. At the bottom of the diagram is shown the processing state near the end of **The Fox and the Crow**. The labeled arcs indicate I-links. The box around **TAU-ULTERIOR** indicates that CRAM has just determined that this TAU is contained in the story.

Figure 5-3 shows CRAM's memory after all three stories have been processed. The 'X'ed arcs indicate the concepts which CRAM uses to index the new instances of the TAUs it has found.

5.2 Expectations

With each MFRAME comes a set of expectations which are activated each time a concept is classified at that node. For example, in the forest micro-world, indexed under **MTRANS-loc** is an expectation,

```
If the thing is something the hearer likes (food)
then EXPECT a goal to have the object
     or a sub-goal for a plan to get the object
```

Expectations can explain an input much more efficiently than running through all the applicable processing rules. In the example above there is a standard **GOAL**, **S-HUNGER**, motivated by acquiring new knowledge about food. It is more efficient to generate this expectation off of **MTRANS-loc** than to perform a **PLAN/GOAL** analysis after the character has already started to achieve the **GOAL**.

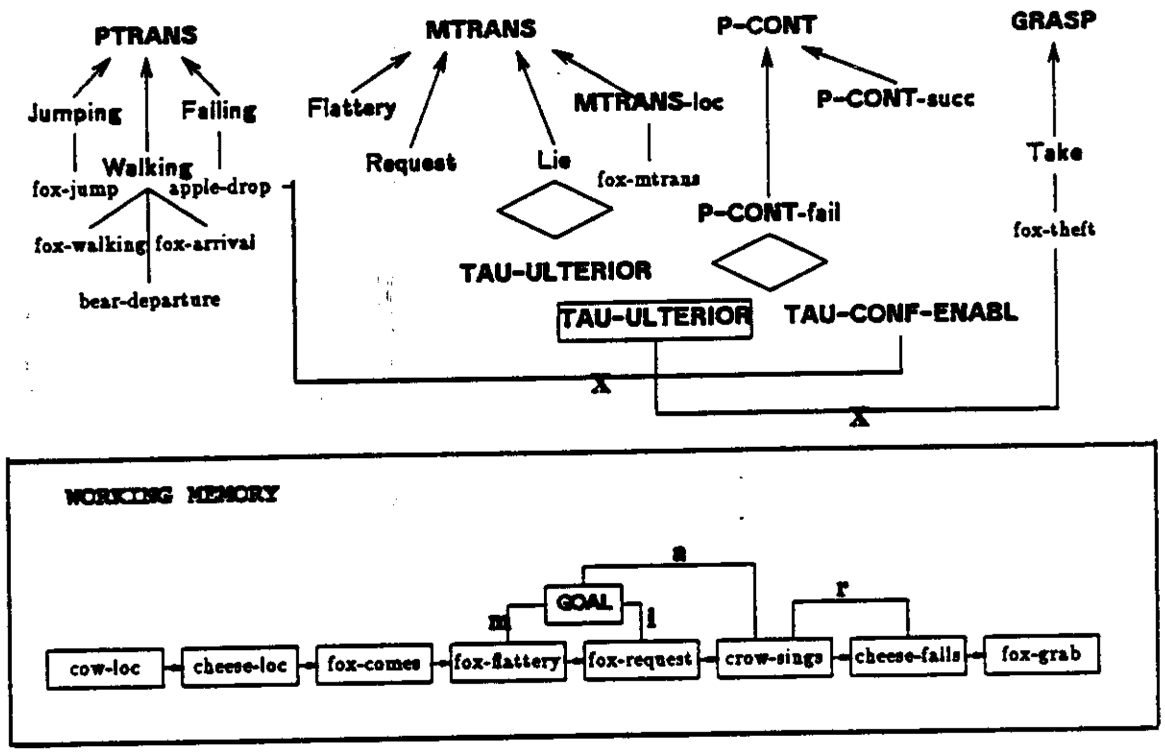

Figure 5-3: CRAM's memory with 2 1/2 stories

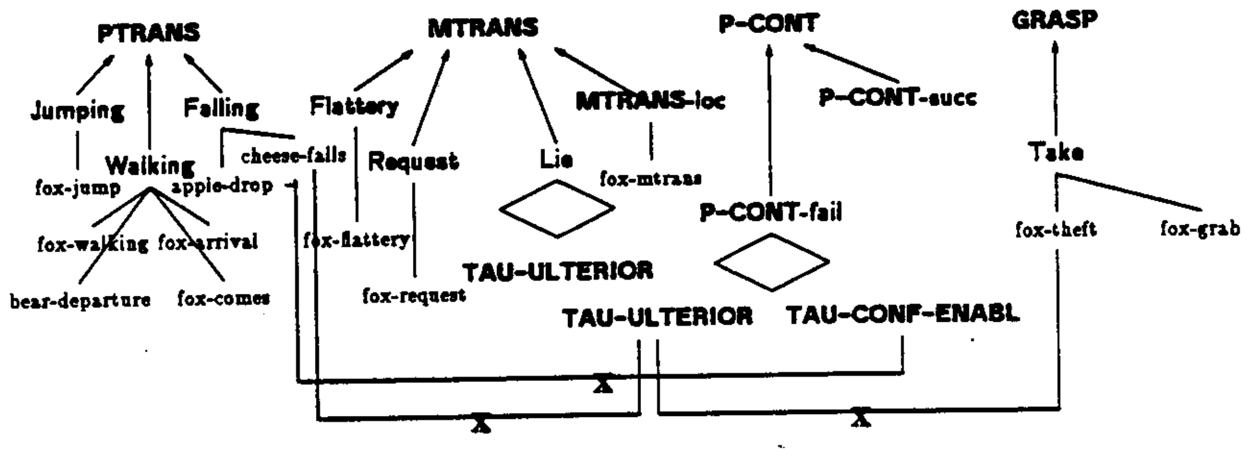

Figure 5-4: CRAM's memory with 3 stories

5.2.1 Expectation indexing - The expectations are indexed off of two places in the memory structure. (1) They are indexed off of MFRAMEs. These expectations are spawned when the new conceptualization is either explained or added to ***CLASSIFICATION-WM*** as shown in Figure 5-1. (2) Expectations are also indexed off of processing rules. When a rule fires, it may spawn some expectations. This is good because the rule acts as an extra condition on the demon's activation. Indexing expectations off of rules takes more knowledge out of the demons, where it is represented as LISP code, and puts in into the declarative 'pattern-&-constraint' notation used by TAUs and processing rules. The 'pattern-&-constraint' notation is explained in the next section.

