A Unified Theory of Inference for Text Understanding

By

Peter Norvig

B.S. (Brown University) 1978

DISSERTATION

Submitted in partial satisfaction of the requirements for the degree of

DOCTOR OF PHILOSOPHY

in

COMPUTER SCIENCE

in the

GRADUATE DIVISION

OF THE

UNIVERSITY OF CALIFORNIA, BERKELEY

Approved: [illegible] [illegible]

Chairman Date

[illegible] 11/25/86

[illegible] 11/25/86

....

A Unified Theory of Inference for Text Understanding

Peter Norvig

A Unified Theory of Inference for Text Understanding

Peter Norvig

Abstract

Natural languages, such as English, are difficult to understand not only because of the variety of forms that can be expressed, but also because of what is not explicitly expressed. The problem of deciding what was implied by a text, of ''reading between the lines'' is the problem of *inference*. For a reader to extract the proper set of inferences from a text (the set that was intended by the text's author) requires a great deal of general knowledge on the part of the reader, as well as a capability to reason with this knowledge. When the ''reader'' is a computer program, it becomes very difficult to represent this knowledge so that it will be accessible when needed.

Past approaches to the problem of inference have often concentrated on a particular type of knowledge structure (such as a script) and postulated an algorithm tuned to process just that type of structure. The problem with this approach is that it is difficult to modify the algorithm when it comes time to add a new type of knowledge structure.

An alternative, unified approach is proposed. This approach is formalized in a computer program named FAUSTUS. The algorithm recognizes six very general classes of inference, classes that are not dependent on individual knowledge structures. Rather, the classes describe general kinds of connections between concepts. New kinds of knowledge can be added without modifying the algorithm. Thus, the complexity has been shifted from the algorithm to the knowledge base. To accommodate this, a powerful knowledge representation language named KODIAK is employed.

The resulting system is capable of drawing proper inferences (and avoiding improper ones) from a variety of texts, in some cases duplicating the efforts of other systems, and in other cases improving on them. In each case, the same unified algorithm is used, without tuning the program specifically for the text at hand.

Acknowledgements

I would like to thank first the members of my thesis committee: Charles Fillmore, Lotfi Zadeh, and especially my advisor, Robert Wilensky, who suggested the topic for this work, and provided guidance throughout.

Robert Wilensky also deserves thanks for providing a stimulating environment to do AI research. The following members of the Berkeley AI Research group contributed to many of the ideas in this thesis, and were interesting people to have around: Rick Alterman, Yigal Arens, Michael Braverman, Margaret Butler, David Chin, Charley Cox, Joe Faletti, Paul Jacobs, Marc Luria, Tony Maida, Jim Martin, Jim Mayfield, Lisa Rau, and Nigel Ward. Berkeley's strength extends beyond the CS Department, and I have benefited from an exposure to various members of the Cognitive Science community, particularly Charles Fillmore, Paul Kay, and George Lakoff.

I would like to thank my parents, Gerda Norvig and the late Torsten Norvig, for kindling my interest in language and science, and for giving me the opportunities to do these things. I can't thank them enough, but at least I can say ''Hi Mom.'' I am also pleased to be able to acknowledge my siblings, Marc and Laura. Besides my family, the people who have been the most important to me in recent years are Gary and Rachel Atkinson, Katy Hayes Dauhoa, Tom Hahn, Melanie Killen, Krys and Steve Upstill, and Gay White. Thanks, guys.

Finally, I would like to thank the taxpayers of the United States for providing me with financial support, which was administered by the National Science Foundation under grants IST-8007045 and IST-8208602, the Office of Naval Research under contract N00014-80-C-0732, the Defense Advanced Research Projects Agency (DOD), ARPA Order #4871, monitored by Space & Naval Weapons Systems Command under contract #N00039-84-C-0089. I was also supported by grant 1-443952-58941 from the Alfred P. Sloan Foundation, and by the Computer Science Department of the University of Southern California. I am grateful for the computer and secretarial support I have received, particularly from Sharon Tague.

Table Of Contents

Chapter 1: Introduction

Making Proper Inferences from Texts

This thesis addresses the problem of understanding written English texts. The reader of a text is faced with a formidable task: recognizing the individual words of the text, deciding how they are structured into sentences, determining the explicit meaning of each sentence, and also making *inferences* about the likely implicit meaning of each sentence, and the implicit connections between sentences. This study is primarily concerned with the problem of inferencing, and touches on lexical and syntactic issues only in that they interact with the problem of inferencing.

An *inference* is defined to be any assertion which the reader comes to believe to be true as a result of reading the text, but which was not previously believed by the reader, and was not stated explicitly in the text. Note that inferences need not follow logically or necessarily from the text; the reader will often jump to conclusions that seem likely but are not 100% certain. The terms *logical inference* and *plausible inference* will be used to differentiate between inferences that are certain and non-certain.

People are very good at interpreting texts and making inferences. They generally do not notice when the text is under-specified and they have to make inferences to resolve ambiguities, or to gain a fuller understanding of the text. As an example, consider the following text, excerpted from a book of fairy tales [9]. It will be referred to as text (1).

> In a poor fishing village built on an island not far from the coast of China, a young boy named Chang Lee lived with his widowed mother. Every day, little Chang bravely set off with his net, hoping to catch a few fish from the sea, which they could sell and have a little money to buy bread.

A reader of text (1) should be able to make inferences like these:

(2a) There is a sea which is used by the villagers for fishing, surrounds the island, and forms the coast of China.
(2b) Chang intends to trap fish in his net, which is a fishing net.
(2c) The word *which* in *which they could sell* refers to the fish.
(2d) The word *they* in *they could sell* refers to Chang and his mother.

There are four important properties of the inferences listed in (2): they are *non-explicit, plausible, relevant,* and *easy*. To elaborate:

- The inferences are not explicitly stated in the text.

- They are *plausible*; not only is it possible for a reader to believe (2a-d) after reading (1), but it seems likely that any reader would.

- They are *relevant*, in that they serve to tie together concepts mentioned in the text.

- They are *easy* inferences; they seem to be made without conscious effort.

Inferences that meet these four criteria will be called *proper inferences*. To understand a text, a reader must make the proper inferences, and avoid making improper inferences. A representative set of *improper* inferences for (1) is listed below:

(3a) The villagers fish on a river in the middle of the island.
The island is on a lake which is near the coast.
(3b) Chang will use the net, which is a butterfly net,
as a deposit on a motor boat to go out fishing.
(3c) The word *which* in *which they could sell* refers to the sea.
(3d) The word *they* in *they could sell* refers to the fish.
(3e) The square root of 169 is 13.
(3f) Chang has a grandmother (who is perhaps deceased).
(3g) Chang lived with his mother.
(3h) Chang is wearing blue pants.

Most readers find it difficult to take seriously the inferences in (3a-d). They often have to go back to the text to see that (3a-d) are indeed possible at all; that they are not explicitly contradicted by the text. In fact, each of (3a-d) is perfectly consistent with everything stated in the text, they are just less plausible than the corresponding inferences in (2a-d). There are also other problems with some of them: in (3b) the motor boat seems irrelevant, and (3a) is a convoluted example that would fail on both the relevance and easiness criteria.

While (3e) is highly plausible, in fact is 100% certain, it is completely irrelevant. Although (3f) refers to a character in the story, it still is not a relevant inference because it does not tie together concepts from the text; it just adds peripheral information. We could go on from (3f) and infer that Chang's grandmother had a grandfather, and that he had a pancreas, and that his pancreas secreted insulin, and so on. In each case we have a highly plausible inference which is connected in some way to either the text itself or a previous inference. However, it is not enough for a fact to be inducible, or even deducible, from the text. The intuition is that we don't think of (3e,f) after reading the text. This intuition is formalized by the relevance criterion, which says that (3e,f) are not proper inferences because the connection between them and the text fails to add anything to the interpretation of the story.

Example (3g) is not considered an inference because it was stated explicitly in the text.

Text (1) was taken from a book where it was accompanied by some pictures. The illustrator presumably made inference (3h), because that is how Chang is depicted. I will speak of (3h) as an *idiosyncratic inference*. People bring many such inferences to the interpretation of a text, but are still able to distinguish idiosyncratic inferences from proper ones. In order to draw a picture, it was necessary to choose some attire for Chang, but another choice, say, brown shorts, would have done as well. On the other hand, the picture in question also showed Chang with a fishing net; here a butterfly net could not have been substituted. The reader is aware that the author of the text intended for him to infer that the net is a fishing net, but did not have any intention one way or the other with regards to the color of Chang's clothes. Herb Clark [30] makes a similar distinction, speaking of authorized and unauthorized inferences.

There is an implicit contract between the author and reader wherein the author agrees to make explicit enough of the story so that the reader, if he searches for the set of proper inferences, will be able to recover the information the author wanted him to recover about the story, and make sense of the text. In a perfectly-structured text there will always be easy, plausible connections between each pair of adjacent sentences. When these connections are missing, or when inferences prove to be incorrect, it is usually a signal that the writer is being humorous, ironic, mysterious, has a different view of the world, or is just being confusing. Indeed, much of what makes texts interesting is the intentional flaunting of this implicit contract.

Note that plausibility is a relative term. In the phrase *which they could sell* from (1), it is more plausible that *which* refers to the fish than to the sea, and that *they* refers to Chang and his mother rather than the fish. When faced with a choice of possible referents, it is possible to decide which is better using default assumptions like *the actors of selling events should be people*, and *the object of selling events should be products which are conventionally bought and sold.* When faced with no good referent, sometimes these assumptions have to be violated. Given sentence (4) below, the reader would be forced to infer that *which* refers to the sea and *they* refers to the fish, even though this constitutes a very unusual event in the real world. In the domain of fairy tales, it is more common to have animals acting in a manner similar to human beings, but this would still be an unusual event. The point is that the sentence is acceptable, despite its unlikeliness, and moreover, it is easier to determine the better of two proposed interpretations of the sentence than it is to decide if any single interpretation is acceptable. In other words, there appears to be no threshold of acceptability.

(4) The fish hoped to acquire a sea which they could sell.

It should be clear by this point that making proper inferences requires a great deal of knowledge. The reader must know the meanings of individual words, as well as the grammatical rules of the language. More importantly, the reader must have specific world knowledge about the subject matter. For text (1), this would include knowledge of spatial relations *(in, on, near)*; geography *(village, island, coast, sea, China)*; familial relations *(boy, mother, widow)*; commercial transactions *(buy, sell, have, money)*, as well as other sources of knowledge. Collectively this will be called *common sense knowledge*, to distinguish it from expert and grammatical knowledge. Without this knowledge, we would be unable to decide among alternative interpretations of the text. For example, inference (2b) above, *Chang intends to trap fish in his net*, comes from our knowledge of nets, not from the structure of the sentence. If the sentence had been (5) instead, we would not make the inference that Chang intends to trap fish in his dog.

(5) Chang set off with his dog, hoping to catch a few fish.

The FAUSTUS Approach to Inferencing

My approach to the problem of making inferences from texts has five main components. First, I formally define the notion of proper inference, in terms of the three criteria of ease, plausibility and relevance. I characterize the class of proper inferences, and compare this class to inferences described by other researchers. Second, I present an algorithm for making proper inferences and avoiding improper ones, given a text and a knowledge base. The algorithm is implemented in a program called FAUSTUS (Fact Activated Unified STory Understanding System). The algorithm employs a *marker-passing* mechanism that finds key concepts, a *collision classification* mechanism that determines a set of potential inferences, and an *evaluation* mechanism that determines which potential inferences should actually be made.

As we have just seen, a suitable knowledge base is a prerequisite to making proper inferences. Building the knowledge base comprises the third and fourth components: first defining a knowledge representation formalism, and then using the formalism to model facts about the world. The representation language is called KODIAK (Keystone to Overall Design for Integration and Application of Knowledge), and was developed jointly with Robert Wilensky [140] and others in the Berkeley AI Research group. The fifth and final component is an evaluation of the strengths and weaknesses of the basic approach, and of the implementations (FAUSTUS and KODIAK).

The name FAUSTUS serves three purposes: it obeys the time-honored AI convention of naming systems with inscrutable yet cute acronyms; it conveys the key idea of achieving understanding through knowledge (or facts); and finally it alludes to the difficulty, the hubris, of this goal of achieving all knowledge.

The system is diagramed schematically in Figure 1. First, input sentences are converted to KODIAK representations by a program called a conceptual analyzer. Arens and Wilensky's PHRAN program [133] was used where possible. For some input, PHRAN was not up to the task, so a representation was constructed by hand instead. A major flaw with this approach is the lack of interaction between the understanding component and the analyzer component, but that was not the topic of this thesis, so I have settled for a one-way connection. The understanding component, FAUSTUS, takes in representations and immediately stores them in the story memory. In addition, it makes inferences, based on what is known about the story so far as well as what is known in the general knowledge base. These inferences are also added to the story memory.

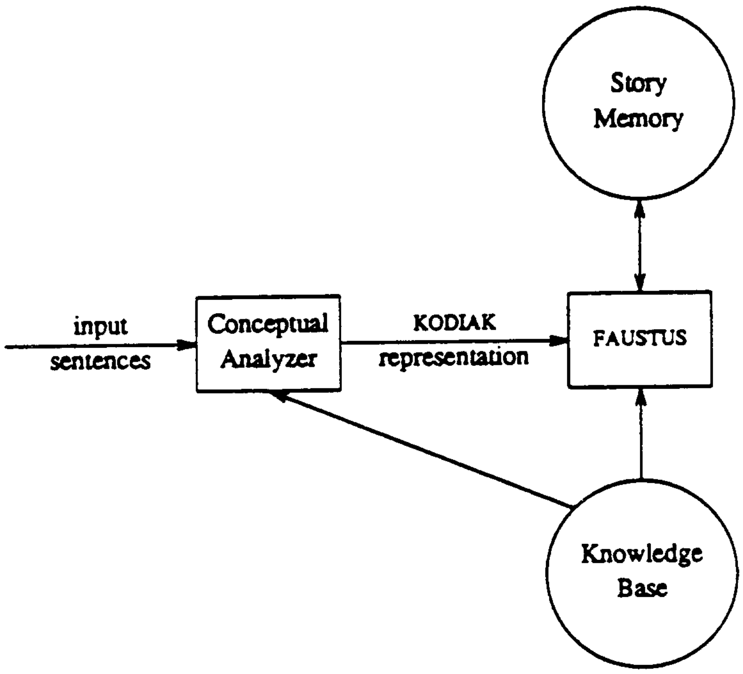

Figure 1: Overview of the FAUSTUS System

Comparison to Previous Approaches

Chapter 2 will cover related systems in detail, while in this section I will introduce just enough previous research to show where FAUSTUS fits in; what it is a reaction to, and what problems it attempts to solve. There have been several recent AI programs that each attack the problem of finding inferences by describing a new type of knowledge structure (script, plan, TAU, plot unit, etc.), showing its pervasiveness in actual texts and efficacy for understanding them, and then designing an inferencing algorithm based on the new type of knowledge structure. For example, Cullingford's [33] Script Applier Mechanism (SAM) made inferences by way of one main technique: identifying a *script* for the situation being described, and inferring the default actions in the script. As a variant of this idea, the FRUMP program, developed by DeJong [34], was based on the notion of *sketchy scripts:* scripts with less allowable variation and less detail than Cullingford's scripts. This allowed FRUMP to process a wider range of input texts, but to extract less information from the texts, and to accept less variation from the pre-stored scripts. The FRUMP program was tested with stories taken directly off the UPI newswire, and was able to correctly summarize over half the news stories that referred to scripts it had knowledge of.

The scriptal approach worked when a script could be identified, and when the story followed the script closely, but failed otherwise. Wilensky's [131] Plan Application Mechanism (PAM) could handle stories that deviated more from stereotypical situations. PAM's control structure had a main loop that reacted to each input sentence by classifying it as either a goal or an action. For every action, PAM would try to find a plan it could be a part of, such that that plan could be explained as arising from a known goal. This assured that many plan/goal connections were found, but it was difficult to find other types of connections. Alterman's NEXUS system [3] made event-concept coherence inferences, and had a similar architecture where it was searching for certain specific types of connections.

These programs were able to make some inferences beyond the scope of their main type of knowledge structure. For example, SAM, PAM and NEXUS each resolved definite noun phrase and pronominal references, but only as a side-effect of the matching process. This meant some references went unresolved; these programs had no instructions to "look harder" when no referent turned up as a result of the matching process. It also meant that it was difficult to integrate syntactic knowledge or other knowledge about constraints on the possible referents into the algorithm.

Each of these programs makes a trade-off between top-down and bottom-up processing. FRUMP was almost exclusively top-down; after deciding what script was relevant, it tried only to fit new input into the existing script structure. In contrast, PAM and NEXUS were more bottom-up in their approach. Both programs read one statement at a time, and tried to explain the new input by finding a connection to previous statements.

To put these programs in perspective, each was designed primarily as an experiment to discover more about the particular knowledge structure they were concentrating on. Each of the programs was successful from this point of view, but none of them could serve as a basis for an extensible system to which it would be easy to add new knowledge without redesigning the entire algorithm. Each program was tied too closely to the particular processing algorithm to allow such extensibility.

There have been attempts to allow flexible control structures. Charniak [23] proposed a demon-based system with arbitrarily complex procedures that could look either forward or backwards to find connections. Dyer [37] describes another demon-based system that makes use of multiple knowledge sources. Although both systems theoretically allow for the addition of new sources of knowledge, both still suffer from three problems:

- They often require what is intuitively the same piece of knowledge to be duplicated in several places in the system, in several forms. I term this the *knowledge-duplication problem.*

- They process different types of knowledge in very different ways. I will call this the *non-uniformity problem.*

- There is little provision for sharing between different knowledge sources, or for weighing different types of knowledge sources in coming to an interpretation. This is the *knowledge incompatibility problem.*

The work that is closest to my effort is presented by Charniak in [28]. He presents an understanding program that uses marker passing to parse, disambiguate and find inferences. The major difference is that I propose a specific set of path shapes and associated inference classes, while Charniak uses a more general resolution theorem-proving based approach. Another difference is that Charniak's knowledge base is four to ten times smaller (depending on how you count nodes and links).

Advantages of the FAUSTUS Approach

The FAUSTUS approach removes the complexity from the rules in the processing system, and places it in the knowledge network. There are several advantages to this approach.

• Declarative knowledge, when organized properly, can be used in several ways, while procedural knowledge by definition can only be used one way. If we are going to go to the trouble of building a large knowledge base, it would be desirable for the knowledge to be applicable to other tasks, such as language production and planning, rather than having all the knowledge being specific to language understanding.

• Another difficulty is dealing with multiple possible inferences at the same time. The FAUSTUS processing algorithm keeps a queue of potential inferences. Because the knowledge base and the possible inferences are in a declarative form, it is relatively easy to combine them, to consider several inferences at the same time (as when two or more possible inferences each suggest a referent for the same pronoun). If the knowledge needed to make inferences were represented procedurally, it would be more difficult to inspect, compare, and merge inferences together. If the procedures were going to have any interaction, they would have to be written as co-routines, and would have to know some of the details of other procedures. This is often confusing and difficult, and would probably require the knowledge base modeler to modify existing inference rules to interact with new rules as they are added.

• It was possible to define the notion of easy, plausible, relevant inferences, and to make some guarantees about the inferences made, guarantees that could not be made with previous systems.

• FAUSTUS employs a general inferencing algorithm that is not dependent on any particular knowledge structure and is relatively simple. Of course, text understanding is still a difficult task. The complexity has not disappeared; it has just moved from the algorithm to the knowledge base. Understanding goal-based stories still requires knowledge of plans and goals; understanding script-based stories still requires knowledge of scripts. The difference is that FAUSTUS is designed to allow incremental additions to the knowledge base, without having to redesign the entire processing algorithm. Thus, I draw on work done by other researchers in describing new knowledge structures, but I incorporate them in a declarative form. This is possible because of the representational power of the KODIAK formalism.

• While similar in many ways to Quillian's [93] spreading activation model, FAUSTUS has the advantage of being able to incorporate grammatical constraints into the marker passing process, rather than checking them only as an afterthought. This helps avoid spurious inferences. This capability is discussed in the first section of Chapter 5.

• FAUSTUS can handle texts based on "script-like" knowledge, as described in [109, 110]. It is not a single-minded script-application program, though, and is capable of reading the word "restaurant" without necessarily expecting the restaurant script to occur. It can also make selective connections between the events that are actually mentioned in the text without making all possible script-related inferences.

• FAUSTUS also handles what have been called plan-based inferences, coherence-based inferences, and several other types. Each of these is covered in Chapter 5. The important point is that, although these different inference classes have been proposed by previous researchers, FAUSTUS makes no distinction between the classes. Yet it has been possible to extend the program to handle these classes because of the generality of the inference types it does distinguish.

The FAUSTUS Algorithm

In this section I briefly present the inferencing algorithm as a six-step process. This presentation will be repeated in Chapter 4 in more detail.

Step 0: Construct a knowledge base defining general concepts like actions, locations, and physical objects, as well as specific concepts like fishing, islands, and nets. This is done once and the same knowledge is applied to all texts, whereas steps 1-5 apply to an individual text. The knowledge base is in the form of a semantic network, in the KODIAK formalism, as will be discussed in Chapter 3.

Step 1: Construct a semantic representation of the next piece of the text. This is done by the PHRAN conceptual analyzer or by hand. In some cases, the resulting representation is vague, and FAUSTUS resolves some particular kinds of ambiguities in the input using two non-marker-passing inference classes.

Step 2: Pass markers from each concept in the semantic representation of the input text to adjacent nodes, following along links in the semantic net. Markers start out with a given amount of *marker energy*, and are spread recursively through the network, spawning new markers with less energy, and stopping when the energy value hits zero. Each marker points back to the marker that spawned it, so we can always trace the marker *path* from a given marker back to the original concept that initiated marker passing.

Step 3: Suggest Inferences based on marker collisions. When two or more markers are passed to the same concept, a *marker collision* is said to have occurred. For each collision, look at the sequence of links along which markers were passed. Each link has a *primitive link type* associated with it, and the list of primitive link types determines the *shape* of the marker path that lead to the collision. We look at the two halves of the marker path involved in the collision, and if the total path shape matches one of five pre-specified shapes, then an inference is suggested. Suggested inferences are kept in a list called the *agenda*, rather than being evaluated immediately. (Primitive link types will be discussed below, and in more detail in Chapter 3.)

Step 4: Evaluate potential inferences on the agenda. The result can be either making the suggested inference, rejecting it, or deferring the decision by keeping the suggestion on the agenda. If there is explicit contradictory evidence, an inference can be rejected immediately. If there are more than one potential inferences competing with one another, as when there are several possible referents for a pronoun, then if none of them is more plausible than the others, the decision is deferred. If there is no reason to reject or defer, then the suggested inference is accepted, and new concepts are added to the model of the text.

Step 5: Repeat steps 1-4 for each piece of the text.

Step 6: At the end of the text there may be some suggested inferences remaining on the agenda. Evaluate them to see if they lead to any more inferences.

I now illustrate these steps with a specific example, taken from story (1) above. I will show how the program infers from the first sentence that there is a body of water which surrounds the island, and serves as the location for the village's fishing.

Step 0 says to construct a knowledge base. A portion of this is shown in Figure 2. The complete knowledge base is more than 60 times larger than this portion. The representation language is discussed in Chapter 3; for now, suffice it to say that concepts (depicted in boxes) are associated with each other by primitive representational links. The link 'S' means 'has as a slot,' 'D' means 'is dominated by (is a sub-category of),' and 'C' means 'is constrained to be a member of this category.' Thus, Figure 2 says the following: a coast has (at least) two things associated with it: a land-border which must be a land-mass, and a water-border which must be a body-of-water. An island is a kind of land mass, and it is surrounded by something which must be a body-of-water. Finally, an instance of fishing must have a location, which is a body-of-water.

Step 1 is to construct a representation of the input sentence. This consists of an interconnected network of individual concepts, which are marked as being instances of more general concepts in the knowledge base. This would include an instance of the action of fishing, and an instance of the concept island. To indicate that these instances stem from the input, they are given names consisting of the concept followed by the number of the input sentence in which they appeared. In this case, that would be fishing.1 and island.1.

After constructing the representation of the input, **Step 2** is to pass markers from

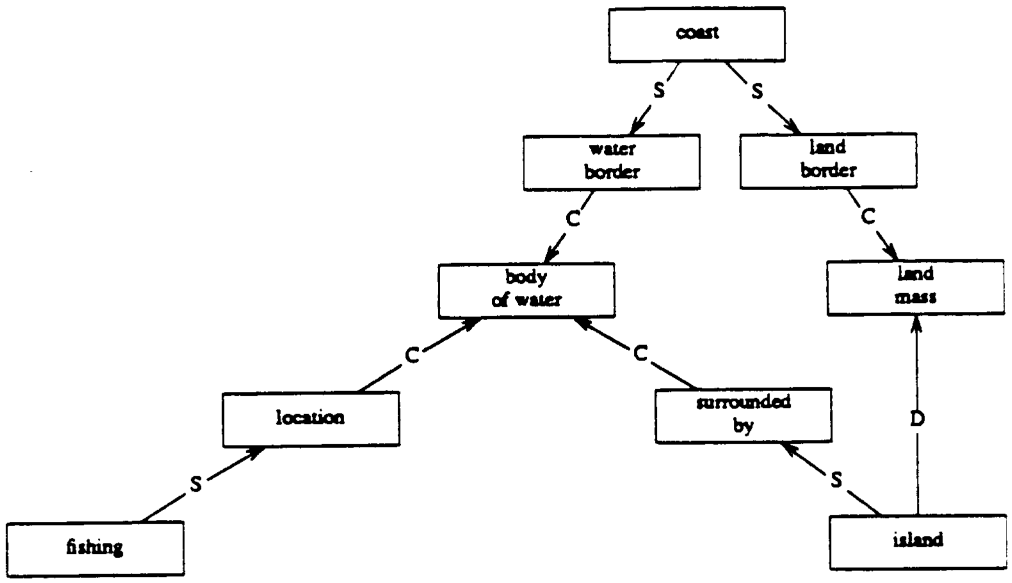

Figure 2: Part of the Knowledge Base Network

each concept in the input to neighboring concepts in the network, and recursively on to other concepts, following primitive links. An arbitrary number of markers can end up being passed from each concept in the input. In fact, since the passing is recursive, there would be an infinite number of markers if not for a set of rules that limit where and when markers can be passed. The exact rules for marker passing are covered in Chapter 4, and the implementation of the rules in Chapter 6.

Step 3 is to suggest inferences based on marker collisions. Figure 3 shows a collision at the concept body-of-water. Marker collisions denote concepts that are related to two different concepts in the text. Thus, they are relevant to the text, in some way. Just how relevant, and what inference they suggest, is determined by the shape of the marker path. In Figure 3, both halves of the marker collision have the shape →I→S→C→. It happens that such a path shape does have a suggested inference associated with it. The suggestion is to find or introduce a new concept and two new relations relating it to the two concepts in the input. This is shown in Figure 4, where the suggested new concepts are depicted in italic font.

If the suggested inferences were evaluated immediately, some would be accepted, only to have a better suggestion turn up later. Therefore, the suggestions are stored in a queue called the agenda, and are evaluated along with potential competing suggestions. This is **Step 4**. The suggestions that are eventually accepted are printed out by the program, and changes are made to the network to reflect the inference. The suggestion from

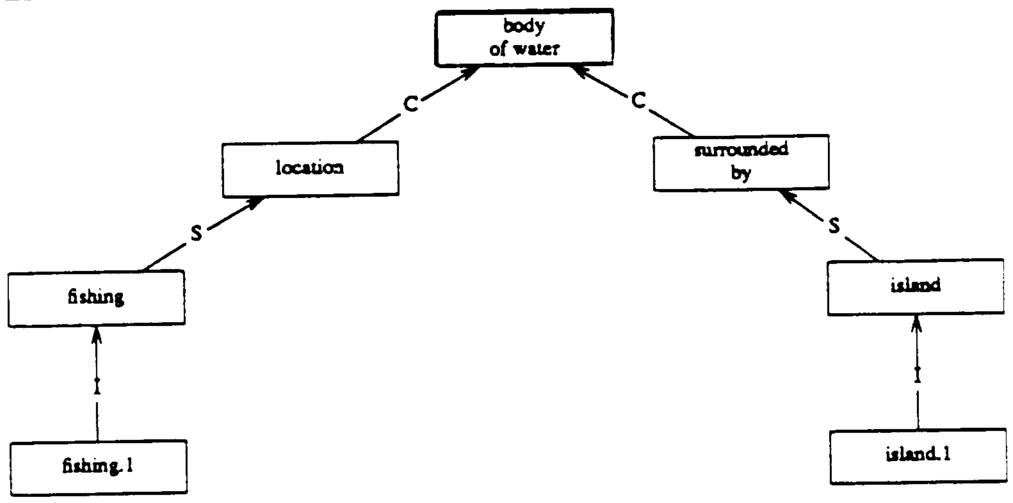

Figure 3: A Marker Collision

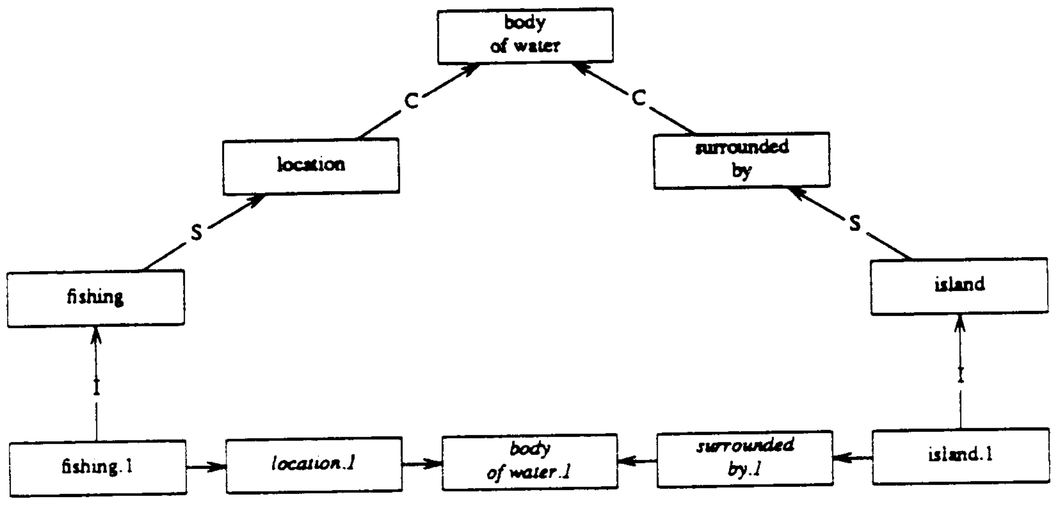

Figure 4: Inference Suggested by Marker Collision

Figure 4 is accepted, because there is nothing to contradict it, and no similar competing suggestion. The actual output from the program is as follows:

```
Inferring: there is a BODY-OF-WATER such that
   it is the LOCATION of the FISHING and
   it is the SURROUNDER of the ISLAND.
   This is a DOUBLE-ELABORATION inference.
```

In summary, the FAUSTUS algorithm breaks down into three main components:

• A set of rules for marker passing. These determine how far, and along which links, markers are spread.

• A list of six important path shapes, which determine the inference classes. Each has a suggested inference associated with it.

• A set of evaluation rules for deciding when to accept a suggested inference. This is necessary when there are multiple competing suggestions that would contradict one another.

The number of basic inference classes, six, is quite small compared to other systems. There are no inference rules that refer to domain concepts. That is, no rule is associated with a concept like 'person' or 'island.' Instead, the inference classes refer to general representational primitives like 'is an instance of' and 'participates in a relation.' These primitive associations are depicted by the links in Figures 2 and 3.

An Annotated Example

This example shows the inferences that are generated by FAUSTUS in the course of processing text (1). FAUSTUS does not receive the text directly as input, instead it is passed a semantic representation of each input sentence; these are shown below as capitalized expressions in parenthesis. The output from FAUSTUS is in `typewriter font`. The output is annotated with comments, in regular font, explaining what is going on. Chapter 5 will go over several examples like this one, showing in more detail how each inference is made.

```
[1] Input: In a poor fishing village built on an island
           near the coast of China,

Rep: (VILLAGE (MOD ← FISHING) (MOD ← POOR)
              (LOCATION ← A ISLAND) WHERE
              (BEING-AT (FIGURE ← ^ VILLAGE) (GROUND ← A ISLAND))
              (BEING-NEAR (FIGURE ← ^ VILLAGE)
                          (GROUND ← A COAST (OF ← CHINA)))

Inferring: a MOD of the VILLAGE is probably the PREDOMINANT-OCCUPATION
   because the FISHING fits it best.
   This is a RELATION-CONCRETION inference.
```

```
Inferring: the VILLAGE is a FISHING-VILLAGE.
   This is a CONCRETION inference.
```

The input says that the village is modified by the concept "fishing" in some unspecified manner. The program determines that fishing should be interpreted as the predominant occupation of the village. It is able to do this because of a collision between two marker paths that begin at "village" and end at "job." One path follows the links that say village.1 is a village, a village is a kind of polity, polities can have a predominant occupation, occupations are jobs of some kind, and fishing can be a job. The other half goes from village.1, which has a mod relation, which is filled by fishing.1, which is a kind of fishing, which can be a job. Associated with a path of this shape is the suggested inference that "mod" should be interpreted as "predominant-occupation." Once this assumption is made, the village can be further classified as a fishing-village. The knowledge base contains other facts about fishing-villages, such as the fact that they are usually near water. Both these inferences are called **concretion** inferences, because they take an abstract representation (like "mod") and interpret it as a more concrete one (like "predominant-occupation").

```
Inferring: a MOD of the VILLAGE is probably the AVERAGE-INCOME
   because the POOR fits it best.
   This is a RELATION-CONCRETION inference.

Rejecting: a MOD of the VILLAGE is probably the OVERALL-QUALITY
   because another possibility, AVERAGE-INCOME, is more specific.
```

Determining how "poor" modifies "village" is difficult not only because the modifying relation is vague, but also because "poor" is ambiguous between "low in wealth" and "low in overall quality." The knowledge base says that people have incomes, polities have average incomes, and objects in general can have an overall quality level. Markers from the instance of "poor" in the input are therefore propagated to the concepts for people, polities, and things. Markers propagating from "village" reach polity and thing, and thus there are marker collisions at those two concepts. Each collision suggests an inference, but when FAUSTUS tries to evaluate the first of these two, it notices there is another inference competing with it, in the sense that accepting one of the two means rejecting the other. The evaluation rule in this case says to accept the inference associated with the relation that has the most specific constrainer, if there is one. In this case, the constrainer of average-income is "polity," which is more specific than the constrainer of overall-quality, which is "thing." Thus, the average-income interpretation is accepted, and the overall-quality interpretation is rejected.

Notice that the possibility of interpreting "poor" as referring to income rather than average-income was never considered, because there was no person mentioned in the input, and thus no marker collision that would suggest that interpretation.

Another possible interpretation is that "poor" modifies "fishing" rather than "village." The whole phrase would then mean 'a village where the fishing was not good.' This interpretation can not be considered by FAUSTUS because the input it gets –

the output of the parser – has already specified the association between modifiers. The parser can return a representation that is ambiguous as to how something is modified, but it can not return a representation that is ambiguous as to what modifies what.

```
Inferring: the CHINA is viewed as a GEOGRAPHICAL-ENTITY.
   This is a VIEW-APPLICATION inference.

Inferring: a OF of the COAST is probably the LAND-BORDER
   because the CHINA fits it best.
   This is a RELATION-CONCRETION inference.
```

Here we see a **view application** inference. The knowledge base defines China as a country, which is a political entity. However, political entities can not have coasts; only geographical entities can. Part of the knowledge base is a general mapping, called a view, stating that political entities can be viewed as the geographical location they have jurisdiction over. So FAUSTUS infers that in this situation, China is being viewed as a geographical entity. After that is done, the ambiguous modifier "of" can be resolved: coasts have two components, a land-border and a water-border; China is known to be a landmass, and thus can only fill one of those roles. What it means to view one concept as another is covered in Chapter 3, while view application inferences are discussed in Chapter 4.

```
Inferring: there is a BODY-OF-WATER such that
   it is the LOCATION of the FISHING and
   it is the SURROUNDER of the ISLAND.
   This is a DOUBLE-ELABORATION inference.

Inferring: there is a BODY-OF-WATER such that
   it is the LOCATION of the FISHING and
   it is the WATER-BORDER of the COAST.
   This is a DOUBLE-ELABORATION inference.
```

Here we see the first inferences that create something new, rather than just further specifying some ambiguous input. The first of these is the inference discussed in the previous section and diagrammed in Figure 4. Although no body of water was explicitly mentioned in the text, concepts that implicitly refer to a body of water were mentioned. In particular, there are three marker paths, starting at the fishing, the island, and the coast, that all collide at the concept body-of-water. Each of these is of the $\rightarrow I \rightarrow S \rightarrow C \rightarrow$ path shape shown in Figure 4. The three paths considered in pairs result in three collisions, and each collision suggests an inference. These are called **double-elaboration** inferences because they elaborate on two concepts at the same time by relating them to a third. Two of the suggested inferences are accepted, and lead to the results printed above. The third suggestion was that the body-of-water is the surrounder of the island and the the water-border of the coast. This suggestion is now redundant because both of its components have already been adopted. Thus, FAUSTUS ignores it. Chapter 4 has a section covering elaboration inferences.

```
[2] Input: a young boy named Chang Lee lived with his widowed mother.

Rep:  (INHABITING  (EXPERIENCER ← A BOY  (MOD ←  YOUNG-AGE)
(NAMED ← CHANG))
                   (WITH ← A WIDOW MOTHER (OF ← ^ BOY))
                   (LOCATION ← ^ VILLAGE))

Inferring: the EXPERIENCER of the INHABITING must be the INHABITER
   This is a RELATION-CLASSIFICATION inference.
```

This is an example of a non-marker-passing inference. The input describes an inhabiting state with the experiencer being the boy. The definition of inhabiting in the knowledge base says that it is a kind of "being-at" state, with an inhabiter that plays the role of the experiencer of the state and the figure of the being-at. In other words, by definition any experiencer of an inhabiting must be an inhabiter. FAUSTUS recognizes this fact and prints the message. It is a non-marker-passing inference because it was detected automatically, by virtue of the definition of inhabiting, rather than by a search procedure involving markers.

```
Inferring: a WITH of the INHABITING is probably the CO-INHABITER
   because the MOTHER fits it best.
   This is a RELATION-CONCRETION inference.

Inferring: a OF of the MOTHER is probably the OFFSPRING
   because the BOY fits it best.
   This is a RELATION-CONCRETION inference.

Inferring: the BOY must be a SON, because it is a OFFSPRING
   This is a RELATION-CONSTRAINT inference.

Inferring: the INHABITING is a FAMILY-LIVING.
   This is a CONCRETION inference.
```

Here there are two more cases of resolving ambiguous modifiers via concretion inferences. A "with" can mark an accompanier, an instrument, or a manner, but in this case there is a very specific type of accompanier, the co-inhabiter, that is compatible with "with." The mechanism for discovering this is a collision at "inhabiting" between a marker path originating at the instance of inhabiting, and the path originating at the instance of with. The path goes through the concept accompanier, and the suggested inference is that the "with" is actually an instance of accompanier. Once it is established that the inhabiting situation holds with the mother and son as participants, then it can be inferred that the inhabiting is an instance of family-living, a more specific situation known in the knowledge base.

```
[3] Input: Every day, little Chang set off with his net,

Rep: (TRAVELING (ACTOR ← CHANG (MOD ← SMALL-SIZE))
                 (WITH ← A NET (OF ← ^ BOY)))

Inferring: the ACTOR of the TRAVELING must be the TRAVELER
   This is a RELATION-CLASSIFICATION inference.

Inferring: a WITH of the TRAVELING is probably the ACCOMPANIER
   because the NET fits it best.
   This is a RELATION-CONCRETION inference.
```

In this case, there is no specific information on how "with a net" could modify an instance of traveling, so the default, the "accompanier case," is selected. Note that the phrase "every day" is ignored completely in the semantic translation. FAUSTUS does not have a sophisticated model of time, and does not deal well with the notion of habitual action.

```
[4] Input: hoping to catch a few fish from the sea,

Rep: (WANTING (EXPERIENCER ← ^ BOY)
              (WANTED ← CATCHING (ACTOR ← ^ BOY)
                                 (PATIENT ← SOME FISH)
                                 (SOURCE ← THE SEA)))

Inferring: the EXPERIENCER of the WANTING must be the WANTER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the CATCHING must be a GOAL-SITUATION, because it is a WANTED
   This is a RELATION-CONSTRAINT inference.
```

Here we see both types of non-marker-passing inferences. First, the experiencer of a wanting state has a more specific name, wanter, which is reported by FAUSTUS. In addition, the catching is explicitly described in the input as being the wanted of a wanting. Things that fill the wanted slot are constrained to be goal-situations, that is, situations that have been considered but have not actually come to pass. FAUSTUS asserts that the catching must belong to this category.

```
Inferring: the CATCHING is a CATCHING-FISH.
   This is a CONCRETION inference.

Inferring: the SEA refers to the BODY-OF-WATER.
   This is a REFERENCE inference.

Inferring: the NET is a INSTRUMENT of the CATCHING-FISH.
   This is a SINGLE-ELABORATION inference.

Inferring: the NET must be a FISHING-NET,
   because it is a CATCHING-FISH$INSTRUMENT
   This is a RELATION-CONSTRAINT inference.
```

The first thing done here is to make the concretion inference that a catching action where the patient is some fish is actually an instance of catching-fish. This is detected because of a marker collision between one marker that starts at fish.4 and goes through fish to catching-fish and then up to catching, and another marker that starts at fish.4, goes to catching.4, and up to catching. The action catching-fish is more specific than catching, and includes other information besides the fact that fish are caught. For instance, it is known that fish are caught either in a net or on a line. Another connection is found by a marker collision at the concept trapping-device. One marker goes from net.3 up the hierarchy to net and to trapping-device. The other marker starts at catching.4, goes to catching, then to the slot catching$instrument (the instrument of a catching action) and on to that slot's constrainer, trapping-device. Note that the two markers did not originate at the same time; such an inference serves to tie sentences together. After it is asserted that the net is the instrument of the catching, a non-marker-passing inference notices that the net can only satisfy the constraint on instruments of catching-fish if it is interpreted as a fishing net.

```
[5] Input: which they could sell

Rep: (SELLING GOAL (ACTOR ← THEY) (PATIENT ← WHICH))

Inferring: the ACTOR of the SELLING must be the SELLER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the SELLING must be the THING-SOLD
   This is a RELATION-CLASSIFICATION inference.

Inferring: 'THEY' refers to the FAMILY.
   This is a REFERENCE inference.

Rejecting: 'THEY' refers to the FISH.
   because the FISH is not a SENTIENT-AGENT.
```

FAUSTUS represents the word "they" as a group of unspecified nature. So markers are passed from the the representation for they.5 up the hierarchy to the concept group. Other marker paths that collide at group originate at the representation for the fish stated in input [4], and the family inferred in input [2]. These later two paths collide with the first one at group, each suggesting a possible referent for "they." The reference is resolved because fish are not considered capable of performing a selling action. Note that if the program had not previously inferred the existence of the family (which was never mentioned explicitly), this inference could not be made.

```
Inferring: 'WHICH' refers to the FISH.
   This is a REFERENCE inference.

Rejecting: 'WHICH' refers to the SEA.
   because it could not be a THING-SOLD.
```

```
Rejecting: 'WHICH' refers to the BOY.
   because it could not be a THING-SOLD.

Rejecting: 'WHICH' refers to Chang.
   because it could not be a THING-SOLD.

Rejecting: 'WHICH' refers to the MOTHER.
   because it could not be a THING-SOLD.

Rejecting: 'WHICH' refers to the FAMILY.
   because it could not be a THING-SOLD.

Rejecting: 'WHICH' refers to the VILLAGE.
   because FISH is more recent.

Rejecting: 'WHICH' refers to the ISLAND.
   because FISH is more recent.

Rejecting: 'WHICH' refers to the COAST.
   because FISH is more recent.

Rejecting: 'WHICH' refers to the CHINA.
   because FISH is more recent.

Rejecting: 'WHICH' refers to the NET.
   because FISH is more recent.
```

The reference inference for the word "which" has many more possible referents. Ten different collisions each suggest a referent, and the evaluation algorithm must choose between them. Several possibilities can be rules out because they can't play the role of thing-sold, a role that the word "which" is explicitly filling. Of the remaining possibility, exactly one, the fish, was more recently mentioned than all the others. Thus, it is selected as the referent, and the others are rejected.

```
Inferring: there is a HAVING such that
   it is a RESULT of the CATCHING and
   it is a PRECONDITION of the SELLING
   This is a DOUBLE-ELABORATION inference.
```

Here we have the introduction via a double elaboration inference of a new "having" state, wherein the family has possession of the fish. This state was inferred because it mediates between two other actions: it is the result of catching the fish, and is a precondition for selling them.

```
[6] Input: and have a little money

Rep: (HAVING GOAL (EXPERIENCER ← ^ GROUP) (PATIENT ← A MONEY))
```

```
Inferring: the EXPERIENCER of the HAVING must be the HAVER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the HAVING must be the HAD
   This is a RELATION-CLASSIFICATION inference.

Rejecting: the HAVING mentioned in [6] is a PRECONDITION of the SELLING.
   because of a mis-match.

Inferring: the HAVING mentioned in [6] is a RESULT of the SELLING.
   This is a SINGLE-ELABORATION inference.

Inferring: the MONEY is the PRICE of the SELLING.
   This is a SINGLE-ELABORATION inference.
```

Here another instance of "having" is explicitly mentioned. FAUSTUS finds two single-elaboration connections between having and selling, but since the selling action above already has its precondition met, this one can only be the result.

```
[7] Input: to buy bread.

Rep: (BUYING GOAL (ACTOR ← ^ GROUP) (PATIENT ← BREAD))

Inferring: the ACTOR of the BUYING must be the BUYER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the BUYING must be the THING-BOUGHT
   This is a RELATION-CLASSIFICATION inference.

Inferring: the BUYING is a PRECONDITION of the HAVING mentioned in [6].
   This is a SINGLE-ELABORATION inference.

Inferring: the MONEY is the PRICE of the BUYING.
   This is a SINGLE-ELABORATION inference.
```

Note that single-elaboration paths have found that the money can fill the price role in both the buying and selling. A more realistic interpretation might be that the money goes into the family cache, and is used a little at a time to buy bread, but FAUSTUS assumes that exactly the same money that was received from selling the fish is then used to buy bread.