The demons take the input concept and use its slots to create a new concept. The demons are run at every cycle to see if the expected input has occurred. When it does, the I-link is made and the new input is considered explained in the sense of the **EXPLAINED?** test of Figure 5-1.

5.3 Processing Rules

Processing rules are constrained causal and intentional links. Two example processing rules, **'&drop-rule'** and **'&too-high-rule'** are given in Figure 5-5.

The first rule states that a character jumping up can cause him to drop what he is carrying. The second rule states that the location of an object can thwart a goal to get it, if the object is above the character who wants it.

The process of rule application is as follows:

```
if both the 'ante' and 'conseq' patterns are found
   in working memory
   and the constraints are true
then form the I-link indicated between the two concepts
```

5.4 Reasoning in Hierarchical Memory

This section describes how CRAM uses its memory hierarchy to reasoning about concepts. Most of the power of CRAM's reason process comes from the way concepts are indexed in memory and the way knowledge is shared among structures in the hierarchy.

5.4.1 Processing Rule Inheritance - Processing rules express general knowledge about possible situations which might occur in memory. They are not fired unless both the **'ante'** and **'conseq'** parts are matched by elements of working memory and all the **constraints** are satisfied. The reasoning mechanism can be characterized as passive. For this reason it is completely safe to allow processing rules to be inherited from more general down to more specific situations, as opposed to active methods such as hypothesis and test. Active methods do not work well with generalization hierarchies because often more specialized situations make hypotheses created from

```
IF the concepts
   ?ante = (PTRANS actor ?x
                   obj ?x
                   manner JUMP) and
   ?conseq = (PTRANS actor GRAVITY
                     obj ?y
                     from (BODY-PART name ?z
                     owner ?x))
   are found in *WORKING-MEMORY* and the constraint
      location(?y,(BODY-PART name ?z owner ?x))
   is met
THEN assert a RESULT link between ?ante and ?conseq

IF the concepts
   ?ante = (LOC actor ?x
                obj ?y) and
   ?conseq = (GOAL actor ?z
                   obj (POSS-BY actor ?z
                                obj ?x))
   are found in *WORKING-MEMORY* and the constraint
      above(?y,?z)
   is met
THEN assert a THWART link between ?ante and ?conseq
```

Figure 5-5: Example processing rules

more general situations obsolete. An example of this is the forward inference from the **GRASP ACT** that a character has a **D-CONT GOAL** which the **GRASP** achieves. At a more specific node, **GRASP-food**, the proper forward inference is that the character has an **S-HUNGER GOAL** which it is achieving. In the current implementation of CRAM, this mistake is made because there is no memory node for **GRASP-food**.

Inheritance allows for a more efficient memory organization. All that has to be specified is that a node is, for example, a **D-CONT** and all the knowledge applicable to that **GOAL** type is available.

5.4.2 Expectation Non-inheritance - There are two reasons why expectations are not inherited from more general situations. The first was mentioned above. Expectations are an active reasoning mechanism. The second reason is that expectations are created in people by seeing a specific event sequence repeated a number of time under certain circumstances. The circumstances are embodied by all the discriminations it takes to go down the memory hierarchy to a particular MFRAME. The event sequence is the combination of the event which is classified at that node and the ones which are expected. Expectation inheritance would not model the way people create expectations.

5.4.3 I-link Inheritance - Inheritance of I-links is important because we want to accurately model the default reasoning processes of people. People are too good at common sense reasoning to be working very hard at it. That is, they cannot possiblely be cycling through a huge database of inference rules all the time. Therefore we must postulate some memory-intensive organization of real world knowledge about **PLANs**, **GOALs**, **ACTs**, and **STATEs** which will allow CRAM to process concepts efficiently.

In contrast to this design goal, PAM [WILE78] completely instantiated the **PLAN/GOAL** structure for each input until it was completely explained. This created a large memory structure for each story. CRAM, however, allows inputs to exist unconnected to other inputs unless the knowledge for making the connection is right at the surface, i.e. indexed at MFRAME for the specific situation or inherited in causal rules from more general situations.

Inheritance of I-links from higher levels allows defaults to be used when they are needed to satisfy either rule or TAU **'constraints'**. For example, CRAM uses an I-link going one way at a very high level to express that for each **PLAN** there must have been a **GOAL**. If the **GOAL** is ever actually needed, it can be derived by a simple pattern match with the **PLAN** and an instantiation of the **GOAL**. Figure 5-6 shows a portion of the CRAM memory with the default I-links in place. Default I-links are those which can always be expected to hold.

```
                                                     grabbing somthing
+-------------achieves--------------+                 of someone else's
!                                   !                        .
!  D-GOAL<--intended-----PLAN       +-----GRASP  .
!    !         by         !                  !    .
!    X                    X                  X    .
!    !                    !                  !    .
+-->D-CONT<--intended--GRAB-PLAN<-------->GRASP-OBJECT
               by                realiz-      !
                                 ation        X
                                              !
                                      GRASP actor FOX
                                            obj CHEESE
```

X indicates specializations which are indexed under a memory node.

Figure 5-6: CRAM memory with default I-links

5.4.4 Pattern Based Reasoning - Because of the declarative nature of the representation presented here, all the reasoning is heavily dependent on the pattern matcher. For example, in order to derive the implicit goal for a **GRAB-PLAN**, the pattern for the **PLAN**,

```
PLAN actor ?x
     obj (POSS-BY actor ?x
                  obj ?y))
```

is matched with the actual input pattern. The intended-by slot of **GRAB-PLAN**, Figure 5-3, is followed back to **D-CONT** which has as its pattern,

```
GOAL actor ?x
     obj (POSS-BY actor ?x
                  obj ?y))
```

which is then instantiated with the proper character and object from the **PLAN**. I-links to generic structures express default conditions. For this reason there is no complementary intends I-link from **D-CONT** to **GRAB-PLAN** because there are several **PLANs** available for that **GOAL**.

Another example is finding a GOAL from an **ACT** which 'achieves' that GOAL. Referring to Figure 5.3 we can see that there is a default I-link, 'achieves', from the generic **GRASP** to the **D-CONT** node. To get the instantiated goal, take the specific **GRASP**,

```
GRASP actor FOX
      obj CHEESE
```

and match it with the generic **GRASP**,

```
GRASP actor ?x
      obj ?y
```

and instantiate the generic **D-CONT**,

```
GOAL actor ?x
     obj (POSS-BY actor ?x
                  obj ?y)
```

with the bindings from above to get

```
GOAL actor FOX
     obj (POSS-BY actor FOX
                  obj CHEESE)
```

5.5 Conclusions

One of the important ideas in this theory of understanding planning failures is that we need a tight, well-defined memory representation and a good clean processing model in order for a TAU to be represented non-procedurally. As we saw in **Chapter 2**, we need to make statements in representing TAUs about both the state of processing (**TAU-ULTERIOR**) and the state of the memory representation (**TAU-TOO-RISKY**). This process model provides those features.

CHAPTER 6
Implementation Details and Trace

The current implementaion of CRAM has the following knowledge structures:

29 memory nodes
5 TAUs
10 causal rules
11 demons
15 constraint types

The code for the demon management is an adapation of the McDYPAR code [DYER83, MOOR84, DOLA84]. The representation system used was **T-CD** [DOLA84]. The following is an annotated trace of the CRAM model running on **The Fox and the Bear**. The entire run, with all trace facilities turned on, took just over 3 minutes on an Apollo DN300 in T. The majority of the CPU time was spent in the pretty printer.

```
> (index-story fox/bear-story)
```

Figure 2-1 is repeated here. It gives an abbreviated representation for the story. The text for the story is in **Section 1.1**.

```
LOC actor BEAR obj STREAM
M-FISHING actor BEAR
POSS-BY actor BEAR obj FISH
PTRANS actor FOX to STREAM
MTRANS actor FOX to BEAR
          obj (LOC actor HONEY
                   obj TREE)
PTRANS actor BEAR from STREAM
GRASP actor FOX obj FISH
```

```
** Starting to index a new story **
```

Each conceptualization in the story is labeled with a mnemonic name to help the reader in interpreting the trace. None of the names are used in the processing

```
Indexing #{REF.1073 &BEAR-LOC}
```

The conceptualization for "The Bear was at the stream."

```
Classifying (LOC ACTOR (BEAR)
                 OBJ (PLACE NAME (STREAM)))
```

When no bottom-up inferences are assumed, classification starts at the very top of the MFRAME tree.

```
Starting at #{REF.285 &MFRAME}
```

Note how the classification process moves down to more specific levels.

```
    Moving to #{REF.289 &STATE}
    Moving to #{REF.433 &LOCATION}
    Moving to #{REF.803 &PLACE-LOCATION}
```

&PLACE-LOCATION is the memory node for a thing's being at a location. It indexes knowledge about how things get to places.

```
Placed at #{REF.803 &PLACE-LOCATION}
```

In the current implementation of CRAM the entire conceptualization and only the conceptualization is used to index the new MFRAMEs.

```
#{REF.1073 &BEAR-LOC} placed, indexing off
  #{REF.803 &PLACE-LOCATION} with
(LOC ACTOR (BEAR)
     OBJ (PLACE NAME (STREAM)))

Running demons
```

Explaination fails because there are no other MFRAMEs in working memory to which to connect this structure.

```
Attempting to explain #{REF.1073 &BEAR-LOC} in terms of,
#{REF.803 &PLACE-LOCATION} and one of *CLASSIFIER-WM* nodes.

#{REF.1073 &BEAR-LOC} not explained. Adding another node
   to *CLASSIFIER-WM*
#{WM-NODE.2223 #{REF.1073 &BEAR-LOC}}

Indexing #{REF.1074 &BEAR-FISHING}
```

This is the representation for "The Bear was fishing." As we saw in **Section 4.4** there is no primitive representation for this concept. **NO-ACT** is an **ACT** used as a place holder for such representations.

```
Classifying (NO-ACT)
```

It is assumed that the parsing process would immediately place any mention of fish at the memory node **M-FISHING**.

```
Starting at #{REF.1001 &M-FISHING}

Placed at #{REF.1001 &M-FISHING}

#{REF.1074 &BEAR-FISHING} placed, indexing
  off #{REF.1001 &M-FISHING} with
(NO-ACT)

Running demons

Attempting to explain #{REF.1074 &BEAR-FISHING} in terms of
#{REF.1001 &M-FISHING} and one of *CLASSIFIER-WM* nodes.

Trying to explain #{REF.1074 &BEAR-FISHING} in terms
   of *CLASSIFIER-WM*
```

The explanation process works up the memory tree from the node in working memory in parallel. As an example, if we had a memory hierarchy like this,

```
       1          2
      /|\        / \
     / | \      /   \
    3  4  5    6     7
   / \
  8   9
```

and nodes 6 and 9 were in working memory, the knowledge structures associated with each node would be examined in the following order.

6 and 9, then
2 and 3, then
1

That way, a very general explanation rule is not applied when there is a more specfic one available from another node.

```
Explaining #{REF.1074 &BEAR-FISHING} in terms of
(#{REF.1074 &BEAR-FISHING} #{REF.1073 &BEAR-LOC})

Explaining #{REF.1074 &BEAR-FISHING} in terms of
(#{REF.1001 &M-FISHING} #{REF.803 &PLACE-LOCATION})

Explaining #{REF.1074 &BEAR-FISHING} in terms of
(#{REF.285 &MFRAME} #{REF.433 &LOCATION})
```

```
Explaining #{REF.1074 &BEAR-FISHING} in terms of
(#{REF.289 &STATE})
```

There are two expectation demons indexed off of **M-FISHING**, SETTING-EXPECT and PROP-EXPECT to set up expectations for mentions of the STREAM and FISH.

```
    Spawning demon: SETTING-EXPECT.2235
    (SETTING-EXPECT
      #{WM-NODE.2234 #{REF.1074 &BEAR-FISHING}}
      (STREAM))

======================== SETTING-EXPECT.2235 ============
TEST:    "Look for a setting which will be explained by"
         "the MFRAME which spawned this demon"

ACT:     "Connect the two MFRAMEs"
=========================================================


    Spawning demon: PROP-EXPECT.2236
    (PROP-EXPECT #{WM-NODE.2234 #{REF.1074 &BEAR-FISHING}}
                 (FISH))

======================== PROP-EXPECT.2236 ================
TEST:    "Look for the mention of a prop associated"
         "with the MFRAME on which this demon is spawned"

ACT:     "Connect the MFRAME to the first MFRAME"
==========================================================

#{REF.1074 &BEAR-FISHING} not explained. Adding another
   node to *CLASSIFIER-WM*
#{WM-NODE.2234 #{REF.1074 &BEAR-FISHING}}
```