The rest of this thesis is laid out as follows. Previous research is covered in Chapter 2. Chapter 3 presents the knowledge representation language KODIAK, while the inferencing algorithm is described in Chapter 4, and the way the algorithm is used to duplicate inferences made by other systems is covered in Chapter 5. Finally, details about the implementation of the program are included in Chapter 6, and Chapter 7 gives some conclusions.

Chapter 2: Previous Research

Chapter 1 described the topic of this thesis: using common sense knowledge to make inferences from texts. This chapter will review some of the relevant past research on this topic. This will include research into both inferencing techniques and knowledge representation techniques, even when the two have been studied independently. Since I am defining the field so broadly and there has been much previous research done, I will not attempt to cover every relevant research effort. Rather, I will concentrate in detail on a few representative approaches.

This chapter will present related work and comment on some of the strengths and weaknesses of the work, but will not present solutions to the problems uncovered. That will come in subsequent chapters, with chapter 3 covering the knowledge representation formalism in detail and chapter 4 covering the inferencing algorithm. Chapter 5 shows how examples that were processed by other systems (including some that are presented in this chapter) can be handled by the FAUSTUS system.

Before presenting the previous research, I will attempt to classify the work I cover, as well as the work I ignore. First, I make a division between inferencing and representation techniques, while acknowledging that the two are often intertwined.

There has been a large body of work in many fields on the subject of inferencing, but I will concentrate on work related to the three criteria for common sense inferences: plausibility, ease, and relevance. The first two criteria separate my work from most expert system research; there they are dealing primarily with inferences that are difficult to make, and require long chains of reasoning. There are millenia of work in mathematical logic, but most of that fails to meet the relevance as well as the ease criteria; in almost all logics, proofs are acceptable regardless of their length, and it is always permissible to conjoin or disjoin any two terms to infer a third. There are some exceptions to this approach within logic, notably Anderson and Belnap's [5] work on relevance logic.

It turns out that most previous work relating to common sense inferencing in AI has been done under the guise of natural language processing. This does not imply that common sense inferences are language-specific. For example, when we read about a character going into a store, we may infer that the character wants to buy something. This inference stems from knowledge about stores and buying, not from any linguistic knowledge about the word *buying*. In fact, the knowledge that *people buy things in stores* can be used in many places. It can be used to understand motivations for going to the store. We can also turn the fact around; if we know someone wants a commodity, we can reason he may go to a store to get it. This reasoning can go on in reading a story, in viewing a silent movie with no words at all, in interpreting someone's actions in the real world, or in planning our own course of action.

I will call the type of knowledge that enables these inferences *common sense knowledge*. Knowledge of this type has been incorporated into natural language understanding systems, planning programs, some expert systems, and language generation programs. All will be examined in turn.

Previous Research in Common Sense Inferencing

The research covered in this section attempts to find plausible inferences from single sentences (or in some cases from pairs of sentences), but not from connected discourse. As we shall see, there are many approaches to this problem.

Procedural Inference Molecules

Rieger [95] recognized that understanding a situation can involve making many different types of inferences. Furthermore, when presented with a situation, it is not immediately clear what inferences will be important. In Rieger's model, the understander takes an input and represents it in conceptual space. From there the understander starts generating inferences, each inference spreading out from the input or from the previous inference in a "multi-dimensional inference space." When two contradictory inferences are generated, the system has to stop and resolve the contradiction. The process is seen by Rieger as an exercise in bi-directional search, using well-understood algorithms (see, for example, Nilsson's [86]).

Rieger makes a point of classifying the types of inferences that can be generated. As an example, consider the situation where person *P* obtains object *X* somehow. Inferences about this situation would be made by posing questions from the sixteen inference classes shown in Figure 1.

For every concept in the knowledge base, sixteen *inference molecules* must be defined, one for each inference class. Each molecule is a LISP procedure, written in the form of a discrimination net of tests. The overall control structure is to observe the input, and for each concept mentioned in the input, call the sixteen inference procedures. Each procedure increases or decreases activation of a particular inference, and when the activation reaches a threshold, the system does a "careful" test and assert.

Figure 2 below is part of the normative inference molecule for the concept own, adapted slightly from [95]. This procedure is intended to figure out just how likely it is that a person P owns an object X. There is a certain amount of appeal to this approach. If we are asked *Does John own a hammer?* and if we know John is a carpenter, then we can in fact be moderately confident in answering yes. However, there are also quite a few problems with this approach.

There are several problems associated with the use of a discrimination net. The discrimination net approach often poses counter-intuitive questions. When asked *Does*

Normative – Is it normal that P has a X?
Specification – From where would P obtain X?
Causative – Why does P obtain X?
Resultative – What does P obtaining X lead to?
Motivational – What results of P obtaining X were intended by P?
Enabling – What would have to be true for P to obtain X?
Missing Enablement – Why would P fail to obtain X?
Enablement Prediction – What could P be able to do now that he has X?
Action Prediction – What plans might P invoke to obtain X?
Function – Infer P will use X for its normal purpose, if it has one.
Intervention – How could C keep P from obtaining X?
Knowledge Propagation – If X knows P obtains X, what else will X know?
State Duration – How long is P likely to keep his X?
Feature – What does P obtaining X tell us about P or X?
Situation – In a particular circumstance, is it more likely for P to obtain X?
Utterance Intent – Why would the author tell me that P obtains X?

Figure 1. Rieger's Sixteen Inference Classes

```
is P a member of a pure communal society, or an infant?
  if so, very unlikely that P owns X otherwise, is X living?
    if so, is X a person?
      is P a slave owner, and could X be a slave?
      if so, likelihood is low but non-zero
      otherwise likelihood is zero
    otherwise, is X an animal or plant?
      if so, is X domestic in P's culture?
        if so, does P have a fear of X's, or an allergy to X's?
          if so likelihood is low
          otherwise, likelihood is moderate
   otherwise, does X have a normal function?
     if so, does P do actions like this function?
       if so likelihood is moderately high
   ...
```

Figure 2: Normative Inference Molecule for (OWNS P X)

John own a hammer? it does not seem that we immediately ask ourselves if John lives in a communal society. The net is organized in a manner that assumes we can answer the top-level questions before we get to the more specific questions near the leaves of the tree. In some cases, we may have specific information that is relevant, but which never gets used because we cannot answer a question that would lead to the node where the question is asked. In other words, discrimination nets are by definition oriented towards top-down processing, and therefore they sometimes ignore useful bottom-up information. Other problems with discrimination nets are discussed by Barsalou and Bower in [11], although most of their comments do not apply to the kind of discrimination net Rieger is using.

Another major problem is that information is not easily shared in this approach. While it is certainly true that if P is allergic to X he is less likely to own X, this is not only a fact about owning. P would also be less likely to be near X, to be holding X, or to like X if he were allergic to it. In Rieger's approach we would have to rewrite the same information four times under owning, holding, being near, and liking. In fact, we might have to rewrite it up to sixty-four times, since there are sixteen inference molecules for each concept, and often the same information must be re-expressed under different inference classes. For example, if asked *does John own six different hammers?* the normative inference molecule for own could infer that the probability is very low, unless John is a carpenter. However, we would also like to be able to infer, given that *John bought six hammers*, that he probably is a carpenter. This would be a causative inference. Thus, the single fact that carpenters own hammers must be represented in several different places. Even definitions, like the fact that buying something causes the buyer to own it, need to be represented multiple times (in this case, as a causative and resultative inference).

Another criticism is that the inference classes are ad-hoc, and serve no purpose other than to remind the knowledge engineer what types of things to consider. One problem is that each inference class hides a varying number of possible inferences. Some classes give rise to one type of inference, but others can be used in multiple ways. Take the 'feature' class, for instance. *John bought a Mercedes* implies something different about John than *John bought a Pinto*. In these examples, a feature of the object determines something about the agent, but the opposite is also possible. In *the art collector bought a painting* we attribute greater value to the painting than in *the child bought a painting*.

In addition, inference molecules are unwieldy to modify. Because the inference molecules are structured like programs rather than like data, adding a new piece of knowledge means editing an existing program, with all the possibilities of introducing bugs due to unforeseen interactions among components.

The final criticism of this approach is that it biases the researcher towards explaining individual sentences. The whole mechanism is geared towards explaining, say, *John bought a hammer*, rather than explaining a complex passage where buying is mentioned in several places. As researchers attempted to handle multi-sentential passages, several new problems arose. One of the most vexing was the reference resolution problem, to be described in the next section.

Rieger is a prime example of drawing distinctions that are not drawn in the FAUSTUS model. Other authors draw different sets of distinctions. For example, it is common to treat pronominal reference resolution or word sense disambiguation as a separate process. In FAUSTUS an attempt has been made to describe inference processes at a higher level: asserting that any two concepts are co-referential, asserting that a concept should be classified under a more specific category, and so on. These general inferencing processes can then be applied to a variety of situations, as long as the knowledge base provides the proper support. The upshot of this is that this chapter will be reporting on research as it was conceived by previous researchers, not necessarily along the lines that are advocated by FAUSTUS.

Reference Resolution Inferences

Reference resolution is the process of deciding what a pronoun or an ambiguous word or phrase in a text refers to. For example, in the phrase *he saw him* the pronouns *he* and *him* are ambiguous; they could be referring to any male. The syntax tells us they refer to different males (otherwise *himself* would have been used), but nothing more. The references can be resolved by consulting the *context* set up by the interpretation of previous sentences to find likely candidates. Almost any information about the likely candidates could be crucial in deciding the referent; reference resolution shares much with inferencing in general.

Charniak studies reference resolution extensively in his thesis, [23]. He tries to tie in reference resolution with other kinds of inference, claiming that *the information which is used to "fill in the blanks" can be directly used to help with reference*. By this he means that there is a certain amount of inference that must be done to understand each sentence and relate it to previous sentences, and that these inferences are also useful for reference resolution.

Charniak goes on to categorize the known types of information that can be used to resolve reference problems. First, there is descriptive information. The pronoun *he* must refer to a male, while *she* must refer to a female, and *the red ball*, clearly enough, must refer to a ball that is red. Charniak points out that other phrases, like *Jack's house* are less precise. Jack's house could be the house he lives in, or the one he owns and rents to someone else.

The concept of recency also plays a role in resolving references. Concepts that have been mentioned or alluded to recently can be referred to with elliptical constructions, while concepts that are not in context cannot be. Charniak estimates that perhaps 90% of pronouns could be resolved correctly by picking the most recent referent with matching gender and number, and then backing up if this choice leads to a contradiction. However, a quick computation, using Charniak's own thesis as the source material, gives a figure closer to 50% for that heuristic.

In addition, selectional restrictions can come into play. In *She landed the 747 safely*, the pronoun *she* might refer to Captain Smith or co-pilot Jones, but it could not normally refer to a female dog that happened to be on board, because piloting a 747 can only be performed by well-trained humans. While it is easy to specify restrictions that are normally appropriate, it is much rarer to find a restriction that is universally applicable. There are three reasons for this. First, in fiction, and especially fantasy, it is reasonable for animals to act like humans, and for other restrictions to be violated. By definition, fiction must differ from reality, and it is very difficult to say *a priori* in exactly what way it can be different, and what things cannot change. Second, we need a mechanism to express hypothetical or counterfactual statements. The passage "*It is false that a dog can pilot a 747*" is a perfectly reasonable and understandable sentence, yet it seems to violate a selectional restriction. Third, expressions that seem to violate selectional restrictions when interpreted literally often have a valid metaphorical interpretation. Wilkes [143] recognizes that selectional restrictions cannot be absolute, and formulates a system of 'preferences' rather than 'restrictions'.

Finally, according to Charniak, there may be other semantic considerations that determine reference resolution. Charniak presents the following example:

(1) Today was Jack's birthday. Penny and Janet went to the store. They were going to get presents. Janet decided to get a top. "Don't do that" said Penny. "Jack has a top. He will make you take *it* back."

Here the problem is to decide what the pronoun *it* refers to in the last line. Recency would suggest the top that Jack owns, while it is only detailed knowledge about owning, tops, wanting, and gift giving that can lead to the determination that *it* refers to the top that Janet is considering buying. The most important fact is that unwanted gifts are sometimes returned to the store at which they were bought. Charniak's algorithm calls for the program to invoke this fact and expect the possibility of the new top being returned to the store. The phrase *take it back* matches this possibility, with the result of the match being both that *it* is identified as the new top, and also that the destination is the store. Thus, the same process that resolves pronoun references also adds additional information to the construal of the story. As a conclusion, Charniak suggests that the best referent "might be the referent which allows the most links to what has already happened." This sentiment will be echoed by many other researchers; the problem is in determining exactly what the links are. (Charniak ends up rejecting this approach, mainly because he feels it is too computationally expensive. However, he never implemented the final version of his thesis model, so the computational complexity is unknown.)

Herb Clark [30] also discusses the problem of reference resolution. He presents example sentence pairs like the following, and discusses the reference relation between the first and second sentence in each pair.

(2a) I met a man yesterday. The man told me a story.
(2b) I met a man yesterday. He told me a story.
(2c) I met a man yesterday. The bastard stole all my money.
(2d) I met two people yesterday. The woman told me a story.
(2e) I walked into the room. The ceiling was high.

(2f) I walked into the room. The windows looked out to the bay.
(2g) I walked into the room. The chandelier sparkled brightly.
(2h) John was murdered yesterday. The murderer got away.
(2i) John died yesterday. The murderer got away.
(2j) John fell. What he wanted to do was scare Mary.
(2k) John fell. What he did was trip on a rock.
(2l) John fell. What he did was break his arm.
(2m) John is a Republican. Mary is slightly daft too.

Sentences (2a-d) are characterized as "direct reference" involving identity, pronominalization, epithets and set membership, respectively. For example, in (2b), the pronoun *he* in the second sentence refers back to the man mentioned in the first sentence. Some authors will speak of the word *he* referring to the phrase *a man*, but since we are more interested in the semantics of the referring relationship than in the syntax, we will adopt the more common terminology where phrases refer to objects, and where pairs of phrases can be co-referential. Sentences (2e-g) are examples of "indirect reference by association" with necessary, probable, and inducible parts, while (2h,i) are "indirect reference by characterization" involving necessary and optional roles. Finally, (2j) involves a reason, (2k) a cause, (2l) a consequence and (2m) a concurrence.

While these examples demonstrate the pervasiveness of the reference problem, the classification system does not shed any light on how to resolve a particular reference. Clark starts to address this question by positing what he calls the *given-new contract* [29] between the author and reader of a text, or the speaker and listener of an utterance. The idea of the given-new contract is that each sentence conveys some old information and some new. The speaker implicitly agrees to construct each sentence so that the listener can compute from memory the unique antecedent that was intended for the given information, and so that the new information is genuinely new.

Clark then suggests the following rule for resolving references: "Build the shortest possible bridge that is consistent with the Given-New Contract." In other words, the listener should make the inference that requires the least assumptions to connect the given and new information in a way that is consistent with the situation. For instance, another example of a concurrence relation shows up in the sentence pair: *Alex went to a party last night. He's going to get drunk again tonight.* The concurrence bridge is the inference that he got drunk at the party last night. However, Clark claims, we could have made the bridging inference that "every time he goes to a party he meets women, and all women speak in high voices, and high voices always remind him of his mother, and thinking about his mother always makes him angry, and whenever he gets angry, he gets drunk." Clearly this chain was not *intended* by the speaker, and thus the listener is not *authorized* to make this inference, in Clark's terminology. The listener is authorized to make only the simplest possible bridging inference.

Clark's theory is relatively complete, in that it addresses the three crucial points of a theory of inference. First, he has a classification of the possible types of inferences. Second, he has a criterion for generating inferences: make inferences that serve to connect given and new information. Finally, he has a metric for deciding among competing, contradictory inferences: the bridge with the smallest number of links is preferred.

This is not completely satisfactory, though. There is no way to decide which of two competing bridges of the same number of links is to be preferred, and there is no allowance for, say, a bridge of three straightforward links to be preferred over one with two highly unusual links. In other words, it seems desirable to have a metric for link strength, and Clark offers no such metric. Even if one could be developed for individual links, it is not clear how these measures should be combined; the strength of a chain may not be the purely additive sum of the strengths of its links. This is illustrated by trying to interpret *flicked the light switch* in the following text:

(3a) John got a book.
(3b) He sat on a couch.
(3c) He flicked the light switch.

One interpretation is that he flicked the switch *on*, which makes the room brighter, thereby illuminating the book, which enables John to attend to the writing in the book, which enables him to understand the contents. This is a bridge of length four which makes use of pre-existing knowledge of switches, lights, reading, and books. Another interpretation is that he flicked the switch *off*, which makes the room dark, which enables him to sleep on the couch. This is a bridge of length two, which uses pre-existing knowledge of switches, lights, sleeping, and couches. However, even though the second path is shorter, it seems less appealing.

Another problem with this approach is that it assumes given and new information can be distinguished by syntax and intonation alone. In some cases this will work; in (4a) the given information is that someone left and the new information Mel performed that action. In (4b) the opposite holds. However (4c) is ambiguous between the two interpretations. When spoken, (4c) might be disambiguated by intonation, but it could also be spoken with neutral intonation.

(4a) It was Mel who left.
(4b) What Mel did was to leave.
(4c) Mel left.

Lockman and Klappholz [71] consider a broad range of inference types, classifying them under the heading *contextual reference*. They include pronominal reference, identity reference, and in general most of the categories considered by Clark. However, they go beyond that, and include examples like (5a) below, where the `walking` in the second sentence is co-referential with the `going to school` in the first sentence.

(5a) Mary went to school early today. She walked the entire three miles.
(5b) Mary went to school early today. She walked up the stairs.
(5c) Mary went to school early today. She arrived at 7:00.

A distinction is made between references that hold between complete sentences, and those that hold between parts of sentences. There is also a reference from *she* to

`Mary` and from *the entire three miles* to the `distance-traveled` of the event in the first sentence, even though the distance is not explicitly mentioned in the first sentence. Lockman and Klappholz are especially interested in this type of reference, and they are not concerned with any inference that can be made solely on the basis of syntactic rules. There are two main types of connections between sentences in their theory: *expansion*, where a sentence provides more information about the preceding one, as in (5a), and *temporal continuation*, where the second sentence is an event that occurs after the first, as in (5b). Expansion comes in two types, expansion of the entire sentence, as in (5a), and expansion of a component of the sentence, as in (5c), where the second sentence refers to the `arriving` component of the going to school, and not to the entire event.

Lockman and Klappholz provide some notion of control structure. For each sentence, they prefer to find a connection to the preceding sentence, but allow the sentence to refer back to preceding sentences. They provide an ordering for searching backwards for an antecedent if the preceding sentences does not provide a satisfactory connection. Unfortunately, they have little to say about how one can decide if an antecedent is satisfactory. Similarly, they provide no way to tell the difference between an expansion and a continuation. Their theory says (5a) and (5b) should be classified differently, but they do not say how to make that classification, and there do not seem to be any syntactic clues to distinguish them.

As we go from Charniak in 1972, to Clark in 1975, to Lockman and Klappholz in 1980, there is an expansion of the term *reference*. At first it applies primarily to pronouns, then to arbitrary noun phrases, and finally to verb phrases and complete sentences. The term is also expanded in that originally the antecedent had to appear explicitly in the text, while the later authors allow implicit antecedents. It would be possible to continue in the vein of Lockman and Klappholz, and try to frame *all* semantic relations between components as references, but other authors have resisted that trend, and have come up with different terminology for semantic relations, as we shall see in the following sections.

Concept Coherence

Alterman [3] presents a story understanding program, NEXUS, that finds connections between events and states mentioned in the text. He defines seven concept coherence relations that form the connections. The relations are *subclass, subsequence, coordinate, antecedent, precedent, consequent,* and *sequel*. Rather than having an encyclopedic knowledge base which attempts to completely define concepts, Alterman calls his knowledge base a *dictionary*. In this dictionary he defines concepts in terms of the seven coherence relations and in terms of default values for case arguments, but does not try to add any other type of information about the concepts. The idea is that this set of relations can be used to make an analysis of the text at an interesting level; other processes besides NEXUS could be used to make a more complete analysis of the text, and to find other classes of inferences.

The control structure is to take each input state or event in turn, and do breadth-first search through the dictionary of concepts, looking for a connection to an existing event or state. When a connection is found, it is checked for consistency, and as soon as a consistent connection is found, the search is stopped and an inference is made. There are rules that cut down the size of the search space by pruning paths that cannot lead to a valid inference, and there is a mechanism that does pronominal reference resolution as a side-effect of the matching process.

In other words, the theory of inference embodied in NEXUS is that events and states are connected by the seven coherence relations, and processing a story means finding the shortest path along these relations that will connect each line in the story to some other line in the story. Each line is expected to have exactly one connection to the rest of the story, no more, no less. Although implemented as a bi-directional breadth-first search procedure, NEXUS could also be seen as a spreading activation or marker-passing approach, which looks for the shortest path between nodes.

Alterman resolves the question of deciding if an event is an expansion of a previously mentioned event with a rule that states that before doing the search for a connection, each new event is first checked to see if it could be a expansion of the previous event. If there is no explicit contradiction, it is assumed that the new event is in fact an expansion.

Hobbs [55, 56] and Mann [75] independently proposed classifications of the types of coherence relations that can hold between sentences. For instance, one sentence can provide an example for the previous sentence, or it can be an elaboration or a generalization. Although both researchers have interesting classification schemes, neither of them is very useful for generating inferences. The systems seem better suited to explaining the options that an author has in constructing a text to make a point than to the task of understanding the connection between two sentences. For example, one of Hobbs' classes is *parallel structure*. An example of a sentence with this structure is *set stack A empty and set link variable P to T*. It is important for the author to know that there is a parallel sentence structure, which uses the word *and*, and which can be used to join two similar ideas. However, for the reader, recognizing the parallel structure does not add any interesting inferences that he could not have made if the two clauses were presented independently.

Case Relation Based Inferencing

Case relations on verbs have had great importance in natural language systems, frame-based representation languages, and in the development of linguistics. This section covers some of the history of systems that have used cases, and discusses a few of the theoretical problems.

An early example of a natural language processing system that used case frames extensively is presented by Hendrix in [52]. It is representative of a simplistic approach to handling case relations, yet it treated them seriously, and derived much of its power from them. The system consists of a parser, a simple generator, a modeling system, and a lexicon. As an example, the lexical entry for *buy* indicates that it is a verb, that it refers to the `exchange` event (which they call a *canonical verb*), and that it has the following case frame:

```
BUY:
 (OK (HUMAN ORGANIZATION) BUYER)
 (OK (PHYSOBJ) THINGBT)
 (FROM (HUMAN ORGANIZATION) SELLER)
 (FOR (MONEY) THINGGIVEN)
 (AT (PLACE) LOC)
 (IN (PLACE) LOC)
 (OK (DAYPART) TIME)
 (IN (DAYPART) TIME)
```

Here the first element of each list is the preposition expected to mark a noun phrase in the input (or OK for an unmarked noun phrase), the third element is the deep case that that noun phrase should map into, and the second element is a disjunction of possible types; the noun phrase must be one of those types to be a valid filler.

Another approach to detecting inferences from single sentences using case relations was work done by Simmons [119-121]. While this work covered a number of points, the part relating to case relations was similar enough to Hendrix's that we will not discuss it further.

There were many problems with Hendrix's system. The only way it could distinguish the subject and dative cases, both marked by OK was by the ordering of entries in the case frame: BUYER comes before THINGBT (the thing bought). Thus, the system could not handle passives, where the order is reversed. There was a great deal of redundancy; the fact that either humans or organizations can be actors in certain types of actions is represented twice in this case frame alone, and will be mentioned many other times in other verbs. Also, the case relations specifying that an event can occur at a particular time and place need to be repeated many times over. One of the motivating ideas behind case systems is to capture generalizations, but Hendrix's approach misses important generalizations. This happens mainly because the level of description is too close to the surface verbs.

Charniak's Case-Slot Identity Theory [25] addresses most of these problems. In this theory, there is no need to translate between surface and deep cases (the deep cases are called *slots* as is the custom in many frame-based representation languages) because surface cases and slots are one and the same. Redundancy is eliminated because slots are inherited. For example, there is a concept called transitive-act which has an agent and a patient slot. Once transitive-act is defined, it is simpler to define more specific actions. For example:

```
[frame: transitive-act
```

```
   slots: agent patient ...]

[frame: reading
   isa: transitive-act
   slots: language ...]
```

Here `reading` inherits the `agent` and `patient` slots from `transitive-act` and adds a new slot, `language`. Charniak used this example for expository purposes; a more complete specification might have `transitive-act` declare only the `patient` slot, and inherit the `agent` slot from `act`, which in turn inherits the `location` and `time` slot from `event`, or some such. Similarly, as Charniak points out, the `language` slot should be inherited from the `linguistic-communication` frame.

The question remains: what is the status of the `language` slot? Charniak claims that since it is a slot, and cases and slots are identical, it must be a full-fledged case, just like agent or patient. However, Charniak does not explain how the `language` case is associated with the word *in*, as in *Fredrica read a book in French*. He could make this association with an explicit declaration, as in Hendrix' PRULES, but this would fail to capture the meaning behind the word *in*. Another solution would be to have the `linguistic-communication` frame inherit from something like the `transfer-along-conduit` frame (see [94]) which has a `medium-of-transfer` slot which is mapped to the word *in*. In general, Charniak represented slots well, but failed to provide a natural association between surface forms and underlying meaning. He also failed to address the question posed in the next section.

How Many Cases Are There?

Consider the following quote from Fillmore's influential article *The Case for Case* [42] :

> *The sentence in its basic structure consists of a verb and one or more noun phrases, each associated with the verb in a particular case relationship. The 'explanatory' use of this framework resides in the necessary claim that, although there can be compound instances of a single case (through noun phrase conjunction), each case relationship occurs only once in a simple sentence.*

Fillmore is making a very strong claim, that each case occurs only once in a simple sentence. This claim is worth investigating, for two reasons. First, if we adopt the claim, it can help us disambiguate case relations, because we will know that each new case relation must be distinct from the previous relations of each simple sentence. Secondly, by comparing acceptable and unacceptable sentences we can tell if two uses mark the same case or not. Fillmore tells us that two noun phrases denoting the same case *must* be conjoined. Conversely, Zwicky and Sadock [150] say that conjunction *cannot* occur with an ambiguous case marker.

Consider the sentences in (6) (some of these are from [54]). Sentences (6a-d) show that the preposition *with* can mark the instrumental case (in two ways), the accompanier case, or the manner case. Sentences (6e,f) together support the analysis that (6a,b) both mark the instrumental case, and hence cannot be used as two separate case relations, but must be conjoined into one noun phrase. Conversely, (6g,h) show that the instrument and manner cases are distinct. The curious example is in (6i,j). If there are distinct instrumental and accompanier cases, then (6i) should be acceptable, and (6j) should be rejected. However, most informants reject (6i), suggesting a problem with Fillmore's claim (unless (6i) is rejected for some reason unrelated to the single-case claim).

(6a) John painted the wall with latex paint.
(6b) John painted the wall with a roller.
(6c) John painted the wall with Mary.
(6d) John painted the wall with reckless abandon.
(6e) ?John painted the wall with latex paint with a roller.
(6f) John painted the wall with latex paint and a roller.
(6g) John painted the wall with latex paint with reckless abandon.
(6h) ?John painted the wall with latex paint and reckless abandon.
(6i) ?John painted the wall with latex paint with Mary.
(6j) ?John painted the wall with latex paint and Mary.

There is an alternative analysis of this situation. An important rhetorical rule (see [49]) can be stated simply as *do not be repetitive*. It is this rule that makes (7a) below unusual, while (7b) is much better. It may be that part of the reason why (6e,i) sound bad is the repetition of the word *with*. As evidence for that, compare (7c) with (6e). Simply substituting *using* for *with* makes the sentence much better, even though the same case relations seem to be represented. Fillmore would not have to accept this as evidence, as the *using* clause makes (7c) a complex sentence, and his claim mentions only simple sentences. In this analysis, (6h) is still bad because it conjoins two semantically distinct slots, but there is no explanation why (6g) seems better than (6e) or (6i).

Case-slot identity claims that there is an instrumental slot somewhere high in the hierarchy (perhaps on `action`), and that both `painting-implement` and `applied-paint` are specializations of the instrumental slot. Thus, it predicts that (6e) above should be acceptable, or at least it makes no claims as to why (6e) is unacceptable.

(7a) ?Ann got a book. Bob got a book. Cathy got a book.
(7b) Ann got a book. Bob got a book, too. So did Cathy.
(7c) John painted the wall with latex paint using a roller.

Another problem shows up when two different case relations necessarily have the same filler. For example, the definition of `selling` states that the actor and the donor are one and the same. The *do not be repetitive* rule blocks us from mentioning the overloaded case twice. Thus, (8a) is acceptable, but (8b) is not.

(8a) John sold the book to Mary.
(8b) ?John sold the book from John.

What Type of Cases Are There?

So far we have seen two approaches to handling case relations. Hendrix emphasizes *grammatical* relations like subject, object, and indirect object, and tries to fit semantics into that framework. Charniak, on the other hand, emphasizes *semantic* relations like agent, recipient, and donor, and fits the grammatical information on top of that. We have yet to see a completely satisfactory way to handle both at the same time. To make matters worse, there are other components that also come into play. Katz [60] distinguishes the grammatical, semantic, and *rhetorical* components, where the later include notions like given, new, topic, and comment. Fillmore adds a fourth way of looking at sentences, the *orientation*, where a concept can be either in perspective or out of perspective. Using an example common to Fillmore and Hendrix, the `commercial-event`, the speaker can put the buyer in perspective by choosing the verb *buy*, as in (9a), the seller by choosing the verb *sell*, as in (9b), or both the buyer and to an extent the money with the verb *pay*, as in (9c). Orientation is a matter of degree; in (9a) and (9b) we could elevate the money by mentioning it in a *for* clause, but the money would still not be as prominent as the subject.

(9a) Iraq bought some bombs from Peru.
(9b) Peru sold some bombs to Iraq.
(9c) Iraq paid $1M for some bombs.

The rhetorical component is similar to the idea of orientation. One example of a rhetorical case is the notion of *topic*. Often, the grammatical case *subject* is both the semantic *agent* and the rhetorical *topic*. Topic is important for problems such as pronoun disambiguation. In (10a) there are two possible referents for *he*, but most informants interpret the pronoun as referring to John, since John is the topic of the first sentence. This preference to choose the topic can be overruled, as in (10b), where *she* refers to Mary because it does not agree in gender with John, or in (10c), where *he* is interpreted by most informants as referring to Bill, because of the added information which fits better.

(10a) John is over there talking with Bill. He's an old friend of mine.
(10b) John is over there talking with Mary. She's an old friend of mine.
(10c) John is over there talking with Bill. He's a good listener.

Script-Based Story Understanding

The basic idea of a *schema* goes back to Bartlett [12]. It is also present in Norman and Rumelhart's work [87]. Minsky's idea of a frame [83] is similar to the schema idea. Although frames were originally conceived for work in computer vision, they have wide applicability, and in fact several natural language systems using frames were built at MIT by Winston [144, 145] and others [44, 118]. Charniak's Ms. Malaprop program [24] is constructed along similar lines. The notion of a *script*, or stereotypical sequence of actions with variable actors, objects, and locations, was proposed and developed by Schank and Abelson [110]. They use the example of the restaurant script to explain the concept. The idea is that unless you knew what typically happens in a restaurant, you would not be able to understand a story like (13). You would not know that John intended to eat some food, or who *they* referred to in *they didn't have any*, or that he probably got angry and left without eating or paying.

(13) John went to a restaurant. He ordered a hamburger. The waiter said they didn't have any. John asked for a hot dog. When the hot dog came, it was burnt. He left the restaurant.

The two major problems to contend with are *script recognition* and *script application*. Recognition is the process of deciding what scripts, if any, are applicable to the current situation. Application involves tracking the current situation in term of the applicable script(s), deciding how each new input relates to the script, and making default inferences for parts of the script that are not explicitly mentioned. For example, in (13) the recognition problem is to notice that the `eat-at-restaurant` script is appropriate. Once the `eat-at-restaurant` script is recognized, script application would lead to inferences such as identifying *the waiter* as "the waiter who is employed by the restaurant and who was assigned to provide service for John". Other inferences are that John probably sat down at a table and looked at a menu before ordering, and that he probably was dissatisfied with the restaurant. Cullingford's [33] SAM (Script Applier Mechanism) program was able to make inferences like these, although it did not seriously address the script recognition problem.

Although limited, the scriptal approach had four main points going for it:

- It emphasized the role of real world *knowledge* as the source of intelligent understanding.

- It had a well-defined control structure whereby inferences were made as they were needed to track progress through the script.

- It showed how *context* could influence the interpretation of subsequent input.

- It dealt with probable *inferences*, rather than strict logical deductions.

Scripts, and indeed, world knowledge in general, seem to have gotten more credit than they deserved. For example, in [38], Dyer states that scripts are useful in word sense disambiguation. He claims as evidence the fact that *ordered* and *to go* have different meanings in a restaurant than in the military, as seen in (14a,b). But these examples have nothing to do with scripts, *per se*. Both of them can be understood unambiguously, out of context, by virtue of the restrictions on semantic case frames of the various senses of the verbs. In other words, it is not the fact that the ordering is occurring in a restaurant or in a military context that gives it its meaning. Rather it is the object of the ordering (a pizza or a person) that determines the interpretation. This is shown in (14c,d).

(14a) John ordered a pizza to go.
(14b) The general ordered the private to go.
(14c) At the restaurant, the customer ordered the waiter to go.
(14d) Working late at the pentagon, the general ordered a pizza to go.

Dyer points out two other major problems of scripts. First, there was no sharing of structure. There was knowledge of tipping imbedded in the restaurant script, but it was unrelated to tipping in other situations, like taking taxis or getting a haircut. Similarly, knowledge about eating could not be shared between the restaurant script and the eat-at-home script.

The second problem was a lack of intentionality. The restaurant script said that the customer looks at the menu and then orders his meal, but it does not explain that the menu tells him what the restaurant has to offer, and thus helps him decide what to order. Dyer solves these two problems by re-representing scripts as MOPs. MOPs are Memory Organization Packets, as described by Schank in [114].

There is another problem of sharing memory that scripts did not cover: representing the commonality between the restaurant script from the customer's point of view *vs.* the waiter's point of view, or the cook's. Each has very different scripts, but there is also a great deal of information that is shared.

Another important frame-based story understander was Charniak's Ms. Malaprop system [24]. Charniak used frames to represent all knowledge in this system, so he avoided the uniformity problem of SAM, but otherwise the system suffered from many of the same problems, and was eventually abandoned.

The major limitation of script-based processing is that it only works for situations for which a known script exists. Whenever the story deviates from a stereotypical situation, which interesting stories must do, script application become impossible. This should not be seen as a limitation of the concept of a script, but rather on the process of script application. SAM had only this one process at its disposal, and therefore it should not be expected to do all of story understanding. Schank and Abelson pointed out this limitation and suggested that goals and plans could be used where scripts failed.

Goal/Plan Based Story Understanding

As a reaction against the limitation of script-based systems, Wilensky [131] designed the PAM (Plan Applier Mechanism) program to track the goals and plans of the characters in the story. This allowed PAM to process a broad class of stories that could not be handled by the script-based approach, and was also a means of controlling inference. The algorithm was to try to interpret each input as either a goal for one of the characters in the story, a plan for achieving a goal, or as an expected input based on previous processing. Each new goal or plan generated *expectations* for what might come next. These were stored in a discrimination net, and matched against all input. Each new input could be explained by matching an expectation, or by being a plan for some known goal, or by being a plan for something which was in turn explainable. Thus, the resulting representation of the input text included a set of *intentional explanations* for each action in the text. The following story is from [131] :

(15a) John was lost.
(15b) He pulled over to a farmer standing by the side of the road.
(15c) He asked him where he was.

PAM processes this story as follows. From (15a) it infers that John will have the goal of knowing where he is. From that it infers he is trying to go somewhere, and that going somewhere is often instrumental to doing something there. From (15b) PAM infers that John wanted to be near the farmer, because he wanted to use the farmer for some purpose. This rather vague inference constitutes the explanation of (15b). Finally (15c) is processed. It is recognized that asking is a plan for knowing, and since it is known that John has the goal of knowing where he is, there is a match, and (15c) is explained. As a side effect of the matching process, the three pronouns (*He, him* and *he*) in (15c) are disambiguated.

Wilensky's model has the following important characteristics:

• There was an underlying theory of human planning behaviour. Although not shown in this example, there was a great deal of detail on what happens when several goals interact.

• There was an assumption that planning behaviour is important in many stories. The important points in a story were taken to be the interesting goal interactions between the characters, or the resolution of a single character's quest to achieve his goal.

• There was a theory of inference. PAM was able to make a fairly long chain of inferences to find a connection between an action (e.g. *Willa picked up the Michelin Guide*) and a known goal (e.g. *Willa was hungry*). PAM would not attempt to infer any connection between, say, two states (e.g. *Willa was hungry.* and *Willa was angry.*), and would not waste time trying to make inferences that could not lead to an intentional explanation.

Granger [45] adopted Wilensky's intentional explanation approach, and added the capability to learn new inference rules from the text, and to recover from incorrect inferences. His system, ARTHUR, (A Reader THat Understands Reflectively) could process stories like the following:

```
INPUT STORY:
      MARY PICKED UP A MAGAZINE.
      SHE SWATTED A FLY.

INPUT QUESTION:
      WHY DID MARY PICK UP A MAGAZINE?
```

```
OUTPUT ANSWER:
        AT FIRST I THOUGHT IT WAS BECAUSE SHE WANTED TO READ IT,
        BUT ACTUALLY IT'S BECAUSE SHE WANTED TO GET RID OF A FLY.
```

PAM could understand some stories where a character changed his goal or plan, by keeping its expectation vague, or by entertaining several explanations simultaneously. However, it could not back up and retract an erroneous inference once it was committed to the inference. Granger's program added the ability to recover from such a mistake. Such a capability is becoming more standard in AI programs, due to works like Doyle's on truth maintenance system [35, 36].

Story Skimmers

All the programs mentioned so far processed only a small number of texts, and adding a new text to the repertoire usually meant a great deal of work on the part of the programmers of the system. DeJong's FRUMP (Fast Reading Understanding and Memory Program) system [34] was designed to be different. The program interpreted news stories taken directly off of the UPI news wire. The approach was to use *sketchy scripts*, which were knowledge structures similar to Cullingford's scripts, but without as much detail. Sketchy scripts include only the important likely events in a situation. The system has the same two subproblems that Cullingford's program had (script recognition and script application) but in DeJong's case it is not necessary to process all of the input. Once a script is selected, predictions are made for the likely events in that script, and the program ignores any input that does not fit the predictions.

Liebowitz's IPP (Integrated Partial Parser) operated in a manner similar to FRUMP, but it also had the ability to generalize and to "learn" new scripts.

Both programs owe at least part of their success to the fact that newspaper stories follow prescribed formats, and have the specific purpose of reporting the important facts relating to a story.

Integrated Story Understanding Systems

There have been attempts to put several of these capabilities together. The most notable in this class is Dyer's BORIS [38] system (Better Organized Reasoning and Inference System). It used a demon-based control structure and was able to find many different classes of inferences on the same text. BORIS incorporates and distinguishes seventeen classes of knowledge: object primitives, scripts, settings, goals, plans, affects, themes, interpersonal relationships, physical states, events, social acts, memory organization packets, thematic abstraction units, scenes, scenarios, reasoning, and beliefs.

These structures interact in certain predefined ways. For example, inferences can be made to connect events to goals (as in Wilensky's approach), but emotional affect can never be directly related to settings. Thus, if given the passage *"At the restaurant Bill punched John in the mouth. John got mad."* and the question *"How did John feel at the restaurant?"*, BORIS would not be able to directly recall John's affect. Instead, it would have to first infer the `eat-at-restaurant` scenario from the restaurant setting, then recall that the punching event occurred as an unexpected event in the restaurant scenario, then infer some goal of John's, and finally infer his affect.

Thus, Dyer has a theory of memory search which is dependent on the knowledge structures and their interconnections. He has adequate representational power to cover a wide range of input. He has some guidelines on how to generate possible inferences. However, the inference demons can be criticized as being *ad-hoc*. Here are some sample inference demons:

```
(16a) IF an ACT at the DESTINATION SETTING of a transition
         scenario is enabled by that SETTING
      THEN build an ENABLES link between the GOAL achieved
         and the CHANGE OF PROXIMITY goal in the transition scenario

(16b) IF the word just read is a pronoun and a HUMAN
         is found with matching GENDER and CASE
      THEN bind the concept to that HUMAN

(16c) IF the word just read is a name, then
         IF character exists with matching GENDER and FIRST-NAME
         THEN return that CHARACTER
         ELSE create a new CHARACTER

(16d) IF the word just read is 'glass' then
         IF it is followed by a LIQUID
         THEN glass is used as a MEASURE (e.g. a glass of coke)
         ELSE glass is used as a MATERIAL (e.g. a glass plate)

(16e) IF the story refers to a MEAL and
         a HUMAN is found modified by the preposition 'with'
      THEN that HUMAN is an EATER in the MEAL
```

There is a great range in the specificity of these rules. The first one is fairly general. It could be used to infer from *John went to Shea stadium* that he intended to watch a ball game, and that he went to the stadium in order to watch the game. The same rule could be applied to numerous other situations. The rule is automatically self-extensive in that every time the system learns about a new action that is enabled by a setting, the rule will automatically apply to the setting. However, the rule can only be used in one way; it cannot be used to infer from *John wanted to watch a ball game* that he should go to a ball park, or even to infer that John is watching the game from *John is at Shea stadium*.

Some rules are incomplete. For example, (16b) and (16c) do not state what to do if several matching humans are found. Other rules are just plain wrong. For example, (16c) cannot handle Aricia is a pretty name and (16d) would fail on glass wine glass. Of course, it is always possible to come up with new examples that a particular system cannot handle, and it should not be considered a fundamental flaw that BORIS could not handle these particular examples. A more serious problem is that many rules are written at a too specific level. For example, rather than having one rule saying that the preposition with can be used to designate an accompanier in an event, there would have to be many rules similar to (16e), approximately one for each verb. In addition to its lack of generality, (16e) is still wrong in that it would interpret *Bill was talking with the waiter* as meaning the waiter was an eater in the meal.

Dyer also addresses the problem of deciding what is an adequate explanation for an event (although not for a concept in general). There are two main traditional approaches: (1) settle for the first adequate explanation found, and (2) search for all possible explanations. A problem not addressed by Dyer is deciding when an explanation is adequate, but leaving that aside, he correctly points out that searching for all possible explanations is computationally prohibitive, while settling on the first explanation can lead to an incomplete interpretation of the text, since some events have multiple motivations. Dyer decides the best approach is a modification of strategy (1): find the first explanation *at each of four levels:* scriptal, goal/plan, thematic, and role. For example, in the main example story Dyer uses, two old friends meet for lunch to discuss some business. The meeting would be understood three ways: in terms of the restaurant script, the friendship theme, and the businessman role. While this approach may be a good heuristic in many cases, it will fail in others. An event could just as easily have multiple explanations at the same level. For example, in *"John was hungry when it started to rain. He ducked into a restaurant."*, a single action can be interpreted as a plan towards two goals: staying dry and having food.

In summary, Dyer's work is more a model of memory organization than it is a model of story understanding or inference. The types of knowledge structures and the connections between them are well worked out, but the class of inferences covered is under-specified, due to the arbitrary nature of the inference demons. Dyer's is the only work discussed here that can infer multiple connections between concepts in the story, but there is no clear statement of what inferences will be found. One view is that this is just in the nature of natural language understanding: it is a complex task, and one should not expect simple statements of principles. Another view is presented in the next section.