At this point the *CLASSIFIER-WM* has the following structure,

```
              &BEAR-LOC        &BEAR-FISHING
                  !                  !
              wm-node.2234     wm-node.2223
```

```
Indexing #{REF.1078 &BEAR-HAS-FISH}
```

&BEAR-HAS-FISH is the representation for "The Bear has some fish."

```
Classifying (POSS-BY ACTOR (BEAR)
                      OBJ (FISH))
Starting at #{REF.285 &MFRAME}

    Moving to #{REF.289 &STATE}
    Moving to #{REF.432 &POSSESSION}
```

The node &POSSESSION is the structure for possesion of objects by actors. The only knowledge contained there is that possesion achieves the **GOAL D-CONT**.

```
Placed at #{REF.432 &POSSESSION}

#{REF.1078 &BEAR-HAS-FISH} placed, indexing
   off #{REF.432 &POSSESSION} with
(POSS-BY ACTOR (BEAR)
         OBJ (FISH))

Running demons
```

The demon set up to recognize the occurence of FISH is fired. The BEAR possessing fish is explained as a result of the M-FISHING occurence. Because the current input was explained with an expectation, it is not processed with any causal rules.

```
    Executing demon: PROP-EXPECT.2236
Linking #{REF.1074 &BEAR-FISHING} and
        #{REF.1078 &BEAR-HAS-FISH}
 with RESULT mode FORWARD
    Killing demon: PROP-EXPECT.2236
```

This time the new concept is not added to the ***CLASSIFIER-WM*** but connected to another node in the memory. The working memory now looks like this,

```
                        &BEAR-FISHING
                              |
                              r
                              |
        &BEAR-LOC       &BEAR-HAS-FISH
            |                 |
        wm-node.2234    wm-node.2223
```

where the 'r' indicates the 'resulting' I-link.

```
Indexing #{REF.1047 &FOX-ARRIVAL}
```

&FOX-ARRIVAL is the representation for "The Fox walked over to the stream."

```
Classifying (PTRANS ACTOR (FOX)
                   TO (PLACE NAME (STREAM))
                   OBJ (FOX)
                   MANNER (WALKING))
Starting at #{REF.285 &MFRAME}

    Moving to #{REF.288 &ACT}
    Moving to #{REF.378 &PTRANS}
```

This is classified as a generic **PTRANS**.

```
Placed at #{REF.378 &PTRANS}
```

The FOX arrives on the scene.

```
#{REF.1047 &FOX-ARRIVAL} placed, indexing
   off #{REF.378 &PTRANS} with
(PTRANS ACTOR (FOX)
        TO (PLACE NAME (STREAM))
        OBJ (FOX)
        MANNER (WALKING))

Running demons

Attempting to explain #{REF.1047 &FOX-ARRIVAL} in terms of
#{REF.378 &PTRANS} and one of *CLASSIFIER-WM* nodes.

Trying to explain #{REF.1047 &FOX-ARRIVAL} in terms
   of *CLASSIFIER-WM*
```

Here we see CRAM working up the memory hierarchy from the nodes in the ***CLASSIFIER-WM***. It starts at the bottom with the nodes which are actually in the memory,

```
Explaining #{REF.1047 &FOX-ARRIVAL} in terms of
(#{REF.1047 &FOX-ARRIVAL} #{REF.1078 &BEAR-HAS-FISH}
 #{REF.1073 &BEAR-LOC})
```

It continues by going up from each of those nodes in parallel.

```
Explaining #{REF.1047 &FOX-ARRIVAL} in terms of
(#{REF.378 &PTRANS} #{REF.432 &POSSESION}
 #{REF.803 &PLACE-LOCATION})
```

Whenever a conceptualization matches the **ante** or **conseq** parts of a rule, the match is announced before the constraints are examined. The constraints for &MOVEMENT-RULE are not satisfied by this situation.

```
Attempting to explain #{REF.1047 &FOX-ARRIVAL} in terms
   of #{REF.188 &MOVEMENT-RULE}
```

The &WALKING-ALONG-RULE tries to explain a character's arrival at some place in terms of a previous mention of the character just "walking along" to no place in particular.

```
Attempting to explain #{REF.1047 &FOX-ARRIVAL} in terms
   of #{REF.177 &WALKING-ALONG-RULE}

Explaining #{REF.1047 &FOX-ARRIVAL} in terms of
(#{REF.288 &ACT} #{REF.433 &LOCATION})

Explaining #{REF.1047 &FOX-ARRIVAL} in terms of
(#{REF.289 &STATE})

#{REF.1047 &FOX-ARRIVAL} not explained. Adding another node
   to *CLASSIFIER-WM*
#{WM-NODE.2249 #{REF.1047 &FOX-ARRIVAL}}
```

The working memory now looks like this,

```
                            &BEAR-FISHING
                                  |
                                  r
                                  |
&BEAR-LOC               &BEAR-HAS-FISH          &FOX-ARRIVAL
    |                             |                   |
wm-node.2234              wm-node.2223          wm-node.2249
```

The FOX tells the BEAR about the location of some honey.

```
Indexing #{REF.1082 &FOX-MTRANS}

Classifying (MTRANS ACTOR (FOX)
                    TO (BEAR)
                    OBJ (LOC ACTOR (HONEY)
                             OBJ (PLACE NAME (TREE))))
Starting at #{REF.285 &MFRAME}

     Moving to #{REF.288 &ACT}
     Moving to #{REF.377 &MTRANS}
     Moving to #{REF.458 &MTRANS-LOCATION}
```

MTRANS-LOCATION is the memory structure for telling someone about the location of something. This memory structure indexes an expectation for the recipient of the information to attempt to get the object.

```
Placed at #{REF.458 &MTRANS-LOCATION}

#{REF.1082 &FOX-MTRANS} placed, indexing
   off #{REF.458 &MTRANS-LOCATION} with
(MTRANS ACTOR (FOX)
        TO (BEAR)
        OBJ (LOC ACTOR (HONEY)
                 OBJ (PLACE NAME (TREE))))

Running demons

Attempting to explain #{REF.1082 &FOX-MTRANS} in terms of
#{REF.458 &MTRANS-LOCATION} and one of *CLASSIFIER-WM*
nodes.

Trying to explain #{REF.1082 &FOX-MTRANS} in terms of
*CLASSIFIER-WM*

Explaining #{REF.1082 &FOX-MTRANS} in terms of
(#{REF.1082 &FOX-MTRANS} #{REF.1047 &FOX-ARRIVAL}
 #{REF.1078 &BEAR-HAS-FISH} #{REF.1073 &BEAR-LOC})

Explaining #{REF.1082 &FOX-MTRANS} in terms of
(#{REF.458 &MTRANS-LOCATION} #{REF.378 &PTRANS}
 #{REF.432 &POSSESION} #{REF.803 &PLACE-LOCATION})

Explaining #{REF.1082 &FOX-MTRANS} in terms of
(#{REF.377 &MTRANS} #{REF.288 &ACT} #{REF.433 &LOCATION})

Explaining #{REF.1082 &FOX-MTRANS} in terms of
(#{REF.289 &STATE})
```