Story Understanding Principles

A slightly different tradition of research concentrates on characterizing the constraints on the story understanding process, rather than describing an algorithm in detail. For example the following principles were presented in Wilensky's [136] and in an earlier version of FAUSTUS [88].

The Principle of Coherence: Build a coherent construal of the input.

The Principle of Concretion: Always use the most specific interpretation of the input as is possible.

The Principle of Least Commitment: Make only the minimal assumptions necessary to interpret the input.

The Principle of Exhaustion: Make sure that all of the input is accounted for in the construal of the text.

The Principle of Parsimony: Construct representations which maximize the connections between inputs.

The Principle of Poignancy: Determine the point of what is being said.

The principle of coherence instructs the understander to find connections between various parts of the text. Causal relations are particularly important here. The principle of concretion says to go beyond a strict interpretation of the text, and induce details. The principle of least commitment limits the understander from inventing characters, objects, events and explanations that are unrelated to the text. The principle of exhaustion assures that an explanation does not use just part of the input, if using more of the input could produce a better, or more complete explanation. The parsimony principle says to find connections in such a way that the connectivity is maximized and the number of new objects introduced is minimized. Finally, the principle of poignancy assumes that the story teller has a reason for presenting the text, and that this is worth discovering.

The difficulty with applying these principles to any particular text is that they contradict each other, and it is never clear how to resolve the contradictions. Wilensky suggests a *ceteris paribus* interpretation of each principle. Given this approach, the principle of concretion, for example, would be interpreted as if it were *always use the most specific interpretation that is consistent with the other principles*. Consider the principle of least commitment, which was originally suggested by David Marr [76] for use in computer vision programs. This principle instructs the understander not to jump to unnecessary conclusions. The principle shows up often in AI research. Sacerdoti's non-linear planner, NOAH [102, 103], was also based on the least-commitment strategy, although he did not formulate it as a principle. In the abstract, the principle sounds like a good idea, but it contradicts the principle of concretion and the principle of parsimony, which each instruct the understander to make unproven assumptions, if the assumptions will lead to a more specific or better connected interpretation.

Accepting the *ceteris paribus* interpretation, we still need some type of comparison mechanism for mediating the contradictions between principles. For example, in interpreting *John went from New York to San Francisco in six hours* it would be a proper use of the principle of concretion to assume he went by plane, but it would be a violation of the principle of least commitment to assume he sat in seat 3A of a Boeing L1011. Some mechanism must determine there is enough evidence for the first inference but not for the second.

The parsimony principle has a particularly long geneology. In the early fourteenth century, William of Occam proclaimed "it is vain to do with more what can be done with less." This principle is now known as *Occam's Razor*. Paul Kay and Charles Fillmore [43, 61] formulate a parsimony principle for inferencing, as well as what they call the parsimony promotion principle:

Whenever it is possible to link two separate scenarios into a single larger scenario by imagining them as sharing a common participant, the ideal reader does so.

Select schemata in such a way as to give the parsimony principle the widest possible scope of operation.

Kay illustrates these principles with the following example:

(17) One day a chef went to Fisherman's Wharf and bought some fish from a fisherman.

He claims that an ideal reader of (17) will make the following inferences:

(17a) The chef will cook the fish at his restaurant.
(17b) The fisherman caught the fish.
(17c) The fisherman is a commercial fisherman.
(17d) The chef used the restaurant's money.
(17e) The purpose of the chef's trip was to buy fish.
(17f) The transaction took place on Fisherman's Wharf.

These inferences are not explicitly stated in the text, and they are all probable but not necessary inferences. The parsimony principle applies to this sentence because it involves four separate schemata: `chef-cooking`, `traveling`, `commercial-event`, and `fishing`. These end up being linked together in various ways: the `chef` in the `chef-cooking` is the `traveler` in the `traveling` event and the `buyer` in the `commercial-event`; the `fish` is the `food` in the `commercial-event`, the `merchandise` in the `commercial-event`, and the `catch` in the `fishing`. The parsimony promotion principle leads to the interpretation of the fisherman as a commercial fisherman rather than a sport fisherman, since that interpretation leads to more sharing of participants between schemata.

There are limits to the parsimony principle that are not addressed by Kay. For instance, it would not be valid to envision the fisherman as the diner in the chef's restaurant, or the chef as the first mate on the fisherman's boat, even though such envisionments would link scenarios.

Granger [45] independently presents his own parsimony principle as follows:

The best representation of a story is the one requiring the fewest number of goals to explain the actions of a story character.

Wilensky's parsimony principle is thus a generalization of Granger's. The intent of Granger's principle is similar to Kay's and it has similar problems in its applicability. The principle could be applied to (17) to infer (17a) and (17e). This would be a parsimonious explanation because we know that chefs normally have the goal of acquiring food to cook in their restaurants. It is certainly a more parsimonious explanation than an interpretation where the chef went to Fisherman's Wharf with the goal of visiting the Wax Museum, and then just happened to see some nice fish, which he bought to take home to his mother in law. However, we can invent even more parsimonious explanations. Perhaps the chef has only the single goal of, say, impressing his girl friend, or of serving God, and all his actions are in service of this goal. The parsimony principle as Granger states it gives no limits to interpretations of this sort.

Drew McDermott's TOPLE story understanding system [79] presented a calculus for measuring parsimony, although he did not use those terms. He dealt with structures called *belief rings*, and distinguishes *tension-causing* and *tension-reducing* elements within the belief structures. McDermott draws a parallel between his approach and the ideas of good form and stable organization of the Gestalt psychologists [64].

Finally, Fillmore [43] acknowledges Harvey Sacks [104] and Yorick Wilks [142, 143] as having independently formulated similar parsimony principles.

Story Grammars

The above research has been primarily aimed at the "understanding" part of story understanding, and has all but ignored the "story" part. How does the fact that one is reading a story (as opposed to watching the news or reading a journal article) affect the interpretation? The basic ability to do *common sense inferencing*, to recognize a situation and the implications of that situation, is a necessary precondition to understanding a story, but it is not the only precondition. There are some inferences that the reader makes precisely because he is reading a story, inferences that he would not make if he observed in the real world the same situation that is reported in the story.

Consider what we must know to understand a typical murder mystery story. First, we know that poison can kill people and that murder is a crime. These are facts that are of relevance to every day life; we avoid swallowing poison, or giving it to our friends. Secondly, we know some things about the capacity for being mystified. We know that problems hold our interest if they deal with important issues (such as life and death), and if they are neither too obvious nor too obscure. Thus, a story which relates events in chronological order, starting with the butler buying poison and sneaking it into the master's food is not a good murder mystery. We expect that the identity of the murderer will be kept hidden until the end of the story, but that there will be ample clues along the way. Third, we know something of the structure of mystery stories. We know that if it seems obvious who the murderer is early in the story, then the author is probably trying to mislead the reader, and the real murderer will be someone else. Such facts are relevant to understanding a story, but not to, say, witnessing or planning an actual murder. At another level is language specific knowledge. We must be able to understand the words and grammatical patterns of English. Note that the first levels did not require this; we can understand a pantomime or silent movie; or understand the consequences of drinking poison without any recourse to language.

There have been many attempts to define the concept of "storiness," going back to Propp's [92] analysis of folk tales (as formalized by Lakoff [66]) and continuing more recently with Rummelhart [101], Thorndyke [126, 127], and Mandler & Johnson [72, 74]. The initial idea was that just as sentences can be decomposed into words according to grammatical rules, so stories must be decomposable according to story-grammar rules. The question then is what are the components of a story? For spoken sentences the lowest level components are phonemes, which are combined into words, which fit into categories such as noun and verb, and can be combined into higher level categories such as noun phrase and verb phrase. This type of analysis is compelling for sentences because it is relatively easy to isolate and classify words, and because the grammatical structure can lead to much of the meaning of the sentence. However, note that the grammatical structure of a sentence does not determine the complete meaning of the sentence.

In the case of stories, a grammatical analysis is more problematic, because there is no easy way to classify the components. Wilensky & Black [132] point out that story grammars have relied on constructs such as *causes*, *initiations*, and *motivations*, which are on quite a different level than *nouns*, *verbs* and *adjectives*. In parsing a sentence, one can appeal to the lexicon to determine that *shoelace* is a noun, but in "parsing" a story there is no similar recourse to determine that *Mary was sad* could be a motivation for a subsequent action on her part. This could only be determined by some unspecified understanding process. The problem is that there is a circularity; the story cannot be parsed until it is understood, but story grammarians claim that the understanding is achieved through parsing.

On the other hand, Wilensky & Black seem to concentrate on the problem of understanding *situations* that are described in a straightforward fashion, and do not really address the difference in understanding a *story text* based on that situation. Thus, they have no explanation of how formulaic expressions like *once upon a time* or *they lived happily ever after* make their way into stories. This is one of the complaints made by Mandler & Johnson in [73]. Wilensky provides a more complete exposition of story points and a criticism of story grammars in [137].

What is needed is a theory of understanding that takes into account the structure of stories. Unfortunately, most current models have been either theories of understanding that ignore storiness, or theories of storiness that ignore the understanding process.

Point Based Story Understanding

Wilensky's [111, 135, 137] theory of story points was proposed as an alternative to the story grammar approach. The theory concentrates on human dramatic situations that involve interacting goals. The important points of a story are defined to be the interesting goal interactions. A story point analysis of a text could produce a summary of the important points, once a thorough interpretation of the goal structure was made.

Lehnert's [70] theory of plot units is very similar. The main difference is that she allows affect, as well as plans and goals, to figure into the analysis. She uses Roseman's [99] treatment of affect, which identifies five dimensions of affect (desirability, attainment, certainty, deservedness and agency) and about thirteen primary emotions (joy, hope, relief, *etc*). A character's emotional reaction to an event can be predicted by ranking the event on each of these dimensions. Similarly, given a character's affective state, likely consequences can be predicted.

One problem with Lehnert's approach is that it depends only on positive or negative changes, and does not recognize that some feelings are more important than others. Thus, the passage *"John got a new pencil. Then it broke."* would be given exactly the same plot unit analysis as *"John married Mary. Then she died."* Wilensky's story points theory suffers from the same problem. In both cases, the analysis would be that John had a recurring goal, achieved subsumption of that goal, and then suffered a loss of his goal-subsumption state. There would be no indication that the death of a spouse is more important than the loss of a pencil. Wilensky mentions that the value of a point is a function of the goals involved, but he does not elaborate on determining how important particular goals are, other than to say the importance of goals are given. Another difficulty with both theories is that for long stories there is no good way to compare different parts, and decide which parts are more important. Lehnert has a summarization mechanism that generates as a summary the plot unit or units most heavily connected to other units. However, she does not allow the summary to abstract away from the literal text, nor does she distinguish storiness from coherency.

Schank [112] had the idea of allocating inferencing resources based on the interestingness of the text at hand. Some concepts, like death, sex, and money are intrinsically interesting, in this theory, while other concepts could be interesting if a personal relatedness was perceived by the understander. Under this theory, the understander would make more inferences about interesting concepts, and hence they would figure more prominently in the resulting interpretation of the text.

Previous Research in Memory Models

This section will review research in AI and psychology on models of human memory. There are a number of reasons why AI researchers should be concerned with results in psychology. From a scientific point of view, one of the goals of AI is to better understand human cognition. From an engineering point of view, humans are the best, and only, example of working intelligent systems, and thus are worth examining. Finally, because language is a means of communicating mental concepts between humans, understanding language requires an understanding of human mental processes. At the end of this section is a short discussion of how these results from psychology have been applied to computer models of knowledge representation.

Network Models of Human Memory

Network models depict semantic memory as a large network of associated concepts. The concepts are normally called *nodes,* and the associations are called *links*. In most models there are a small number of different types of links. In some models the links themselves are full-fledged concepts, in others they are primitive elements, with semantics known to the processes that interpret the network. All other concepts are defined circularly, in terms of their links to other concepts.

In any memory model, the representation of concepts must be related to processes like recall, memorizing, inference, learning, and forgetting. Most research to date has concentrated on recall and inference.

Quillian's model, presented in [93] and [31], is one of the first memory models, yet is fairly representative. Although the model explains some experimental data well, it was not intended to account for data. Instead, the model was designed primarily to facilitate programming of a language understanding system, TLC, the Teachable Language Comprehender.

Quillian's memory system had five primitive link types: superordinate, modifier, conjunction, disjunction, and property. Using these, the concept `client` can be defined as a node with a superordinate link to `person`, and a modifier link to a node which might be called `employs-professional`. This node is in turn defined with a superordinate link to `employs`, and a property link saying that its first argument is `professional`, and its second another property with superordinate `by` and first argument `client`. Quillian chose not to give the nodes names in the actual network, using an external dictionary instead, but the system is easier to understand if names are explicitly used.

The Spreading Activation Model

Quillian's system viewed memory search as a process of *spreading activation.* Given two or more starting concepts, a search procedure spreads out from each of the starting nodes along links to adjacent nodes. An activation tag is left at each node visited by this procedure. Eventually, an intersection will occur where one node receives tags from two starting nodes. At this point an evaluation procedure is invoked to determine if the intersecting occurred because of some inference which should be made, or if the intersection is purely accidental, and should be ignored.

The idea of blindly spreading out from the starting point and searching for inferences is similar to Rieger's approach. The difference is that in Quillian's case, no inference is actually made until there is an intersection.

TLC attempted to deal with three classes of inferences: word sense disambiguation, property composition, and anaphoric references. It attacked these problems with a single mechanism. For example, if TLC is processing the phrase *lawyer's client*, it first builds a node for lawyer. Activation tags are spread from this entry, but as there has been no previous input, there is no chance for an intersection. In processing *client* activation tags are again spread to neighboring nodes, and an intersection occurs along the path `client - employs - professional - lawyer`. At this point a decision process is called which checks the syntactic relation of the words in the text to determine if this path should be incorporated into the representation of the text, or if the path is purely coincidental. In this case, the path would be accepted. Other paths might be rejected. For example, the phrase *the fall leaves* might lead to an intersection where `leaves` are recognized as an object of `to fall`, but that path would be rejected in favor of the path where the season `fall` is a modifier of `leaves`. One important limitation of Quillian's approach is that he did not make a clear distinction between words and the concepts they represent.

Although not shown in these examples, the same mechanism that infers connections such as the one between lawyer and client is used to choose the best sense of ambiguous words, and to resolve anaphoric references. For each word in the text, TLC sets up a new node which initially has a set of pointers to candidate senses. The candidates are all the dictionary entries for the word, and any existing instances of those entries that were mentioned previously in the text. The system spreads activation tags from each candidate; the one that first connects in an acceptable path is declared the winner. Thus, if TLC were to continue processing with the phrase *she argued a case* the word *she* would initially have three candidate senses, the lawyer, the client, and the generic `female`. The same process that connects *she* to *argued* would pick out the lawyer discussed previously as the correct sense.

The Quillian model is defended and extended in Collins and Loftus [32]. To explain various psychological data they introduced refinements to the model. In their extended theory activation tags have strengths associated with them, which decay over time and distance. An intersection is redefined as a node where the accumulated activation exceeds a certain threshold of activation.

The theory was extended primarily to counter the feature model of Rips, Shoben and Smith [96, 97]. They showed, for instance, that subjects were faster at answering *Is a robin a bird?* than *Is a penguin a bird?* According to Collins and Quillian's [31] presentation, the response time should be proportional to the number of links traversed in the memory search. If in both cases the only relevant traversal was the superordinate link to `bird`, then both questions should take the same time to answer. Rips, Shoben and Smith thus propose a feature model, where instead of comparing superordinate links, one compares the properties of the the two concepts at hand. The decision is made through a two-step process that first compares a small set of "defining" features, and then if a decision has not been made, compares the complete set of features.

Collins and Loftus point out that Quillian did not intend the superordinate link to be the overriding, or even the primary factor used to determine matches. Instead, categorization can in general be determined by weighing any number of pieces of positive and negative evidence. Besides superordinate links, properties can be contrasted, prototypes or examples can be compared, counterexamples can be searched for, and so on. Following these provisions, a network model can perform similarly to a feature-set model. One can be considered a notational variant of the other. This should not be surprising to anyone who has ever attempted a computer implementation of such a model; one obvious way to implement features is with property lists, which are just nodes and links. McDermott expands on this point in [81].

There were still some problems with the Quillian model. His syntactic rules were impoverished. He could not easily extend the test for *lawyer's client* to handle *the lawyer's new client*. Quillian's networks were missing many of the representational capabilities of more recent proposals, and I think many of the problems with his system stem not from his basic approach, but from the lack of representational expressiveness he had at his disposal at the time. This showed up at three levels. One, there was confusion about levels of description, about the intensional/extensional distinction, and about the representation of sets. These types of problems are discussed in [14, 15, 147]. Two, the vocabulary of domain-level concepts that, for example, Dyer uses in [38] (scripts, plans, goals, plot units, service triangles, etc.) was unavailable then. And finally, a new way of looking at categorization has developed since then, due to works like [67, 68, 98, 149].

Quillian freely admits that certain types of knowledge, such as mental imagery and perceptual-motor capabilities, were "far beyond our present scope." These remain beyond the scope of current research to this day. Despite these shortcomings, the Quillian model had a number of important points, some of which have been lost in the intervening 15 years.

• The emphasis is on a very extensive memory network, rather than on complex processing strategies. The processing consists mainly of searching memory for closely related concepts.

• Word sense disambiguation, property composition, and anaphoric reference resolution all emerge from this searching strategy.

• The strategy employs an automatic, autonomous search procedure coupled with a controlled decision procedure.

• Processing is based on semantic relationships. Syntax serves a secondary role.

Other Network-Like Memory Models

Although Quillian's model is typical of many proposition-based network models of semantic memory, it is by no means the only such model. Kintsch presents a model [63] that has more examples worked out than Quillian, and has more of a background in psychology. He covers a number of important concepts, like quantification, modality, presupposition, and time. However, as a ten-year old work, the findings are not up to date, and are not worth discussing here. Similar remarks apply to Norman and Rumelhart's model [87].

John Anderson has presented two major models of semantic memory. The HAM model [4], developed with Gordon Bower, included a computer implementation that could accept statements like *In a park a hippie touched a debutante* and questions like *Who was touched by the hippie?* and produce answers like *The debutante*. The model was motivated largely by experimental results on list-learning tasks, and thus had less to do with inference and understanding than the other models discussed here.

The ACT model [6-8] is in Anderson's words a model of factual memory, and is not primarily a model of semantic memory. Like HAM, it is more concerned with the mathematical specifics of encoding, storage, and retrieval than with higher-level inference processes.

Another important knowledge representation formalism of the 1970's is Schank's Conceptual Dependency model [105, 110]. While representations in this formalism look quite different than Quillian or Rumelhart's diagrams, the differences are largely cosmetic. One difference that is important is Schank's commitment to reductionism. In [106], he describes fourteen primitive acts which can be combined, he claims, to specify all verbs of action. The key point was that there are a well-defined set of inferences that can be made from each primitive act. For example, when something is moved, its location is no longer the source location but rather becomes the destination location. The problem with primitive acts is that there are many non-primitive acts that have inferences associated with them that cannot be derived from a composition of the primitives. For example, it is a fact that if a person moves through the air for a distance of 300 miles, he probably was in an airplane. This is not a fact about the primitive concept moving, however, so in CD there is no good place to store this fact, and no good way to represent it.

Schank extended and modified Conceptual Dependency greatly in his theory of Dynamic Memory [113]. Here he was concerned with the types of memory confusions one makes over a period of time, and with how one recalls and reconstructs past events. For example, why is it that we could confuse what happened while waiting in the dentist's office with what happened in the doctor's office? How do one remember the last time renting a car? Much of the theory is an attempt to make up for the unfortunate decision to utilize the primitive acts to the extent of ignoring perfectly good non-primitive concepts. Like Conceptual Dependency, the Dynamic Memory theory has a tendency to categorize and enumerate rather than to explain or motivate. Kolodner extended this work, concentrating on episodic memory, in [65].

Connectionist Models of Memory

Fahlman's NETL architecture [40] led to a small renaissance of the spreading activation approach. The trend at that time had been to consider inferencing as a problem in heuristic search, and to discover ways of limiting the search space to avoid the combinatorial explosion. Fahlman's conclusion was that syntax-directed heuristics could never reduce search effectively in the general case, but special-purpose parallel hardware could allow blind search to take place in a reasonable amount of time. This was a popular approach because it matched, on some level, the architecture of the brain, and also because it matched the architecture of supercomputers that are just starting to be designed.

Fahlman concentrated on question verification (e.g. *Does a nautilus have a shell?*) and retrieval (e.g. *What kinds of molluscs are there?*) rather than on making plausible inferences. His system was designed to do all processing in parallel, but for the types of inferences we are concerned with in this report, that is clearly not possible. To choose a pronominal referent, for example, it is possible to gather candidates in parallel, but choosing between them requires sequential comparisons. To see this, recall Charniak's example of the birthday-present story, given as (1) above. The last sentence is *He will make you take* it *back.* A parallel process could decide that possible referents for *it* are the top Jack already has and the top Janet is considering buying. The parallel process could even give some score to each referent indicating how plausible it is. However, only a sequential process could compare these scores and pick out the best referent.

Waltz and Pollack [128, 129] work in a new sub-field called *connectionist parsing*. They take spreading activation models to an extreme, allowing them to do all the work, with no higher-level decision processes. This requires highly tuned weightings for values of activation and inhibition between nodes. It is not clear yet if this approach will be feasible for more than very simple examples. Small and Cottrell [123, 124] have done similar work in connectionist parsing. The idea is to train a network to automatically acquire the "right" weights. This is done by presenting sample inputs, letting the network compute an output, and then using a process called back-propagation to reconcile any difference between the network's output and the expected output. The obvious advantage of such a system is that the experimenter need not understand the internal workings of the resulting system; indeed, he may not be *able* to understand it, even if it does work. The disadvantage lies in the fact that language understanding and commonsense inferencing appear to require many levels of processing, and it is not known if connectionist models will ever be able to generate the necessary intermediate levels between the input and output. Connectionists are encouraged by the intuition that human brains seem to have a connectionist architecture, and infants learn to reason and comprehend with only a few years of learning time. However, this neglects the four million or so years of evolution required to generate the initial configuration of the brain.

Marker-Passing-Based Research

Granger, Eiselt and Holbrook [46] present an integrated parser/understander called ATLAST. This system has three main components: the capsulizer, which takes care of lexical access and local syntax; the proposer, which searches for inferences by spreading activation; and a filter, which evaluates inferences and handles more complex syntax. Each component runs in parallel, and can communicate with the others. For example, the filter can tell the proposer to stop looking once it has accepted an inference.

Charniak [28] presents the system that is perhaps most similar to FAUSTUS. The Wholy Integrated Marker Passer, or WIMP, parses, disambiguates and draws inferences from texts. The examples presented have all been one or two sentences. WIMP passes markers and finds marker collisions in a manner very similar to FAUSTUS. The rules for passing are somewhat different, involving what Charniak calls *zorch* strength. The idea is that marker energy is divided equally among all paths leading out of a node. When this energy falls below 1 it is truncated to 0, and marker passing stops. Collision evaluation is different as Charniak uses a more general and formal resolution theorem-proving based approach, while I give a set of pre-defined marker paths which have interesting inferences associated with them. Conflicting suggested inferences are settled by accepting the one with the shortest path length (or maximum path zorch, in his terms). Another difference is that Charniak passes markers only from open-class words, while I pass markers from all input, including prepositions, and thus can disambiguate vague case relations. Finally, WIMP's knowledge base is much smaller than FAUSTUS's, and it seems the program has only been tested on one to three sentence texts. This system is a continuation of earlier papers [26, 27] which propose that spreading activation could be a good starting point for a model, like Quillian's, that made maximal use of semantic information, and only checked syntax when necessary. However, Charniak uses an ATN parser which is much more powerful than Quillian's *ad hoc* syntactic formalism.

Hendler [51] extends Charniak's approach to address the problem of problem solving in a spreading activation model. He is primarily concerned with avoiding backtracking in planning; this is done by having a marker-passing mechanism make appropriate suggestions as to what to try first.

Granger [47, 48] is doing similar research, but he concentrates on the problems of lexical access and word sense and case slot disambiguation, rather than on higher level inferences.

One important unanswered question is how these systems will scale up as the size of the knowledge base is increased. This is a problem both in terms of execution speed, and in terms of unpredicted interactions as new concepts are added to the knowledge base. In the case of connectionist models, we will not know how easy it is to add a large number of concepts until the next generation of hardware becomes available. Waltz's models takes on the order of several minutes to run a small example with a few dozen nodes. Charniak's 1985 model [27] has about 75 generic concepts, and runs somewhat faster, since spreading activation models are at a higher level than connectionist ones.

The FAUSTUS data base has roughly 8 times as many concepts as Charniak's, and the transition in FAUSTUS from 60 to 600 concepts resulted in a qualitatively different system. A system with 10 times more concepts would be qualitatively different again.

How Large Must a Memory Be?

Many AI researchers have tried to speculate as to how many facts or concepts are known by the average person, and how many are needed to perform "intelligently." In [82], Minsky concludes that "a million, if properly organized, should be enough for a very great intelligence. If my argument does not convince you, multiply the figures by ten." It is not clear if the last sentence is meant to be applied recursively, or only once. John McCarthy has stated that as few as 100,000 facts might do. David Waltz gave the figure of two million, which is ordinary in its magnitude, but extraordinary in that he seems to be the first to start an estimate with anything other than the digit '1'. There are several ways of arriving at an estimate, or at least of setting bounds. For a lower bound, a small dictionary has about 20,000 words; someone who knew half of the words, knew only one 'fact' about each word, and knew nothing else would have 10,000 facts. As a more realistic estimate, Elizabeth Bates has estimated that the average person has a vocabulary of 40,000 words; this would include proper nouns and names that are not in a dictionary. If we accept this estimate and guess that one knows an average of 10 facts about each word, and if one knows twice as many non-verbal facts as verbal ones, then we arrive at a figure of 1,200,000 facts.

A 32 year old adult who learned one fact per second throughout her life would have acquired a billion facts. This does not account for forgetting, for generalization of learned facts, or for the reduced rate of learning during sleep. In addition, one fact per second seems to be a high rate of learning; as Simon points out [122], there have been a wide variety of psychological experiments that show a maximum learning time of about one syllable every ten seconds, and that is achieved only when the subject is concentrating on the learning task. On the other hand, these experiments dealt with nonsense syllables; presumably meaningful information would be learned faster. As a rough estimate, assume that the rate-of-acquisition argument leads to an upper bound of about one billion facts. The number of neurons in the human brain is about ten billion. Most current neurological theories assume that information is stored by patterns of neuron activation, rather than in individual neurons, and it is unknown what these patterns are like. But if we arbitrarily assume ten neurons per fact, we arrive at an upper bound of a billion facts from two different sources.

In summary, if we accept the given assumptions, we can be almost certain that the total number of facts in a human mind is between 10^4 and 10^{10}, and fairly confident that it is between 10^6 and 10^9. To try to narrow the gap between these bounds any more seems a little premature, since we are not even sure what exactly should count as an individual 'fact,' let alone a million of them, and since even AI systems that are called "large" number their facts in the thousands, not millions or billions. There has been little or no published speculation along these lines; the estimates in these paragraphs from McCarthy, Waltz, and Bates are from personal communication.

Summary of Previous Research

A theory of inference must be able to answer three questions: what classes of inferences are supported? What control structure is used to generate individual inferences? If contradictory inferences are suggested, how is the conflict resolved? Most of the researcher reported here attempts to handle the first question. Rieger, Lockman & Klappholtz, Clark, and Kay make this the primary thrust of their work. Most of the AI researchers address the second question. Unfortunately, the third question remains elusive; no researcher makes it the main point of his work, and some fail to address the question at all.

There is a trend among the AI text understanding systems (Alterman, Cullingford, Wilensky, Granger, DeJong, Lehnert) that should not go unnoticed. In each case, a new type of knowledge structure (*e.g.* coherence relation, script, plan, plot unit) was developed, along with a set of inference rules appropriate for that knowledge structure. However, in each case the inference rules were in the form of a new top-level inferencing algorithm; there was no way to incorporate different types of rules in one system. For example, Wilensky's PAM was designed as an improvement on Cullingford's SAM, but PAM had a completely new top-level algorithm that could not easily incorporate the class of inferences that SAM made. Furthermore, PAM could not make certain inferences that should have been within its power to make. For example, PAM could find the connection between the two sentences in (18a) and determine who *he* refers to because the sentences involve a character's actions and affect. PAM could not determine what *it* refers to in (18b) because there are no goals or plans involved. Yet both examples are similar in that the first sentence causes the second. PAM could not represent this commonality.

(18a) *Bill hit John. He cried.*
(18b) *The ball hit the vase. It broke.*

Dyer's work is an exception to this trend towards championing a single new knowledge structure, since he allows inference rules for various knowledge types at various levels to be operating in the same system. However, Dyer suffers from the same problem as Rieger; he presents very specific inference rules that can only be used in one particular context, even though they represent knowledge that should be applicable in a variety of ways. Furthermore, he has no unified control structure; instead he relies on demon activation to do the right thing.

I have pointed out two flaws in past systems that both stem from the same source: the inability to represent knowledge in a neutral declarative fashion and use that knowledge whenever it is applicable. In the next chapter I present a knowledge representation system that allows representations to exist at the proper level of abstraction, and allows commonalities in representation to be reflected by commonalities in processes.

Chapter 3: Knowledge Representation

An important claim made in chapter 1 was that the inferencing algorithm for a text understanding system could be made simpler, if the system could represent declaratively a variety of knowledge structures that were handled procedurely in other systems. In this chapter I formulate the criteria that such a knowledge representation language (henceforth, *RL*) would have to meet, and describe KODIAK, a RL which meets these criteria.

There is a distinction to be made between what goes into the RL itself, that is, what are the primitives of the language, and what sorts of concepts are represented using these primitives. The first process, ***RL design***, is the topic of this chapter. The second process is often referred to as ***modeling the domain*** of interest. Some of the concepts modeled in the FAUSTUS system will be discussed in chapter 5.

Another important trichotomy separates the RL, which is a descriptive notation, from the operations that manipulate those descriptions, and the application programs that use those operations. The combination of RL and operators I call the representation system (RS). This terminology is not in general use; in fact sometimes even those who warn most strenuously of the dangers of confusing descriptive issues with processing issues are guilty of using the term "representation language" to mean RL at some times and RS at others. For example, one will often find mention to "an efficient knowledge representation language." In my terminology this would be nonsensical; a RL can lend itself to efficient implementation of a RS, which can in turn lead to efficient implementation of a parser, or theorem prover, or whatever, but the RL itself cannot be "efficient" or "inefficient."

A RL capable of supporting a text inferencing system must be able to represent three types of knowledge. First, the RL must be able to define terms and describe facts about the domain. Second, it must allow for the representation of the meaning of the input text. It must be able to represent the meaning of the text as far as it is known, and represent ambiguities in the text until they are resolved. Third, the RL must be able to represent inferences derived from the text. There is by no means universal agreement on this characterization of RL's. Some RL designers, such as Brachman and Moser [16, 85], feel there should be two distinct languages, one for defining terms in the domain, and another for making assertions involving those terms. In a separate article [18], Brachman argues that text can be represented within the assertional RL, making use of a *syntaxonomy*, a taxonomy of syntactic entities. A similar approach is taken in [125]. Moser does not discuss the representation of text; perhaps he would favor a third language for that.

Another issue of debate is the range of applicability of a RL. It is often easier to define RL's, and to build models in a RL, if the use of the representations is restricted. For example, in one large natural language understanding project being developed jointly at BBN and ISI, there are two separate representations of English grammar, an ATN grammar for parsing and a systemic grammar for generation of language. The idea is that each representation is better for the task it was designed for, and that it would be more difficult to specify a single grammar that could handle both tasks. If the goal is to apply available technology and produce a working program, this may be the best approach. However, if the goal is to make the most effective use of knowledge, then it would be better to have a system where the same knowledge base can be used for a variety of purposes: to understand stories about a given situation, to generate stories or answer questions about that same situation, or to act in a reasonable fashion when faced with a similar situation. Herb Simon defines *understanding* as: *understanding a piece of knowledge K means using K whenever it is appropriate to do so.*

KODIAK [138] is a RL which was designed to allow the modeling of a large knowledge base that could be applied to a variety of tasks. In other words, representations in KODIAK should be constructed to address the question *how can I accurately model my conception of the domain* rather than the question *how can I describe the facts I need to get this particular application to work.* An earlier version of FAUSTUS had a knowledge base that it shared with a common-sense planning program, PANDORA [41]. The PHRAN parser used in FAUSTUS has always shared its syntactic knowledge with the language generation program, PHRED [59]. When the generator was redesigned as Jacobs' ACE, sharing knowledge was still a priority. KODIAK is also used as the RL for UC [139], a consultation system that answers natural language questions about the UNIX file system. Thus the idea of sharing knowledge between systems that perform different tasks is well established at Berkeley, and is also an important thrust of recent work at Brown [54, 146].

Few other researchers have addressed the problem of sharing linguistic and real-world knowledge to this extent. However, there have been several attempts to allow one grammar to serve for both parsing and generating tasks. The programming language PROLOG is designed so that a procedure can in some cases solve for any one of its missing arguments, when the others are supplied. For example, we might be able to write "$X = Y + Z$" and solve for any of X, Y or Z whenever the other two are given. In actual practice, PROLOG does not always perform this way, and in fact simple arithmetic is one of the areas in which it fails to exhibit the desired behaviour. However, several grammatical formalisms have been been developed that can be manipulated by PROLOG in just this manner. Pereira and Warren [91] introduced *definite clause grammars* or DCG's, which are designed for parsing using PROLOG, but could also be used for generation. More recently [130], they describe *extraposition grammars*, and show how they can be applied to the problem of translating natural language queries to a data base. Martin Kay's *functional unification grammar* [62] has the property of reversibility with respect to translation; sentence *a* in language *A* can translate to sentence *b* in language *B* if and only if *b* can also translate to *a*. Jacob's ACE formalism [57] has the explicit goal of providing a direction-independent association between language and meaning, which is also somewhat independent of the grammatical formalism used.

KODIAK, like Conceptual Dependency (CD) theory [105], represents a conceptual level of analysis that is independent of the actual words and syntactic constructions in the sentence. It is a meaning representation language, not a word or sentence representation language. Thus, the output of the conceptual analyzer, and the input to FAUSTUS is a meaning representation, not a syntactic parse tree.

Unlike CD, a single KODIAK knowledge base does not claim to be an *interlingua*, a universal language to represent any thought expressible in any language. Rather, the knowledge encoded in a KODIAK knowledge base is designed as a model of one particular person's conceptual knowledge. Different people in different cultures speaking different languages and having different experiences will conceptualize the world differently, so we should not expect there to be one "correct" representation of all knowledge.

One guiding principle is that any concept that is important enough to have a word denoting it is important enough to be modeled as a full-fledged concept, not just as a composition of other concepts. Thus, we have explicit representations for concepts like buying and selling; we are not forced to decompose them into primitive actions as one would in CD.

Conversely, if a concept is represented in KODIAK, it means the concept is of some import. Other representation languages like KL-ONE that were concerned primarily with defining terms explicitly often ended up generating spurious intermediate level categories, such as `left-handed-person-with-two-supervisors-and-red-hair`. In KODIAK there is a difference between forming a category like this, and forming a description that happens to fit into several categories.

CD also makes a commitment to the idea of *reductionism*: providing a small set of primitive concepts that can be composed to represent any other concept. This is closely related to the *interlingua* issue, but they are ultimately orthogonal issues. KODIAK makes no such commitment to reductionism; in fact it promotes a proliferation of concepts, each defined recursively in terms of the others, but with no pre-determined set of primitive concepts.

Criteria for Representation Languages

The most basic criterion of a RL is what McCarthy and Hayes [78] call *epistemological adequacy*. By that they mean the RL must have sufficient expressive power to represent the concepts the modeler wants to define. If, for example, the modeler is interested in the domain of algebra, then it would be useful for the RL to have mechanisms for dealing with sets and numbers built in as primitives. This notion has been rediscovered by a number of researchers; McDermott [80] calls it *analytic adequacy* and Woods [148] calls it *expressive adequacy*.

As another example of expressive power, in KL-ONE one can assert a minimum and maximum on the number of role fillers (of each role) that a concept can take. For example, we can say that a mammal has a minimum of two and a maximum of four legs, and that a person has exactly two legs. A language with more expressive power might allow the modeler to assert that all animals have an even number of legs (except perhaps starfish). However, adding expressive power like this can have repercussions in terms of efficiency and inferential power of various operations in the RS. If the only restriction allowed on the number of role fillers is a maximum and a minimum, then there is an easy algorithm for determining if all instances of a given category must be instances of another category. This is called the *subsumption* relation in KL-ONE.

Determining subsumption becomes more difficult if we allow more expressive power. Suppose we wanted to define the concept `prime-pose` to mean a pose that an `animal` adopts whereby a prime number of legs are bent and a prime number of legs are straight. Let us also define `prime-animal` as an animal that is capable of adopting the `prime-pose`. Then, asking the RS if all `animals` with more than two legs are `prime-animals` would be equivalent to asking it to prove Goldbach's conjecture, a well-known and long unsolved problem in mathematics. This is unfortunate, because we would like all operations in the RS to be easily computable. For more examples of how easy it is for the computational complexity of queries to the RS to get out of hand as we increase the expressive power of the RL, see [17]. He also gives a proof of a subsumption algorithm there.

Epistemological adequacy is a matter of degree, rather than of an absolute adequate/inadequate distinction. But just as important as the number of concepts we can represent is the ease with which they can written, extended, shared, and modified. There is a parallel here between RL's and programming languages. The "epistemological adequacy" of ADA or COMMON LISP is no better than machine language, yet much time and effort has been spent in developing these new languages, because they make it easier to express complex algorithms and data structures.

McCarthy and Hayes also introduce the criterion of *heuristic adequacy*. A problem solver meets this criterion if it can use the facts it has represented whenever it is appropriate to do so. As mentioned above, the approach taken in KODIAK is to encourage the domain modeler to define concepts in a form that is neutral with respect to the intended use. If this is done consistently, heuristic adequacy will take care of itself.

Efficiency of operations in the RS is another criterion, but it is not one I will be overly concerned with, since it is not directly a property of the RL. The data bases involved in understanding short texts are (unfortunately) still quite small, and the inferencing mechanisms I am using are designed to be efficient, so that a wide range of RS implementations will all be efficient enough. In general, it behooves the RS designer to worry about efficiency to a degree, but it is important to maintain the distinction between efficiency issues and representation issues. For example, in the PEARL representation language [134], there was a mechanism for saying that every time an instance of a given concept was created, allocate storage and fill in the default value for certain relations involving that concept, but do not allocate storage for other relations. But this set of relations with pre-allocated storage is also the set of relations that is used in matching one instance with another. This means that a representation issue is mixed up with an implementation issue. A similar confusion occurs in the FRL language [100]. In KODIAK, all relations can come into play for matching purposes, regardless of how they are implemented.

The next criterion is *inferential adequacy*, what McDermott called *power*. For most RL's, inference takes the form of logical proof: if the assertions P and Q are in the knowledge base, and if it is given that $P \ \& \ Q \rightarrow R$, then the system had better be able to deduce R, and not deduce $\tilde{}R$. In other words, completeness and consistency are the guidelines against which inferential adequacy is judged. Unfortunately, this is a very harsh judge; if there are more than a few dozen facts in the system, then it is strongly believed that there cannot be *any* algorithm that can solve this problem in the general case, without taking years of computing time. Technically, we say that the satisfiability problem for predicate calculus is NP-complete, and therefore it is in practice impossible to guarantee completeness and consistency for any RS with more than a handful of assertions. We must accept some degradation in either the epistemological or inferential adequacy of any RS. For humans, the balance is tipped heavily towards epistemological adequacy; we rarely have trouble finding some way to think about a concept, but there are many, many valid logical deductions that we are not able to deduce. Humans also seem to have an ability to handle contradictory assertions without much trouble. This is in marked contrast to propositional logic, where "contamination" by an inconsistent belief is a constant problem. If P and $\tilde{}P$ are both believed true, then *any* proposition can be proved true.

In the domain of text understanding, logical deductions are not as important as plausible inferences. Also, I am not interested in proofs of general properties, but rather in specific inferences involving individual concepts. Thus, if I were to adopt the definition of `animal` that asserts an even number of legs, and defined `prime-pose` as above, I would only be interested in questions like *could Clyde, an elephant with four legs, adopt the prime-pose?* Obviously, it is easier to answer a question like this than to prove Goldbach's conjecture. Since my plausible inferencing algorithm looks only at paths of length less than n, where n is small, it does not matter if the computations are expected to grow exponentially for large n.

I have chosen not to explicitly define a logic which could, for example, deduce R from $P \ \& \ Q$ and $(P \ \& \ Q \rightarrow R)$. Instead, I have defined an inferencing mechanism for making plausible inferences triggered by connections along short paths, and now need to define *plausible* and *short*. The idea of *conceptual distance* makes sense for RL's that are based on, or can be interpreted as, semantic nets. KODIAK is such an RL. The conceptual distance between two concepts is defined as the shortest path between the nodes representing those concepts, where the length of a path is computed by summing a cost function over the links in the path. Conceptual distance is asymmetric; to reuse an old example, the distance from `island` to `body-of-water` is less than the distance from `body-of-water` to `island`. This is because part of the definition of `island` states that every `island` is surrounded by a `body-of-water`, but not every `body-of-water` need surround an island. The distance from A to C is always greater than or equal to the sum of the distances from A to B and B to C, if we assume B was in the network throughout. However, introducing a new concept B' to the network can conceivably produce a shorter path between A and C. Thus, conceptual distance is *non-monotonic*; it can change over time. The distance from A to A is always zero. Note that conceptual distance is distinct from *similarity*. The concepts `island` and `body-of-water` may be close, but that does not imply any similarity between the two. On the other hand, similar objects will normally be conceptually close.

Consider the task of the domain modeler. She or he must represent the concepts of interest using whatever primitives are in the language, along with new concepts she defines. In doing this she has certain intuitions about how to represent each concept. However, she also has intuitions on the conceptual distance between pairs of concepts. When there is a grave discrepancy between her intuitive judgement of distance and the distance computed by the system, she has three choices: change the function for computing the distance between concepts, change the representation of one or both concepts, or ignore the discrepancy and hope that her intuitions were wrong, and that the system is still able to make the right plausible inferences involving these concepts. In modeling concepts for FAUSTUS, I found the first choice was quickly ruled out after a brief shakedown period, because changing the cost function would effect the distance between every other pair of concepts in unpredictable ways. In many cases I was able to detect modeling errors when a path between two concepts was too short and triggered an erroneous inference, or was too long and missed an important plausible inference. In other cases there seemed to be no good way to change the distance between two concepts without perverting their meanings in some way.

Wilensky [141] re-iterates the principle of epistemological adequacy, extends the notion of heuristic adequacy, and introduces the criteria of interpretability and uniformity. Interpretability is the inverse of epistemological adequacy. The idea is that anything that can be stated in the language must have a ''meaning'' or interpretation, although that meaning may be false or nonsensical in the real world. The principle of uniformity states that a single language that can express anything is better than a collection of ad hoc languages.

Wilensky also introduces the principle of cognitive correspondence: ''a particular representation for a particular item must be supported by its correspondence to how that item is cognized.'' In previous systems, there was always the assumption that, for example, ''John ate lunch'' would be represented in predicate form as `eat(John,lunch)` rather than `lunch(eat,John)`. The cognitive correspondence principle recognizes that both choices are possible, that the choice between them is not arbitrary, and that the reason for preferring to associate verbs with predicates is an underlying cognitive reality of such an association.

The KODIAK Representation Language

Now that we have an understanding of the criteria for evaluation RL's, we can investigate the RL used in the FAUSTUS system, KODIAK, and discuss the design decisions that went into KODIAK. The KODIAK language was developed jointly by Robert Wilensky and several members of the BAIR group at Berkeley. The version described here differs in some details from the version discussed by Wilensky in [141].

Representations in the KODIAK language are composed of instances of three types of primitive objects, and eight primitive associations between those objects. When seen as a semantic network, the objects are called *nodes* and the associations *links*. Representations are formed by inserting links between nodes. When seen as a programming system, the primitive types are operators of no arguments that return new objects, and the associations are operators of two arguments that assert relations between the objects. Objects have names as a convenience for the system modeler, but the names are not used for purposes other than identification. The primitive object and link types are shown in Figure 1 along with a brief description of each one.

The Three Primitive Object Types

An *absolute* is any concept that can be modeled in its own right, that is, any concept that it makes sense to speak of as an individual entity. Absolutes need not represent physical objects; they can be actions, events, situations, or abstract ideas. Examples include `person`, `island`, `action`, `walking`, and `talking`. Attached to the concept `person` will be primitive associations to describe assertions that are true of all instances of `person`, but also information that is true for most instances but not all, or is true for only a few instances. In NETL [40] the concept that I call `person` would be called `typical-person`. The intent in KODIAK is to model the general idea of `person`, but to be able to distinguish typical instances from non-typical instances and from non-instances. There is no attempt to define a set of necessary and sufficient conditions for defining `person`; instead we describe `person` as best we can, and rely on the inferencing algorithm to classify any instances as a `person` or non-person. In general classification will depend not only on the properties of `person`, but on the properties of similar

Absolutes - concepts, *e.g.* `person, action, purple, government`
Relations - relations between concepts, *e.g.* `actor-of-action`
Aspectuals - formal parameters for the relations, *e.g.* `actor`

Dominate - a concept is a subclass of another class
Instance - a concept is an instance of some class
View - a concept can be seen as another class
Constrain - fillers of an aspectual must be of some class
Argument - associates aspectuals with a relation
Fill - an aspectual refers to some absolute
Equate - two concepts are co-referential
Differ - two concepts are not co-referential

Figure 1: Primitives in KODIAK

concepts as well. For example, if the only types of animal defined in the model were person, mouse, and fish, then when confronted with a description of a gerbil, the system would probably classify it is a mouse.