Here we see the expectation of movement spawned. This is an example of how expectations compile into one step, a multiple step GOAL/PLAN sequence. The justification for the inference is still contained in default links for **D-CONT** and **WALK-PLAN**.

```
    Spawning demon: PREDICT-MOVEMENT.2264
    (PREDICT-MOVEMENT #{WM-NODE.2263
                        #{REF.1082 &FOX-MTRANS}})
#{REF.1082 &FOX-MTRANS} not explained. Adding another node
   to *CLASSIFIER-WM*
#{WM-NODE.2263 #{REF.1082 &FOX-MTRANS}}
```

After this node has been added to the working memory it looks like this,

```
&BEAR-FISHING
      !
      r
      !
```

```
&BEAR-LOC        &BEAR-HAS-FISH    &FOX-ARRIVAL  &FOX-MTRANS
    |                  |                |            |
wm-node.2234      wm-node.2223     wm-node.2249  wm-node.2263
```

The BEAR leaves for the TREE.

```
Indexing #{REF.1086 &BEAR-DEPARTURE}

Classifying (PTRANS ACTOR (BEAR)
                    FROM (PLACE NAME (STREAM))
                    TO (PLACE NAME (TREE))
                    OBJ (BEAR))
Starting at #{REF.285 &MFRAME}

     Moving to #{REF.288 &ACT}
     Moving to #{REF.378 &PTRANS}
     Moving to #{REF.531 &LEAVING}
Placed at #{REF.531 &LEAVING}

#{REF.1086 &BEAR-DEPARTURE} placed, indexing
    off #{REF.531 &LEAVING} with
(PTRANS ACTOR (BEAR)
          FROM (PLACE NAME (STREAM))
          TO (PLACE NAME (TREE))
          OBJ (BEAR))

Running demons
```

The PREDICT-MOVEMENT demon fires. This is an inference which would have taken several step in a PAM type model.

```
     Executing demon: PREDICT-MOVEMENT.2264
Linking #{REF.1082 &FOX-MTRANS} and
          #{REF.1086 &BEAR-DEPARTURE}
    with RESULT mode FORWARD
```

An expectation has taken care of explaining this concept. Now the working memory is this,

```
                  &BEAR-FISHING                    &BEAR-DEPARTURE
                        |                                 |
                        r                                 r
                        |                                 |
&BEAR-LOC        &BEAR-HAS-FISH    &FOX-ARRIVAL    &FOX-MTRANS
    |                  |                |                |
wm-node.2234      wm-node.2223     wm-node.2249    wm-node.2263
```

The demon DEPARTURE is used to maintain consistency. It asserts in the data base that the BEAR is no longer at the STREAM.

```
    Spawning demon: DEPARTURE.2303
    (DEPARTURE #{WM-NODE.2263 #{REF.1086 &BEAR-DEPARTURE}})

    Killing demon: PREDICT-MOVEMENT.2264
```

The FOX grabs the fish.

```
Indexing #{REF.1091 &FOX-THEFT}

Classifying (GRASP ACTOR (FOX)
                   OBJ (FISH))
Starting at #{REF.285 &MFRAME}

    Moving to #{REF.288 &ACT}
    Moving to #{REF.376 &GRAB-ACT}
Placed at #{REF.376 &GRAB-ACT}

#{REF.1091 &FOX-THEFT} placed, indexing
   off #{REF.376 &GRAB-ACT} with
(GRASP ACTOR (FOX)
       OBJ (FISH))

Running demons


    Executing demon: DEPARTURE.2303
```

MFRAME-OBJECT.2306 is the MFRAME containing the conceptualization for the BEAR no longer being at the STREAM.

```
Linking #{REF.1086 &BEAR-DEPARTURE} and
        #{MFRAME-OBJECT.2306 (MFRAME ...)}
   with RESULT mode FORWARD


    Killing demon: DEPARTURE.2303
Attempting to explain #{REF.1091 &FOX-THEFT} in terms of
#{REF.376 &GRAB-ACT} and one of *CLASSIFIER-WM* nodes.

Trying to explain #{REF.1091 &FOX-THEFT} in terms
   of *CLASSIFIER-WM*

Explaining #{REF.1091 &FOX-THEFT} in terms of
(#{REF.1091 &FOX-THEFT} #{MFRAME-OBJECT.2306 (MFRAME ...)}
 #{REF.1047 &FOX-ARRIVAL} #{REF.1078 &BEAR-HAS-FISH}
 #{REF.1073 &BEAR-LOC})

Explaining #{REF.1091 &FOX-THEFT} in terms of
```

```
(#{REF.376 &GRAB-ACT} #{REF.433 &LOCATION} #{REF.378 &PTRANS}
 #{REF.432 &POSSESION} #{REF.803 &PLACE-LOCATION})

Attempting to explain #{REF.1091 &FOX-THEFT} in terms
   of #{REF.273 &TAKE-RULE}
Matched against #{REF.1078 &BEAR-HAS-FISH}
Trying constraint #{CONSTR.279 NON-EQUALITY}

Connecting a mop to the current input
   #{REF.1078 &BEAR-HAS-FISH}-->
   #{WM-NODE.2234 #{REF.1078 &BEAR-HAS-FISH}}
```

The FOXs theft of the FISH is linked in as a DISABLEMENT of the BEARs possesion of the FISH.

```
Linking #{REF.1091 &FOX-THEFT} and
        #{REF.1078 &BEAR-HAS-FISH}
   with DISABLE mode FORWARD
```