A *relation* is a concept that holds between two instances of absolutes. For example, given the absolutes talking and person, we could define the talker-of-talking relation to hold between instances of talking and instances of person. Each relation can be thought of as a function of two parameters; the formal parameters are called *aspectuals*. In this case, the aspectuals might be called talker and talkers-action. The definition would say that every talking action must have exactly one talker, who must be a person. However, not every person need be related to a talkers-action, and some may be related to more than one.

The Eight Primitive Associations

The Dominate Association

The primitive association *dominate* is used to define hierarchical relationships between concepts. For example, the assertion (dominate animal elephant) means that elephant inherits all relations and aspectuals from animal, although further assertions may be made to differentiate elephants from other animals. If we had defined a part-of relation to hold between animal and animal-head, then elephant would also have to participate in a part-of relation with an animal-head, but we could further differentiate the relation to require an elephant-head.

The description above defines individual dominate links, but there is more that we can say about groups of dominate links. For example, not only can we can assert that the concepts male-person and female-person are dominated by person, but furthermore we can assert every person must be one or the other, but not both. Together male-person and female-person form an *exhaustive partition* of the concept person. It is a partition because no instance of person can belong to both categories, and it is exhaustive because there are no other possible categories; every person must be one or the other. This information is used by the matcher: if person.1 is a male-person, and person.2 is a female-person, then the matcher can conclude that they cannot refer to the same person. Partitions are implemented with an optional third argument to the dominate relation, as described below.

Another example of an exhaustive partition is old-person and young-person, for some suitable definition of old. Note that there is nothing to be said about matching *overlapping* concepts; concepts that appear in distinct partitions. Given that person.1 is a male-person, and person.2 is a old-person we cannot decide if person.1 and person.2 are the same or different. An example of a non-exhaustive partition would be the various species of animal: dog, cat, elephant, mouse, etc. Here there is always a chance that there is another kind of animal that we have not heard of yet, and so knowing that a given animal is not a dog, cat, or elephant is not enough to prove that it must be a mouse. On the other hand, if we know that animal x is a dog, then we want to be able to conclude that it is not a cat, and make no conclusion about it being, say, a male-animal or female-animal.

We are now in a position to see just how partitions are represented in KODIAK. Two different implementations were tested. In the first implementation, we would have assertions such as these:

```
(dominate person male-person p)
(dominate person female-person p)
(dominate person old-person q)
(dominate person young-person q)
(dominate animal elephant r)

(exhaustive-partition p)
(exhaustive-partition q)
(non-exhaustive-partition r)
```

Here, p, q, and r are *partition indicators*. Partition indicators support two operations. First, they can be tested for equality. The system can determine that male-person and female-person are in the same partition with respect to person, but that old-person is in a different partition. The other operator is partition type: p and q can be declared to be exhaustive partitions, while r is non-exhaustive. The current implementation of KODIAK provides two user interfaces (one textual, the other graphical) to make such declarations easy. As an implementation detail, partition indicators were implemented as integers, rather than symbols, since all that is important is that they be atomic.

While this implementation was reasonable and worked adequately, the introduction of a new data type, partition indicators, tended to complicate code that would otherwise be more straightforward. Therefore, a second implementation was tried which made use of differs links rather than partition indicators. To duplicate the information given above in this implementation, the following assertions would be used:

```
(dominate person male-person)
(dominate person female-person)
(dominate person old-person)
(dominate person young-person)
(dominate animal elephant)

(differs male-person female-person)
(differs old-person young-person)
(dominate animal other-species-of-animal)
(differs elephant other-species-of-animal)
```

Partitions are indicated by enumerating each pair of differing members. Non-exhaustive partitions are indicated by inventing a new absolute, in this case other-species-of-animal as a catch-all. The disadvantage of this approach is that it requires on the order of ^{2} differs links for a partition of n objects, while the previous approach requires only n assertions. In practice, partitions are small, and the second approach seemed to perform just as well, although no formal benchmarks were taken.

For relations and aspectuals, the hierarchical relationships are even more complicated than they are for absolutes. Suppose we would like to represent the fact there is a concept called event that can have any number of participants. Furthermore, an action is a kind of event, where the actor is one of the participants, and a talking is a kind of action where the talker is the actor. There may be other participants in an action besides the actor, but there can be no other actors in a talking besides the talker. Participant dominates actor, which in turn dominates talker, but we say that talker specializes actor, while actor does not specialize participant. In other words, the talker is *the* actor, but the actor is only *a* participant. This is diagrammed below in Figure 2. The (s) attached to a D link indicates specialization, while the lack of an (s) indicates a non-specializing link. Specialization is implemented with the exhaustive/non-exhaustive partitioning mechanism described above. The ability to represent this difference turns out to be important for handling prepositions and other case-relation related problems, as we shall see in chapter 5.

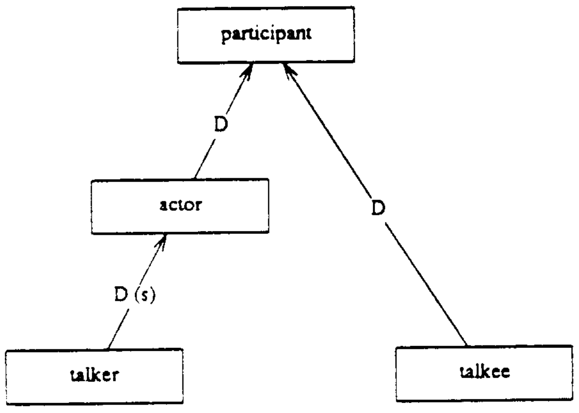

Figure 2: Specializing and Non-Specializing Dominate Links

The Instance Association

The other primitive hierarchical association is *instance*. The assertion (instance elephant Clyde) means that Clyde inherits all relations and aspectuals from elephant, and furthermore all the constraints on the aspectuals of elephant must hold for any of the fillers of the inherited aspectuals of Clyde.

Inheritance through the dominate/instance hierarchy is transitive; every concept that is dominated by elephant is also dominated by animal, and every concept that dominates animal dominates elephant. If Clyde is an instance of elephant, then Clyde is also an instance of every concept that dominates elephant. However, it is not the case that every concept that has animal as an instance also has elephant as an instance, or vice-versa.

It is important to keep the absolute person distinct from the absolutes set-of-all-people and homo-sapiens. We can assert that a person has two legs, that the set-of-all-people has cardinality of about 4 billion, and that homo-sapiens is an instance of species, and is dominated by hominidae. The three concepts are related to each other by primitive associations, but are separate concepts. If we apply the rules for inheritance strictly, it becomes obvious that these must be distinct concepts. Otherwise, if we asserted that homo-sapiens evolved one million years ago, and that Fred is an instance of homo-sapiens, we would be forced to conclude that Fred evolved one million years ago. Other researchers (*e.g.* [10, 39]) have tried to avoid this dilemma by positing more complicated inheritance rules, involving meta-properties that are not inherited, or are inherited differently than other properties. In KODIAK inheritance always works the same way, but we are forced to introduce new concepts for species and set-of-all-people and so on.

The Argument Association

The *argument* primitive association is used to pair aspectuals with relations. Every relation must have exactly two aspectuals, and every aspectual is associated with one unique relation. For example the talker-of-talking relation would have two aspectuals as arguments, which might be called talker and talking-action. An instance of talking might then be related to an instance of person by an instance of the talker-of-talking relation. This would also entail two instances of the aspectuals, and, as we shall see in the next paragraph, two *fill* associations. It should be noted that this formulation of relations and arguments differs in detail from the formulation given by Wilensky in [140, 141].

The Fill Association

The *fill* primitive association holds between an aspectual and an absolute that is used to fill the value of that aspectual. Both the aspectual and the absolutes must be instances; if the aspectual is an instance of aspectual `A` then the absolute must be an instance of each of the constrainers of `A`, as defined by the *constrain* association, which will be covered in the next section. Using the example from the paragraph above, the top diagram in Figure 3 depicts the arguments and constrainers of the `talker-of-talking` relation, and then shows the fillers of an instance of that relation, `talker-of-talking.1`. In the diagrams below (and throughout this thesis), the convention is that aspectuals appear as circles, and absolutes as rectangles. In Figure 3, relations as double circles, but that convention is not adopted throughout. Associations appear as arrows labeled with a single character: D, I, V, C, A, F, =, or ≠. The second and third diagrams in Figure 3 are abbreviations representing the same information as in the top diagram, but in less explicit notations.

The Slot Pseudo-Association

The middle diagram in Figure 3 uses the *slot* notation, labeling a link with an S. The intent is that we can consider `talking` to be a frame with a `talker` slot which must be filled by a `person`. This is just a convenient notation; the actual implementation is in terms of relations, aspectuals, and argument links. The bottom diagram in Figure 3 takes this one step further, placing the aspectual name as the label on the link. These notations will be used freely in diagrams, but they should always be construed as abbreviations for underlying A and F links. The abbreviations have the advantage of making the diagrams less cluttered, but they have the disadvantage of a built in bias: the arrow says that `person.1` is the `talker` of `talking.1`, but it could as easily have said that `talking.1` is a `talking-action` of `person.1`.

The Constrain Association

The *constrain* primitive association holds between an aspectual and an absolute, and asserts that all fillers of instances of the aspectual must be instances of the absolute. If there is more than one constraint put on a single aspectual, then fillers of the aspectual must satisfy all the constraints simultaneously. There is no way to represent a disjunction of constraints directly. For example, there is no direct way to say that the `limbs` of an animal must be filled by either `arms` or `legs`. Instead, we would have to have parallel hierarchies for absolutes and aspectuals, each saying that `arm` and `leg` are dominated by `limb`.

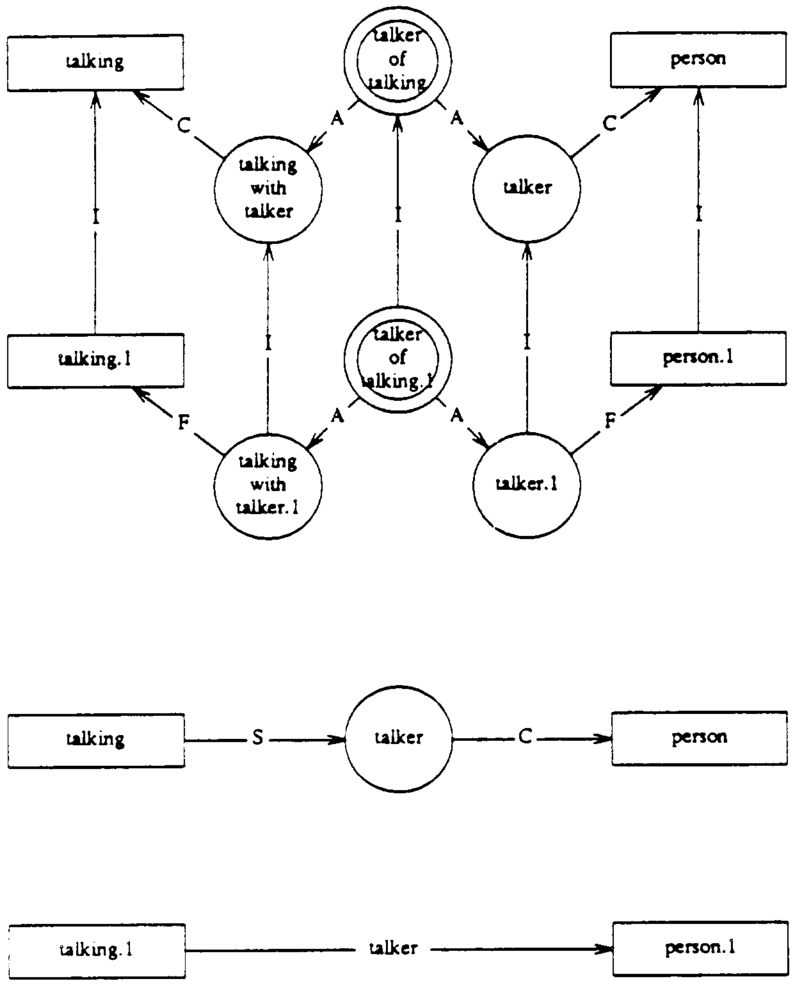

Figure 3: The `talker-of-talking` Relation

Number Restrictions and Quantifiers

By now we see that relations in KODIAK are rather complex entities. Besides the relation itself, there are the arguments, which are aspectuals, and the constrainers, which are absolutes. KODIAK is designed to represent each of these components as a distinct manipulatable object. With this we can describe relations to the point where, if a relation held in a model of the world, we would know what kinds of objects it would have to hold between. What is missing is an indication of which relations are likely or unlikely to hold. For instance, we would like to be able to say that *all* islands must participate in a surrounding relation with a body-of-water, but that only *some* bodies-of-water need surround islands. In KODIAK his is done by adding a *quantifier* to the appropriate constraint link. The allowed quantifiers are *all, most, some,* and *none*. To the matcher, the quantifiers *all* and *none* are significant in ruling in or out a match, while to the marker passing mechanism, the cost of traversing an inverse constrain link is less if the quantifier is *all* or *most* than if it is *some* or *none*. The four quantifiers were chosen because of their use in these two procedures; as far as the current implementation of FAUSTUS is concerned, a more complete quantifier system would be of no additional use. Of course, there could be other applications that depend critically on a better-defined quantifier system. For such applications, extensions could be made.

Number restrictions in KODIAK are handled the same way as in KL-ONE; for each aspectual a minimum and maximum number may be specified. These determine the number of fillers that an instance must have, if they do in fact have any. There is an interaction between quantifiers and number restrictions. For example, we can say that only some `reptiles` have `ears`, but if they do have any, they must have exactly two. Similarly, we can assert that only some bodies-of-water participate in surrounding relations with islands, but those that do can participate in any number of such relations. Number restrictions are inherited through the hierarchy; If aspectual B is dominated by aspectual A, then the maximum number restriction on B must be less than or equal to the maximum number restriction on A, and the minimum must be greater than or equal for specializing aspectuals. For non-specializing aspectuals, this constraint on the minimum number restrictions does not hold.

Some examples should make this clearer. In Figure 2 above, the `talker` aspectual is dominated by `actor`, and the dominate relation is a specializing dominate. This means that a talker is *the* actor of a talking action. There is only 1 actor for an action, and it would be an error for the number restriction on talker to be anything more or less than 1. In contrast, a non-specializing dominate asserts that, for instance, an accompanier is *a* participant in a situation. So it is not an error that the minimum number restriction on accompanier is less than on participant; it is assumed that a situation will have some other kind of participant. Of course, it is possible for two or more people to act together, but in that case we say there is one single `group` that is playing the role of the actor.

```
(number-restrict actor 1 1)
(number-restrict talker 1 1)
(number-restrict participant 1 ∞)
(number-restrict accompanier 0 ∞)
```

As another example, consider the `water-surrounding-island` concept. It requires exactly one surrounded island, and one or more surrounding bodies of water. Thus, we can state that Great Britain is surrounded by the Atlantic and the North Sea, but stating that both Hawaii and Oahu are surrounded by the Pacific would require two different assertions (or one `water-surrounds-archipelago` assertion). Below we have some of the low level calls needed to state these facts; again recall that the KODIAK user interfaces make it unnecessary to work at this level.

```
(quantifier (constrain surrounded-island island) all)
(quantifier (constrain surrounding-water body-of-water) some)
(number-restrict surrounded-island 1 1)
(number-restrict surrounding-water 1 ∞)
```

We also have some flexibility in specifying how an instance manifests a particular aspectual. Consider the problem of dealing with anomalies like three legged elephants. In languages with strict classification, like KL-ONE, we can form the concept `elephant` and the concept `quadruped`, but it would be incorrect to try to state a relation between them, in the KL-ONE formalism itself. To say that most but not all elephants have four legs means going outside the language and using a distinct assertional language.

In KODIAK we have the flexibility to say that most elephants have four legs, but somehow this does not capture completely the relation between elephants and their legs. The number of legs is not a statistical property. Suppose we defined the typical American family as having four children, and asserted that the Jones are an American family with three children. In this case there is no need for an explanation as to why the Jones only have three children, and it does not make any sense to ask *which child is missing?* In contrast, if we assert that Clyde is an elephant with three legs, then we expect an explanation of some sort: perhaps Clyde had a birth defect or an amputation. Also, it makes sense to say *Clyde is missing his left front leg*. Therefore, rather than asserting that most elephants have four legs, in KODIAK we assert that all elephants have four legs, but we describe the way in which animals have legs: that there are specific places on the body where legs are expected, and that they are attached by tendons, ligaments, and skin. We include the fact that there are non-standard ways to ''have'' a leg. One way is to have a missing leg because of a birth defect, and another way is to have a missing leg because of an amputation, just as one way to have a child is to have a deceased child, or an adopted child.

If we came across a new animal that looked just like an elephant except it had an extra pair of legs protruding from its midsection, it would not fit the definition of elephant. We might want to classify it as an elephant, reasoning that it came closer to being an elephant than anything else, but that would be a matter of classification rather than definition.

Of course, some properties are in fact properly represented using the quantifier *most* rather than *all*. For example, it would be a mistake to assert that all birds can fly. Instead we assert that most birds can fly. This gets inherited by all subcategories of bird, but we can still say that a penguin is a bird, and no penguins can fly.

Ideally, the KODIAK knowledge base could update itself automatically to account for inconsistencies. If the modeler originally asserted that all birds can fly, and then added the fact that penguins can not fly, the system could change the quantifier for birds flying from all to most. While such an approach might work for examples like this, in general the problem of learning new concepts is complex; it is not obvious what to change, what variables to generalize over. FAUSTUS is concerned only with the problem of interpreting input text properly, and thus we will not be concerned with the problem of learning new generalizations about the domain. This means that the domain modeler will have to be more careful, and will be expected to do a certain amount of debugging. FAUSTUS checks for certain inconsistancies in the domain model and prints appropriate error messages, but it doesn't make or suggest corrections on its own.

The Equate Association

The *equate* primitive association comes in two kinds; one that holds between two absolutes, and one that holds between two aspectuals. The first kind means that the two absolutes are co-referential. If we had one concept representing Bill's girlfriend, and another for John's sister, and then learned they were one and the same, we would insert an equate link between them. The other use of equate is to assert that two aspectuals must be filled by the same absolute. For example, the concept `suicide` can be defined as a kind of `killing` where the `killer` and the `victim` are equated. (There may be additional facts about `suicide` that further differentiate it from `killing`. For example, killing can be done accidently, while suicide, it seems, can not.) Equate is an equivalence relation; it is symmetric, transitive, and reflexive. The equate association can also be used to model more complex facts about the domain, as we shall see shortly.

The Differ Association

In contrast to equate is the *differ* primitive association. This is used to assert that two absolute instances are explicitly not co-referential, or that two aspectuals cannot be filled by the same instance. Differ is symmetric, but is neither transitive nor reflexive.

Unlike PLANNER, CONNIVER, FRL, PEARL, OPS5, and many other RLs, there is no notion of *variable* in KODIAK. Aspectuals can be used to achieve some of the purposes of variables: they can be defined to be equal or different, and they can have quantifiers associated with them. The scope of every aspectual is universal; for example there is only one node denoting the `actor` of an `action`; to represent the actor of, say, a `driv-ing` action, we would have to create a different node, which we could name `driver`, and which would be dominated by `actor`. Then to represent a particular instance of driving with a particular driver, we would have to create another node, say, `driver.1`, which would be an instance of `driver`. The lack of variables leads to a proliferation of concepts, but it makes it easy to refer to these concepts unambiguously.

At this point, one might ask why the equate and differ links are included as primitives of the system, and why, for instance, a "similar-to" link is not included. The answer is that together the two link types provide the basic capability that other systems derive from the use of *symbols*, and the `eq` and `neq` predicates, to use LISP terminology. In a language like FRL, two frames match if they are in the same branch of the type hierarchy and if all their slot fillers match. Two slot fillers match if they are either frames that match, or are the same symbol. In KODIAK, there is no notion of slots, and no notion of bottoming out at symbols. Two concepts match if they are the same concept or are equated, they fail to match if they differ, and otherwise we have to look at the relations they participate in to see if they match, or if any of their ancestors differ.

The View Association

The final primitive association is called *view*. It is used to interpret one concept as if it were another type of concept. The representation of views is more complex than the other representational primitives, involving both an association and a relation with accompanying aspectuals. Views will be explained below, after the semantics of KODIAK relations have been presented in more detail.

The Relational Regress Problem

The relational regress problem as follows: Anytime we have a relation holding between two absolutes, it is possible to re-model that relation as an absolute in its own right, with two new relations holding between it and the two original absolutes. We need a principled means of deciding when to stop – when to use relations and when to use absolutes. Consider the problem of modeling the situation wherein a person knows a fact. One solution is simply to define a relation, `knows`, that holds between a `person` and a `fact`. We would further assert that a person can know any number of facts, and a fact can be known by any number of people, and we could add more semantics to define `knows` in terms of `believes` and `true`. Another solution is to represent `knowing` as a kind of `stative-situation` where the `be-er` of the `situation` is a `person` who we will call the `knower`, and the `object` is a `fact` we will call the `known`. In other words, we define a new absolute, `knowing`, and two relations involving that absolute. We could go yet another step, first defining the `knowing` situation as an absolute, and then defining `known` as an absolute which is another situation wherein something is known and `knower` as a situation wherein somebody knows. We would then need to define four new relations to mediate between these three new absolutes. Figure 4 depicts the three possible representations graphically, using the abbreviated notation.

The answer to the relational regress problem is suggested by the fact that absolutes can participate in other relations, while relations cannot participate in other relations; they are unqualified. Thus, if we adopted the first representation shown in Figure 4 above, and if we wanted to say that `person.32` knows `fact.37`, it would have to be an unqualified instance of knowing. On the other hand, if we adopted the middle solution, we would be able to represent `person.32` knowing `fact.37` for a period of time, but then forgetting it. This is possible because `stative-situation` has a `duration` relation associated with it. Another important consideration is that it makes sense to have a knowing situation where the knower is unspecified, as in *Someone must know who the murderer is* or where the known fact is unspecified, as in *John knows something*. For

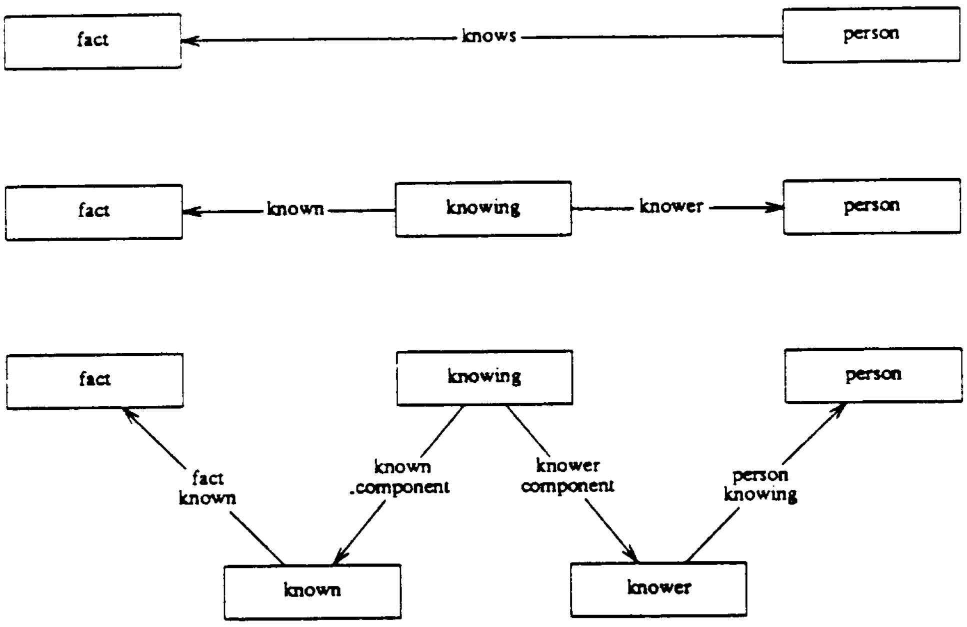

Figure 4: Three Levels of the Relational Regress

these two reasons we see the first representation in Figure 4 is too restrictive, and we need at least the second. The first permits only assertions of the form "there is a situation wherein X knows Y," while the second allows "there is a situation wherein X knows Y, and furthermore this situation held from last Tuesday until the present."

The third possibility from Figure 4 permits some rather bizarre representations. We could use it to assert "there is a single situation wherein X knows Y on Monday and Z knows W on Tuesday." It seems wrong to call something like this a single situation, so it must be that this representation is not restrictive enough. The second representation of knowing correctly captures the intuition that these should be represented as two distinct `knowing` situations. It is through tests like the `duration` test and this multiple instance test that the modeler determines how to represent concepts. The guiding principle is as follows:

> Represent as relations the inalienable intrinsic properties of an absolute, and represent other properties through separate stative absolutes.

Modeling Facts About The Domain

Suppose we wanted to model two simple facts about the action of `giving`:

(1) A precondition of giving is that the giver has the object given.
(2) The result of a giving is that the recipient has the object given.

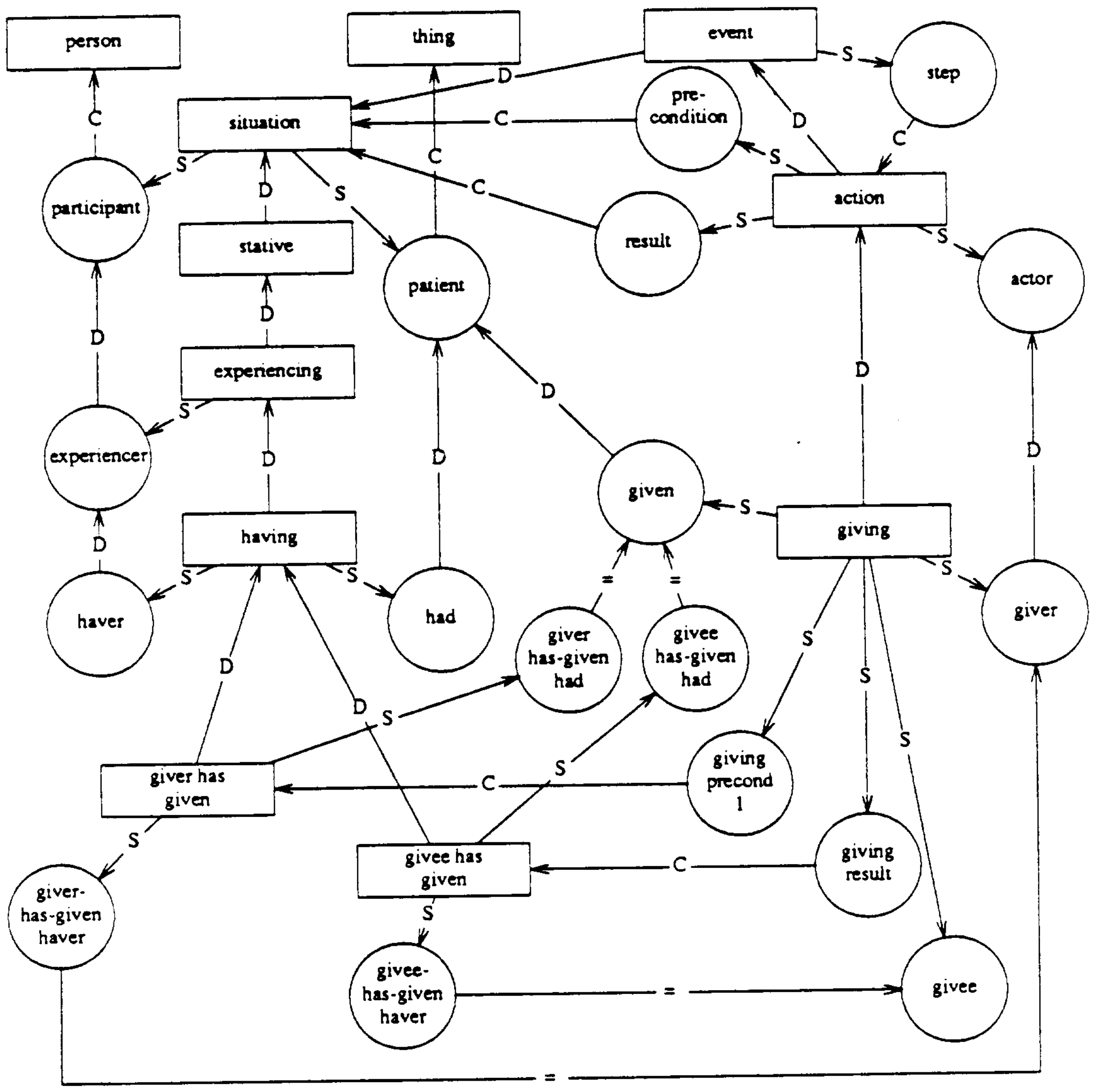

Figure 5: Modeling Facts About the Concept Giving

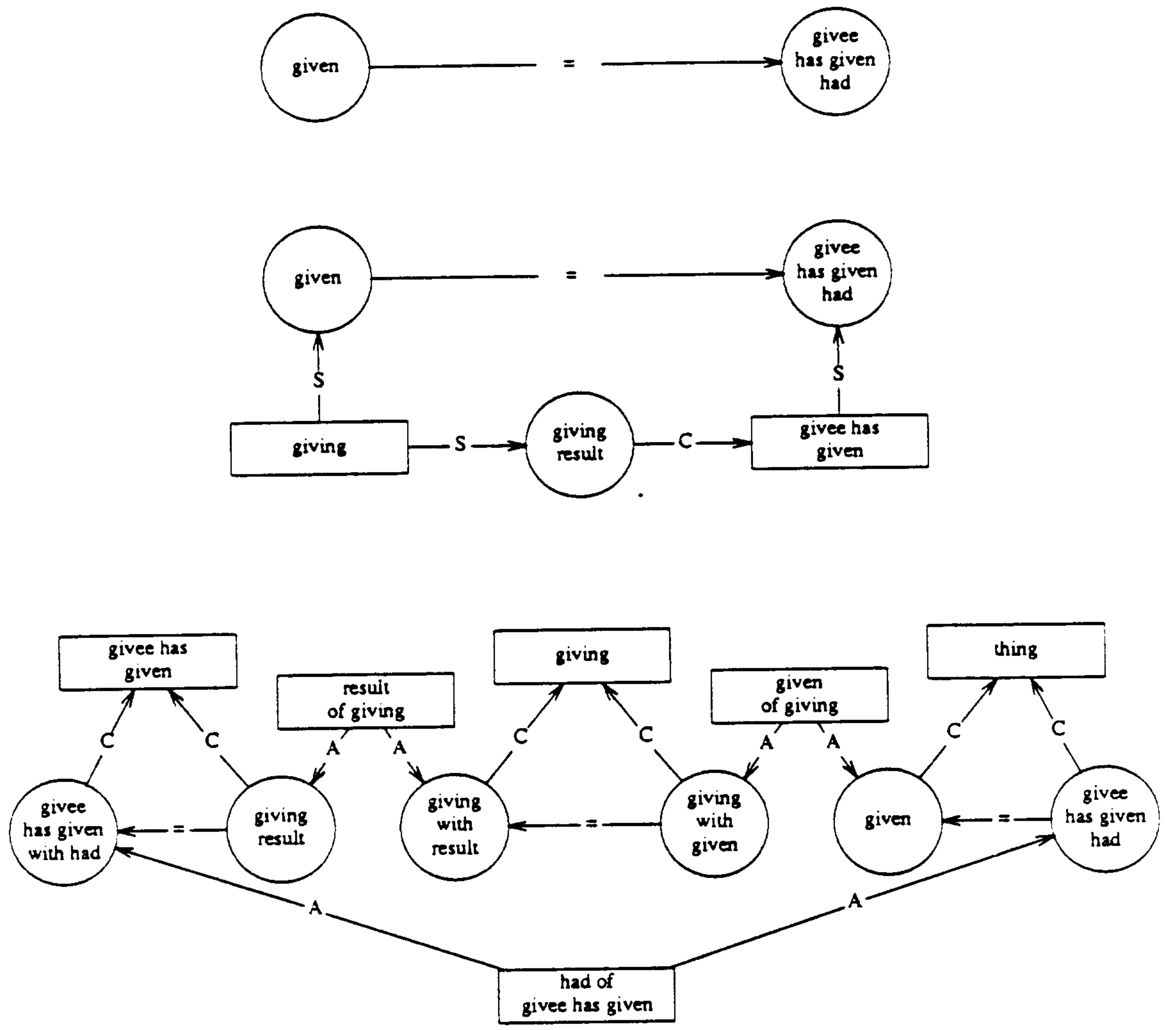

Figure 6: Details of Equate Links

The relation between having and giving is shown in Figure 5. The concept having is defined to be a kind of stative situation, in particular it is a being, where the be-er is called the haver. The be-er is a participant in a situation, and thus must be a person. Besides having participants, situations can also have a patient, and the patient of a having is called the had, and must be a thing. The other kind of situation is an event, and one kind of event is an action. Actions have pre-conditions and a result. They also have an actor, which is a participant (although that link is not shown in the diagram). Giving is an action where the actor is called the giver, the patient is called the given, and there is another participant called the givee. (In the current world model, giving is also dominated by transferring, but that is not shown here.) Giving also has a pre-condition, namely giver-has-given and a result, givee-has-given, which are both kinds of having. However, the result of a particular instance of giving can not be just any instance of a giver-has-given, it has to be one with the right object and the right recipient. We would not want the result of John giving a book to Mary to be that Bill has a pencil. Equate links are used to make the proper description. The equate links assert that the haver of the result of a giving must be the same as the givee of that giving, and the object given must be the same as the object had by the givee. Two similar equate links hold for the precondition.

Since Figure 5 makes use of the abbreviated slot notation, it will be useful to look at the equate links in more detail in Figure 6. One of the facts represented in Figure 5 is that, in a giving event, the thing given is the same as the thing which the recipient (the givee) has as a result of the giving. This is shown as the equate link between given and givee-has-given-had. It is shown as a single equate link in the top of Figure 6. In the middle of Figure 6, we show a portion of the network from Figure 5. This can be read as "the thing given in a giving is the same as the thing had in the result of a giving."

The single equate link in the slot notation is actually just an abbreviation for a more complex structure involving three equate links. This is shown in the bottom of Figure 6. The equate link on the right is the same equate we have seen before. It says that the given object is the same as the object had by the givee. We also need the equate link in the middle stating that this is true only when we are dealing with the given and the result of the same giving, and the one on the left stating that the had and the inverse result must be for the same instance of having, namely the one involving the givee.

There is also a more lisp-like interface for representing concepts. The facts relating to giving can be expressed in the following form:

```
(A GIVING (↑ TRANSFERRING)
  (giver SENTIENT-AGENT (↑ donor actor))
  (givee SENTIENT-AGENT (↑ recipient))
  (given INANIMATE (↑ transferred))
  (giving$result GIVEE-HAS-GIVEN (↑ result))
  (giving$precondition GIVER-HAS-GIVEN (↑ precondition))
  (= given (had giving$result))
  (= given (had giving$precondition)))
```

Here the first line means that giving is dominated by transferring. The second line means that giver is a slot of giving which is constrained to be a sentient-agent. Furthermore, giver is dominated by donor and actor. The next four lines describe more slots, and the last two lines describe the two equate relations.

Representing Input Text

So far we have discussed how to describe facts, that is, concepts from the domain model, which are represented in long-term memory. Some new considerations come into play when we attempt to represent input text in KODIAK. First of all, there is a gross difference in the types of links that are used. Definitions usually make use of dominate, constrain, argument, equate, and differ links, while input text is represented primarily with instance, argument, and fill links.

More importantly, the representations of input sentences produced by the conceptual analyzer are not guaranteed to be well-formed KODIAK forms. Consider sentences (4a-d), and their representations in KODIAK. In each case KODIAK will detect a *constraint violation* in that the representations involve relations where the constraints are not satisfied by the fillers of the aspectuals.

(4a) The man arrested the criminal.
`(arresting (actor ← the man) (patient ← the criminal))`

(4b) The commissioner arrested the criminal.
`(arresting (actor ← the commissioner) (patient ← the criminal))`

(4c) The Red Sox killed the Yankees.
`(killing (actor ← the RedSox) (patient ← the Yankees))`

(4d) The chair laughed.
`(laughing (actor ← the chair))`

First of all, each of these representations purports to attach general slots to specific actions. For instance, (4a) specifies an `actor`. But the correct name for the actor of an arresting is `arrester`, not `actor`. KODIAK allows the representation shown, and reports that the actor is interpreted as the arrester. This is called a *relation classification* inference.

There is another problem with (4a), however. There is a constraint violation in that an `arrester` is constrained to be a `law-enforcement-official`, whereas in this sentence the arrester was specified just as a `man`. In this case, FAUSTUS can satisfy the constraint simply by making the new assertion that the `man` is in fact a `law-enforcement-official` as well as being a man. This is called a *constraint classification* inference. (This assumes a model where citizen's arrest does not exist, or where it has to be marked explicitly, as in *The man performed a citizen's arrest.*)

Compare (4a) with (4b). In (4b), we should make the same inference as in (4a), that the actor is a `law-enforcement-official`. However, in this case we would want to make the additional inference that the `commissioner` is a `police-commissioner`, and not, say, the commissioner of baseball. This is called a *relation concretion* inference. Both (4a) and (4b) involve inferring new instance links, but in (4b) we add a link that is not strictly necessary.

The option of inferring a new instance link is not always open. In (4c) there is a constraint violation because the actor and patient of a `killing` are baseball teams, which are `organizations`, whereas they should be an `animate-agent` and a `living-thing`, respectively. Since `organizations` and `living-things` are mutually exclusive categories in the hierarchy, there is no link that could be added to resolve this violation. Instead, something must be changed, as we shall see in the next section.

Viewing One Concept As Another

One way to correct for the constraint violation in (4c) is to *view* the teams as referring to the players on the teams, thereby interpreting the sentence as if it were *The players on the Red Sox team did something that caused the players on the Yankees team to die*. The other possibility is to reinterpret the relation. That is, we could view the `killing` as an instance of `defeat-convincingly`, thereby interpreting the sentence as *The Red Sox baseball team defeated the Yankees baseball team by a wide margin*. Both these reinterpretations are made possible only by the existence of a *view link*– a representation that it is possible to view one type of object as another. In the first case, there is a metonymic view stating that the name of an organization can be used to stand for the members of the organization. In the second case, a view states that one meaning of the word *kill* is to `defeat-convincingly`.

Figure 7 shows the representation of the view wherein an organization stands for the members of that organization. The V link in the figure indicates that a organization can be viewed as a group of people. This is a `referential-view`, meaning that in this view the organization is used to refer to the group of people. However, it is not the case that any organization can be used to refer to any group of people; rather, an organization refers to the people who are members of that organization.

The notation may seem complex, but actually there is a reason behind every link and node in Figure 7. First, the V link is there to indicate that a view can exist between a `organization` and a `group-of-people`. However, unlike, say, a D link which specifies that every instance of one concept must be an instance of another, the view link is contingent. Thus, we must be able to differentiate between assertions that are true of all `organizations` and assertions that only hold for organizations that are being viewed as people. That is why we need the `organization-viewed-as-people` and `people-via-organization-view` nodes. There is an important relation between these two nodes: the organization that is the source of the view must be the same as the organization that the people are members of. This is represented with the equate link, and an `organization-of` relation (shown here with the abbreviated slot notation).

Not only is the V link contingent, it is also non-unique. There might be several ways in which one concept could be viewed as another. For that reason, we need the arrow from the "V" to a relation that indicates exactly *how* the organization is to be viewed as a people. In this case it is via the `organization-for-people-view`, which is a kind of `referential-view`.

Since views are just a special kind of relation, they can form a hierarchy of their

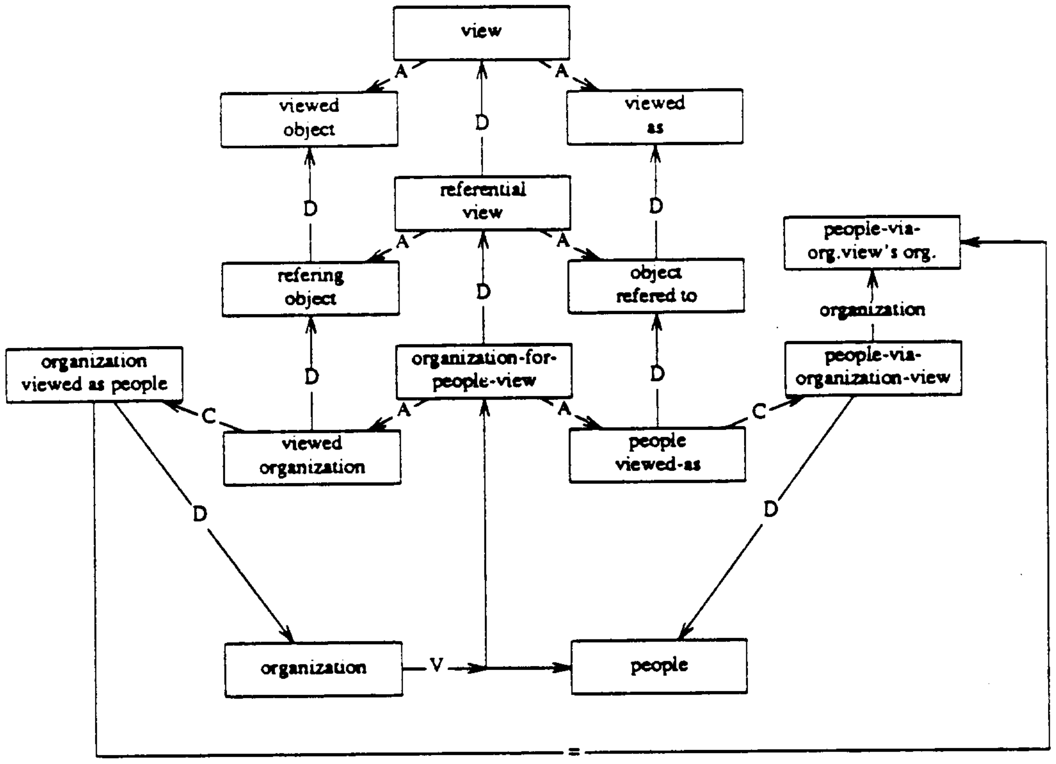

Figure 7: The Organization for People View

own. There are three main types of views: *referential* views, *transferential* views, and *focusing* views. Each will be covered in detail in chapter 5.

In general, views can be chained together; to interpret (5) we first map the `Kremlin` to `soviet-government` by applying the `place-for-organization-view`, and then map `soviet-government` to `soviet-leaders` by applying the `organization-for-people-view`.

(5) The Kremlin took offense at Reagan's latest remarks.

There have been several AI systems that provided support for the idea of viewing one concept as another, or of reinterpreting input to match constraints. The MERLIN system [84] allowed *forced matches* where one concept could be viewed as another, but this was done through a general syntactic matching procedure, rather than by applying stored views. Winston [144] had a similar procedure for understanding similes. Carbonell [20] has done work on analogical reasoning, and has recognized the importance of processing metaphorical language [19, 21]. The KRL language [13] allowed multiple views of an object, but did not distinguish between true ISA relations and views; thus it is not clear how easy it would be to work with the system in practice. KODIAK is the first representation language that I know of to have a design goal the explicit representation of views that prescribe how certain concepts can be interpreted as others.

The question remains of how to choose between multiple applicable views. Chafe [22] makes a claim for the centrality of verbs in determining what a sentence says. He claims that when faced with an anomalous sentence like (4d), *the chair laughed*, "what we do is to interpret chair as if it were abnormally animate, as dictated by the verb. What we do not do is to interpret laugh in an abnormal way as if it were a different kind of activity, performed by inanimate objects." KODIAK has no such notion of centrality of verbs, and could take either one of Chafe's alternatives, depending on what views are available. That is why it is important to represent views explicitly, rather than relying on some general processing mechanism to derive new interpretations on the fly.

Representing Creation Time and Story Time

All objects in the KODIAK knowledge base (including links) have a *creation time* associated with them, which marks the time they were introduced to the system. For objects in the permanent knowledge base, the creation time is zero, and for objects in the representation of the text, or inferred from the text, the creation time is the number of the input sentence that was last processed when the object was created. There are a set of time relations (`before`, `after`, *etc.*) adapted from Allen's time model [1,2] that are semi-primitives in that the KODIAK interpreter can look at the creation-time slot to evaluate these relations. It is possible to specify temporal ordering information explicitly, and a sentence like *After the party, John went home* would be represented with a `after` relation. If no temporal ordering is specified in the input, it is assumed that the order of sentences in the text is the same as the order of events in the underlying story.