After this MFRAME is linked to the &BEAR-HAS-FISH, the working memory looks like this,

```
        BEAR-FISHING          &FOX-THEFT             BEAR-DEPARTURE
                    \        /                             |
                     r     d                               r
                     |     |                               |
&BEAR-LOC          &BEAR-HAS-FISH     &FOX-ARRIVAL    &FOX-MTRANS
   |                     |                 |               |
wm-node.2234       wm-node.2223      wm-node.2249    wm-node.2263
```

TAKE-RULE has the demon FIND-P-CONT-FAIL indexed under it.

```
     Spawning demon: FIND-P-CONT-FAIL.2324
     (FIND-P-CONT-FAIL
           #{WM-NODE.2234 #{REF.1078 &BEAR-HAS-FISH}}
           ((CHEESE) (HONEY)
            (APPLE) (GRAPES) (FISH))
           ((FOX) (CROW) (BEAR)))

========================== FIND-P-CONT-FAIL.2324 ============
TEST:     "Look at the object of the current concept."
          "See if it is one of the things in OBJECT-LIST."
          "Check to see if it got away from an actor in"
          "ACTOR-LIST."

ACT:      "Set up a (P-CONT manner (FAIL)) and assert it"
          "into memory."
```

```
=================================================================

placing new input,
   #{REF.1091 &FOX-THEFT}-->
   #{WM-NODE.2234 #{REF.1078 &BEAR-HAS-FISH}}


    Spawning demon: D-CONT.2334
    (D-CONT #{WM-NODE.2234 #{REF.1091 &FOX-THEFT}})

======================== D-CONT.2334 ===================
ACT:     "Assume that since a character grabbed"
         "something that he wants it."
==============================================================
```

There is a last pass in the program to spawn and test any demons which are attached to the last conceptualization found.

```
**Searching for TAUs which resolve explanation splinters**


    Executing demon: D-CONT.2334
Linking #{REF.1091 &FOX-THEFT} and
        #{MFRAME-OBJECT.2337 (MFRAME ...)}
    with INTEND mode BACKWARD


    Killing demon: D-CONT.2334

    Executing demon: FIND-P-CONT-FAIL.2324
Linking #{MFRAME-OBJECT.2306 (MFRAME ...)} and
        #{MFRAME-OBJECT.2357 (MFRAME ...)}
   with THWART mode FORWARD
```

P-CONT-fail has two invocations of TAU-RECOGNITION indexed under it, one for **TAU-TOO-RISKY** and one for **TAU-ULTERIOR.**

```
    Spawning demon: TAU-RECOGNITION.2371
    (TAU-RECOGNITION
        #{WM-NODE.2234 #{MFRAME-OBJECT.2357 (MFRAME ...)}}
        #{REF.789 &TAU-TOO-RISKY})

======================== TAU-RECOGNITION.2371 ===========
SPAWN:   "Search BINDING-SPEC for a form which matches"
         "the mop on which this TAU was spawned. Run the"
         "constraints of the TAU to get additional"
        "binding."
TEST:   "Look at incoming concepts to see if they satisfy"
        "anymore constraints."

ACT:    "Index the TAU in the MFRAME on which it was"
        "spawned."
==============================================================


    Spawning demon: TAU-RECOGNITION.2372
    (TAU-RECOGNITION
        #{WM-NODE.2234 #{MFRAME-OBJECT.2357 (MFRAME ...)}}
        #{REF.790 &TAU-ULTERIOR})

======================== TAU-RECOGNITION.2372 ============
SPAWN:  "Search BINDING-SPEC for a form which matches"
        "the mop on which this TAU was spawned. Run the"
        "constraints of the TAU to get additional"
        "binding."
TEST:   "Look at incoming concepts to see if they satisfy"
        "anymore constraints."

ACT:    "Index the TAU in the MFRAME on which it was"
        "spawned."
==============================================================
```

This is the first invocation of TAU-RECOGNITION.

```
TAU recognition spawned on
   #{MFRAME-OBJECT.2357 (MFRAME ...)}
   for #{REF.789 &TAU-TOO-RISKY}
```

First it matches the conceptualization from which it was spawned to the one under which is was indexed.

```
Matched #{VAR.1147 ?P-CONT} against
        #{MFRAME-OBJECT.2357 (MFRAME ...)}
```

Once it has found all the concepts contained in the spec before the indexing concept, it can try some of the constraints.

```
Attempting constraints
Trying constraint #{CONSTR.1163 THWARTING}
Trying constraint #{CONSTR.1164 RESULTING}
```

One of the INTENTION links specified in the descripion of TAU-ULTERIOR is not there.

```
Trying constraint #{CONSTR.1165 INTENTION}
failed
```

Here are the bindings which TAU-RECOGNITION process has found so far.

```
Bindings in effect at spawn time,
((#{VAR.1147 ?P-CONT} . (GOAL MANNER (FAIL)
                              OBJ (POSS-BY ACTOR (BEAR)
                                           OBJ (FISH))))
 (#{VAR.1149 ?Y} . (FISH))
 (#{VAR.1148 ?X} . (BEAR)))
```

Here is the second invocation of TAU-RECOGNITION.

```
TAU recognition spawned on
    #{MFRAME-OBJECT.2357 (MFRAME ...)}
   for #{REF.790 &TAU-ULTERIOR}

Matched #{VAR.1168 ?P-CONT} against
        #{MFRAME-OBJECT.2357 (MFRAME ...)}

Attempting constraints
```

Since the TAU demons did not get spawned until after the story was completely processed, all the constraints for &TAU-ULTERIOR are satisfied at this time.

```
Trying constraint #{CONSTR.1183 THWARTING}
Trying constraint #{CONSTR.1184 RESULTING}
Trying constraint #{CONSTR.1185 RESULTING}
Trying constraint #{CONSTR.1186 NO-CONNECTION}

Bindings in effect at spawn time,
((#{VAR.1175 ?DO2} . #{Head PTRANS})
 (#{VAR.1172 ?Z} . (FOX))
 (#{VAR.1171 ?GRASP} . (GRASP ACTOR (FOX)
                              OBJ (FISH)))
 (#{VAR.1173 ?MTRANS} .
    (MTRANS ACTOR (FOX)
            TO (BEAR)
            OBJ (LOC
                  ACTOR (HONEY)
                  OBJ (PLACE NAME (TREE)))))
 (#{VAR.1174 ?ACT2} . (PTRANS ACTOR (BEAR)
                              FROM (PLACE NAME (STREAM))
                              TO (PLACE NAME (TREE))
                              OBJ (BEAR)))
(#{VAR.1176 ?ACT1} . (LOC MODE (NEG)
                          ACTOR (BEAR)
                          OBJ (PLACE NAME (STREAM))))
(#{VAR.1168 ?P-CONT} . (GOAL MANNER (FAIL)
                             OBJ (POSS-BY ACTOR (BEAR)
                                          OBJ (FISH))))
(#{VAR.1170 ?Y} . (FISH))
(#{VAR.1169 ?X} . (BEAR)))


   Killing demon: FIND-P-CONT-FAIL.2324
```