Each object also has a *creation status*, which tells if the object was a *fact* in the knowledge base, an *input* from the text, a *necessary inference* or a *possible inference*. Creation status is useful when there is conflicting information: it allows the program to give more weight to a known fact than to a possible inference. Both links and concepts have a creation status.

Representing Semantic Cases

In Chapter 2, I mentioned that there are at least three types of case roles: grammatical case, semantic case, and rhetorical case. KODIAK is well-suited to capture the different types of case relations, because of its ability to represent multiple inheritance between relations. We could represent in KODIAK the fact that the "I" in *I smeared mud on the wall* is both subject and agent, (and perhaps topic as well) and that the agent of a `smear` is the `smearer`. In some frame-based RL's, such distinctions cannot be expressed.

Despite the capability provided by KODIAK, I have chosen to implement only semantic case relations in the knowledge base used by FAUSTUS, for two reasons. First, the conceptual analyzers ("parsers") available to the project cannot produce representations with rhetorical or orientational cases, and by definition an analyzer eliminates grammatical information. Second, as indicated in Chapter 2, some of the interactions between the cases are extremely subtle. I leave the problem of incorporating the different types of cases as an important issue for future research.

The PHRAN conceptual analyzer is capable of interpreting straightforward case relations. For example, in *John gave Mary a book,* PHRAN determines that John is the actor, Mary is the recipient, and the book is the (semantic) object. Given that level of analysis, FAUSTUS then determines that the actor of a giving is the giver, that the actor is also the donor, and similar inferences for the other cases.

Representing semantic case relations by themselves is still a complex task. It is not enough to isolate a small number of cases, such as the actor of an action and the destination of a motion. The problem is that there is information that would get lost if only a few distinguished cases were recognized. For example, in `driving` the actor (the driver), the object (the passengers), and the instrument (the vehicle) all end up at the destination. This is not true for other forms of movement, like `throwing` or `guided-missile-launching`, where the actor does not move. If we did not have detailed knowledge about the case relations, we would make errors in inferencing in instances like this. Of course, it is not important that we use the name `driver`; the name `actor of driving` would do as just as well, although it is a little more verbose. The important thing is that there be a separate concept about which we can assert the necessary information.

Difficulties arise when there are ambiguous case markers, ambiguous attachment points, or vague cases. Consider sentences (6a-d). Each sentence has a similar form, but in (6a) the noun phrase following *with* is an accompanier, in (6b) it is an instrument, in (6c) a manner, and in (6d) a modifier attached to the object, spaghetti, not the action of eating. Because of the nature of the representations passed from PHRAN the problem of attachment cannot be addressed by FAUSTUS. It must receive a representation where `pesto` is attached to either `spaghetti` or `eating`; there can be no representation that is neutral between the two. However, it is possible to claim that the senses of *with* in (6a-c) are polysemous variants of one root `with`, and to make FAUSTUS choose between them. That is in fact what has been done; the representation for (6a-c) each are of the form shown in (6e). FAUSTUS is able to take this representation and arrive at a more specific interpretation of the sentence in each case. The details are covered in Chapter 4 in the section "Concretion Inferences."

(6a) John ate spaghetti with Frank.
(6b) John ate spaghetti with a fork.
(6c) John ate spaghetti with gusto.
(6d) John ate spaghetti with pesto.

(6e) `(eating (actor ← John) (object ← spaghetti)`

```
(with ← ...))
```

Ambiguity, Vagueness, and Polysemy

FAUSTUS starts with a semantic representation of the input text, and makes inferences based on that representation. That makes the representation an important object of study, and makes this chapter necessary. So far we have been acting largely as if the process of translating from text to representation is straightforward and automatic. This is far from true, for a variety of reasons.

There will be some sentences that cannot be satisfactorily handled by this strict pipeline process of syntactic analysis first, inferencing second. In general, the pipeline approach will fail for *ambiguous* input, but will succeed for *vague* input. A good introduction to the difference between ambiguity and vagueness is Zwicky and Sadock's *Ambiguity Tests and How to Fail Them* [150] (although their definition of ambiguity includes some examples that would be considered vague in my system). For example, consider the following sentences:

(7a) They saw her duck.
(7b) Her duck was seen by them.
(7c) They saw two ducks.
(7d) When Mary was in the hospital, John took her flowers.
(7e) When Mary was in the hospital, flowers were taken to her by John.
(7f) When Mary was in the hospital, her flowers were taken by John.

In sentence (7a) *her* can be either an objective or genitive pronoun, and *duck* can be a verb or a noun. One test for ambiguity involves applying the passive transformation, yielding (7b). The fact that (7b) has only one interpretation means that (7a) was ambiguous, and not just vague. Another test involves conjunction; the fact that (7c) cannot refer to one water fowl and one ducking movement also supports the analysis that (7a) is ambiguous.

Another example is (7d), which can undergo two different transformations to arrive at (7e,f), each of which has a different interpretation. This means that (7d) is ambiguous too. (7a,b) are from [150], while (7d) is adapted from Burns and Allen.

Now suppose we accept the interpretation of (7a) where *duck* refers to a water fowl. Then the sentence is still vague, in that it does not specify if the duck is male or female, large or small, alive or cooked. (8a) is the representation for *her duck* under this interpretation; it is unspecified with respect to gender, size, and other characteristics. (8b) is the representation for the other interpretation of *her duck*; there is no representation that PHRAN can return that is unspecified between the two of them.

(8a) `(a water-fowl (possessor ← a female))`
(8b) `(a downward-movement (actor ← a female))`

I have also made allowances to treat certain *polysemous* words as if they were vague, rather than ambiguous. Polysemes are words with many related senses, as opposed to *homonyms*, which are words with unrelated senses. For example, *to have* is a highly polysemous verb, as indicated by the small sampling of senses in (9) below. The fact that (9e) is unacceptable indicates that (9a-d) use different senses of the word *had*, not just one vague sense. Nevertheless, we would like to be able to make inferences about sentences like these without requiring the parser to do all the work. This is done by introducing the notion of an *abstract polysemous concept*. These are objects that can be returned by the parser as valid representations, but which are marked as requiring *concretion* to a more specific sense. That is, the inferencing mechanism is required to choose a particular sense at some later time.

(9a) Gayle had dinner.
(9b) Gayle had a Fiat.
(9c) Gayle had a hard time.
(9d) Gayle had a baby.
(9e) *Gayle had dinner, a Fiat, a hard time and a baby.

Note that even after we have concreted an abstract polysemous concept to a particular sense, there can still be vagueness in that sense. In (9b) we do not know if Gayle owned the Fiat, was leasing it, or merely had permission to use it. It would be reasonable to answer *Did she own or rent the car?* with *I do not know*, but it would not be reasonable to answer *Did she eat, give birth to, or own the car?* with *I do not know*. Abstract polysemous concepts capture the difference between these two cases, and force the system to commit itself to an answer to the later question. At the same time, the system can take advantage of the commonality between the various senses of *having*: each sense takes an actor and an object, and denotes some relation between them.

This same type of ambiguity can show up without an explicitly ambiguous word in the sentence. For example the phrase *John's girlfriend* has many of the same properties as *John has a girlfriend*, and would be handled the same way in FAUSTUS. Jacobs [58] and Wilensky [141] suggest that this can be handled by a "relation as possession" view. This has not been completely worked out in my version of KODIAK.

Ambiguity also shows up as a problem in *attachment* of a phrase to the rest of the sentence. In (10) the phrase *with a telescope* can be attached to the hill, the man, or the seeing. When PHRAN encounters ambiguities like this, it arbitrarily chooses one interpretation. In some cases I have been forced to add *ad hoc* semantic/linguistic patterns to PHRAN to help it resolve ambiguous sentences correctly. For still other sentences, I have bypassed the parser completely, coding by hand my best guess as to what the semantic analysis would have been, had PHRAN been able to make the correct choice.

(10) I saw the man on the hill with a telescope.

Summary Of KODIAK Features

The components of the KODIAK language are objects, which are divided into concepts and links. There are three primitive concept types, absolute, aspectual and relation, and eight primitive link types: argument, fill, dominate, instance, view, equate, differ, and constrain. Representations are formed by connecting concepts with links. However, in addition to these basic eleven primitives, there are several additional features in KODIAK, enough to make it worthwhile to list them all here. First, each object has a *creation time* associated with it, saying when it entered the knowledge base, and a *creation status* stating if it is a given fact, an input, or an inference.

Aspectuals have *number restrictions* which tell how many times an instance of a class of absolute can enter into a particular relation. Aspectuals can also have *quantifiers* which tell how common it is for instances of a class of absolute to enter into a particular relation.

In an earlier implementation, every dominate link had a *partition indicator* associated with it; other links with the same indicator determined the set of concepts in a given partition. Each partition indicator was marked as being exhaustive or non-exhaustive. In the current implementation, this feature is not supported; the same effect is achieved using *differs* links.

View links are more complicated than other links in that they are associated with a relation (which must be dominated by `view`). Thus, views are in a sense three-place predicates, while all other links are two-place.

Finally, there are three additional ways of storing information with concepts. This information has no effect on the meaning of the representations, but is used by the marker passing algorithm, and by the input/output interfaces to the knowledge base. First, each concept has a list of *markers* associated with it. A marker is itself a fairly complex data structure; each marker has a pointer to the concept it marks, the previous concept it came from and the link it traveled along to get there. Markers have no effect on the interpretation of a concepts meaning, but they are used by the FAUSTUS text interpretation algorithm, as discussed in chapter 4.

Second, every concept has a *name*, which exists solely for the user's convenience, and has no meaning to KODIAK.

Third, every object (including links) has an association list or a-list on which it can store arbitrary information. This is used in a variety of ways. Certain concepts are marked as being *promiscuous*, meaning they are connected to many other concepts. This idea and terminology is due to Charniak [26]. Promiscuity is discussed in chapter 4. Each object also has a creation time associated with it. A creation time of 0 means the object is part of the initial knowledge base; a creation time of 1 means the object was created due to the first input, and so on. Since most objects have creation time 0, it would be somewhat wasteful to allocate space to store this information in every object.

Instead, the program arranges that 0 is the default, and stores positive creation times in the a-list. Similarly, the object's *status* is defined to be ''given'' if its creation time is 0, ''input'' if its creation time is positive, and ''inferred'' if there is an entry in the a-list stating it was derived from an inference. Finally, the a-list is used to store information on which suggestions are relevant to a concept; e.g. which suggestions have referents for a given reference, or elaborations for a given concept.

Chapter 4:
The Inferencing Algorithm

In Chapter 1 we saw that the task of understanding a text can be broken down into two components: making proper inferences, and avoiding improper ones. In Chapter 2 we saw that top-down algorithms, such as DeJong's text-skimmer FRUMP, tended to miss proper inferences when the input did not correspond to what was expected. In contrast, bottom-up inferencing schemes, such as Rieger's, tended to generate too many inferences, many of them irrelevant.

In this Chapter I will present the inferencing algorithm used in the FAUSTUS system, and show how it balances the conflict between these two tendencies. Another problem brought out in many of the programs presented in Chapter 2 is that they tie the processing algorithm to one particular knowledge structure, and therefore make it difficult to modify or extend the system. The FAUSTUS algorithm is designed to address this problem as well. The goal of the algorithm can be stated simply as follows:

> Make those and only those inferences that serve as a connection tying together two concepts, such that the connection is a simple one, and is the most plausible such connection.

In this definition the conflict between too few top-down inferences and too many bottom-up inferences has been restated as a conflict between simplicity, connectivity and plausibility. The resolution of the conflict is to use marker passing to spread out from the input in a bottom-up fashion, but to only make inferences based on marker collisions, thus regaining some of the guidance provided by top-down algorithms.

In simplest terms, the algorithm is to read the input, translate it into a conceptual network representation, and pass markers to neighboring concepts in the network. When two markers reach the same concept, suggest an inference based on the path the markers took to get there. After passing markers and suggesting inferences, evaluate each suggested inference and accept each inference unless there is a reason to reject it.

An Example

To make things less abstract, I use as an example a simple three-line text, presented as (1). This example will show some of the marker path shapes, collisions types, and inferences involved in processing the text.

(1a) Bill had a bicycle.
(1b) John wanted it.
(1c) He gave it to him.

Understanding text (1) means making inferences like the following:

(2a) The word *it* in (1b) refers to the `bicycle`.
(2b) The word *he* in (1c) refers to `Bill`.
(2c) The word *it* in (1c) refers to the `bicycle`.
(2d) The word *him* in (1c) refers to `John`.
(2e) Bill having the bicycle is a precondition for giving it.
(2f) The giving results in John having the bicycle.
(2g) This new having satisfies John's goal of wanting the bicycle.
(2h) John wanting the bicycle was the reason for Bill giving it.

Different readers might have slightly different interpretations; the point of (2a-h) is that they give a likely interpretation and indicate the range and depth of the inferences that should be made.

The trace of FAUSTUS' processing of this text follows. In this trace, an option to the program has been set to increase the verbosity of the output. After possibly setting options, the user calls the function `do-story`, which pops up a menu from which the user chooses the story to be processed. The interaction with the menu is not shown in the trace. The actual computer output is in `typewriter font`; commentary is in regular roman font.

```
> (do-story)

          Bill's Bicycle

[ 1] Bill had a bicycle.

Rep: (HAVING (EXPERIENCER ← BILL) (PATIENT ← A BICYCLE))
```

After the title of the text, "`[ 1]`" labels the first line of the English text as it is presented to the conceptual analyzer, and "`Rep:`" labels the resulting representation as it is presented to FAUSTUS.

```
Inferring: the EXPERIENCER of the HAVING must be the HAVER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the HAVING must be the HAD
   This is a RELATION-CLASSIFICATION inference.

Int: (HAVING.1 (↑ HAVING) (haver ← PERSON.1) (had ← BICYCLE.1))
```

Above we see two non-marker-passing inferences made as during the process of turning the analyzer's representation into a KODIAK network. The resulting internal KODIAK representation is labeled with "`Int:`". There is a straightforward one-to-one correspondence between the ''Rep:'' and the ''Int:'' – the difference is that the later refers to individual concepts, and has been made more specific by relation classification and relation constraint inferences. The meaning of these terms will be covered later in this chapter.

At this point, markers are passed from the concepts in the internal representation to neighboring concepts. The blank space following the ''Passing Markers and Suggesting Inferences:'' below indicates that no inferences were suggested. When marker passing is complete, some summary statistics are printed. Then, since there are no suggestions to evaluate, we move on to the second input sentence.

```
Passing Markers and Suggesting Inferences:

Concepts marked: 195 new, 195 total
Collisions: 91 new, 91 total
Suggested inferences: 0 new, 0 total
Accepted inferences: 0 new, 0 total

[ 2] John wanted it.

Rep: (WANT-TO-HAVE (EXPERIENCER ← JOHN) (PATIENT ← IT))

Inferring: the EXPERIENCER of the WANT-TO-HAVE must be the WANTER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the WANT-TO-HAVE must be the THING-WANTED
   This is a RELATION-CLASSIFICATION inference.

Int: (WANT-TO-HAVE.2 (↑ WANT-TO-HAVE) (want-to-have$wanter ← PERSON.2)
                     (thing-wanted ← INANIMATE.2))
```

Again there are two inferences made in the course of going from the analyzer's Rep to FAUSTUS's internal (Int: ...) form. At this point we are ready to pass markers:

```
Passing Markers and Suggesting Inferences:  1 2
```

Each time there is a marker collision that leads to a suggested inference, FAUSTUS numbers the suggestion, prints the number, and places the suggestion on an agenda. We see above that inferences number 1 and 2 have been suggested. The next step is to evaluate the suggestions:

```
Evaluating Suggestions:

Rejecting: the HAVING is a OUTCOME of the WANT-TO-HAVE.
   because of a mis-match.
   This is a SINGLE-ELABORATION inference.
   It is #1, due to the collision:
```

```
   WANT-TO-HAVE.2→elaboration→STATIVE←ref←HAVING.1

Inferring: 'IT' refers to the BICYCLE.
   This is a REFERENCE inference.
   It is #2, due to the collision:
   BICYCLE.1→ref→INANIMATE←ref←INANIMATE.2
```

Wanting to have something can lead to having it, and the marker passing mechanism found a connection which suggests that Bill's having the bicycle is an outcome of John's wanting it. When it comes time to evaluate this suggestion, however, FAUSTUS checks it more carefully and decides there is a mis-match: the outcome should be a having with John as the haver, not Bill. Therefore, the suggestion is rejected.

The other suggestion was triggered by a collision at the concept INANIMATE. A bicycle is a kind of inanimate object, and the representation for 'it' is also an inanimate. Since 'it' needs a referent, this type of collision suggests that 'it' refers to the bicycle. When we evaluate this suggestion, there is no evidence to contradict it, and no competing referents for 'it', so the suggestion is accepted.

Now we print the statistics for the second sentence and move on to the third:

```
Concepts marked: 25 new, 220 total
Collisions: 108 new, 199 total
Suggested inferences: 2 new, 2 total
Accepted inferences: 1 new, 1 total

[ 3] He gave it to him.

Rep: (GIVING (ACTOR ← HE) (PATIENT ← IT) (RECIPIENT ← HIM))

Inferring: the ACTOR of the GIVING must be the GIVER
   This is a RELATION-CLASSIFICATION inference.

Inferring: 'HE' must be a SENTIENT-AGENT, because it is the GIVER
   This is a RELATION-CONSTRAINT inference.

Inferring: the PATIENT of the GIVING must be the GIVEN
   This is a RELATION-CLASSIFICATION inference.

Inferring: the RECIPIENT of the GIVING must be the GIVEE
   This is a RELATION-CLASSIFICATION inference.

Int: (GIVING.3 (↑ GIVING) (giver ← MALE.3) (given ← INANIMATE.3)
      (givee ← MALE.3B))
```

In this sentence there are three relations, actor, patient, and recipient. Each gets classified as a more specific relation, namely, giver, given, and givee. Again, these are called relation classification inferences. We also have here an instance of the other kind of non-marker-passing inference, the relation constraint inference. The idea is that anything that plays the role of a giver must be a sentient-agent – a person or some kind of agency or organization acting as a person. In the input, the giver is identified only as 'he' – a male animal, but not necessarily a person. FAUSTUS makes the inference that 'he' does in fact refer to a sentient-agent (and therefore, a person).

```
Passing Markers and Suggesting Inferences:  3 4 5 6 7 8 9 10 11
```

Marker passing results in nine collisions that spawn suggestions. These suggestions are numbered 3-11. The next step is to evaluate them, along with any suggestions that may be remaining from the previous iterations. In this case, it turns out that the previous suggestions were all either accepted or rejected, but in general some suggestions can be deferred and tried again.

```
Evaluating Suggestions:

Inferring: the 'IT' mentioned in [3] refers to the BICYCLE.
   This is a REFERENCE inference.
   It is #5, due to the collision:
   BICYCLE.1→ref→INANIMATE←ref←INANIMATE.3

Inferring: there is a HAVING such that
   it is the RESULT of the GIVING and
   the WANT-TO-HAVE is the SATISFIES of it.
   This is a DOUBLE-ELABORATION inference.
   It is #8, due to the collision:
   GIVING.3→elaboration→HAVING←elaboration←WANT-TO-HAVE.1
```

Suggestion #5 is straightforward; since only one inanimate object, the bicycle, has been mentioned, it is the only candidate for the referent of 'it.' FAUSTUS therefore accepts the suggestion that they are co-referential. Suggestion #8 is a double-elaboration collision with origins at the want-to-have in sentence 2 and the giving in sentence 3. The suggestion is that there is a new 'having' situation wherein Bill has the bicycle, and this is the result of the giving and satisfies Bill's goal of wanting-to-have it.

```
Inferring: the WANT-TO-HAVE is the REASON of the GIVING.
   This is a SINGLE-ELABORATION inference.
   It is #9, due to the collision:
   GIVING.3→elaboration→STATIVE←ref←WANT-TO-HAVE.2

Inferring: the HAVING mentioned in [1] is a PRECONDITION of the GIVING.
   This is a SINGLE-ELABORATION inference.
   It is #10, due to the collision:
   GIVING.3→elaboration→STATIVE←ref←HAVING.1

Rejecting: the HAVING mentioned in [1] is a RESULT of the GIVING.
```

```
   because of a mis-match.
   This is a SINGLE-ELABORATION inference.
   It is #11, due to the collision:
   GIVING.3→elaboration→STATIVE←ref←HAVING.1
```

Suggestions #9 - #11 are all single elaboration inferences related to the giving action. These are due to marker collisions along the paths representing the following three facts: that somebody wanting something can be a reson for giving it to them; that having something is a necessary precondition of giving it; and that giving something results in someone else having it. The first two of these are accepted, thereby forming connections between the third sentence and the first two. Suggestion #11 is that Bill having the bicycle is the result of his giving it. The evaluation routine notices a mis-match in that the result of the giving was already filled (by inference #8) with an instance of having where John has the bicycle. Therefore, suggestion #11 is rejected.

```
Inferring: 'HIM' refers to John.
   This is a REFERENCE inference.
   It is #3, due to the collision:
   PERSON.2→ref→SENTIENT-AGENT←ref←MALE.3B

Rejecting: 'HIM' refers to Bill.
   This is a REFERENCE inference.
   It is #4, due to the collision:
   PERSON.1→ref→SENTIENT-AGENT←ref←MALE.3B

Rejecting: 'HE' refers to John.
   This is a REFERENCE inference.
   It is #6, due to the collision:
   PERSON.2→ref→SENTIENT-AGENT←ref←MALE.3

Inferring: 'HE' refers to Bill.
   This is a REFERENCE inference.
   It is #7, due to the collision:
   PERSON.1→ref→SENTIENT-AGENT←ref←MALE.3
```

There are two reasons why a suggestion can be rejected. The first is a mis-match between two objects, as in #11 above. The other reason is that there are several mutually exclusive suggestions, of which only one can be accepted. It may be that none of the suggestions involves mis-matches, but we still want to be able to choose the "best" alternative. As examples of this, both John and Bill have been suggested as possible referents of the pronoun 'him' in sentence 3. They are also both possible referents of the pronoun 'he'. At the time these suggestions were made, there was no reason to prefer one over the other. FAUSTUS tries to delay making a decision until more information is available, so it makes two passes over the suggestions, taking care of these suggestions in the second pass. That is the reason why #3 comes after #11 above. In this particular case, the wait was worthwhile, because inferences #8 and #11 have added the information necessary to choose between the referents in each case. Inference #11 said that Bill having the bicycle was a precondition of the giving. But for this particular kind of precondition for giving, the knowledge base states that the haver of the having must be the same as the giver of the giving. The haver is Bill, so when the suggestions are evaluated, the matcher quickly rules out John in #6 and accepts Bill in #7. Similarly, inference #8 says that the result of the giving is a state where John is the haver of the bicycle, and that the haver and the givee are the same. Therefore, the matcher accepts John in #3 and rejects Bill in #4.

Note that it is a fact about English that the referents for 'he' and 'him' in sentence 3 are necessarily distinct, and would necessarily be co-referential if 'himself' were used instead of 'him.' FAUSTUS does not make use of this information. Other programs, such as Wilensky's PAM and Alterman's NEXUS also handled examples like this without making use of this information.

After evaluating all the suggestions, the program prints the following statistics and stops.

```
Concepts marked: 59 new, 279 total
Collisions: 34 new, 233 total
Suggested inferences: 9 new, 11 total
Accepted inferences: 6 new, 7 total
```

Now that we have seen an example of the algorithm in action, we can define it more precisely. The algorithm is a six step process:

Step 0: Construct a knowledge base with good conceptual coverage of the subject matter that the text is likely to cover. The knowledge base is in the form of a semantic network, in the KODIAK formalism, as discussed in Chapter 2.

Step 1: Construct a semantic representation of the next piece of the text. The PHRAN conceptual analyzer is used for this, when possible. For some texts, PHRAN was unable to handle the input sentences correctly, and a representation was constructed by hand. Check the relations in the semantic representation for constraint violations, and make classifications or concretions as necessary to resolve the conflicts.

Step 2: Pass markers from each concept in the semantic representation of the input text to adjacent nodes, following along links in the semantic net. Each of the eight primitive link types in KODIAK has a cost associated with traversing it. Markers start off with a constant *energy value*, and are propagated recursively until the total cost of traversing links exceeds the given energy value.

Step 3: Suggest Inferences based on marker collisions. When two or more markers are passed to the same concept, a *marker collision* is said to have occurred. Collisions suggest possible inferences, but the type of inference suggested is dependent on the *shape* of the marker *path* (and is independent of the marker energy). Each marker keeps track of where it came from, so it is possible to trace backwards from any marker to its origin. This trace is called the marker's *path.* The *shape* of the path is the sequence of link types traversed along the path. Suggested inferences are kept in a list called the *agenda*, rather than being evaluated immediately.

Step 4: Evaluate potential inferences on the agenda. The result can be either making the inference, rejecting it, or deferring the decision by keeping the potential inference on the agenda. If there is explicit contradictory evidence, an inference can be rejected immediately. If there are more than one potential inferences competing for the same actual inference, as when there are several possible referents for a pronoun, then if none of them is more plausible than the others, the decision can be deferred. If there is no reason to reject or defer, then the inference is accepted, and new concepts are added to the model of the text. Note that the checks for plausibility can involve a complex pattern-matching procedure. The details are given below in the section on Step 4 below.

Step 5: Repeat steps 1-4 for each piece of the text.

Step 6: At the end of the text there may be some potential inferences remaining on the agenda. Evaluate them to see if they lead to any more inferences.

One way to categorize and evaluate the FAUSTUS system is as a theory of inference. Chapter 2 presents three components that such a theory must contain. It must have a classification of the types of inferences handled by the theory, a control structure for determining which inferences will be considered, and an evaluation metric for resolving conflicts and for deciding which inferences will actually be made. This last component is important for situations in which we have to choose between two possible inferences, such that accepting either one would contradict the other. Examples of this are when there are several potential referents for a pronoun, or several fillers for a case slot. It turns out that it is much easier to make the most plausible choice by looking at the set of potential inferences all at once than by looking at them one at a time. It also turns out that sometimes the best referent or filler does not come along until some time after the reference or case slot is first mentioned. For these reasons, we need the complication of queuing potential inferences on the agenda, rather than evaluating them as soon as they are suggested.

In the following sections, I will discuss how FAUSTUS handles these three components. This will be done in the context of a specific example text, presented in the next section.

Another way to evaluate the FAUSTUS system is to see how well the inferences generated by the algorithm satisfy the four criteria for proper inferences set out in Chapter 1. Recall that the first, implicit criterion was non-explicitness, and the three main criteria were relevance, easiness, and plausibility.

First, the algorithm avoids making inferences that were explicitly mentioned in the text because the marker passing algorithm has a simple rule that prohibits traversing a link that is part of the representation of the input. Thus, although markers start at the set of concepts representing the input, there is no intramural marker passing in this set; every marker starts its path by going up an instance link to the category above.

Second, the inferences are guaranteed to be relevant to at least two concepts in the model of the text, because inferences are only triggered when collisions occur. In other words, I am *defining* relevance by saying that an inference is relevant if and only if it is related (by marker paths) to two or more concepts in the model of the text.

The third criterion is easiness. This is guaranteed by making the initial energy value on markers small in comparison with the cost of traversing links. Thus, only short paths will be generated before a marker path runs out of energy. The degree to which the length of a path corresponds to a human reader's intuitive judgement of the ease with which a inference can be made can be argued, but at least there is an explicit definition of easiness, which guarantees an upper bound on the amount of computing time required.

The final criterion is plausibility. Every inference that has been suggested due to a marker collision will be accepted, unless there is specific evidence contradicting it. Contradictions are checked by a unification-like matching routine and can involve things like mis-matching types on constraints or too many fillers for an aspectual. The matcher is discussed in the section on Step 4 below. Thus we see that the FAUSTUS inferencing algorithm guarantees that all inferences will be implicit inferences, not known facts, and that they will be easy, relevant, and plausible, at least according to the definitions of these terms laid out above.

Step 0: Representing the Knowledge Base

To make these inferences requires some knowledge about bicycles, people, having, wanting, giving, and so forth. Representing this knowledge is step 0 in the algorithm. The representation of these concepts is in no way dependent on the text of story (1). In fact, when it came time to make FAUSTUS process this text, the only information that had to be added was the fact that a bicycle is a vehicle; all the other knowledge had already been defined for use in other texts.

I show a pictorial representation of a section of the knowledge network in Figures 1 and 2. The diagram follows the graphical notation for the KODIAK representation language described in Chapter 3, and can be paraphrased in English as follows. First, for Figure 1: In the lower half we see `wanting` is a kind of `stative` situation, which has a `wanter` and a `wanted-state`. One kind of `wanting` is `wanting-to-have`, where the `wanted-state` is constrained to be a particular kind of `having`, namely `wanter-has-thing`, which requires that the `haver` of the `having` must be the same person as the `wanter` of the `wanting` and the thing `had` in the `having` must be the same as the `wanted-thing` of the `wanting-to-have`. In the upper left, `Bill` and `John` are `names`, and there is a mapping from `names` to the `people` they refer to. The upper right shows that a `bicycle` is a kind of `vehicle`, and a `vehicle` is a `artifact` and also a `functional-object` whose `purpose` is `traveling`. Every `functional-object` has a `purpose`, which must be an `action` of some kind. Finally, in the lower right we see that a certain kind of `event` can have a `result`, which is a kind of `stative`. In addition,

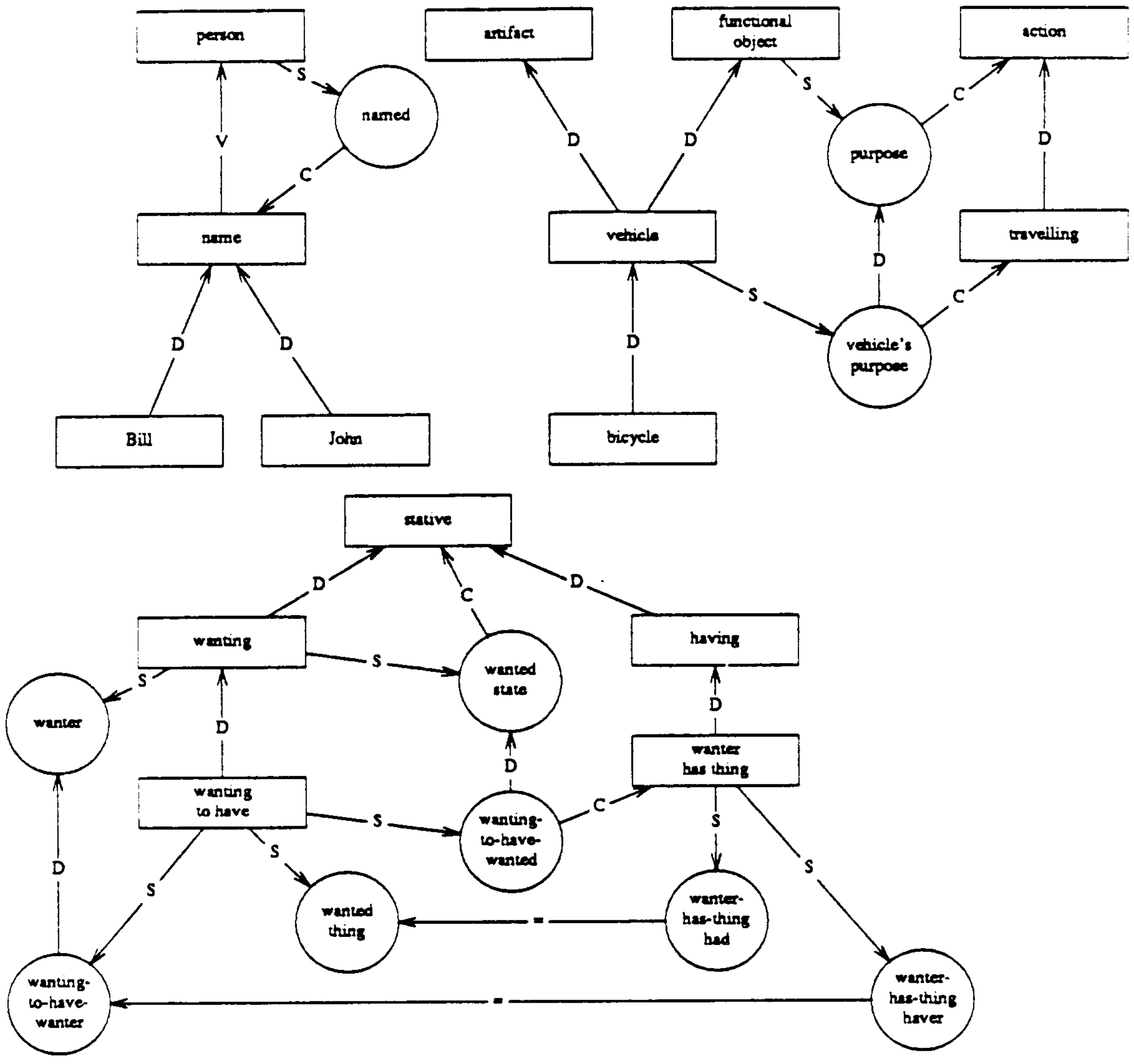

Figure 1: Domain Knowledge: Person, Bicycle, Result, and Wanting

there is something called a `cause`, and one particular type of cause can hold between `events` and the `results` of those events. There is also a particular `before` relation, called `cause-before-effect` which holds between the `cause` and the `effect`. `Before` has special semantics to the KODIAK interpreter in that it can only hold if the time of the `before` part is less than the time of the `after` part. The time of a state or event can be determined either by assertions that come explicitly in the input text, or by the default assumption that the order of presentation of the text is the same as the order of events in the world. In other words, when KODIAK is trying to determine if *A* occurred before *B*, it checks first to see if they participate in any explicit `before` or `after` relation, and if they do not, it goes on to look at the input times associated with *A* and *B*.

Figure 2 describes the relationship between `giving` and `having`. It also appears as Figure 5 in Chapter 2, and is described in detail there.

Figure 3 describes two complex events related to `giving`, namely `gift-giving`

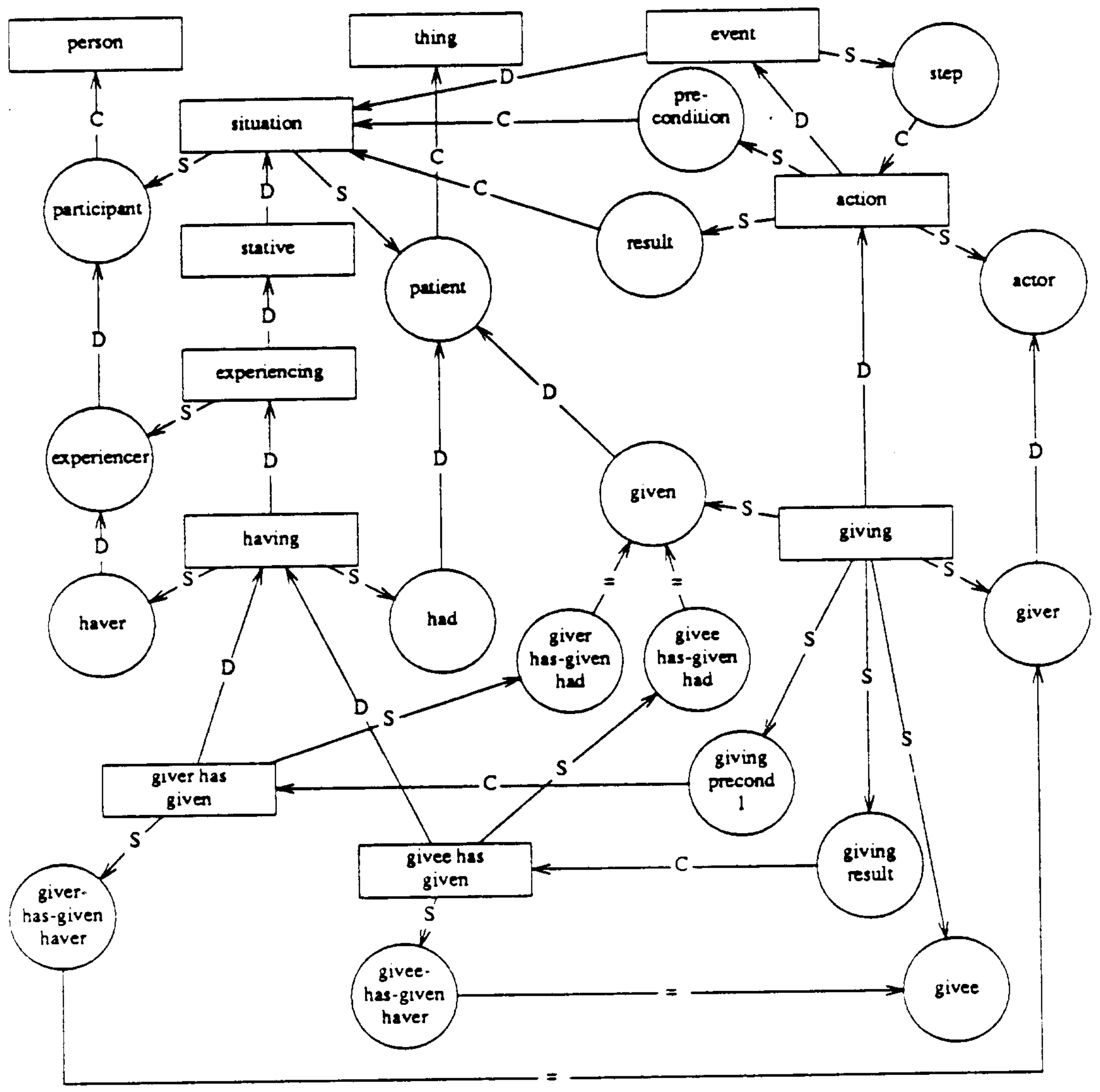

Figure 2: Domain Knowledge: Having and Giving

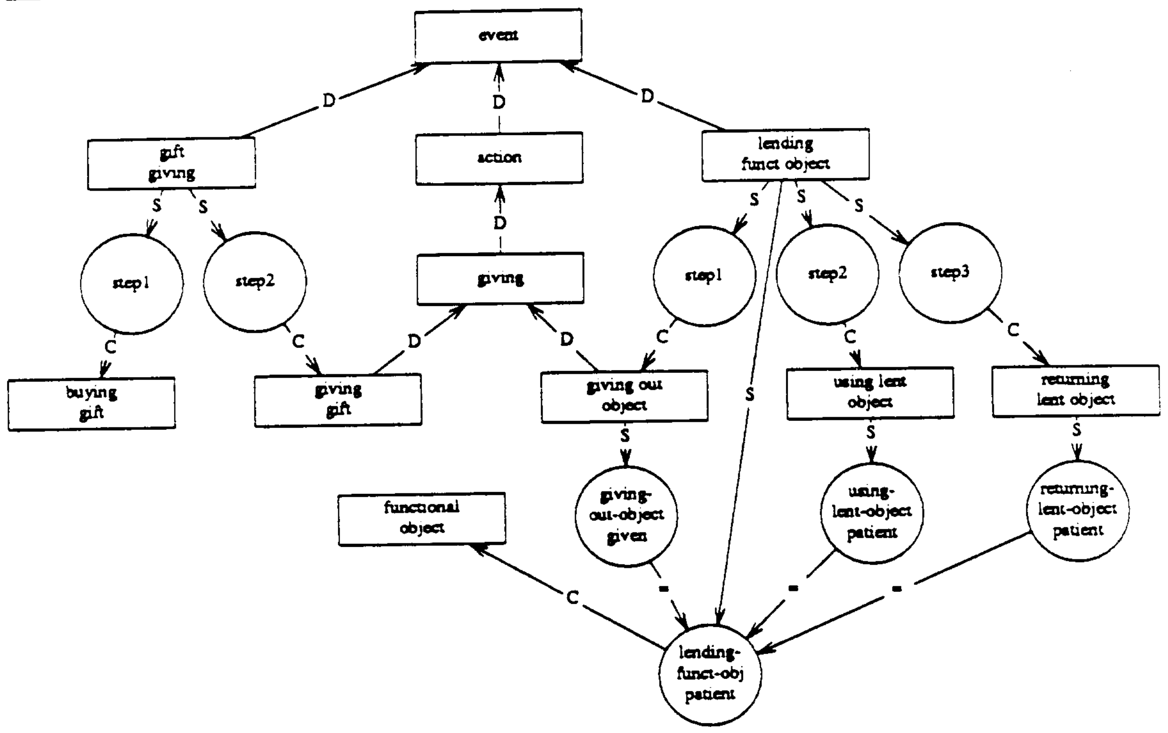

Figure 3: Domain Knowledge: Gift Giving and Lending

and `lending-functional-object`. `Gift-giving` is defined as having two steps; `buying-gift` and `giving-gift`. The two steps are defined in more detail in the FAUSTUS knowledge base, but the detail is not shown here. The other event is `lending-functional-object`, which has three steps. First the lender gives out the object; then the lendee uses it for its intended purpose; then he returns it. One fact about this situation is depicted: the patient of the lending is the same as the object given, the object lent, and the object returned.

Step 1: Representing the Input Text

Now that we have seen the representation of some of the background knowledge necessary to understand text (1), we move on to step 1 of the algorithm: the representation of input text. The three sentences are represented in Figure 4, in the abbreviated notation. Not all of the instance links are shown; `bicycle.1` should be an instance of `bicycle`, `male.3b` should be an instance of `male`, and so on. The notation `having.1`, for example, means that the event was mentioned in sentence 1; unless mentioned otherwise it will be assumed that this occurred before events in sentence 2.

Non-Marker-Passing Inferences

In Chapter 3 I stated that certain inferences happen immediately, when the input text is processed, rather than as a result of the marker passing algorithm. Such inferences fall into two classes: *relation classification inferences* and *relation constraint inferences*. Both of these are demonstrated in the following sentence, an excerpt from text (5), which will be shown in its entirety below.

```
[ 2] A girl started talking to him.

Rep: (TALKING (ACTOR ← A GIRL) (PATIENT ← HIM))

Inferring: the ACTOR of the TALKING must be the TALKER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the TALKING must be the TALKEE
   This is a RELATION-CLASSIFICATION inference.

Inferring: 'HIM' must be a PERSON, because it is the TALKEE
   This is a RELATION-CONSTRAINT inference.

Int: (TALKING.2 (↑ TALKING) (talker ← GIRL.2) (talkee ← MALE.2))
```

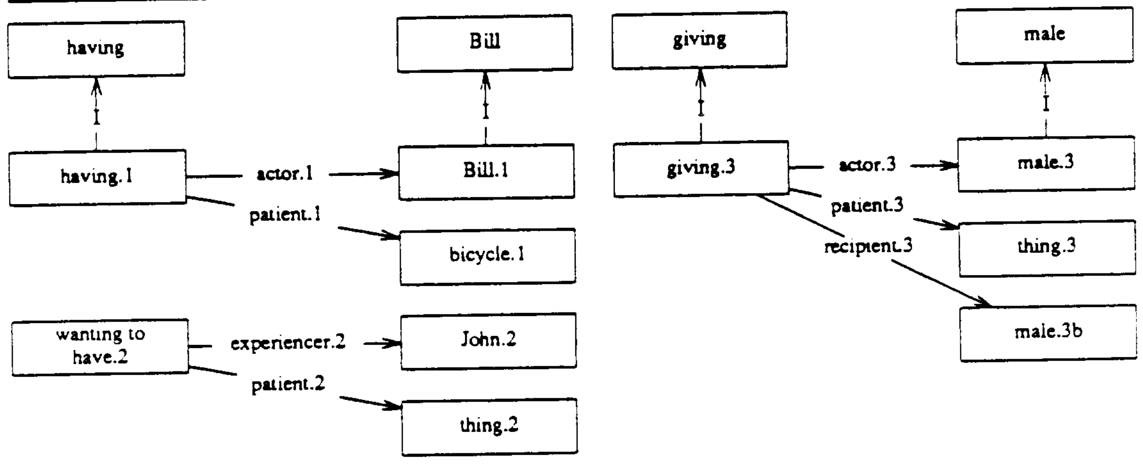

Figure 4: Representation of Input Text

Relation classification is the simpler of the two classes of non-marker-passing inferences. The girl is specified as the actor of a talking. FAUSTUS reports that the actor of a talking is called a `talker`. This is important because there are facts about talkers that are not true about actors in general. Technically speaking, this is a *logical inference*, as defined in Chapter 1. Every actor of a talking is necessarily a talker as a matter of definition. This is as opposed to the *plausible inferences* we have been talking concentrating on up to now. In a similar manner, the patient of the talking is classified as a talkee.

The other class of non-marker-passing inference is the relation constraint inference. In the example above, *him* is specified as the patient of a talking event. *Him* maps to the concept *male*, which is defined as an animal whose gender is the male sex, while `talking` is defined as a kind of communication where the talker and talkee must be people. Therefore, if this male is to be a talkee, he must be a male person, not just a male animal of any kind. FAUSTUS makes the inference that this is the case. Note that these assertions are made during the process of constructing the internal KODIAK representation of the input, which is denoted with the `Int:` line. After that point markers are passed from each of the concepts in the internal representation, and plausible inferences are suggested.