Here CRAM recognizes that &TAU-ULTERIOR is contained in this story.

```
   Executing demon: TAU-RECOGNITION.2372
TAU #{REF.790 &TAU-ULTERIOR} recognized.
Instantiating with,
(TAU ISA &TAU-ULTERIOR
  BINDINGS
    ((#{VAR.1175 ?DO2} . #{Head PTRANS})
     (#{VAR.1172 ?Z} . (FOX))
     (#{VAR.1171 ?GRASP} . (GRASP ACTOR (FOX)
                                  OBJ (FISH)))
     (#{VAR.1173 ?MTRANS} .
        (MTRANS ACTOR (FOX)
                TO (BEAR)
                OBJ (LOC ACTOR (HONEY)
                         OBJ (PLACE NAME (TREE)))))
     (#{VAR.1174 ?ACT2} .
          (PTRANS ACTOR (BEAR)
                  FROM (PLACE NAME (STREAM))
                  TO (PLACE NAME (TREE))
                  OBJ (BEAR)))
     (#{VAR.1176 ?ACT1} .
          (LOC MODE (NEG)
               ACTOR (BEAR)
               OBJ (PLACE NAME (STREAM))))
     (#{VAR.1168 ?P-CONT} .
        (GOAL MANNER (FAIL)
              OBJ (POSS-BY ACTOR (BEAR)
                           OBJ (FISH))))
     (#{VAR.1170 ?Y} . (FISH))
     (#{VAR.1169 ?X} . (BEAR))
     (#{VAR.1169 ?X})))


   Killing demon: TAU-RECOGNITION.2372
```

CHAPTER 7
Conclusions and Future Work

7.1 Conclusions

Abstract knowledge about planning is hard to capture. TAUs are used to capture abstract knowledge about bad planning, the same knowledge that people express using adages. Knowledge about bad planning can be used to critque plans, and it can also be turned around in counter-planning to trick other planners into making mistakes. What has been shown in this thesis is that we can have a specialization hierarchy of planning errors, and we can chunk smaller planning errors together to make larger ones. The relationships that arise among TAUs as a result of these operations are used to cross-index episodes with multiple planning errors for better recall.

We have also shown that TAUs can serve as a very abstract indexing scheme for stories in episodic memory. TAUs can index analogically similar stories because they represent information at the **GOAL/PLAN** level and not at the **ACT/STATE** or context feature level .

We have also presented an implementaion of TAUs which uses a **'binding-spec'** and **'constraints'**. The **'binding-spec'** is an ordered list of concepts. The **'constraints'** are relations among the concepts. This works well because the order information aids in the recognition process. Once a TAU has been activated, incoming concepts only have to be compared against one concept, the next undetected concept in the **'binding-spec'**. This has ramfications for TAU indexing. If the overhead for TAU recognition is low many TAUs can be indexed at single memory node.

7.2 Future Work

This thesis has left a number of avenues open for further exploration.

representing what is common sense
interaction of TAUs and parsing
TAU learning and the organization of thematic memory
analogical reasoning driven by TAU recognition

* Context feautures are observable surface features of a situation such as "the tall man" or "in the restaurant".

7.2.1 Common Sense - Poor planning often involves not using "common sense". Recognizing planning failures requires knowing which specific knowledge is common sense. Two constraints were used to capture part of this meta-knowledge in this work. They both had to do with detecting whether or not a particular causal link was "obvious". The constraint names used for these tests were:

> **not-obvious-result** and
> **obvious-result.**

The '**not-obvious-result**' constraint tests to see if CRAM has currently made any causal connection between two concepts. The '**obvious-result**' constraint checks to see if the causal link between two concepts was easy to derive, i.e. took a small number of rule applications.

In both the above cases the factors for determining whether or not some knowledge is common sense are based on the internal state of CRAM, not on an unchanging standard of what is obvious. By assuming that what CRAM has represented is "common sense", we get around the problem of representing "common sense" itself. It is very likely that people do the same type of introspective computation when determining whether or not something is obvious. However, people also have specific knowlege which they know is not common sense. An example of this type of knowledge is technical knowledge.

7.2.2 Parsing - The current implementation of CRAM does not include a parser. In order to be accepted as a valid cognitive model, CRAM needs to accept natural input rather than input distilled from English by a human translator. The issue here is one of representation validity. Since the representation for TAUs is declarative, it depends on all the "right" links being present between concepts, and on concepts being expressed in the "right" representation. The only way to show that a representation is not ad hoc is to be able to "parse into" that representation from natural language.

There does not seem to be any interaction between TAUs and language understanding at the word level. Once a TAU is spawned, it can set up expectations that may affect word disambiguation, but it is not likely that people "parse into" TAUs except for the case of parsing adages themselves.

7.2.3 Representation - The representation presented in this thesis is very sparse; it only represents those features which are required by the pattern matcher for recognizing TAUs. It is not likely that CRAM's current representation will be adequate when natural language input is used. A more robust representation system would be Wilensky's PEARL system [WILE83], where primitives such as **PTRANS** and **MTRANS** have their own hierarchy. The hierarchy of primitives is respected by the pattern matcher, which allows more specific concepts to match more general ones, but not the other way around. This is a more robust representation format than the simple slot/filler representation used by CRAM.

7.2.4 Learning and Thematic Memory - Generating a new TAU through specialization is fairly simple. A **GOAL** failure is found that instantiates two TAUs, one of which contains the other; a specialization is formed by adding the constraints of the contained TAU to the containing TAU. However, there is a problem: Once a specialization has been created, how does a program check that it has never been generated before?

Acquiring new TAUs through combination is much more complex. In **Chapter 2** we saw that we could combine **TAU-ULTERIOR** and **TAU-VANITY** to get **TAU-SUCKERED**. In general, given a goal failure which instantiates two different TAUs, we can chunk the two TAUs together to form a new composite TAU, using the elements of story to establish a correspondence between the elements of the two TAUs.