Step 2: Passing Markers

The next step is to pass markers from each of the new concepts in Figure 4 to neighboring concepts in the network. Markers will end up being propagated to a large number of concepts, including some not represented in Figure 4. Think of marker passing as spawning new markers which get spread around the network, and not as moving a single marker from node to node. The exact rules for propagating markers through the network are as follows. Start at each concept mentioned in the input with a constant amount of *marker energy*. The normal value for this parameter is 5. Next, start passing markers from each input concept to its neighboring concepts recursively until the marker energy is depleted. Never pass markers along links that are part of the representation of the text, and never pass markers back along the link just followed. Other than that, the links to pass along depend on the type of the concept and the cost in marker energy depends on the type of link.

From aspectuals, pass markers along constrainer links and inverse argument links, at a cost of one unit of marker energy. From relations, pass markers to aspectuals, also at a cost of one unit of marker energy. From absolutes, pass markers along constrainer links to the constrained aspectual, but only when the quantifier on the link is *all* or *most*. For example, there is a `water-surrounds-island` concept which states that *all* islands are surrounded by water, but only *some* bodies of water surround islands. The marker passing algorithm would traverse the link from `island` to `surrounded` to `body-of-water`, but would not go the other way from `body-of-water` to `surrounder` to `island`. This is an example of an asymmetry in conceptual distance: the distance from `island` to `body-of-water` is less than the distance from `body-of-water` to `island`.

We also pass markers to parents in the hierarchy (along D and I links) at no cost. The reason there is no cost associated with this link is that we do not want to allow intermediate nodes in the hierarchy to eclipse a potential inference. Suppose we had the assertion that `lassie` is an instance of `animal`. This could lead to inferences like `lassie has-as-part a head`. But note we could have represented `lassie` as an instance of `collie`, which is a kind of `large-dog`, which is a kind of `dog`, which is a kind of `canine`, which is a kind of `mammal`, which is a kind of `animal`. Now there is a chain of five links between `lassie` and `animal`, but she is still just as much an `animal` as before. To make sure this fact is reflected, we make the cost of traversing dominate links zero. Another way to look at this is that the marker passing scheme directly reflects the fact that dominate is transitive. Of course, there are other relations that are transitive, but that have no such privileged status. This means that FAUSTUS will in general fail to make certain inferences involving other transitive relations. However, there is some evidence that this is not unreasonable; that people do not automatically find the transitive closure of all relations. Langacker [69], for example, points out the oddity of sentences like *A body has 56 knuckles and 20 nails* as opposed to *A finger has 3 knuckles and 1 nail*. This is evidence that people are reluctant to apply *has* transitively.

Ordinarily, markers are not passed along view links. However, when there is a constraint violation, markers can pass from the concepts associated with the offending relation along view links, at a cost of one unit. This aspect is covered below in the section on *view application inferences*.

Marker passing continues to spread from each concept in the representation of the input until the marker energy reaches zero. The point of marker passing is to detect *marker collisions*, concepts that receive markers from two different origin concepts, along two different marker paths. Marker collisions suggest possible inferences, according to the rules laid down in the section on Step 3 below.

Anti-Promiscuity Cutoffs

Without some restraints, the marker passing mechanism as described would end up checking an enormous number of potential inferences. For example, every `situation` can have a set of `preconditions`, which are constrained to be `statives`, which is dominated by `situation`. If we allowed the marker passing mechanism to work unchecked, we would get elaboration collisions for every pair of instances of `situation` mentioned in the text, each collision suggesting that one is the `precondition` of the other. The problem is clear: at a lower level, we have definitions that say what types of `stative` can be a precondition for what type of `action`, but these distinctions are lost as we go up the hierarchy. They will be discovered again, and most of the potential inferences will be rejected, when the matching routines are applied. Unfortunately, that requires a lot of wasted computation. Worse than that is the possibility of introducing improper inferences. This problem is important because one of the design decisions in FAUSTUS was to have a two-step filtering process where not too many unreasonable inferences get suggested.

This problem is addressed by Charniak [27], who suggests the *anti-promiscuity* rule: do not pass markers to concepts that have more than *n* links attached to them, for some reasonably large value of *n*. I will call this the *static anti-promiscuity solution*. The problem with this approach is that adding new nodes to the middle of the hierarchy can disturb the link counts, and change the computation unpredictably. For example, suppose there is a high-level concept called `thing`, which dominates fifty different concepts. This is just the type of concept we would like to identify as a promiscuous one. However, now suppose we add two new concepts into the hierarchy just below `thing`, namely `animate-thing` and `inanimate-thing`. Then `thing` will no longer be promiscuous under Charniak's formulation. It may be that the effect of having `thing` be promiscuous is achieved if `animate-thing` and `inanimate-thing` still have enough links to be promiscuous, and if marker passing only proceeds up the hierarchy. But then the introduction of other intermediate nodes would just push the problem down one level.

Because of this difficulty, I have adopted the *dynamic anti-promiscuity solution*, which works as follows. First run the algorithm on a representative sample of texts. Then count the markers that accumulate at each concept, and declare the *m* concepts with the most markers as promiscuous concepts. In this approach, the introduction of new concepts like `animate-thing` and `inanimate-thing` will not change the total number of markers that ultimately arrive at `thing`. Both solutions have an element of arbitrariness; Charniak must choose a value for *n* and a topology of the netowrk, while I must both choose a value for *m* and a representative sample of texts.

Another change in dynamic anti-promiscuity is that I do not stop passing markers to promiscuous concepts; I just stop making certain classes of potential inferences at those concepts. This solution is appealing for several reasons. First, if we are assuming a parallel implementation of marker passing, then there is no cost in continuing passing markers. Even in a sequential simulation of parallelism, the promiscuous nodes are near the top of the hierarchy, and thus the cost of continuing to spread is not high. The sequential part of the algorithm– sifting through the collisions and evaluating inferences– would be slowed down by a proliferation of markers, so that is where the anti-promiscuity rule comes in to play. Spurious elaboration inferences can be ruled out, but we can still consider other inference classes, like referential inferences, that seem to require marker collisions at very high levels in the hierarchy.

To see that high-level collisions are sometimes important, consider the first two sentences of text (1), repeated here as (3). The word *it* involves the representation of something like `physical-object`, or perhaps something even more abstract; *it* can sometimes refer to a situation or an idea. If `physical-object` were marked as a promiscuous concept, then under Charniak's scheme there would be no way to get the collision that would generate the inference that *it* refers back to the bicycle. In my scheme, a collision would be detected at a promiscuous concept, but this would be a *referential* collision, a type that allows collisions at promiscuous concepts, if exactly one of the marker origins is explicitly marked as a reference. Reference collisions like the one at `person` from `Bill` and `John` are thrown out because `person` is a promiscuous concept. We will see other collision types that also refuse to suggest an inference if they occur at a promiscuous concept.

(3) Bill had a bicycle. John wanted it.

We can now state the complete set of constraints on the generation of elaboration path inferences. First, unlike referential inferences, if the collision occurs at a promiscuous concept, then no inference will be suggested. Also, if the slot in question is already filled, there is no sense suggesting that it be filled again. In the case of a double elaboration collision, the inference will only be suggested if the two origins are different; we would not suggest that a situation could be a precondition or result of itself. In the case of a double elaboration inference, the suggested inference will only be satisfied by creating a new instance which fills both slots. Creating two new instances, one for each slot, would violate the *relevance* criteria for inferences in that the new instances would only be connected to one other concept in the representation of the text. Furthermore, it would violate a kind of parsimony principle– we don't want to introduce two new objects when one would do.

In [28], Charniak presents a theory of marker passing which requires what he calls *predictive power* for inferences. Inferences are only made when the number of true predictions resulting from the inference is at least as many as the number of new assumptions required. This rule is in accordance with the rules of relevance and parsimony, but by Charniak's own admission, there are several ''ugly'' details. Part of the problem is that it is not clear what should count as an assumption, or as a prediction. I agree that the details can be quite ugly; in my model I have no general theory of predictive power; instead I have different constraints on what can be introduced by each inference class.

Step 3: Suggesting Inferences - FAUSTUS' Inference Classes

When a marker collision is detected, the first thing that happens is the marker paths leading to the collision are classified according to their *marker path shape*. Each half of the path is classified independently, and then the collision evaluation function dispatches on the combination of two shape classes to a pre-defined inference class which may or may not suggest a potential inference to put in the agenda. There are currently five path shapes, and six inference classes. The inferences can suggest adding a new absolute to the construal of the text, adding a new relation between absolutes, construing one object to be co-referential with another, or classifying some absolute or relation to be a member of some class. In effect, the claim is that these are the only kind of inferences that need be made (for a certain level of understanding of the text), and that the six inference classes are sufficient to generate the appropriate construals.

In story-understanding systems such as BORIS [38] and in most expert-system programs, there can be hundreds of inference rules, and adding new knowledge means adding new rules. In FAUSTUS, adding new knowledge is done declaratively, without changing or adding any of the six basic inference classes. Thus, the term *inference class* in FAUSTUS is very different from the inference rule in traditional expert systems.

Potential inferences take the form of entries on an *agenda*, or queue of suggested inferences. Each entry in the agenda has an inference class, and some specific information that depends on the type. For example, the suggestion of finding a referent for a pronoun would have information giving a list of possible referents. Each suggestion, regardless of its type, can have an invocation time before which it cannot be invoked, and an expiration time after which it is automatically removed from the agenda. Each suggested inference also keeps track of how long it has been in the agenda, its scheduling priority, and the two marker paths that led to the collision that placed it on the agenda. When a suggested inference is run, it does one of three things: *succeeds* and builds new representations into the world model, *fails* and removes itself from the agenda without building any representations, or *defers* and puts itself back into the agenda.

The implementation of FAUSTUS checks for collisions between two paths of exactly the right shape. It would also be possible to check all collisions by examining the two halves of the path that led to the collision, and seeing if their concatenation is one of the valid path shapes. In Chapter 6, I explain why it was deemed more efficient to look for path halves rather than complete paths, and why this will not miss any important collisions. However, this was an implementation decision, and the important thing is that certain paths, however they are recognized, suggest certain inference classes.

There are also two inference classes that are not associated with path shapes, but occur as an immediate result of interpreting the input.

FAUSTUS supports a simple regular-expression definition language for defining interesting path shapes. In this notation, the arrow ($\rightarrow$) means that the markers follow a link of the given type from one concept to the next. The link types are I for instance, D for dominate, V for view, C for constrain, F for fill, A for argument, = for equate and $\neq$ for differ. The markers must traverse the links in the right direction; inverse links are denoted by the superscript ($^{-1}$). For example, A^{-1} means an inverse argument link, a link from an argument to its relation rather than from a relation to its argument. The other convention in this notation is that a (*) is the Kleene star operator, denoting any number of repetitions of the preceding link type.

The possible inference classes with their associated path shapes are shown in Figure 5. Each inference class will be discussed in turn in the sections that follow.

Elaboration Inferences

Elaboration inferences are those that build a new relation between two absolutes, filling in a slot of some absolute with another absolute. As we shall see, there are two varieties of elaboration inferences; one creates a new instance as one of the absolutes, the other uses existing instances. The definition of the elaboration path shape is as follows:

(4) $\text{origin} \rightarrow I \rightarrow D^* \rightarrow C^{-1} \rightarrow A^{-1} \rightarrow A \rightarrow C \rightarrow D^* \rightarrow \text{collision}$

In other words, the elaboration path starts at a concept mentioned in the text, follows an I link, then any number of D links, then an inverse C link and an inverse A link to arrive at a relation. From there the path goes along an A link to an argument, up a C

Inference Classes	**Path 1**	**Path 2**
Double Elaboration	Elaboration	Elaboration
Elaboration	Elaboration	Ref
Reference Resolution	Ref	Ref
View Application	Constraint	View
Concretion	Elaboration	Filler
Relation Concretion	Elaboration	Filler

Path Name	**Path Shape**
Elaboration	$\text{origin} \rightarrow I \rightarrow D^* \rightarrow S \rightarrow C \rightarrow D^* \rightarrow \text{collision}$
Ref	$\text{origin} \rightarrow I \rightarrow D^* \rightarrow \text{collision}$
View	$\text{origin} \rightarrow I \rightarrow D^* \rightarrow V \rightarrow D^* \rightarrow C^{-1} \rightarrow A^{-1} \rightarrow \text{collision}$
Constraint	$\text{origin} \rightarrow I \rightarrow D^* \rightarrow C^{-1} \rightarrow A^{-1} \rightarrow \text{collision}$
Filler	$\text{origin} \rightarrow F^{-1} \rightarrow A^{-1} \rightarrow A \rightarrow F \rightarrow I \rightarrow D^* \rightarrow \text{collision}$

Non-Marker-Passing Inference Classes
Relation Classification
Relation Constraint

Figure 5: Path Shapes and Inference Classes

link to the constrainer of the argument, and then again along any number of D links to arrive at the concept where the collision occurred. The combination of going down an inverse C link and then along two A links is an 'S' link in the abbreviated notation, so the elaboration path could also be defined as (5):

(5) $\text{origin} \rightarrow I \rightarrow D^* \rightarrow S \rightarrow C \rightarrow D^* \rightarrow \text{collision}$

Once the path shape is defined, the next step is to define how this shape interacts with others. It turns out there are two relevant path collisions that can suggest inferences: when two elaboration paths collide, or when an elaboration path collides with a *referent* path. Referent paths will be covered in the next section. Their shape is defined with equation (6), which denotes a path starting at a concept mentioned in the input, and traveling up the taxonomic hierarchy, first by an instance link and then by any number of dominate links.

(6) $\text{origin} \rightarrow I \rightarrow D^* \rightarrow \text{collision}$

When an elaboration path and a referent path collide, the suggested inference is that the concept at the origin of the referent path might be the filler of the slot in the elaboration path. For example, Figure 6 shows a referent path from `having.1` colliding at `having` with an elaboration path from `giving.3`. The suggested inference is that `having.1` might be a `precondition` of `giving.3`. In this particular case, the suggested inference will eventually be accepted, because there is no evidence to contradict it, and no better alternative filler for that slot.

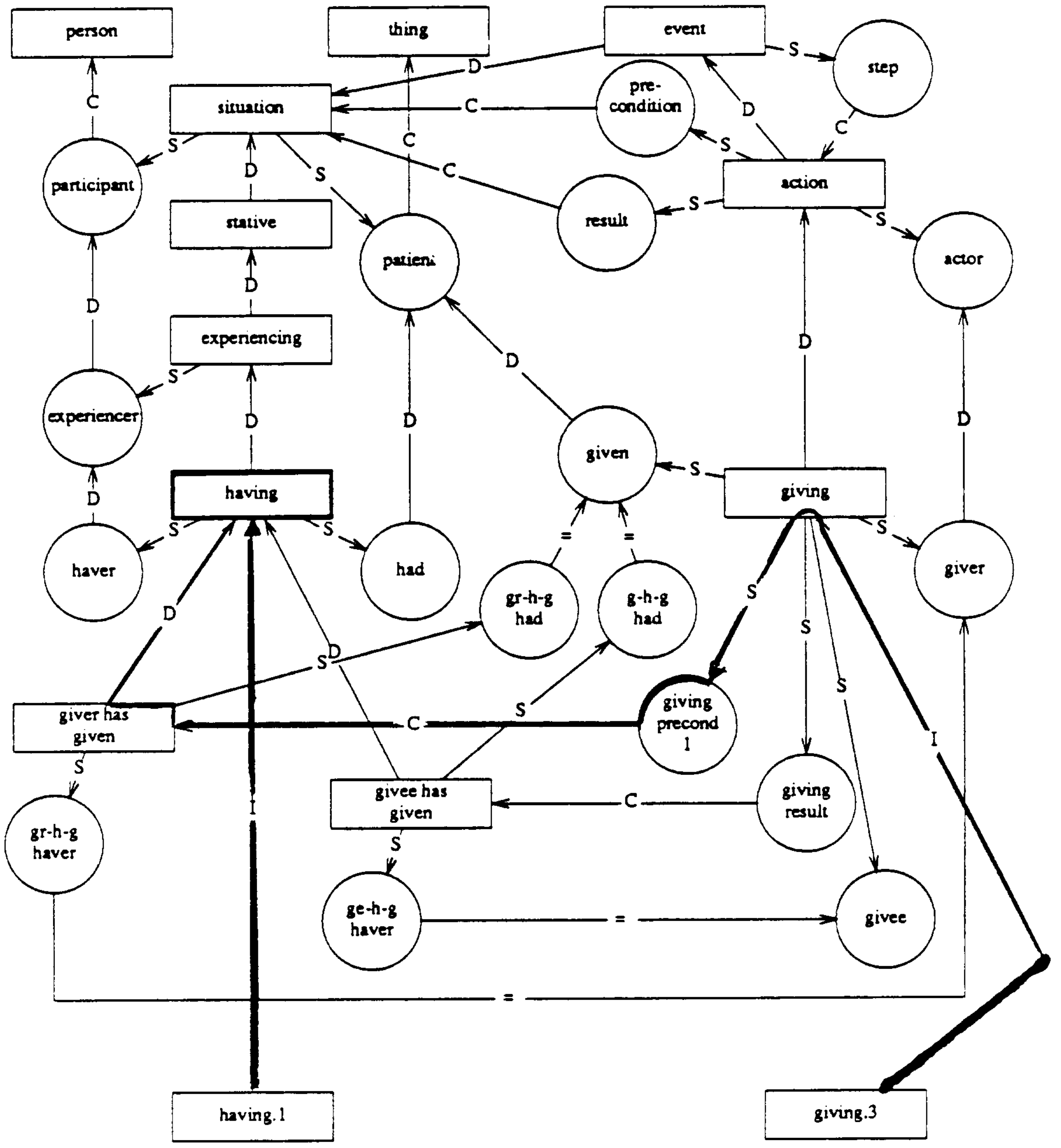

Figure 6: Elaboration/Referent Path Collision

Figure 7 shows the result of an elaboration inference: a new relation, `giving-precondition1.3`, is built (at the bottom middle of the diagram) indicating that `having.1` is in fact a precondition of `giving.3`.

Not all suggested inferences are accepted. There is another collision at `having` suggesting that `having.1` is the result of `giving.3`. This elaboration inference will be rejected when it comes time to evaluate it because the match is not satisfied. There is an `after` relation that must hold between a `situation` and its `result`, but `having.1` occurs before, not after `giving.3`.

The other type of inference involving an elaboration path is the *double elaboration* inference, when two elaboration paths collide. Although there are no good examples of this in the sample text, it does show up in the Fisherboy story, text (1) of Chapter 1, in the phrase *hoping to catch a few fish from the sea, which they could sell.* The connection between `catching` the fish and `selling` the fish is an instance of `having` the fish. Unlike the previous examples, there is no direct link between the two, and there are no instances of `having` mentioned in the text. Instead, there is a two-step connection: the `having` is a result of the `catching` and a precondition of the `selling`. One might guess that this difference would require completely different marker paths and inference classes than the case of simple elaborations, but it turns out that the same shape marker paths are involved. Markers pass from `catching` *via* `result` to `having-quarry` to `having`, and from `selling` *via* `precondition` to `having-merchandise` to `having`. Both of these paths have the shape $I \rightarrow S \rightarrow C \rightarrow D$, which matches the specification in (8). The collision causes the potential inference to be queued, and eventually a new instance of `having` is created to fill both parts of the connection.

Double elaboration inferences always wait in the queue one time unit before they are evaluated. The reason for this is that the next sentence might provide an explicit filler for the slots, in which case it would be incorrect to make up a new instance for that purpose. The *raison d'etre* of the agenda is to delay making any decisions until pairs of inferences like these have time to hook up with each other.

Double elaboration inferences solve a problem that Charniak [23] addressed with a much more complicated approach. He was concerned with texts like the following:

(7) Janet was going to get a present for Jack.
She needed some money.

(8) Janet needed some money.
She was going to get a present for Jack.

In both cases, the reader should infer that Janet needed the money because of getting the present. Charniak's solution was to attach to the concept `get-present` a procedure, or demon, which would look for an instance of `needing-money`, and if found, assert that the needing and getting are causally related. As (7) and (8) show, the demon must be able to look both forwards and backwards. The problem with this shows up in cases like (9):

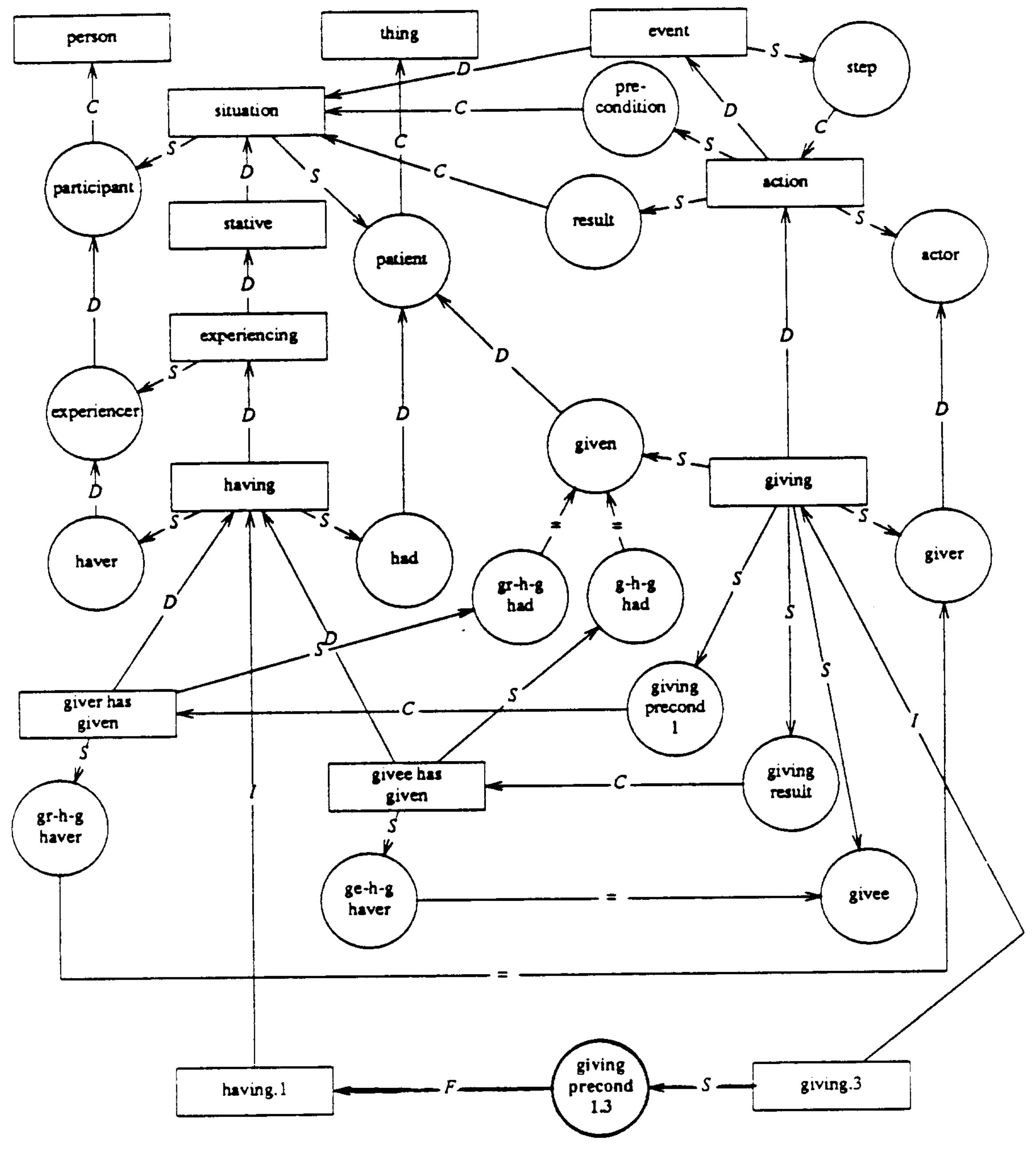

Figure 7: Elaboration Inference Made

(9) Janet was going to get a present for Jack.
She went to get her piggy-bank.

Here, the demon on `get-present` will not find the `need-money` it is looking for, and thus can make no inference. But this is not the only demon around. In Charniak's formulation there is also a demon on `get-piggy-bank` which is also looking for an instance of `need-money`, and, once found, will assert that needing money is the reason for getting the piggy-bank. To make the right inferences in the case of (9), Charniak must introduce a complicated mechanism called *demon-demon interaction.*

In FAUSTUS, we take a declarative, rather than a procedural approach to such problems. There is no need for demons, and thus there is no need to instruct demons which direction to look, or how they should interact with one another. Instead, we need only add to the knowledge base an `enables` relation between `have-money` and `get-present`, and a `reason` relation between `get-piggy-bank` and `have-money`. Once that is done, the double elaboration inference class will find the right connection in (9), and instantiate the right inferences.

Referential Inferences

Referential inferences allow, among other things, a pronoun or definite noun phrase to refer to a concept previously represented in the text model. Referential inferences add equate (=) links to the representation. Equation (6), repeated below as (10), defines the referential path:

(10) origin $\rightarrow$ I $\rightarrow$ D* $\rightarrow$ collision

There are actually three distinct classes of referential paths, although all of them fit the specification in (10). The difference is in the *determiner* associated with the concept at the origin of the path. If the determiner is indefinite, the path is classified as a *reference* path; if the determiner is definite, the path is a *referent* path, and if the determiner is unspecified, the path is classified as simply a *ref* path. Noun phrases in English usually have a article, like *a* or *the* which can give an indication of how they are used, but verb phrases usually have no such surface indication. Whenever a *reference* or *ref* path collides with a *referent* or *ref* path the potential inference suggested is that the concept at the origin of the first path is a reference to the concept at the origin of the second path. If more than one referent is suggested they are all considered in turn when the suggested inference is taken off the agenda and run.

To understand how the referential path inferences work, consider text (11).

(11a) A boy was walking down the street with a friend.
(11b) A girl came up to talk to him.

The sole reference path starts at the concept `male.2`, which is the interpretation of the word *him* in (11b). This reference path collides with referent paths originating at `boy.1` and `friend.1`. Each of these collisions results in a suggestion that the origin of the reference path (the concept `male.2`) refers to the origin of the referent path. This reference path will participate in other collisions, such as the ones from `street.1` and girl.2, but these will generate no suggestion, since they are mutually exclusive with the concept male.2.

After processing (11b), it is time to evaluate the suggested inferences. For each reference, we first count the number of suggested referents that still match the referent. If there is exactly one, it is accepted, and an equate link is added to show this. If there are more than one possibility, as in this case, we find the subset of the possibilities that match the reference in the highest number of features or relations. A boy is defined as a child, a male, and a person, whereas a friend is defined as a person who participates in a friendship relation. Therefore, boy.1 matches male.2 on two counts, male and person, while friend.1 matches only as a person. So in this case, the inference would be that *him* refers to the *boy*.

Suppose there were no unique referent chosen by the process above. The next step is to fall back on the ideas of *recency* and *focus*. Recency is the number of sentences that have passed since a possible referent was last mentioned, and focus is the idea that concepts that have served as slot fillers for salient case slots like actor and patient are more likely to be referents than concepts that were fillers of peripheral case slots, or no slots at all. For example, suppose that in (11a), the boy was walking down the street with *his father* rather than *his friend*. Then there would be two possible referents for the *him*, both matching as a male and as a person. They also would have the same recency, since this is scored only by sentence number, not by position within sentence. The only distinguishing factor is that the boy fills an actor slot, and actor is defined to be a focused-slot, so the boy would be chosen as the referent over the father. One more rule for resolving reference collisions is that ref/ref collisions are ignored if they occur at a promiscuous node.

A trace of FAUSTUS making some reference inferences follows:

```
> (do-story)

        Walking Down the Street


[ 1] A boy was walking down the street with a friend.

Rep: (WALKING (ACTOR ← A BOY) (PATH ← THE STREET) (WITH ← A FRIEND))

Inferring: the ACTOR of the WALKING must be the WALKER
   This is a RELATION-CLASSIFICATION inference.

Int: (WALKING.1 (↑ WALKING) (walker ← BOY.1) (path ← STREET.1)
                (with ← FRIEND.1))

Passing Markers and Suggesting Inferences:  1
```

```
Inferring: a WITH of the WALKING is probably the ACCOMPANIER
   because the FRIEND fits it best.
   This is a RELATION-CONCRETION inference.
   It is #1, due to the collision:
   WALKING.1→filler→PERSON←elaboration←WALKING.1

[ 2] A girl started talking to him.

Rep: (TALKING (ACTOR ← A GIRL) (PATIENT ← HIM))

Inferring: the ACTOR of the TALKING must be the TALKER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the TALKING must be the TALKEE
   This is a RELATION-CLASSIFICATION inference.

Inferring: 'HIM' must be a PERSON, because it is the TALKEE
   This is a RELATION-CONSTRAINT inference.

Int: (TALKING.2 (↑ TALKING) (talker ← GIRL.2) (talkee ← MALE.2))

Passing Markers and Suggesting Inferences:  2 3

Evaluating Inferences:

Inferring: 'HIM' refers to the BOY.
   because it is the best match.
   This is a REFERENCE inference.
   It is #2, due to the collision:
   BOY.1→ref→PERSON←ref←MALE.2

Rejecting: 'HIM' refers to the FRIEND.
   because it is not the best match.
   This is a REFERENCE inference.
   It is #3, due to the collision:
   FRIEND.1→ref→PERSON←ref←MALE.2
```

Here, because the boy is male, it matches 'him' better than the friend, which is not marked for gender (that is, is not an instance of either male or female).

```
> (do-story)

          Walking Down the Street #2
```

```
[ 1] A boy was walking down the street with his father.

Rep: (WALKING (ACTOR ← A BOY) (PATH ← THE STREET) (WITH ← A FATHER))

Inferring: the ACTOR of the WALKING must be the WALKER
   This is a RELATION-CLASSIFICATION inference.

Int: (WALKING.1 (↑ WALKING) (walker ← BOY.1) (path ← STREET.1)
                (with ← FATHER.1))

Passing Markers and Suggesting Inferences:  1

Inferring: a WITH of the WALKING is probably the ACCOMPANIER
   because the FATHER fits it best.
   This is a RELATION-CONCRETION inference.
   It is #1, due to the collision:
   WALKING.1→filler→PERSON←elaboration←WALKING.1

[ 2] A girl started talking to him.

Rep: (TALKING (ACTOR ← A GIRL) (PATIENT ← HIM))

Inferring: the ACTOR of the TALKING must be the TALKER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the TALKING must be the TALKEE
   This is a RELATION-CLASSIFICATION inference.

Inferring: 'HIM' must be a PERSON, because it is the TALKEE
   This is a RELATION-CONSTRAINT inference.

Int: (TALKING.2 (↑ TALKING) (talker ← GIRL.2) (talkee ← MALE.2))

Passing Markers and Suggesting Inferences:  2 3

Evaluating Inferences:

Inferring: 'HIM' refers to the BOY.
   because it is in focus.
   This is a REFERENCE inference.
   It is #2, due to the collision:
   BOY.1→ref→PERSON←ref←MALE.2

Rejecting: 'HIM' refers to the FATHER.
   because it is not in focus.
   This is a REFERENCE inference.
   It is #3, due to the collision:
```

```
FATHER.1→ref→PERSON←ref←MALE.2
```

In this case, both the boy and the father match 'him' equally well, since they are both males. The evaluation procedure thus moves on to the next criterion, which is participation in a focused case relation. The boy was the actor in the previous sentence, so he is in focus, and the father is not.

It is an open research question how to combine evidence from various sources; FAUSTUS uses a fairly primitive counting procedure that does not try to answer questions like *how much focus is necessary to offset one time unit of recency?*. The idea of a best match comes from counting up the number of features, or relation fillers, that two concepts have in common. Either two concepts do not match at all, or they have a small number of features in common – usually in the range zero to two. If there are several matches with the same count, then we look to see if there is exactly one that is more recent than the others. If not, we award one point for each participation in a focused case relation and again see if the tie is broken.

One complication in telling references from referents is that the surface article in English is not as reliable as one might hope it would be. Normally, definite noun phrases like *him* or *the boy* are references, while indefinite noun phrases like *a boy* are referents. However, this rule does not always hold; the article *the* can be used to refer to a specific entity that had not previously been mentioned in the text, as long as it is a well-known entity, such as *the sun* or *the president*. It can also refer to some role in a known script or other type of knowledge structure, as when we refer to *the waiter* in a restaurant. This type of inference is handled routinely by FAUSTUS; it is just an elaboration/reference collision involving an instance of `waiter` and the `waiter-of-restaurant` concept.

Another complication is that some phrases are not marked with any article at all in English. This is particularly true for verb phrases. In passages like (12), the *talked* in the second sentence refers to the same event as the *discussed* in the first sentence, but neither event is explicitly marked as definite or indefinite. FAUSTUS is able to make the inference that the two actions are co-referential, using the same mechanism that works for pronouns. The idea of treating actions under a theory of reference is covered in [71]. A trace of FAUSTUS processing (12) follows:

(12) The president discussed Nicaragua. He talked for an hour.

```
> (do-story)

          The President


[ 1] The president discussed Nicaragua.

Rep: (DISCUSSING (ACTOR ← THE PRESIDENT) (CONTENT ← NICARAGUA))
```

```
Inferring: the ACTOR of the DISCUSSING must be the TALKER
   This is a RELATION-CLASSIFICATION inference.

Int: (DISCUSSING.1 (↑ DISCUSSING) (talker ← PRESIDENT.1)       (content
← NICARAGUA.1))

[ 2] He spoke for an hour.

Rep: (TALKING (ACTOR ← HE) (DURATION ← AN HOUR))

Inferring: the ACTOR of the TALKING must be the TALKER
   This is a RELATION-CLASSIFICATION inference.

Inferring: 'HE' must be a PERSON, because it is the TALKER
   This is a RELATION-CONSTRAINT inference.

Int: (TALKING.2 (↑ TALKING) (talker ← MALE.2) (duration ← HOUR.2))

Evaluating Inferences:

Inferring: 'HE' refers to the PRESIDENT.
   This is a REFERENCE inference.
   It is #1, due to the collision:
   PRESIDENT.1→ref→PERSON←ref←MALE.2

Inferring: the NICARAGUA is a COUNTRY such that
   it is the HABITAT of 'HE' and
   it is the COUNTRY of the PRESIDENT.
   This is a DOUBLE-ELABORATION inference.
   It is #3, due to the collision:
   MALE.2→elaboration→PLACE←elaboration←PRESIDENT.1

Inferring: the TALKING refers to the DISCUSSING.
   This is a REFERENCE inference.
   It is #4, due to the collision:
   DISCUSSING.1→ref→TALKING←ref←TALKING.2

Inferring: the DISCUSSING refers to the TALKING.
   This is a REFERENCE inference.
   It is #5, due to the collision:
   DISCUSSING.1→ref→TALKING←ref←TALKING.2
```

Although this example was meant only to illustrate action/action co-reference, several inferences are made. First, FAUSTUS realizes that *he* refers to the president. That is straightforward. But next there is a double-elaboration inference, where Nicaragua is taken to be both the residency of the president, and the country which he presides over. I did not expect this inference to occur; like most readers, I interpreted the text as referring to the president of the United States. However, this is because I am living in the U.S., and the current U.S. president is a salient figure. FAUSTUS does not have this context available to it. Nowhere was it specified to FAUSTUS that the texts are being read in the U.S., so given that (lack of) context, inference #3 is quite reasonable.

View Application Inferences

In Chapter 3 we introduced the necessity of viewing one concept as another in order to make the right interpretation. An example of this shows up in the sentence *The Red Sox killed the Yankees*. There is a constraint violation in that the `Killed-Of-Killing` relation should hold between an instance of `killing` and an `animal`, but the Yankees are defined as a baseball team, which is an organization, and not an animal. To resolve this constraint violation, we need to either interpret the Yankees as a kind of animal, or the killing as a kind that holds between organizations. Views are applied to make interpretations like this, but they are only applied as needed. Ordinarily, markers are not passed along view links. However, if an input representation contains a constraint violation, then the markers originating at each of the concepts involved in the violation are free to traverse view links.

The path shapes used in view application inferences are as follows, and a diagram of the collision is shown in Figure 8.

(13) origin $\rightarrow I \rightarrow D^* \rightarrow V \rightarrow D^* \rightarrow C^{-1} \rightarrow A^{-1} \rightarrow$ collision
(14) origin $\rightarrow I \rightarrow D^* \rightarrow C^{-1} \rightarrow A^{-1} \rightarrow$ collision

Path (13) is a *view path*, and (14) denotes a *constraint path*. These work in tandem much as the referent and reference paths do; there is an inference associated with the intersection of a constraint path and a view path, but neither of them interact with any of the other types of paths. The inference routine associated with their intersection checks to see if viewing the concept at the origin of (13) as an instance of the concept at the collision could rectify the constraint violation that started this passing along view links. If so, the suggested inference is to apply the view.

A trace of FAUSTUS processing the kill example follows:

```
> (do-story)

          Baseball


[ 1] The Red Sox killed the Yankees.

Rep: (KILLING (ACTOR ← THE RED-SOX) (PATIENT ← THE YANKEES))
```

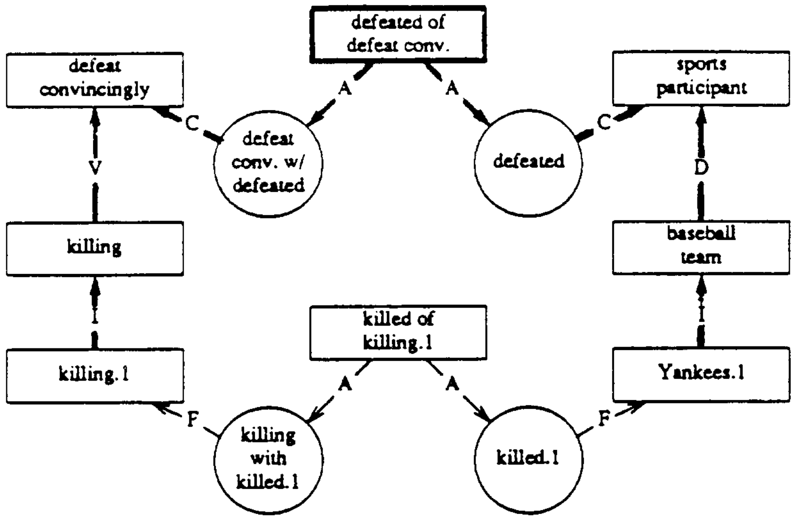

Figure 8: View Application Path Collision

```
Inferring: the ACTOR of the KILLING must be the KILLER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the KILLING must be the KILLED
   This is a RELATION-CLASSIFICATION inference.

Int: (KILLING.1 (↑ KILLING) (killer ← RED-SOX.1) (killed ← YANKEES.1))

Passing Markers and Suggesting Inferences:  1 2

Evaluating Inferences:

Inferring: the KILLING is viewed as a DEFEAT-CONVINCINGLY,
   where the YANKEES is the DEFEATED of it.
   This is a VIEW-APPLICATION inference.
   It is #1, due to the collision:
   KILLING.1→view→Defeated-Of-Defeat-Convincingly←constraint←YANKEES.1

Inferring: the KILLING is viewed as a DEFEAT-CONVINCINGLY,
   where the RED-SOX is the DEFEATER of it.
   This is a VIEW-APPLICATION inference.
   It is #2, due to the collision:
   KILLING.1→view→Defeater-Of-Defeat-Convincingly←constraint←RED-SOX.1
```

When more than one view is applicable, we have to choose between them. Having multiple applicable views is rarer than having multiple possible referents, but it is still important to account for this case. The rule for choosing among multiple applicable views is as follows:

> The first step is to see if one of the applicable views is dominated by another. If so, consider only the most specific, and eliminate the more general view from consideration. If this fails, the next step is to defer making a decision for one time unit. On the next time around, it may be that newly inferred links to the concepts involved will enable a choice to be made.

As an example of multiple applicable views, consider the sentence *The chairman moved the meeting up a week.* Moving something up should take as object a direction or distance, not a time duration. To understand this sentence, we need a view that maps time onto a physical direction or distance scale. There are two such views, one that measures distance from the current time into the future, and one that measures time from some landmark in the future back to the current time. Applying the first view would mean interpreting the sentence as meaning the meeting was postponed, while the second view would have the meeting occurring earlier than originally planned. FAUSTUS's rules would be unable to disambiguate such an example.

There are cases where a view should be applied even when there is no constraint violation. For example, in *Lendl killed Becker at Wimbledon*, there is a valid literal interpretation, but the preferred interpretation still views *killed* as defeat convincingly. Similarly, it is literally true that *no man is an island, entire of itself;* but in processing the Donne quote it would be best to interpret *island* as an `isolated-entity`, rather than a `land-mass`, even without considering the constraint violation in *every man is a piece of the Continent*. FAUSTUS does not handle such cases.

Paul Jacobs [59] used views to handle certain very general relationships in language. The problem he addressed was generating English sentences. James Martin [77] shows that a system that has representations for common conventional metaphors can acquire new knowledge easily from user input that refers to these metaphors. Neither address the problem of multiple applicable views, or of applying views when the literal reading is valid.

Concretion Inferences

In the KL-ONE language, much attention is paid to the process of *classification* (see Schmolze and Lipkis [117]). For example, when given an instance of `traveling` with an `automobile` as `instrument`, the KL-ONE classifier could conclude the `traveling` must also be an instance of `driving`. When given a description of an `animal` with four legs, a trunk, large ears, tusks, and grey skin, the classification algorithm could conclude that the animal must be a `quadruped`, but no more. We would like to be able to do more than that, and infer that the animal is probably an `elephant`. This is a plausible inference, not a logical consequence of the taxonomy, and thus is beyond the scope of KL-ONE classification. Such an inference is called a *concretion* inference in FAUSTUS. It refers to the process of interpreting a concept as something more concrete – less abstract – than is strictly warranted by the representation. Making a concretion inference means adding a dominate or instance link from the concept to a category. Concretion was first discussed in [136] and [88, 89].

To demonstrate how concretion works in FAUSTUS, we will be examining the following sentence:

(15) John cut the lawn.

FAUSTUS' processing of this sentence is as follows:

```
> (do-story)

          Cutting


[ 1] John cut the grass.

Rep: (CUTTING (ACTOR ← JOHN) (PATIENT ← THE GRASS))

Inferring: the PATIENT of the CUTTING must be the THING-CUT
   This is a RELATION-CLASSIFICATION inference.

Int: (CUTTING.1 (↑ CUTTING) (actor ← PERSON.1) (thing-cut ← GRASS.1))

Passing Markers and Suggesting Inferences:  1

Evaluating Inferences:

Inferring: the CUTTING is a LAWN-CUTTING.
   This is a CONCRETION inference.
   It is #1, due to the collision:
   GRASS.1→filler→CUTTING←elaboration←GRASS.1
```

In (15), FAUSTUS infers that this is an instance of `lawn-cutting`, not just any kind of cutting. This is important, because there are several specific facts associated with cutting the lawn. For example, it is likely that John used a lawnmower as an instrument, and that he cut the blades of grass horizontally, to a roughly uniform height. If we stopped at `cutting`, and did not make the concretion inference, we could not make the lawnmower interpretation. It would be just as likely that the instrument was a chainsaw, and that he hacked the turf into two large pieces.

Concretion inferences are suggested as the result of collisions between an elaboration path and a *filler* path. The elaboration path shape has been seen before as (5), but is repeated here as (16), and the filler path is defined in (17). A diagram of the collision is shown as Figure 9. The details of the collision are that, if the origins of the two paths are the same, and the collision occurs at a non-promiscuous concept, then suggest that the concept at the bottom of the I link in the filler path (in this case, `cutting.1`) is an instance of the concept at the start of the S link in the elaboration path (in this case, `lawn-cutting`). When it comes time to evaluate the suggestion, FAUSTUS checks that the one concept could still be an instance of the other (it has not been made incompatible by the addition of new information). If several incompatible concepts have been suggested as concretions of the same concept, FAUSTUS tries to find the best match by counting matching features, using the same mechanism that was used for finding the best referent. However, there is no tie-breaking formula for concretion inferences (recency and participating in central case relations is not important). In case of a tie, no inference is made.

(16) origin $\rightarrow$ I $\rightarrow$ D* $\rightarrow$ S $\rightarrow$ C $\rightarrow$ D* $\rightarrow$ collision
(17) origin $\rightarrow$ F^{-1} $\rightarrow$ A^{-1} $\rightarrow$ A $\rightarrow$ F $\rightarrow$ I $\rightarrow$ D* $\rightarrow$ collision

In this example it was the absolute – the cutting – that was the object of concretion. It is also possible to concrete the relation between two absolutes. This is done by a *relation concretion* inference. Relation concretion inferences are suggested by the same shape marker collision as 'regular' concretion inferences, but under slightly different circumstances. To see examples of relation concretion, we turn to the "Spaghetti Dinner" texts, given in Chapter 3 as (6a-d), and repeated here as (18a-d). The problem is that the relation `with` is ambiguous. It can be an accompanier, instrument, manner or just a

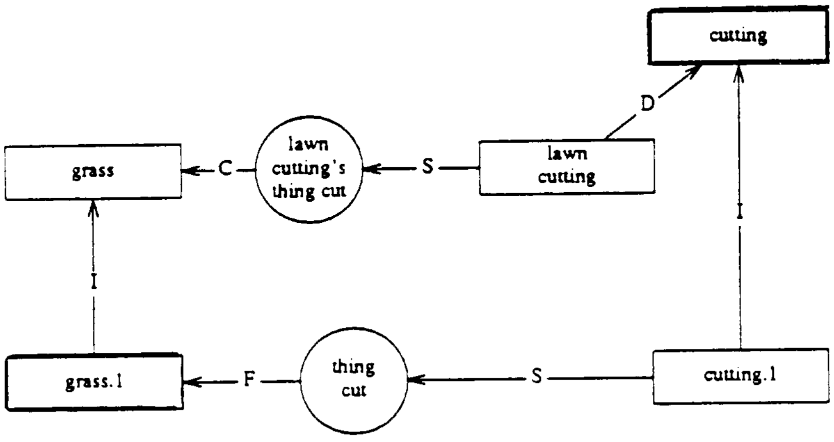

Figure 9: Concretion Path Collision

default "along with" modifier.

(18a) John ate spaghetti with Frank.
(18b) John ate spaghetti with a fork.
(18c) John ate spaghetti with gusto.
(18d) John ate spaghetti with pesto.

In the FAUSTUS output shown below, each line in (18a-d) is processed as a separate text. In (18a), which is Story #1 below, there is a concretion collision at the concept `person`. Compare this collision to the one in Figure 9, where the suggested inference is that `cutting.1` is an instance of `lawn-cutting`. In Figure 10 a corresponding suggestion is not made, because the elaboration path does not go through a sub-category on the path to the `person`. Instead, the suggestion is that the `with` relation should be categorized as an `accompanier`. Another collision, at `phys-obj`, suggests that the relation be interpreted as an `instrument`. In cases where there are multiple possibilities, the evaluation rules favor the more specific interpretation. Since `person` is more specific than `phys-obj`, the `accompanier` interpretation is accepted, and the `instrument` rejected. In (18b), only the `instrument` interpretation is suggested, so it is accepted.

```
> (do-story)

          Spaghetti Dinner #1


[ 1] John ate spaghetti with Frank.
```

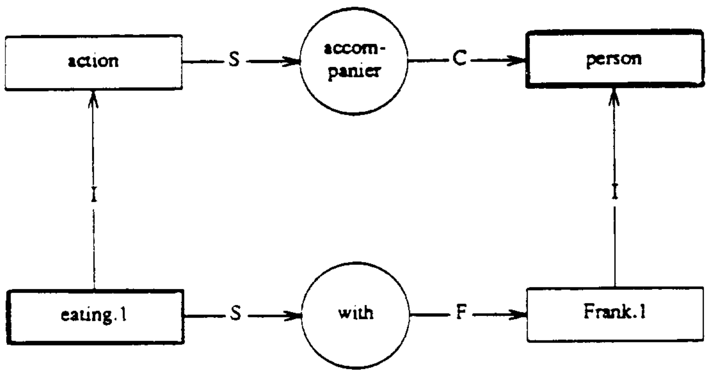

Figure 10: Marker Collision for "Spaghetti Dinner #1"

```
Rep: (EATING (ACTOR ← JOHN) (PATIENT ← SPAGHETTI) (WITH ← FRANK))

Inferring: the ACTOR of the EATING must be the eater
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the EATING must be the eaten
   This is a RELATION-CLASSIFICATION inference.

Int: (EATING.1 (↑ EATING) (eater ← PERSON.1) (eaten ← SPAGHETTI.1)
               (with ← PERSON.1B))

Passing Markers and Suggesting Inferences:  1 2

Inferring: a WITH of the EATING is probably the ACCOMPANIER
   because Frank fits it best.
   This is a RELATION-CONCRETION inference.
   It is #1, due to the collision:
   EATING.1→filler→PERSON←elaboration←EATING.1

Rejecting: a WITH of the EATING is probably a INSTRUMENT
   because another interpretation, ACCOMPANIER, is more specific.
   This is a RELATION-CONCRETION inference.
   It is #2, due to the collision:
   EATING.1→filler→PHYS-OBJ←elaboration←EATING.1
```

Here the word "with" was interpreted as the accompanier of the eating action. The output "a WITH of the EATING is..." may not be perfect English, but it is analogous to (and generated by the same print subroutine as) the phrase "the ACTOR of the EATING must be..." The meaning of the former is that there is a WITH relation that holds between the EATING and Frank, and that this relation is being classified as the ACCOMPANIER relation. The phrase "a WITH" [relation] is used instead of "the WITH" [relation] because there may be more than one such relation. "The ACTOR" indicates that there can be only one actor for an action.

```
> (do-story)

          Spaghetti Dinner #2


[ 1] John ate spaghetti with a fork.

Rep: (EATING (ACTOR ← JOHN) (PATIENT ← SPAGHETTI) (WITH ← A FORK))

Inferring: the ACTOR of the EATING must be the eater
   This is a RELATION-CLASSIFICATION inference.
```

```
Inferring: the PATIENT of the EATING must be the eaten
   This is a RELATION-CLASSIFICATION inference.

Int: (EATING.1 (↑ EATING) (eater ← PERSON.1) (eaten ← SPAGHETTI.1)
               (with ← FORK.1))

Passing Markers and Suggesting Inferences:  1

Inferring: a WITH of the EATING is probably a INSTRUMENT
   because the FORK fits it best.
   This is a RELATION-CONCRETION inference.
   It is #1, due to the collision:
   EATING.1→filler→PHYS-OBJ←elaboration←EATING.1
```

We now go on to the next text, where again there is only one suggestion:

```
> (do-story)

          Spaghetti Dinner #3


[ 1] John ate spaghetti with gusto.

Rep: (EATING (ACTOR ← JOHN) (PATIENT ← SPAGHETTI) (WITH ← GUSTO))

Inferring: the ACTOR of the EATING must be the eater
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the EATING must be the eaten
   This is a RELATION-CLASSIFICATION inference.

Int: (EATING.1 (↑ EATING) (eater ← PERSON.1) (eaten ← SPAGHETTI.1)
               (with ← GUSTO.1))

Passing Markers and Suggesting Inferences:  1

Inferring: a WITH of the EATING is probably the MANNER
   because the GUSTO fits it best.
   This is a RELATION-CONCRETION inference.
   It is #1, due to the collision:
   EATING.1→filler→ATTITUDE←elaboration←EATING.1
```

```
> (do-story)

          Spaghetti Dinner #4
```

```
[ 1] John ate spaghetti with pesto.

Rep: (EATING (ACTOR ← JOHN) (PATIENT ← SPAGHETTI (WITH ← PESTO)))

Inferring: the ACTOR of the EATING must be the eater
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the EATING must be the eaten
   This is a RELATION-CLASSIFICATION inference.

Int: (EATING.1 (↑ EATING) (eater ← PERSON.1) (eaten ← SPAGHETTI.1))

Passing Markers and Suggesting Inferences:  1

Inferring: a WITH of the SPAGHETTI is probably the SAUCE
   because the PESTO fits it best.
   This is a RELATION-CONCRETION inference.
   It is #1, due to the collision:
   PESTO.1→filler→FOOD←elaboration←PESTO.1
```

In #4, the "with" modifies the spaghetti, not the eating. Thus the accompanier, instrument, and manner interpretations are not open, since they only hold for actions. The only possibility remaining is the default "along with" interpretation. However, there is a specific version of this relation, which represent the fact that sauces go with other foods. This is the interpretation accepted. Charniak [28] has shown that marker-passing techniques can be used to decide the proper attachment for prepositional phrases, but FAUSTUS cannot address this problem since it is not integrated with the parser, and there is no way to get the parser to produce a representation that is neutral as to attachment.

Step 4: Evaluating Suggestions

As stated earlier, suggestions are placed on a queue called the agenda, and then evaluated after marker passing has been completed for an input. This section explains the details of how the queue is maintained, and how individual suggestions are evaluated.

First, I review the FAUSTUS inferencing algorithm to put the evaluation mechanism in perspective. The marker passing mechanism spreads markers from concepts in the representation of the input to concepts in the knowledge base, and detects marker collisions. Each collision is classified according to its marker path shape. Each collision is then tested according to the rules for its class. If the collision passes the tests, a suggestion is placed on the agenda. Each suggestion contains an indication of where it came from (the marker paths leading to the collision), the inference class, the time it entered the agenda, the particular inference being suggested, a priority, and a list of other suggestions it may be in competition with.

After marker passing has completed, the suggestions on the agenda are evaluated. The result can either be an inference – something new to add to the representation of the text – or it can be a decision to reject the suggestion and make no inference, or to defer and try the suggestion again after the next input.

The main reason for maintaining an agenda rather than just evaluating each suggestion as soon as it is detected is that some suggestions cannot be evaluated in their own right, but must be compared to other, competing suggestions. For example, in text (1), the bicycle story, there were two competing suggestions, one that 'he' referred to John, and the other that 'he' referred to Bill. Both suggestions are plausible, and either one would be accepted if evaluated separately. Therefore, we have to evaluate them together, and choose the best possibility. Evaluation becomes a question of relative plausibility, rather than absolute yes/no acceptance.

All the inference classes offer the possibility of competing inferences. Referents can compete for the same reference, fillers can compete to elaborate the same aspectual, and we can have multiple possible views or concretions of a concept. When a suggestion is put in the agenda, the suggestion also places a pointer to itself in the a-list of the appropriate concept. For example, when the suggestion that 'he' refers to John is placed in the agenda, a pointer to the suggestion is placed under the ''referent'' indicator in the a-list of the concept representing this instance of 'he.' Later, when the suggestion that 'he' refers to Bill is placed in the agenda, it looks at this a-list entry, sees there is something already there, and keeps track of the fact that these two suggestions are now competing with one another. The agenda evaluation procedure can then arrange to evaluate competing suggestions together as a group, and can apply the rules for making a choice among alternatives. The procedure also arranges to evaluate all single suggestions before those with competing alternatives. The rationale for this is that we want to have all possible information available when we are forced to make a choice. By waiting as long as possible, it may be that other suggestions have added information to the representation of the text that can help make the choice.

FAUSTUS has provisions for ordering suggestions according to a priority factor, which can take into account the length of time waiting in the agenda, the inference class, and the list of competing suggestions. Currently this feature is used for only two cases. First, competing sets of suggestions are put at the end of the agenda, as stated above. Second, we place all single elaboration inferences before double elaborations. This means the system will prefer to fill slots with existing objects rather than invent new ones.

It is useful to think of the suggestion mechanism as a three-part filter. Out of the infinite number of possible inferences that could be made, we filter out completely irrelevant ones by only considering inferences due to particular types of marker collisions. Then, out of all the possible inferences due to collisions, we filter out impossible ones. For example, if 'John,' 'Mary,' and 'he' have been mentioned in the text, then there will be a referential collision with markers originating at 'Mary' and 'he' and colliding at the concept `animal`. But this will not suggest an inference because 'Mary' and `male` are in mutually disjoint portions of the hierarchy. Finally, the third filter is a more stringent check of suggestions at evaluation time. A suggestion is made unless there is a blatant contradiction – such as two concepts being in mutually disjoint categories – but the suggestion is only accepted after a full check for more subtle contradictions. As an example, consider again text (1), repeated here as (19):

(19a) Bill had a bicycle.
(19b) John wanted it.
(19c) He gave it to him.

Two suggestions will be that 'he' refers to 'John' and to 'Bill.' Neither of these suggestions could be eliminated by checking the hierarchy, as both John and Bill are males. However, another suggestion is the following inference, which was accepted:

```
Inferring: the HAVING mentioned in [1] is a PRECONDITION of the GIVING.
   This is a SINGLE-ELABORATION inference.
   It is #10, due to the collision:
   GIVING.3→elaboration→STATIVE←ref←HAVING.1
```

This states that the having of the bicycle enables the giving of the bicycle. Giving is defined as a kind of transferring, and transferring is defined as follows.

```
(A TRANSFERRING (↑ ACTION-PROCESS)
  (donor SENTIENT-AGENT (↑ participant))
  (recipient SENTIENT-AGENT (↑ participant))
  (transferred INANIMATE (↑ patient))
  (transferring$result RECIPIENT-HAVING-OBJECT (↑ result) 1 1)
  (transferring$precondition DONOR-HAVING-OBJECT (↑ precondition) 1 1)
  (= recipient (haver transferring$result))
  (= transferred (had transferring$result))
  (= donor (haver transferring$precondition))
  (= transferred (had transferring$precondition))
  (≠ TAKING GETTING GIVING))
```

The important line here is the third from the bottom, which says that the donor of a transferring event must be the same as the haver of the precondition of the transferring. So after inference #10 is asserted, the donor of the giving, 'he,' is equated with the haver of the having, 'Bill.' The evaluation procedure checks equate links, and thus can accept Bill and reject John as the referent of 'he.'

Actually, the matching procedure does more than check equate links. Two concepts match if they are the same identical object, or if they are equated. They fail to match if they are connected by a differ link, or if they are in disjoint portions of the hierarchy. Otherwise, they match if there are no mis-matches among the relations they participate in. This is harder to compute in KODIAK than in frame-based languages, because we have to consider not only the slots of a concept, but also the slots the concepts fills. The match procedure looks at all the relations for one concept, and finds the corresponding relation(s) for the other concept. This comparison may result in an immediate match, if one concept has a relation filled by some instance, and the other concept just has a constraint saying that kind of concept is acceptable. It may result in an immediate mis-match, if the constraint is violated. Or, it may result in the need for a recursive match between two concepts involved in the relation. In that case, the procedure first checks to see if a match has been requested between these two concepts before. If so, the recursion is stopped, to avoid infinite loops. The recursive match makes the assumption that it should return true, with the expectation that eventually the original call to match will return something. If it returns true, then the assumption was warranted, and if not, then the whole match fails, and we still have the correct result.

I now cover the evaluation rules for resolving competing suggestions. For each inference class there is a preliminary check using the match procedure, where some of the competing suggestions can be ruled out immediately. If there is only one suggestion that survives this check, then it is accepted. Otherwise, the following rules hold.

Evaluating Competing Elaboration Suggestions

If more than one filler is suggested for a slot, and one matches the constrainer of the slot in more respects than any others, then it is accepted. This can happen, for example, if the constrainer is dominated by two different concepts, and if only one of the possible fillers is explicitly dominated by both (even if the others are compatible with, or match, both dominators). If there is no best match, all suggestions are deferred.

Evaluating Competing Reference Suggestions

Reference suggestions can also be resolved by finding one referent that matches better than any other. If there is no best match, the next step is to choose the most recently mentioned of the top-scoring matchers. If there are still several possibilities, then if exactly one referent participates in a focused slot relation (such as the actor of an action), then it is chosen. Otherwise, the suggestions are deferred.

Evaluating Competing View Application Suggestions

Since views are related to each other hierarchically, it may be that one suggested view is just a specialization of another. If this is the case, choose the more specific applicable view. After that, the rule is to pick the view where the resulting (viewed-as) category matches the source concept the best. If there is no best match, defer making a decision.

Evaluating Competing Concretion Suggestions

Concretion suggestions are somewhat different in that a given concept can be concreted in several directions at once, so not all concretions for a concept are necessarily in conflict. Thus, the evaluation rule for this class partitions the competing suggestions according to the hierarchy, and chooses the best match from each partition. There is no tie-breaking procedure; the default is to defer and make no decision.

Summary

In this chapter, we have seen how to create the knowledge base and to represent individual inputs. The marker-passing, inference suggestion, and suggestion evaluation parts of the inferencing algorithm were also presented. The algorithm makes use of six inference classes motivated by specific marker path shapes, as well as two non-marker-passing inference classes. The next chapter will present several texts that can be processed using this algorithm.

Chapter 5: Further Examples

As discussed in Chapter 2, there have been many past efforts to make inferences from text. In this chapter, I will show how the FAUSTUS system is capable of equaling the inferencing capabilities of some of these systems. For some of the examples, I introduce a new analysis of the text. In other examples, the contribution here is not in making a new analysis, but rather in FAUSTUS' ability to integrate the capabilities of several systems into one. If FAUSTUS can process stories about Bill's bicycle, Chang's fishing, and John's dining, if it is possible to share knowledge between the stories, and if the knowledge added to process a new story does not interfere with the processing of an old story, then that is a significant finding.

The purpose of this chapter is four-fold: it provides examples that add detail to the ideas expressed in Chapters 3 and 4; it shows that FAUSTUS can be applied to a wide range of problems; it shows how FAUSTUS can accommodate different knowledge sources in a uniform manner, and finally it shows some problems that FAUSTUS cannot handle. The example texts were taken from or suggested by work done by previous researchers. We will proceed in roughly chronological order.

Unconstrained Sentence-Based Inferencing

Several early AI systems addressed the problem of generating all plausible inferences from a single sentence input. One of the first was Quillian's Teachable Language Comprehender [93], or TLC, which took as input single noun phrases or simple sentences, and related them to what was already stored in semantic memory. For example, given the input "lawyer for the client," the program could output "at this point we are discussing a lawyer who is employed by a client who is represented or advised by this lawyer in a legal matter." The examples given in [93] show an ability to find the main relation between two concepts, but not to go beyond that. One problem with TLC was that it ignores the grammatical relations between objects until the last moment, when it applies "form tests" to rule out certain inferences. For the purposes of generating inferences, TLC treats the input as if it had been just *"Lawyer. Client"*. Quillian suggests this could lead to a potential problem. He presents the following examples:

enemy's lawyer	lawyer's enemy
wife's lawyer	lawyer's wife
client's lawyer	lawyer's client
lawyer for the enemy	enemy of the lawyer
lawyer for the wife	wife of the lawyer
lawyer for the client	client of the lawyer

In all the examples on the left hand side, the lawyer is employed by someone. However, among the examples on the right hand side, only the two mentioning *client* should include the employment relation as part of the interpretation. While he suggests a solution in general terms, Quillian admits that TLC as it stood could not handle these examples.

In trace output #1 below, the `for` relation in *lawyer for the client* is first classified as an `employed-by`, because a lawyer is defined as a `professional-service-provider`, which includes an `employed-by` slot as a specialization of the `for` slot. After that a double elaboration inference finds that an `employing-event` can mediate between an employer and an employee. The second example shows that, like TLC, FAUSTUS can find this connection without the `for` relation.

In the third and fourth examples, we see FAUSTUS' solution to Quillian's quandary. In order to get the right connection for examples like #3, *lawyer for the enemy*, Quillian had to define the employ-lawyer relation to hold between a lawyer and any person. In FAUSTUS I define the employing relation as holding between an employer and an employee, and state separately that a client is an employer, and a lawyer is an employee. In #3, then, it is the interpretation of the `for` relation that adds the assertion that the enemy is an employer. After that, the double elaboration inference is free to go through. In #4, the enemy is not interpreted as an employer, so no inference is made.

The important point is that FAUSTUS has a better way of combining information from syntax and semantics. Both TLC and FAUSTUS suggested inferences by spreading markers from all components of the input, and looking for collisions. The difference is that TLC used syntactic relations only as a filter to eliminate certain suggestions, while FAUSTUS incorporates the meaning of these relations into the representation before spreading markers.

```
> (do-story)

          Quillian #1


[ 1] lawyer for the client

Rep: (LAWYER (FOR ← THE CLIENT))

Inferring: a FOR of the LAWYER must be the EMPLOYED-BY
   This is a RELATION-CLASSIFICATION inference.

Int: (LAWYER.1 (↑ LAWYER) (employed-by ← CLIENT.1))

Inferring: there is a EMPLOYING-EVENT such that
   the CLIENT is the EMPLOY-ER of it and
   the LAWYER is the EMPLOY-EE of it.
   This is a DOUBLE-ELABORATION inference.
   It is #2, due to the collision:
   CLIENT.1→elaboration→EMPLOYING-EVENT←elaboration←LAWYER.1
```

```
> (do-story)

          Quillian #2


[ 1] Lawyer.

Rep: (LAWYER)

Int: (LAWYER.1 (↑ LAWYER))

[ 2] Client.

Rep: (CLIENT)

Int: (CLIENT.2 (↑ CLIENT))

Inferring: there is a EMPLOYING-EVENT such that
   the CLIENT is the EMPLOY-ER of it and
   the LAWYER is the EMPLOY-EE of it.
   This is a DOUBLE-ELABORATION inference.
   It is #2, due to the collision:
   CLIENT.2→elaboration→EMPLOYING-EVENT←elaboration←LAWYER.1
```

```
> (do-story)

          Quillian #3


[ 1] lawyer for the enemy

Rep: (LAWYER (FOR ← THE ENEMY))

Inferring: a FOR of the LAWYER must be the EMPLOYED-BY
   This is a RELATION-CLASSIFICATION inference.

Int: (LAWYER.1 (↑ LAWYER) (employed-by ← ENEMY.1))

Inferring: there is a EMPLOYING-EVENT such that
   the ENEMY is the EMPLOY-ER of it and
   the LAWYER is the EMPLOY-EE of it.
   This is a DOUBLE-ELABORATION inference.
   It is #2, due to the collision:
   ENEMY.1→elaboration→EMPLOYING-EVENT←elaboration←LAWYER.1
```

```
> (do-story)

          Quillian #4


[ 1] enemy of the lawyer

Rep: (ENEMY (OF ← THE LAWYER))

Int: (ENEMY.1 (↑ ENEMY) (of ← LAWYER.1))

Passing Markers and Suggesting Inferences:  1

Inferring: a OF of the ENEMY is probably a RELATED-TO
   because the LAWYER fits it best.
   This is a RELATION-CONCRETION inference.
   It is #1, due to the collision:
   ENEMY.1→filler→PERSON←elaboration←ENEMY.1
```

A more elaborate inferencing program was the MARGIE system of Schank, Goldman, Rieger, and Riesbeck [107]. MARGIE performed two separate tasks; paraphrasing sentences, which we will not be concerned with here, and generating inferences, which is relevant. The program took one complete sentence (with restricted syntax) at a time, and generated a list of plausible inferences. An example follows:

INPUT: John told Mary that Bill wants a book. OUTPUT1: A book about what? OUTPUT2: Mary knows that Bill wants a book. OUTPUT3: Bill wants to come to have a book. OUTPUT4: Bill wants someone to cease to have a book. OUTPUT5: Bill wants to read a book. Each of these inferences is reasonable, although it is not clear they necessarily constitute the best set of inferences. For example, it is appropriate in output 1 to wonder about the subject of the book, but it would also be appropriate to wonder if Bill wants a particular title (he wants *Lord Jim*), or if he wants a particular physical instance of a book (he wants that one on the table). Output 4 is particularly suspect; Bill probably does not care if someone else is deprived of a book, and would be just as happy if the book appeared out of thin air without anyone ceasing to have it.

In summary, there are a large number of quibbles one could make about the inferences made by MARGIE. One reason for this is that the input is an isolated sentence, and it is not at all clear what the purpose of the sentence is. If the sentence were imbedded in a larger context, it would be easier to decide which inferences were relevant.

FAUSTUS, it turns out, can make none of the inferences listed in output1-5. FAUSTUS was based on the assumption that the input will be coherent, and the program's job is to find relations between concepts in the input. In this example, output1-5 consist mostly of speculations of where the text might go next, rather than how the text itself coheres. FAUSTUS does not spontaneously generate inferences like output1-5, but rather waits until they are needed to tie the text together. Thus, if the next sentence were *She gave one to him*, FAUSTUS would infer that *she* refers to Mary, *one* refers to a book, and *him* refers to Bill. Furthermore, Mary gave it because Bill wanted it, there is a new situation wherein Bill has the book, and this situation is the result of Mary giving the book, and satisfies Bill's wanting the book. This set of inferences is isomorphic to the set generated by sentence 3 in the *Bill's bicycle* story presented as the first example in Chapter 4.

Despite the fact that FAUSTUS was not designed to handle single-sentence inputs it does perform adequately on the *Lawyer/Client* examples above. We would like an explanation of why it can handle those examples, but not the *book* example. The difference is precisely the existence of the mediating relation `employing-event`, which forms a bridge between `lawyer` and `client`. If there were a `wanting-to-read` that described the situation wherein someone wants a book in order to read it, then FAUSTUS could duplicate output5. The program could correctly infer that it is Bill, not John or Mary, who wants to read a book, and that the book he wants to read is the same book he wants to have.

Script-Based Inferencing

Scripts had to be introduced originally as a separate type of knowledge structure to extend the semantics of Schank's Conceptual Dependency knowledge representation. CD relied heavily on a small set of primitive acts, and it was difficult to organize and contain the inferences that arise from non-primitives. There was a primitive, INGEST which provided a place to store information about substances entering a body, but there was no good place to store information about a specific kind of ingest, like eating, let alone eating at a restaurant. Thus, scripts were introduced as an extension of primitive acts, and MOPs as an extension of scripts. In KODIAK, there are no such arbitrary distinctions, so `eating-at-a-restaurant` is just another event, much like `eating` or `walking`, except that it is more complex, involving multiple agents and multiple substeps, with relations between the steps. Scripts needed the notion of *tracks*, like the fast-food track of the restaurant script, to accommodate variation. In KODIAK possible variants in a script are represented using the normal inheritance hierarchy mechanism.

We do not need any additional mechanisms or formalism to define scripts, or tracks of scripts, but we do still need to represent the same knowledge, in one fashion or another. The following is FAUSTUS' representation of some knowledge related to eating at a restaurant. Because there are many related facts, I use the textual description (as described in Chapter 3) rather than diagrams. In this notation the first line of each expression defines a new absolute. For example, `(A EAT-AT-RESTAURANT (↑ EATING`

CONTRACTUAL-EVENT) means that eat-at-restaurant is an absolute dominated by the concepts eating and contractual-event. A line of the form (diner SENTIENT-AGENT (↑ eater)) says that diner is a slot of eat-at-restaurant, it is dominated by eater, and is constrained to be a sentient-agent. If there are numbers at the end of such an expression, as in (eating$result EATER-IS-FULL (↑ result) 1? 1), they are interpreted as follows. The first 'number,' 1?, means that as a result of a given act of eating, at most one person gets full. The expression 1? is an abbreviation for the range 0 to 1. The second number, 1, means that for a given eater-is-full situation, it can be the result of exactly one eating. An equation of the form (= food-role (patient ordering-food-step)) means that whatever fills the food-role of an eat-at-restaurant must also fill the patient slot of whatever fills the ordering-food-step slot. In the definitions below, facts about food, eating, transferring things from one place to another, and contractual obligations are defined separately from the eat-at-restaurant scenario, but are referred to by it. Thus, my representation is closer to the MOPS approach than to scripts.

```
(A EAT-AT-RESTAURANT (↑ EATING CONTRACTUAL-EVENT)
  (diner SENTIENT-AGENT (↑ eater))
  (waiter-role WAITER (↑ participant))
  (food-role FOOD (↑ patient))
  (eat-at-restaurant$setting RESTAURANT (↑ setting))
  (going-to-restaurant-step TRAVELING-TO-RESTAURANT (↑ step) 1 1?)
  (ordering-food-step ORDERING-R-FOOD (↑ step) 1 1)
  (food-arrives-step TRANSFERRING-R-FOOD-TO-TABLE (↑ step) 1 1?)
  (main-restaurant-step EATING-R-FOOD (↑ step) 1 1?)
  (pay-for-food-step PAYING-FOR-R-FOOD (↑ step) 1 1?)
  (leaving-restaurant-step TRAVELING-FROM-RESTAURANT (↑ step) 1 1?)
  (= eat-at-restaurant$setting (destination going-to-restaurant-step))
  (= eat-at-restaurant$setting (source leaving-restaurant-step))
  (= food-role (patient ordering-food-step))
  (= food-role (patient food-arrives-step))
  (= food-role (patient main-restaurant-step))
  (= food-role (merchandise pay-for-food-step))
  (= diner (traveler going-to-restaurant-step))
  (= diner (traveler leaving-restaurant-step))
  (= diner (party1 eat-at-restaurant))
  (= waiter-role (party2 eat-at-restaurant))
  (= food-arrives-step (party1s-obligation eat-at-restaurant))
  (= pay-for-food-step (party2s-obligation eat-at-restaurant)))

(A EATING (↑ ACTION-PROCESS)
  (eater SENTIENT-AGENT (↑ actor))
  (eaten FOOD (↑ patient))
  (eating$result EATER-IS-FULL (↑ result) 1? 1)
  (eating$precondition EATER-IS-HUNGRY (↑ precondition) 1 1)
  (= eater (experiencer eating$result))
  (= eater (experiencer eating$precondition))
  (≠ EAT-AT-HOME EAT-AT-RESTAURANT))
```

```
(A FOOD (↑ FUNC-OBJ)
  (food$purpose EATING (↑ purpose) 1+)
  (food$with-other-food FOOD (↑ with-modifier) 0+)
  (food$sauce SAUCE (↑ food$with-other-food) 0+)
  (≠ FISH HAMBURGER SPAGHETTI HOT-DOG SAUCE SOUP OTHER-FOOD))

(A FORK (↑ FUNC-OBJ)
  (fork$purpose EATING (↑ purpose) 1+))

(A FISH (↑ ANIMAL FOOD)
  (fish$body-covering SCALY (↑ body-covering))
  (fish$habitat BODY-OF-WATER (↑ habitat)))

(A STORE (↑ BUILDING FUNC-OBJ)
  (≠ RESTAURANT SHOE-STORE FISH-MARKET))

(A RESTAURANT (↑ STORE)
  (restaurant$purpose EATING (↑ purpose)))

(A TRANSFERRING (↑ ACTION-PROCESS)
  (donor SENTIENT-AGENT (↑ participant))
  (recipient SENTIENT-AGENT (↑ participant))
  (transferred INANIMATE (↑ patient))
  (transferring$result RECIPIENT-HAVING-OBJECT (↑ result) 1 1)
  (transferring$precondition DONOR-HAVING-OBJECT (↑ precondition) 1 1)
  (= recipient (haver transferring$result))
  (= transferred (had transferring$result))
  (= donor (haver transferring$precondition))
  (= transferred (had transferring$precondition)))

(A CONTRACTUAL-EVENT (↑ EVENT)
  (party1 SENTIENT-AGENT (↑ participant))
  (party2 SENTIENT-AGENT (↑ participant))
  (party1s-obligation OBLIGATION (↑ step) 1+)
  (party2s-obligation OBLIGATION (↑ step) 1+)
  (= party1 (actor party1s-obligation))
  (= party2 (actor party2s-obligation)))

(A COMMERCIAL-EVENT (↑ CONTRACTUAL-EVENT)
  (customer SENTIENT-AGENT (↑ party2))
  (merchant SENTIENT-AGENT (↑ party1))
  (merchandise COMMODITY (↑ prop))
  (price MONEY (↑ prop))
  (customers-obligation OBLIGATION (↑ party1s-obligation))
  (merchants-obligation OBLIGATION (↑ party2s-obligation))
  (= customer (donor customers-obligation))
  (= merchant (recipient customers-obligation))
  (= price (given customers-obligation))
  (= customer (recipient merchants-obligation))
  (= merchant (donor merchants-obligation))
  (= merchandise (given merchants-obligation))
  (≠ STORE-TRANSACTION FREELANCE-TRANSACTION))

(A PAYING (↑ COMMERCIAL-EVENT TRANSFERRING)
  (payer SENTIENT-AGENT (↑ actor customer donor))
  (payee SENTIENT-AGENT (↑ recipient merchant)))

(A WAITER (↑ PERSON EMPLOYEE)
  (waiter$occupation WAITERING (↑ occupation) 0+))
```

The theory of scripts addressed two basic problems: script recognition and script application. Recognition means deciding what script(s) are appropriate to a given situation. In SAM, finding the proper script was accomplished by some *ad hoc* tricks involving keywords, but this was not intended as a serious proposal. The point of the program was script *application*: given a script to apply to a situation, SAM tried to make a set of inferences. FAUSTUS is able to do better at script recognition. Without specifically marking words like "restaurant" or "waiter" as keywords which invoke scripts, the program is able to use information associated with these words when appropriate, and find connections to events in the text. Unlike SAM, FAUSTUS does not automatically infer an entire script whenever it sees a keyword. Thus, FAUSTUS could handle *John walked past a restaurant* without inferring that he ordered, ate, and paid for a meal.

As for script application, FAUSTUS performs adequately, but for different reasons than SAM. Consider the following example:

```
> (do-story)

          The Waiter


[ 1] John was eating at a restaurant with Mary.

Rep: (EATING (ACTOR ← JOHN) (SETTING ← A RESTAURANT) (WITH ← MARY))

Inferring: the ACTOR of the EATING must be the EATER
   This is a RELATION-CLASSIFICATION inference.

Int: (EATING.1 (↑ EATING) (eater ← PERSON.1) (setting ← RESTAURANT.1)
               (with ← PERSON.1B))

Passing Markers and Suggesting Inferences:  1 2

Evaluating Inferences:
```

```
Inferring: a WITH of the EATING is probably the ACCOMPANIER
   because Mary fits it best.
   This is a RELATION-CONCRETION inference.
   It is #1, due to the collision:
   EATING.1→filler→PERSON←elaboration←EATING.1

Inferring: the EATING is a EAT-AT-RESTAURANT.
   This is a CONCRETION inference.
   It is #2, due to the collision:
   RESTAURANT.1→filler→EATING←elaboration←RESTAURANT.1

[ 2] The waiter spilled soup all over her.

Rep: (SPILLING (ACTOR ← THE WAITER) (PATIENT ← SOUP) (RECIPIENT ←
HER))

Inferring: the ACTOR of the SPILLING must be the SPILLER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the SPILLING must be the SPILLED
   This is a RELATION-CLASSIFICATION inference.

Inferring: the RECIPIENT of the SPILLING must be the SPILLEE
   This is a RELATION-CLASSIFICATION inference.

Int: (SPILLING.2 (↑ SPILLING) (spiller ← WAITER.2) (spilled ← SOUP.2)
        (spillee ← FEMALE.2))

Passing Markers and Suggesting Inferences:  3 4 5 6 7 8 9 10 11

Evaluating Inferences:

Inferring: there is a EAT-AT-RESTAURANT such that
   the SOUP is the FOOD-ROLE of it and
   the RESTAURANT is the SETTING of it.
   This is a DOUBLE-ELABORATION inference.
   It is #3, due to the collision:
   SOUP.2→elaboration→EAT-AT-RESTAURANT←elaboration←RESTAURANT.1

Inferring: there is a EATING such that
   the SOUP is the EATEN of it and
   it is the PURPOSE of the RESTAURANT.
   This is a DOUBLE-ELABORATION inference.
   It is #4, due to the collision:
   SOUP.2→elaboration→EATING←elaboration←RESTAURANT.1

Inferring: there is a EAT-AT-RESTAURANT such that
```

```
   the WAITER is the WAITER-ROLE of it and
   the SOUP is the FOOD-ROLE of it.
   This is a DOUBLE-ELABORATION inference.
   It is #5, due to the collision:
   WAITER.2→elaboration→EAT-AT-RESTAURANT←elaboration←SOUP.2

Inferring: there is a COMMERCIAL-EVENT/a EAT-AT-RESTAURANT such that
   the SOUP is the MERCHANDISE of it and
   the RESTAURANT is the SETTING of it.
   This is a DOUBLE-ELABORATION inference.
   It is #7, due to the collision:
   SOUP.2→elaboration→CONTRACTUAL-EVENT←elaboration←RESTAURANT.1

Inferring: there is a EMPLOYING-EVENT/a COMMERCIAL-EVENT such that
   the WAITER is the EMPLOY-EE of it and
   the SOUP is the MERCHANDISE of it.
   This is a DOUBLE-ELABORATION inference.
   It is #9, due to the collision:
   WAITER.2→elaboration→CONTRACTUAL-EVENT←elaboration←SOUP.2

Inferring: there is a BUYING/a STORE-TRANSACTION such that
   the SOUP is the THING-BOUGHT of it and
   the RESTAURANT is the SETTING of it.
   This is a DOUBLE-ELABORATION inference.
   It is #10, due to the collision:
   SOUP.2→elaboration→COMMERCIAL-EVENT←elaboration←RESTAURANT.1
```

These inferences are shown graphically in Figure 1 and 2 (before and after). The set of inferences made seems reasonable, but it is instructive to contrast them with the inferences SAM would have made on this text. SAM would first notice the word *restaurant* and fetch the eat-at-restaurant script. This script is essentially a linear list of everything that happens in a typical visit to a restaurant, with some allowance for variation. The text is processed by comparing each input to the successive steps in the script. When the input matches a step, the input is unified with that step. This can lead to the resolution of pronominal references and other inferences. In addition, all intervening steps are also inferred. Thus, from the second sentence, SAM would infer that John and Mary entered the restaurant, walked to a table, sat down, got a menu, read the menu, and ordered soup. This approach does not allow for varying level of detail; it forces the program to make all conceivable inferences.

Since FAUSTUS has no global control, there is nothing to tell it to fill in all steps of a script. Instead, it can only make inferences that tend to connect other parts of the representation of the story. In this story, it finds connections to the waiter and soup, but makes no inferences about entering or leaving the restaurant. These are still implicit in the definition of eat-at-restaurant, and we could in principle ask for the filler of the actor slot of the entering-restaurant-step, and FAUSTUS would build a representation showing

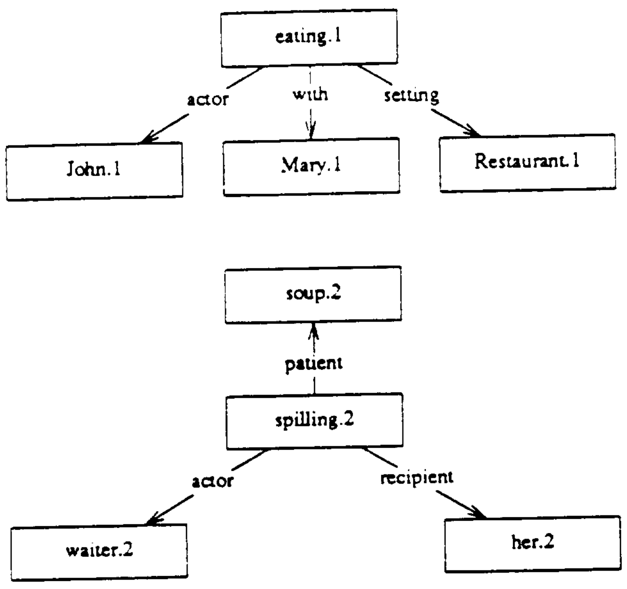

Figure 1: "The Waiter": Input

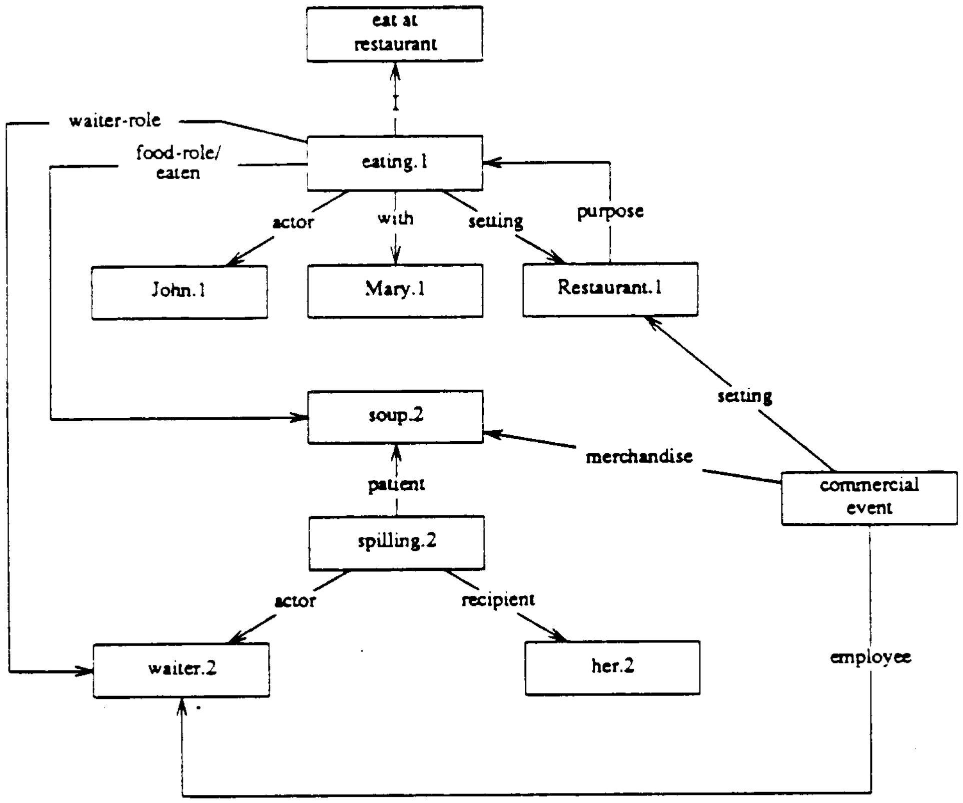

Figure 2: "The Waiter": Inferences

that John entered the restaurant. FAUSTUS has no natural language query system to facilitate this, although there are KODIAK functions that can be called to make these queries.

We can also see that FAUSTUS handles failed scripts properly. Given the *nonsequitur* story (1), taken from [110], FAUSTUS makes the usual relation classification inferences, and resolves the pronominal references, but suggests no other inferences.

(1a) John went to a park.
(1b) He asked the midget for a mouse.
(1c) He picked up the box and left.

```
> (do-story)

          The Midget in the Park


[ 1] John went to a park.
```

```
Rep: (TRAVELING (ACTOR ← JOHN) (DESTINATION ← A PARK))

Inferring: the ACTOR of the TRAVELING must be the TRAVELER
   This is a RELATION-CLASSIFICATION inference.

Int: (TRAVELING.1 (↑ TRAVELING) (traveler ← PERSON.1)
                  (destination ← PARK.1))

Passing Markers and Suggesting Inferences:

[ 2] He asked the midget for a mouse.

Rep: (ASKING (ACTOR ← HE) (PATIENT ← THE MIDGET) (CONTENT ← A MOUSE))

Inferring: the ACTOR of the ASKING must be the TALKER
   This is a RELATION-CLASSIFICATION inference.

Inferring: 'HE' must be a PERSON, because it is the TALKER
   This is a RELATION-CONSTRAINT inference.

Int: (ASKING.2 (↑ ASKING) (talker ← MALE.2) (patient ← MIDGET.2)
               (content ← MOUSE.2))

Passing Markers and Suggesting Inferences:  1 2

Evaluating Inferences:

Inferring: 'HE' refers to John.
   because it is the best match.
   This is a REFERENCE inference.
   It is #1, due to the collision:
   PERSON.1→ref→PERSON←ref←MALE.2

Rejecting: 'HE' refers to the MIDGET.
   because it is not the best match.
   This is a REFERENCE inference.
   It is #2, due to the collision:
   MIDGET.2→ref→PERSON←ref←MALE.2

[ 3] He picked up the box

Rep: (GRASPING (ACTOR ← HE) (PATIENT ← THE BOX))

Inferring: the ACTOR of the GRASPING must be the TAKER
   This is a RELATION-CLASSIFICATION inference.

Inferring: 'HE' must be a SENTIENT-AGENT, because it is the TAKER
```

```
   This is a RELATION-CONSTRAINT inference.

Inferring: the PATIENT of the GRASPING must be the TOOK
   This is a RELATION-CLASSIFICATION inference.

Int: (GRASPING.3 (↑ GRASPING) (taker ← MALE.3) (took ← BOX.3))

Passing Markers and Suggesting Inferences:  3 4

Evaluating Inferences:

Inferring: the 'HE' mentioned in [3] refers to John.
   because it is the best match.
   This is a REFERENCE inference.
   It is #3, due to the collision:
   PERSON.1→ref→PERSON←ref←MALE.3

Rejecting: the 'HE' mentioned in [3] refers to the MIDGET.
   because it is not the best match.
   This is a REFERENCE inference.
   It is #4, due to the collision:
   MIDGET.2→ref→PERSON←ref←MALE.3

[ 4] and left.

Rep: (DEPARTING (ACTOR ← ^ MALE))

Inferring: the ACTOR of the DEPARTING must be the MOVER
   This is a RELATION-CLASSIFICATION inference.

Int: (DEPARTING.4 (↑ DEPARTING) (mover ← MALE.3))

Passing Markers and Suggesting Inferences:
```

The question remains of how well FAUSTUS can handle script recognition. It has an advantage over SAM in that it need not instantiate a complete script every time it hears a keyword, like "restaurant" in "John walked past a restaurant." Problems arise when more than one script is applicable at the same time. The program is good at recognizing clues for a script locally, but cannot make a good global choice between two or more scripts when there are multiple clues for each possibility. Consider the following examples:

(2a) Mary walked down the aisle.
(2b) She picked up a can of tuna from the first pew.

The idea is that we want to be able to disambiguate (2a). The "tuna" leads to a double-elaboration collision that suggests there is a supermarket-shopping event such that the walking down the aisle is a step of it, and the tuna is the merchandise of the shopping. Another collision suggests the walking is a step of a church-wedding event, where the pew is a fixture of the church. FAUSTUS is unable to reason that it is more likely for a can of tuna to unexpectedly be in a church than for a pew to be in a supermarket. FAUSTUS does not realize that the two suggestions are in competition with each other, and would accept whichever suggestion happened to be evaluated first, and reject the other one. This does not seem like a major failing, as texts like (2) are problematic even for human understanders.