It would be helpful in testing theories of TAUs to have a large database of stories that contain TAUs indexed into a semantic memory. It is difficult to test a particular process, such as TAU learning, without having a large number of examples. The indexing schemes presented in this thesis, both for MFRAMEs and TAUs, are primitive. Part of the reason they work is the limited size of the domain. To test a better, more psychologically valid memory organization, a larger number of stories needs to be handled.

7.2.5 Analogical Reasoning - In **Section 1.1** it was pointed out that recognizing a particular TAU gives an analogical mapping from one story to any other story that instantiates that TAU. How can analogical stories aid in understanding the current story? One simple way is that the analog may have contained TAUs which were not found in the current story. With this new information, those TAUs can be activated in the current context.

A question which requires much more investigation is "What expectations can be generated from an analogical story?". It has been shown that once a TAU is activated, expectations are generated from the **'binding-spec'**. Are there other, more detailed expectations which can be extracted from an analogical situation?

BIBLIOGRAPHY

[BEWI73]	Bewick, T., Illustrator, *Treasury of Aesop's Fables*, Avenel Books, New York, 1973.
[CARB83]	Carbonell, J. G., "Learning by Analogy: Formulating and Generalizing Plans from Past Experience", in *Machine Learning: An Artificial Intelligence Approach*, R. Michalski, J. Carbonell, and T. Mitchell (Eds), Tioga Publishing Company, Palo Alto, Calif., 1983.
[CHAR80]	Charniak, Eugene, Riesbeck, C. K., McDermott, D. V., *Artificial Intelligence Programming*, Lawrence Erlbaum Associates, Hilsdale, New Jersey, 1980.
[CULL78]	Cullingford, R. E., *Script Application: Computer Understanding of Newspaper Stories*, Technical Report 116, Yale University, Department of Computer Science, 1978, Ph.D. Disseration.
[DEJO79]	Dejong, G. F., *Skimming Stories in Real Time: An Experiment in Intergrated Understanding*. Technical Report 158, Yale University. Department of Computer Science, 1979. Ph.D. Dissertation.
[DOLA83]	Dolan, C. P., "Reasoning Analogically", Term Project Winter 1983.
[DOLA84]	Dolan, C. P., *Representations for Conceptual Objects: T-CD², Discrimination Net, and Demon Control Structures*, Tools Note 1, UCLA Artificial Intelligence Laboratory, 1984.
[DYER81]	Dyer, M. G., "The Role of TAUs in processing Narratives", *Proceedings of the Third Anual Conference of Cognitive Science Society*, Berkely, Calif., 1981.
[DYER83]	Dyer, M. G., *In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension*, The MIT Press, Cambridge, Mass., 1983.

[DYER83a] Dyer, M. G., "Understanding Stories through Morals and Remindings", in *Proceedings of the Eight International Joint Conference on Artificial Intelligence*, 1983.

[GENT83] Gentner, D., "Structure-Mapping: A Theoretical Framework for Analogy", *Cognitive Science*, Vol 7, 1983.

[LEBO80] Lebowitz, M., *Generalization and Memory in an Integrated Understanding System*, Technical Report 186, Yale University, Department of Computer Science, Ph.D. Dissertation, 1980.

[LEHN82] Lehnert, W. G., "Plot Units: A Narrative Summarization Strategy", in *Strategies for Natural Language Processing*, W. Lehnert and M. Ringle (Eds), Lawrence Elbaum Associates, Hillsdale, New Jersey, 1982.

[LEHN83] Lehnert, W. G., Dyer, M. G., Johnson, P., and Yang, C., "BORIS - An In-Depth Understander of Narratives", *Artificial Intelligence*, Vol 20, No 1, 1983.

[KOLO80] Kolodner, J. L., *Retrieval and Organization Strategies in Conceptual Memory: A Computer Model*, Technical Report 187, Yale University, Department of Computer Science, Ph.D. Disseration, 1980.

[MEEH79] Meehan, J., *The Metanovel: Writing Stories by Computer*, Technical Report 74, Yale University, Computer Science Department, Ph.D. Dissertation, 1979.

[MOOR84] Moore, J. D., *Implementing McDypar in T*, Tools Note 2, UCLA Artificial Intelligence Laboratory, 1984.

[RIEG79] Reiger, C., Small, S., "Word Expert Parsing", in *The Proceedings of the Seventh International Joint Conference on Artificial Intelligence*, 1979.

[RIES75] Riesbeck, C. K., "Conceptual Analysis", in Schank (Ed) *Conceptual Information Processing*, American Elsevier, New York, 1975.

[RUME75] Rumelhart, D. E., "Notes on a Schema for Stories", in *Language, Thought, and Culture: Advances in the Study of Cognition*, D. Bobrow and A. Collins (Eds), Academic Press, New York, 1975.

[SCHA72] Schank, R. C., "Conceptual Dependency: A Theory of Natural Langauge Understanding", *Cognitive Psychology*, Vol 3, No 4, 1972.

[SCHA77] Schank, R. C. and Abelson, R. P., *Scripts, Plans, Goals, and Understanding: An Inquiry into Human Knowledge Structures*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1977.

[SCHA81] Schank, R. C. and Reisbeck, C. K., *Inside Computer Understanding: Five Programs plus Miniatures*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981.

[SCHA82a] Schank, R. C., *Dymanic Memory: A Theory of Reminding and Learning in Computers and People*, Cambrdige University Press, Cambridge, 1982.

[SCHA82b] Schank, R. C., *READING AND UNDERSTANDING: Teaching from the Perspective of Artificial Intelligence*, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1982.

[STEIN79] Stein, N. L. and Glenn, C. G., "An Analysis of Story Comprehension in Elementary School Children", in *New Directions in Discourse Processing*, R. O. Freedle (Ed), Ablex Publishing Corporation, New Jersey, 1979.

[WILE78] Wilensky, R. *Understanding Goal-Based Stories*, Technical Report 140, Yale University, Department of Computer Science, 1978, Ph.D. Dissertation.

[WILE81] Wilensky, R. "PAM" in *Inside Computer Understanding*, R. Schank and C. Riesbeck (Eds), Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1981.

[WILE83] Wilensky, R., *Planning and Understanding: A Computational Approach to Human Reasoning*, Adison-Wesley Publishing Company, Reading, Mass., 1983.

[WINS79] Winston, P. H, "Learning by Creating and Justifying Transfer Frames", in P. Winston (Ed) *Artificial Intelligence: An MIT Perspective*, MIT Press, Cambridge, Mass., 1979.