Plan-Based Inferencing

In the previous section we saw that FAUSTUS was able to make what have been called "script-based inferences" without any explicit script-processing control structure. This was enabled partially by adding causal information to the representation of script-like events (like eating at a restaurant). The theory of plans and goals as they relate to story understanding, specifically the work of Wilensky [131], was also an attempt to use causal information to understand stories that could not be comprehended using scripts alone. Consider story (3):

(3a) John was lost.
(3b) He pulled over to a farmer standing by the side of the road.
(3c) He asked him where he was.

Wilensky's PAM program processed this story as follows: from (3a) it infers that John will have the goal of knowing where he is. From that it infers he is trying to go somewhere, and that going somewhere is often instrumental to doing something there. From (3b) PAM infers that John wanted to be near the farmer, because he wanted to use the farmer for some purpose. This rather vague inference constitutes the explanation of (3b). Finally (3c) is processed. It is recognized that asking is a plan for knowing, and since it is known that John has the goal of knowing where he is, there is a match, and (3c) is explained. As a side effect of the matching process, the three pronouns in (3c) are disambiguated. Besides resolving the pronouns, the two key inferences are that John has the goal of finding out where he is, and that asking the farmer is a plan to achieve that goal.

In FAUSTUS, we can arrive at the same interpretation of the story by a very different method. (3a) does not generate any expectations, as it would in PAM, and FAUSTUS cannot find a connection between (3a) and (3b), although it does resolve the pronominal reference, because John is the only possible candidate. Finally, in (3c), FAUSTUS makes the two main inferences. It recognizes that being near the farmer is related to asking him a question by a `precondition` relation (and resolves the pronominal references while making this connection). FAUSTUS could find this connection because both the asking and the being-near are explicit inputs. The other connection is a little trickier. The goal of knowing where one is was not an explicit input, but "where he was" is part of (3c), and there is a collision between paths starting from the representation of that phrase and another path starting from the asking that lead to the creation of the `plan-for` between John's asking where he is and his hypothetical knowing where he is.

The important conclusion, as far as FAUSTUS is concerned, is that both script- and goal-based processing can be reproduced by a system that has no explicit processing mechanism aimed at one type of story or another, but just looks for connections in the input as they relate to what is known in memory. For both scripts and goals, this involves defining situations largely in terms of their causal structure. This approach fails when the connections are tenuous; that is, when the story makes large jumps.

There are examples of goal-based stories that PAM handles that FAUSTUS can not. Another of Wilensky's examples is shown here as (4):

(4a) Willa was hungry.
(4b) She picked up the Michelin guide.

From this PAM infers that Willa will use the Michelin guide to choose a restaurant and find out where it is, rather than, say, to eat the guide. Making this inference is a multi-step process (in Wilensky's notation it is seven steps) and thus is beyond FAUSTUS' ability. Of some consolation is the fact that the inference is also beyond the ability of some humans.

One way to make FAUSTUS handle (4) would be to include a direct mediating relation between being hungry and picking up a restaurant guide. As Wilensky points out, it is virtually impossible, and certainly impractical to list all such possible relations in advance. Wilensky's alternative is to explicitly track the goals of each character, arriving at the answer by a problem-solving approach. It would be against the spirit of FAUSTUS to allow an inferencing procedure designed specifically for plans and goals to drive the understanding process.

There is a way to modify FAUSTUS that does not involve adding a knowledge-class-specific inferencing mechanism. As we saw in the MARGIE example above, one of the things FAUSTUS does not do is to make forward inferences from an input without any connections to other representations. However, certain assertions seem to strongly suggest forward inferences, and it is probably a mistake for FAUSTUS to ignore them. Thus, after reading *Willa was hungry*, it seems natural to infer she will want to eat something. Similarly, after reading *John told Mary something*, one would not be out of line to infer that Mary now knows what John told her. The problem of where to draw the line still remains. We still do not want to infer that Mary has a grandmother, or that she has a pancreas that secretes insulin. FAUSTUS draws the line as strictly as possible by ruling out all such forward inferences. A better approach would be to tease apart presuppositions, assertions, probable implications, and improbable implications, and just make the salient inferences. This is a topic for future research.

Coherence Relation-Based Inferencing

In this section we turn to inferences based on coherence relations, as exemplified by this example proposed by Kay and Fillmore [61]:

(5) An alpinist bought a pair of boots from a cobbler.

From the definition of `buying` one could infer that the alpinist now owns the boots that previously belonged to the cobbler and the cobbler now has some money that previously belonged to the alpinist. However, a more complete understanding of (5) should include the inference that the transaction probably took place in the cobbler's store, that the boots probably are ones that he made (or perhaps bought with intent to sell) and that the alpinist will probably use the boots in his avocation, rather than, say, give them as a gift to his sister. The first two of these can be derived from concretion inferences once we have described what goes on at a shoe store. The problem is that we want to describe this in a neutral manner-- to describe not "buying at a shoe store" which would be useless for "selling at a shoe store" or "paying for goods at a shoe store" but rather the general "shoe store transaction." This is done by first defining the `commercial-transaction` absolute, which dominates `store-transaction` on the one hand, and `buying`, `selling` and `paying` on the other. Each of these last three is also dominated by `action`. Assertions are made to indicate that the `buyer` of `buying` is both the `actor` of the `action` and the `merchant` of the `commercial-transaction`. The next step is to define `shoe-store-transaction` as a kind of `store-transaction` where the `merchandise` is constrained to be `shoes` and the `merchant` is constrained to be a `cobbler`.

Some of the information described above is shown in Figure 3, along with a representation of sentence (5). The figure highlights the two paths in the collision that leads to the inference that the buying is a `shoe-store-transaction`.

The complete trace of FAUSTUS processing the two examples from Kay is shown below. In addition, I show a trace of a non-sequitur example with no coherence. This demonstrates that FAUSTUS can avoid making spurious inferences.

```
> (do-story)

          The Cobbler and the Alpinist


[ 1] A cobbler sold a pair of boots to an alpinist.

Rep: (SELLING (ACTOR ← A COBBLER) (PATIENT ← A BOOT)
              (RECIPIENT ← A ALPINIST))

Inferring: the ACTOR of the SELLING must be the SELLER
```

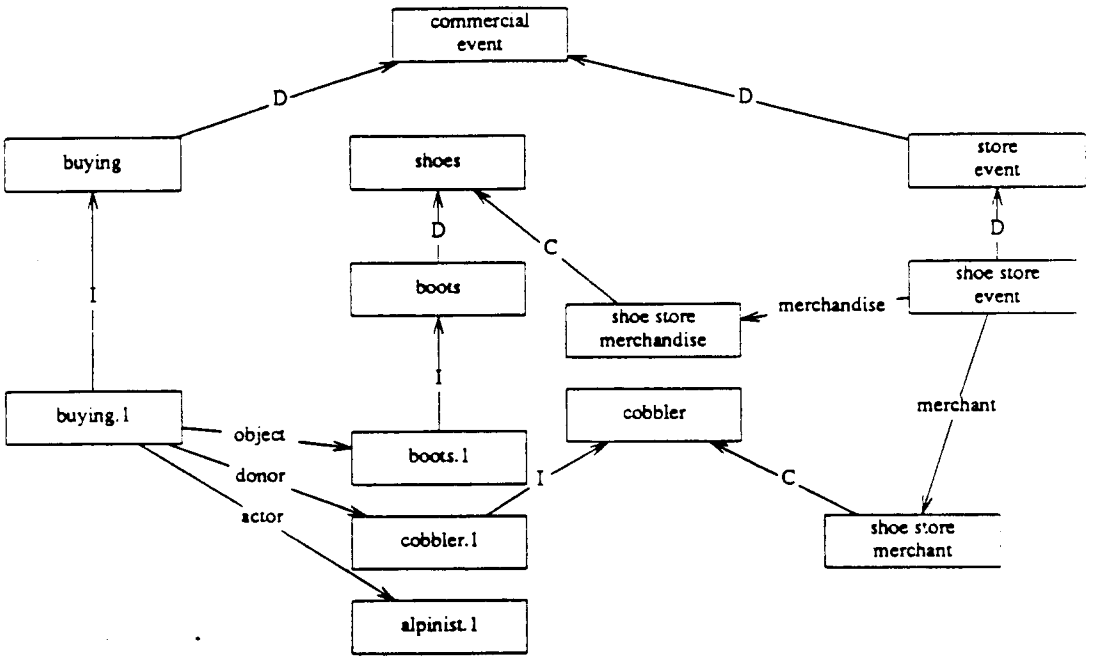

Figure 3: The shoe-store-event

```
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the SELLING must be the THING-SOLD
   This is a RELATION-CLASSIFICATION inference.

Inferring: the RECIPIENT of the SELLING must be the SELLEE
   This is a RELATION-CLASSIFICATION inference.

Int: (SELLING.1 (↑ SELLING) (seller ← COBBLER.1) (thing-sold ← BOOT.1)
                (sellee ← ALPINIST.1))

Passing Markers and Suggesting Inferences:  1 2 3 4 5 6 7 8 9 10

Evaluating Inferences:

Inferring: the SELLING is a SHOE-STORE-TRANSACTION.
   This is a CONCRETION inference.
   It is #1, due to the collision:
   BOOT.1→filler→COMMERCIAL-EVENT←elaboration←BOOT.1
```

```
Inferring: there is a WALKING such that
   it is the PURPOSE of the BOOT and
   the ALPINIST is the MOVED of it.
   This is a DOUBLE-ELABORATION inference.
   It is #3, due to the collision:
   BOOT.1→elaboration→MOVING←elaboration←ALPINIST.1

Inferring: the SELLING is a SHOE-STORE-TRANSACTION such that
   the COBBLER is the MERCHANT of it and
   the BOOT is the MERCHANDISE of it.
   This is a DOUBLE-ELABORATION inference.
   It is #4, due to the collision:
   COBBLER.1→elaboration→SHOE-STORE-TRANSACTION←elaboration←BOOT.1
```

```
> (do-story)

          The Chef and the Fisherman


[ 1] A chef bought a fish from a fisherman on fisherman's Wharf.

Rep: (BUYING (ACTOR ← A CHEF) (PATIENT ← A FISH) (DONOR ← A FISHERMAN)
             (SETTING ← FISHERMANS-WHARF))

Inferring: the ACTOR of the BUYING must be the BUYER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the BUYING must be the THING-BOUGHT
   This is a RELATION-CLASSIFICATION inference.

Inferring: the DONOR of the BUYING must be the BUYEE
   This is a RELATION-CLASSIFICATION inference.

Int: (BUYING.1 (↑ BUYING) (buyer ← CHEF.1) (thing-bought ← FISH.1)
               (buyee ← FISHERMAN.1) (setting ← FISHERMANS-WHARF.1))

Passing Markers and Suggesting Inferences:  1 2 3 4 5 6 7 8 9 10

Evaluating Inferences:

Inferring: the BUYING is a FISH-MARKET-TRANSACTION.
   This is a CONCRETION inference.
   It is #1, due to the collision:
   FISHERMAN.1→filler→COMERCIAL-EVENT←elaboration←FISHERMAN.1
```

```
Inferring: there is a COMMERCIAL-FISHING such that
   the FISH is the FISHED of it and
   it is a OCCUPATION of the FISHERMAN.
   This is a DOUBLE-ELABORATION inference.
   It is #4, due to the collision:
   FISH.1→elaboration→FISHING←elaboration←FISHERMAN.1

Inferring: the BUYING is a FISH-MARKET-TRANSACTION such that
   the FISH is the MERCHANDISE of it and
   the FISHERMAN is the MERCHANT of it.
   This is a DOUBLE-ELABORATION inference.
   It is #6, due to the collision:
   FISH.1→elaboration→FISH-MARKET-TRANSACTION←elaboration←FISHERMAN.1
```

```
> (do-story)

          The Lawyer and the Doctor


[ 1] A lawyer bought a fish from a doctor.

Rep: (BUYING (ACTOR ← A LAWYER) (PATIENT ← A FISH) (DONOR ← A DOCTOR))

Inferring: the ACTOR of the BUYING must be the BUYER
   This is a RELATION-CLASSIFICATION inference.

Inferring: the PATIENT of the BUYING must be the THING-BOUGHT
   This is a RELATION-CLASSIFICATION inference.

Inferring: the DONOR of the BUYING must be the BUYEE
   This is a RELATION-CLASSIFICATION inference.

Int: (BUYING.1 (↑ BUYING) (buyer ← LAWYER.1) (thing-bought ← FISH.1)
               (buyee ← DOCTOR.1))

Passing Markers and Suggesting Inferences:
```

Chapter 6:
Implementation Details

This chapter presents some of the details of the actual FAUSTUS program. I discuss the implementation of KODIAK first, then FAUSTUS, and conclude with some statistics on the time and space requirements of the system. The programs are written in Common Lisp on a Texas Instruments Explorer lisp machine. Versions of the KODIAK program also run in Zetalisp on Symbolics lisp machines and in Franz Lisp under UNIX as part of the UC project at Berkeley.

Implementation of KODIAK

For completeness, I will briefly describe the implementation of KODIAK, although there were less design considerations of import here, and the implementation did not bring to light as many issues. The basic data type is *object*, which has subtypes *link* and *concept*, which in turn has subtypes *aspectual, absolute,* and *relation*. All data types were defined using the defstruct facility.

Links have three fields: the concept they are attached *to* and *from*, and the *type* of link (I, D, A, etc). Concepts are more complex, being composed of a *name* field, a list of *markers*, and two alists. The first of these lists the links attached to the concept, with each link type being a separate key in the alist. The second alist lists directly the concepts that are at the other ends of these links. This is redundant information, and thus wastes some space, but being able to map over these lists made the code somewhat faster, and certainly simpler than constantly having to find the other end of links.

Aspectuals, absolutes, and relations have no fields beyond those inherited from concept, but type checking is used to distinguish between them in many places.

The basic type *object* has one field, an a-list. This is used to hold the following facts about an object:

- Creation time.
- Creation status (inference, input, or given).
- Candidates for inference, such as the list of possible fillers of a slot.
- An indication that the object is a promiscuous concept.
- Number restrictions on aspectuals.
- The determiner (a, the) associated with certain inputs.

For most objects, each of these is nil (or some other default value). Thus, it was decided that it would be a waste of space to allocate separate fields for each one within each object. Instead, entries are stored in the a-list only for those concepts that have values different from the default.

Functions were defined to store and retrieve from any of the entries in the links- and linked-to- alists. For example, for the D link type, the following functions were defined.

- *dominate* - asserts that one concept dominates another
- *dominate-p* - asks if two concepts are in a dominate relation
- *dominateds* - returns a list of concepts dominated by the argument
- *dominators* - returns a list of concepts that dominate the argument
- *links-to-dominateds* - returns the links rather than the concepts themselves
- *links-to-dominators* - returns the links rather than the concepts themselves

Similar functions are defined for each link type. This was done with a function defining macro, so that the definitions are in the form of a table, given below, which provides documentation for the user. The last argument to def-link-fncs indicates if the predicate for that link type is transitive. For example, the definition of constrain-p will be constructed so that it holds if there is a constrain link between the absolute and the aspectual, or if there is some concept which constrains the aspectual, and which is dominated by the absolute.

```
(def-link-fncs argument  (aspectual relation) relations     aspectuals   A)
(def-link-fncs fill-in   (concept aspectual)  filled        fillers      F)
(def-link-fncs equate    (concept1 concept2)  equates       equates      =)
(def-link-fncs differ    (concept1 concept2)  differs       differs      ≠)
(def-link-fncs constrain (absolute aspectual) constrains    constrainers C t
(def-link-fncs dominate  (upper lower)        dominateds    dominators   D t
(def-link-fncs instance  (upper lower)        instances     categories   I t
(def-link-fncs view      (target source)      view-sources  view-targets V t
```

There are also predicates that search the D and I link hierarchy to ask if one concept is above or below another by an arbitrary number of connected links, and to return a list of all the parents or children of a concept. This is complicated by the existence of multiple inheritance. The algorithms used keep a queue of concepts that have already been visited, to avoid returning a list with multiple occurrences of the same concept, and to make sure that the list is a valid topological sort of the lattice. Another possibility would be to pre-compute the complete list of recursive dominators for each concept, but I wanted to avoid a lot of pre-computation, as the next paragraph explains.

KODIAK of course includes a facility for defining a knowledge base. There was a strong motivation to allow for fast updating of the knowledge base, because it was assumed that most of the debugging time would be spent changing the knowledge base. This assumption proved true throughout the course of the project. Initially, it was easier to write functions to load the entire knowledge base at once, making consistency checks at the end. The plan was that at some point the knowledge base would become large enough that I would have to write functions to allow incremental updating of a few concepts at a time. However, even though the knowledge base grew to moderate size, it was still possible to do a load in under 40 seconds, so I never wrote the incremental updating code.

There is also a graphical interface to the knowledge base, but this has been used only for viewing the network, not for modifying it.

The final important algorithm is one that, given an absolute and an aspectual, finds all the slots of that absolute that the aspectual could be referring to. For example, with the absolute `eat-at-restaurant` and the slot `participant`, the function find-slots would return the list `(diner accompanier waiter)`. Diner and waiter are slots of eat-at-restaurant directly, but accompanier comes from the concept action, which dominates eat-at-restaurant. Thus, to find all the slots, the function must either search up the hierarchy from eat-at-restaurant, or down from participant. The later approach was chosen. But eat-at-restaurant is also defined as an action and a contractual-event, and we want to make sure not to include actor of action or party1 or party2 of contractual-event in the resulting list; since these refer to the diner and the waiter, and we don't want to be redundant. Therefore we have to be careful in our search to include only the most specific answers. Find-slots was defined as the most specific subset of concepts below the slot, satisfying the predicate that each candidate slot must be one of a pair of aspectuals such that the constrainer of the other one is a recursive parent (or is equal to) the parent in question. The function most-specific-below was then defined as follows. This function proved useful for a variety of purposes.

```
(defun most-specific-below (concept predicate)
  "Return the most specific subset of the children of concept satisfying the predicate."
  (when (funcall predicate concept)
    (or (delete-duplicates
          (mapcan #'(lambda (c) (most-specific-below c predicate))
                  (children concept)))
          (list concept))))
```

There are a number of miscellaneous functions within KODIAK, bringing the total to about 150 data types and functions.

Implementation of FAUSTUS

The FAUSTUS program of course makes use of the functions and data types defined in KODIAK. It also defines three new data types: story, marker, and suggestion. A story has a title and a text field. The text is an alternating list of strings and translations of those strings into KODIAK representation. A marker has fields for the marker it is associated with, the link traversed to get to that concept, the previous marker in the marker chain, and the shape of the marker path from the origin to the current concept. A suggestion has fields for the suggested inference number, the time the suggestion was entered into the queue, the time it must wait before being considered, the time after which it is automatically discarded, and its priority in the queue. It also has fields to record the two marker paths that lead to the collision. Specific types of suggestions have fields relevant to what is being suggested. For example, a reference suggested inference has fields for the reference and referent.

New stories are defined with the macro *defstory*, which takes as arguments a title and any number of elements representing the text of the story. If the title represents a new story, it is pushed on a global list of stories. If the title is of a known story within that list, the text of that story is updated.

The main function, *do-story*, takes two optional arguments, one indicating the story to do, and the other an indication that the knowledge base needs to be reloaded before starting the story. If these arguments are not provided, a menu pops up asking the user to choose from the global list of stories, and another menu asks if the knowledge base needs to be reloaded. The function *do-story* then processes each line of the story in turn, building the KODIAK representation for the line, passing markers, and then running the agenda of suggested inferences.

As mentioned above, every concept has a field for the markers attached to it. This is implemented as an alist, where the key for each entry in the list is the marker shape. The important marker shapes are those that participate in collisions that lead to inferences, and those already have names. In addition, I made up names for marker shapes corresponding to intermediate points along such paths. The advantage of this will become evident in a moment.

Given a concept, *c*, and a marker path of shape *s* leading to *c*, we pass markers from *c* by first looking at the alist of links from *c*. For each link type *l* among the keys of the alist, we consult a simple finite state automaton (FSA) with current state *s* and transition *l*. The entry in the FSA may be nil, meaning that no such link can lead to a valid path shape, and we can ignore all the links in this element of the alist. Alternately, the FSA entry may be a symbol representing a new path shape, meaning we should pass markers to each concept in this element of the alist, and the resulting path shape will be the new entry. From there we have to recursively pass markers again. For example, the FSA says that if *l* is D and *s* is elaboration, ref, constraint, or filler, then the resulting path is valid, and the new shape will still be *s*. If *s* was any other shape, then no D link should be considered. Finally, the FSA entry may be a predicate, in which case we have to go through each link in the alist entry, applying the predicate to each one to see if we should in fact continue passing markers. The predicate, when given an individual link to consider, should either return nil or a new path shape. This capability is needed, for example, to allow us to only pass along inverse C links with a quantifier of *most* or *all*.

Thus, one advantage of having alists for both markers and links is that we can quickly zero in on the combination of markers that will lead to valid path shapes, without having to exhaustively consider all possibilities. As it turns out, even though the marker passing algorithm was described as using the concept of marker energy, the finite state automaton can be defined in such a way that we don't need to keep track of marker energy at all. The arithmetic is essentially compiled away into the transitions between states in the FSA.

The other advantage of alists of markers is in resolving collisions. When we add a marker to a concept, we push or create the proper alist entry, and check for meaningful collisions. We don't have to look at all the previous markers, just ones with a marker shape that combines with the current marker shape to suggest an inference. Thus, if the new marker has the ref shape, we only look at the previous ref shape markers (suggesting referential inferences) and the previous elaboration shape markers (suggesting single elaboration inferences). All other previous markers are ignored. The intermediate marker shapes introduced for the sake of the FSA above never suggest inferences, so we don't need to waste any time on them.

A different approach to finding interesting marker collisions would be to allow collisions to occur anywhere along the path, as long as the two halves of the collision add up to the correct whole. The efficiency advantages listed above would be largely lost if we were to take this approach, so I look for collisions at one particular point, involving two specified shapes for the component paths. This means less searching, although it means I have to be diligent to define the marker passing rules (in the form of the FSA) to insure that no potential path will be cut off before it reaches the designated collision point. It turned out to be fairly easy to implement those precautions.

When it comes time to run the agenda, evaluating the suggested inferences, there are quite a few rules to consider. These were explained in Chapter 4, and the implementation is fairly straightforward, so it will not be mentioned here. FAUSTUS also contains a sub-facility for generating semi-English paraphrases of KODIAK concepts. This is seen in the output traces, and again is of no theoretical or technical import.

Statistics: Time and Space Requirements

The size of FAUSTUS' knowledge base is shown in the list below. In some ways it is hard to compare this to other systems, since KODIAK encourages a proliferation of concepts. For example, what counts as a relation, two aspectuals, and two arguments links in KODIAK might be recorded as a single relation in another system. A reasonable number for comparison is the total number of absolutes, relations, and equate links, since equates in some way represent a certain kind of "fact" in the system that is not captured by the other counts. Currently, the total of these three is 627. A list of counts for various types of objects in the knowledge base follows:

Type	Count
Absolutes	346
Relations	216
Aspectuals	432
D links	1113
C links	432
A links	432
V links	11
= links	82
≠ links	573
Total concepts	990
Total links	2643
Total objects	3633

The next table gives the number of markers, collisions, and inferences generated over the course of two separate texts, "John was lost" and "Bill's Bicycle." Most of the concepts that were marked once were also the site of a collision. Only about 5% of the collision sites result in suggested inferences, but a much larger percentage (about half) of the suggested inferences are accepted. This is good news for a potential parallel implementation, in that only the suggested inferences – 5% – need be evaluated sequentially. The marker passing and rejecting of collisions could conceivably be done in parallel.

Text:	Lost	Bicycle
Concepts marked:	261	279
Collisions:	247	233
Suggested inferences:	13	11
Accepted inferences:	6	7

It turns out that the majority of markers are on aspectuals, as the following table shows. It was taken from the processing of the "Bill's Bicycle" text.

	Relations	Absolutes	Aspectuals
Collisions:	32	48	153
Marked:	50	55	174

Running times range from under 2 seconds for single sentence texts to about 1 minute for 4 and 5 sentence texts and two minutes for the 7 input Fishing Village text. The number of collisions, and hence total processing time, is roughly proportional to the square of the length of the text. It takes 40 seconds to re-load the knowledge base. These times are for compiled Common Lisp on a Texas Instruments Explorer.

The KODIAK program is 822 lines (38K bytes) of source code, while FAUSTUS is 1005 lines (46K), and the knowledge base is 1013 lines (27K).

Chapter 7: Conclusions

In a sense, FAUSTUS was an experiment in self-deprivation. In a rule- or demon-based system, such as BORIS, one occasionally gets the suspicion that the system designer can just add one more rule to account for each new difficulty as it arises, as long as he or she is careful about interactions with previous rules. One way to do that is to make the new rule very specific, so that it accounts for just the one situation at hand. To instill confidence in its overall abilities, a system should have some way of constraining such *ad hoc* rules.

In FAUSTUS, the basic inference classes were decided upon early on. While there was certainly a great deal of debugging involved in getting the code for these right, it was never a possibility to alter the rules for the sake of making another example run. Thus, debugging consisted of changing facts in the knowledge base: redefining concepts and the relations between them. Since these concepts and relations were not tied to particular processing rules, there was less temptation to add *ad hoc* patches, as it would be obvious in the representation that this was done. The result was a more robust knowledge base, one that was not slanted towards particular examples.

FAUSTUS was also an experiment in taking the idea of relevant inferences to an extreme. In this regard, the experiment can be considered a success. With no automatic forward inferences and no top-down control structure to guide inference, FAUSTUS was able to duplicate the performance of a variety of systems predicated on processing specific knowledge classes. This is a testimony both to the power of the unified approach to knowledge representation, and also to the high degree of coherency in the texts studied.

Another point of success was that it was possible, over the course of the project, to extend the knowledge base to new examples, without having to make extensive changes for each addition. In other words, it was possible to settle on a representation for core concepts such that these representations were usable and extensible over a variety of texts.

Problems

FAUSTUS is missing some desirable capabilities that exist in other current systems. It cannot back up from choosing an inference that later proves to be incorrect. It does not integrate parsing and inferencing. However, there is no inherent reason why the knowledge-based approach could not be extended to include these capabilities.

There were three problems that arose during the course of the project that seem more significant than the ones mentioned above, and which constitutes significant areas for future research. The first is that the program has no notion of what constitutes a complete construal of the input. For each text, FAUSTUS will find a certain set of inferences, but will not be able to say if the text is coherent, confusing, or nonsensical. This has several ramifications. First, FAUSTUS is unable to search harder for an explanation for an unusual event, because it has no notion that events need explanations. Thus, while PAM was able to find a chain of seven inferences mediating between *Willa was hungry* and *She picked up the Michelin guide*, FAUSTUS does not know to search for such a chain. What FAUSTUS needs is a theory of *interestingness*, as in [112, 115] or [53] to coax the inferencing process into investigating some areas in more detail than others. This theory of interestingness should include provisions for forgetting the earlier part of long texts, or more precisely, focusing, in the sense of Grosz' [50], on the currently active parts. Without such a capability, FAUSTUS could not handle texts much longer than the ones presented here.

The second major problem is in the power of the representation language. In the quest to get the representations just right, a number of important representational issues had to be addressed, while others had to be side-stepped. For example, the notion of quantifier scope was not addressed. Sentence (1) below is usually interpreted as meaning there are different languages spoken by each of the participants, while sentence (2) refers to two specific languages. KODIAK has no way of representing the difference between these two interpretations.

(1) Everyone here speaks two languages.
(2) Two languages are spoken by everyone here.

Another representational problem is specifying degrees of usualness or unusualness. KODIAK provides quantifiers to say that *some* or *most* concepts of a category participate in a given relation, but we would like to have more flexible representation mechanisms. Future research should consider the representation of prototypical knowledge, of metaphors and metonymies, of case frame relations, and of the relation between syntactic and semantic knowledge.

The final major problem is choice. FAUSTUS contains rules for choosing between suggestions that are explicitly marked as competing with each other. There is room for improvement on these, but for the most part, the rules seem adequate. The real problem is when suggestions that aren't marked as explicitly competing end up affecting each other. For example, finding the proper pronominal referent from a set of possibilities may be affected if one of the possibilities is elaborated. On the other hand, choosing the proper elaboration of a slot from a set of possibilities may be affected if one of the possibilities is equated to a reference. In general it is a combinatorially hard problem to decide which choices to consider first, and which combination of choices yields the best interpretation. Another way of looking at it is that marker passing finds suggested inferences locally, but does not make suggestions for a global interpretation.

Despite these problems, the FAUSTUS project has shown it is possible to develop an algorithm to draw proper inferences from a range of texts, using only a small set of inference classes and a conceptual knowledge base.

References

1. Allen, J. F., "Modelling Events, Actions, and Time", *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, Ann Arbor, 1982, 5-6.
2. Allen, J. F. and Hayes, P. J., "A common-sense theory of time", *Proceedings of the 9th IJCAI*, Los Angeles, 1985, 528-531.
3. Alterman, R., "A Dictionary Based on Concept Coherence", *AI Journal 25*, 2 (1985), 153-186.
4. Anderson, J. and Bower, G., *Human Associative Memory*, Winston-Wiley, Washington, D.C., 1973.
5. Anderson, A. R. and Belnap, N. D., *Entailment: The Logic of Relevance and Necessity*, Princeton University Press, Princeton, NJ, 1975.
6. Anderson, R. C., Spiro, R. J. and Montague, W. E., *Schooling and the Acquisition of Knowledge*, Erlbaum Associates, Hillsdale, N.J., 1976.
7. Anderson, J., "A Spreading Activation Theory of Memory", *Journal of Verbal Learning and Verbal Behavior 22*, 3 (1983), 261-295.
8. Anderson, J., *The Architecture of Cognition*, Harvard Univ. Press, Cambridge, MA, 1983.
9. Anonymous, *My big book of fairy stories*, G.G.S. Ltd., Manchester, 1972.
10. Attardi, G. and Simi, M., "Consistency and Completeness of OMEGA, a Logic for Knowledge Representation", *Proceedings of the Seventh IJCAI*, Vancouver, 1981, 504-510.
11. Barsalou, L. W. and Bower, G. H., "Discrimination Nets as Psychological Models", *Cognitive Science 8* (1984), 1-26.
12. Bartlett, F. C., *Remembering: A Study in Experimental Social Psychology*, Cambridge University Press, Cambridge, 1932.
13. Bobrow, D. and Winograd, T., "An Overview of KRL: A Knowledge Representation Language", *Cognitive Science 1* (1977), 346.
14. Brachman, R. J., "What's in a concept: structural foundations for semantic networks", *Int. J. Man-Machine Studies*, 1977, 127-152.
15. Brachman, R. J., "What IS-A Is and Isn't: An Analysis of Taxonomic Links in Semantic Nets", *Computer 16*, 10 (1983), 30-36.
16. Brachman, R. J., Levesque, H. J. and Fikes, R., "KRYPTON: Integrating Terminology and Assertion", *Proceedings of AAAI-83*, Washington, D.C., 1983, 31.
17. Brachman, R. J. and Levesque, H. J., "The tractibility of subsumption in frame-based description languages", *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, 1984, 34-37.
18. Brachman, R. J. and Schmolze, J. G., "An overview of the KL-ONE knowledge representation system", *Cognitive Science 9*, 2 (1985), 171-216.

19. Carbonell, J. G., ''Metaphor: An Inescapable Phenomenon in Natural Language Comprehension'', in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982, 415-454.

20. Carbonell, J. G., ''Derivational Analogy and its Role in Problem Solving'', *Proceedings of AAAI-83*, Washington, D.C., 1983, 64.

21. Carbonell, J. G. and Minton, S., ''Metaphor and Common-Sense Reasoning'', Report CMU, Dept. of Computer Science, Carnegie-Mellon University, Pittsburgh, PA-CS-83-110, 1983.

22. Chafe, W. L., *Meaning and the Structure of Language*, Univ. of Chicago Press, Chicago, 1970.

23. Charniak, E., ''Toward a Model of Children's Story Comprehension'', AI-Tech. Rep.-266, MIT AI Labs, Cambridge, MA, 1972.

24. Charniak, E., ''On the use of framed knowledge in language comprehension'', *Artificial Intelligence 11*, 3 (1978), 225-266.

25. Charniak, E., ''The Case-Slot Identity Theory'', *Cognitive Science 5*, 3 (1981), 285-292.

26. Charniak, E., ''Passing Markers: A Theory of Contextual Influence in Language Comprehension'', *Cognitive Science 7*, 3 (1983), 171-190.

27. Charniak, E., ''A Single-Semantic-Process Theory of Parsing'', unpublished ms., Brown University Dept. of Computer Science, Providence RI, 1985.

28. Charniak, E., ''A neat theory of marker passing'', *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986, 584-588.

29. Clark, H. H. and Haviland, S. E., ''Psychological processes as linguistic explanation'', in *Explaining linguistic phenomena*, D. Cohen (editor), Hemisphere Publishing, Washington D.C., 1974.

30. Clark, H. H., ''Bridging'', in *TINLAP*, R. C. Schank and B. Nash-Webber (editor), Cambridge Mass, 1975, 169-174.

31. Collins, A. M. and Quillian, M. R., ''Retrieval time from semantic memory'', *Journal of Verbal Learning and Verbal Behavior 8* (1969).

32. Collins, A. M. and Loftus, E. F., ''A spreading activation theory of semantic processing'', *Psychological Review 82*, 4 (1975).

33. Cullingford, R. E., ''Script Application: Computer understanding of newspaper stories'', Research Report #116, Yale University Computer Science Dept., 1978.

34. DeJong, G., ''An Overview of the FRUMP System'', in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982, 149-177.

35. Doyle, J., ''A Truth Maintenance System'', *Artificial Intelligence 12*, 3 (1979), 231-272.

36. Doyle, J., ''The Ins and Outs of Reason Maintenance'', *Proceedings of the 8th IJCAI*, Karlsruhe, West Germany, 1983, 349-351.

37. Dyer, M. G. and Lehnert, W. G., "Question Answering for Narrative Memory", in *Language and Comprehension*, J. L. Ny and W. Kintsch (editor), North-Holland, Amsterdam, 1982.

38. Dyer, M. G., "In-Depth Understanding", Research Report #219, Yale University Computer Science Dept., 1982.

39. Etherington, D. W. and Reiter, R., "On Inheritance Hierarchies With Exceptions", *Proceedings of AAAI-83*, Washington, D.C., 1983, 104.

40. Fahlman, S. E., *NETL: A System for Representing and Using Real-World Knowledge*, MIT Press, Cambridge, 1979.

41. Faletti, J., "PANDORA -- A Program for Doing Commonsense Planning in Complex Situations", *Proceedings of AAAI-82*, Pittsburgh, 1982.

42. Fillmore, C. J., "The Case for Case", in *Universals in Linguistic Theory*, E. W. Bach and R. T. Harms (editor), Holt, Rinehart & Winston, New York, 1968, 1-88.

43. Fillmore, C. J., "Ideal Readers and Real readers", Report #5, Berkeley Cognitive Science, 1983.

44. Goldstein, I. P. and Roberts, B., "Using Frames in Scheduling", in *Artificial Intelligence: An MIT Perspective*, vol. I , P. H. Winston and R. H. Brown (editor), MIT Press, Cambridge, MA, 1982, 255-286.

45. Granger, R. H., "Adaptive Understanding: Correcting Erroneous Inferences", 171, Yale Univ. Dept. of Computer Science, New Haven, CT, 1980.

46. Granger, R. H., Eiselt, K. P. and Holbrook, J. K., "The Parallel Organizatiuon of Lexical, Syntactic, and Pragmatic Inference Processes", *First Conference on Theoretical Issues in Conceptual Information Processing*, Atlanta, GA, 1984.

47. Granger, R. H., Eiselt, K. P. and Holbrook, J. K., "Parsing with parallelism: a spreading-activation model of inference processing during text understanding", Technical Report #228, Dept. of Information and Computer Science, UC Irvine, 1984.

48. Granger, R. H., Holbrook, J. K. and Eiselt, K. P., "Interaction effects between word-level and text-level inferences: on-line processing of ambiguous words in context", *Proceedings of the Sixth Annual Conference of the Cognitive Science Society*, Boulder, CO, 1984.

49. Grice, H. P., "Logic and conversation", in *Syntax and Semantics, Vol. III: Speech Acts*, P. Cole and J. L. Morgan (editor), Academic Press, New York, 1975.

50. Grosz, B., "The representation and use of focus in dialogue understanding", Ph.D. dissertation, University of California at Berkeley, 1977.

51. Hendler, J., *Integrating Marker-Passing and Problem-Solving: A Spreading Activation Approach to Improved Choice in Planning*, Department of Computer Science, Univ. of Maryland, College Park, MD, 1986.

52. Hendrix, G. G., Thompson, C. W. and SLocum, J., "Language processing via canonical verbs and semantic models", *Proceedings of the 3rd IJCAI*, Stanford, 1973.

53. Hidi, S., Baird, W. and Hildyard, A., ''That's Important but is it Interesting? Two Factors in Text Processing'', in *Discourse Processing*, A. Flammer and W. Kintsch (editor), North-Holland, Amsterdam, 1982.

54. Hirst, G. and Charniak, E., ''Word Sense and Case Slot Disambiguation'', *Proceedings of AAAI-82*, Pittsburgh, 1982, 95-98.

55. Hobbs, J. R., ''Coherence and coreference'', *Cognitive Science 3*, 1 (1979), 67-90.

56. Hobbs, J., ''Towards an Understanding of Coherence in Discourse'', in *Strategies for Natural Language Processing*, Lehnert and Ringle (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982.

57. Jacobs, P. S. and Rau, L. F., ''Ace: Associating Language with Meaning'', *Proceedings of the ECAI*, Pisa, Italy, 1984.

58. Jacobs, P. S., ''A Knowledge-Based Approach to Language Generation'', PhD Thesis, also Report No.86/254, Computer Science Div., University of California, Berkeley, 1985.

59. Jacobs, P. S., ''PHRED: A generator for natural language interfaces'', Report No. 85/198, Computer Science Div., University of California, Berkeley, CA, 1985.

60. Katz, J. J., *Semantic Theory*, Harper and Row, New York, 1972.

61. Kay, P., *Three Properties of the Ideal Reader*, Berkeley Cognitive Science Program, 1981.

62. Kay, M., ''Functional unification grammar: a formalism for machine translation'', *10th International Conference on Computational Linguistics*, Stanford, CA, 1984, 75-78. 22nd ACL.

63. Kintsch, W., *The Representation of Meaning in Memory*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1974.

64. Koffka, K., in *Principles of Gestalt Psychology*, Harcourt, Brace & World, NY, 1963.

65. Kolodner, J. L., *Retrieval and Organizational Strategies in Conceptual Memory: A Computer Model*, Lawrence Erlbaum Associates, Hillsdale, NJ, 1984.

66. Lakoff, G. P., ''Structural Complexity in Fairy Tales'', *The Study of Man 1* (1972), 128-150.

67. Lakoff, G. and Johnson, M., *Metaphors We Live By*, University of Chicago Press, Chicago, 1980.

68. Lakoff, G., ''Categories and Cognitive Models'', Berkeley Cognitive Science Report No. 2, University of California, Berkeley, Berkeley, CA, 1982.

69. Langacker, R., ''An Introduction to Cognitive Grammar'', *Cognitive Science 10*, 1 (1986), 1-40.

70. Lehnert, W. G., ''Plot Units: A Narrative Summarization Strategy'', in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982, 375-414.

71. Lockman, A. and Klappholz, A. D., ''Toward a procedural model of contextual reference resolution'', *Discourse Processes 3* (1980), 25-71.

72. Mandler, J. M. and Johnson, N. S., "Remembrance of Things Parsed: Story Structure and Recall", *Cognitive Psychology*, 1977, 111-151.

73. Mandler, J. M. and Johnson, N. S., "On Throwing Out the Baby with the Bathwater: A Reply to Black and Wilensky's Evaluation of Story Grammars", *Cognitive Science 4*, 3 (1980), 305-312.

74. Mandler, J. M., "Recent Research on Story Grammars", in *Language and Comprehension*, J. L. Ny and W. Kintsch (editor), North-Holland, Amsterdam, 1982.

75. Mann, W. C. and Thompson, S. A., "Relational Propositions in Discourse", Report #ISI/RR-83-115, ISI, Marina Del Ray, CA, 1983.

76. Marr, D. and Marr, D., "Artificial Intelligence – A personal view", *Artificial Intelligence 9* (1977), 37-48.

77. Martin, J. A., "Views from a kill", *Proceedings of the Cognitive Science Society*, Amherst, Mass., 1986.

78. McCarthy, J. and Hayes, P. J., "Some philosophical problems from the standpoint of artificial intelligence", in *Machine Intelligence 4*, Meltzer and Michie (editor), Edinburgh, 1969, 463-502.

79. McDermott, D., "Assimilation of new information by a natural language understanding system", MIT AI Tech. Rep.-291, 1974.

80. McDermott, D., "Planning and acting", *Cognitive Science 2* (1978), 71-109.

81. McDermott, D., "Artificial Intelligence Meets Natural Stupidity", in *Mind Design*, J. Haugeland (editor), MIT Press, Cambridge, MA, 1981, 143-160.

82. Minsky, M., in *Semantic Information Processing*, MIT Press, Cambridge, Mass, 1967.

83. Minsky, M., "A framework for representing knowledge", in *The psychology of computer vision*, P. Winston (editor), McGraw-Hill, 1975.

84. Moore, J. and Newell, A., "How Can Merlin Understand?", in *Knowledge and Cognition*, L. W. Gregg (editor), Lawrence Erlbaum Associates, Baltimore, 1973, 201-252.

85. Moser, M. G., "An Overview of NIKL, The New Implementation of KL-ONE", Report No. 5421, Bolt, Beranek and Newman, Inc., Cambridge, MA, 1983.

86. Nilsson, N. J., *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill, New York, 1971.

87. Norman, D. A. and Rumelhart, D. E., *Explorations in Cognition*, Freeman, San Francisco, 1975.

88. Norvig, P., "Frame Activated Inferences in a Story Understanding Program", *Proceedings of the 8th IJCAI*, Karlsruhe, West Germany, 1983.

89. Norvig, P., "Six Problems for Story Understanders", *Proceedings of AAAI-83*, Washington, DC, 1983.

90. Norvig, P., "A Unified Theory of Inference for Text Understanding", Ph.D Thesis, University of California, Berkeley, 1986.

91. Pereira, F. C. N. and Warren, D. H. D., "Definite clause grammars for language analysis--- a survey of the formalism and a comparison with augmented transition networks", *Artificial Intelligence 13*, 3 (1980), 231-278.

92. Propp, V., *Morphology of the folktale*, University of Texas Press, Austin, 1968.

93. Quillian, M. R., "The teachable language comprehender: A simulation program and theory of language", *Communications of the ACM*, 1969, 459-476.

94. Reddy, M., "The Conduit Metaphor", in *Metaphor and Thought*, A. Ortony (editor), Cambridge Univ. Press, Cambridge, 1979.

95. Rieger, C. J., "Understanding By Conceptual Inference", Tech. Rep.-353, University of Maryland Computer Science Dept., 1975.

96. Rips, L. J., Shoben, E. J. and Smith, E. E., "Semantic distance and the verification of semantic relations", *Journal of Verbal Learning and Verbal Behavior 12* (1973), 1-20.

97. Rips, L. J., Shoben, E. J. and Smith, E., "Structure and process in semantic memory: A featural model for semantic decisions", *Psychological Review 81*, 3 (1974), 214-241.

98. Rosch, E. and Lloyd, B. B., *Cognition and Categorization*, Lawrence Erlbaum Associates, 1978.

99. Roseman, I., *Cognitive aspects of emotion and emotional bevavior*, Dept. of Psychology, Yale University, New Haven, CT, 1979. Unpublished manuscript.

100. Rosenberg, S., "HPRL: A Language for Building Expert Systems", *Proceedings of the 8th IJCAI*, Karlsruhe, West Germany, 1983, 215-217.

101. Rumelhart, D., "Notes on a schema for stories", in *Representation and Understanding*, D. G. Bobrow and A. Collins (editor), Academic Press, New York, 1975, 211-236.

102. Sacerdoti, E., "The non-linear nature of plans", *Fourth IJCAI*, Tsibilisi, 1975, 206 - 214.

103. Sacerdoti, E., in *A Structure for Plans and Behavior*, NY, 1977, Elsevier.

104. Sacks, H., "On the analyzability of stories by children", in *Directions in Sociolinguistics*, J. Gumperz and D. Hymes (editor), Holt, Rinehart and Winston, 1972.

105. Schank, R. C., "Conceptual Dependency: A Theory of Natural Language Understanding", *Cognitive Psychology*, 1972, 552-631.

106. Schank, R. C., "The fourteen primitive actions and their inferences", AIM-183, Stanford University Computer Science Department, Stanford, California, 1973.

107. Schank, R. C., "Identification of conceptualizations underlying natural language", in *Computer Models of Thought and Language*, R. C. Schank and K. M. Colby (editor), Freeman, San Francisco, CA, 1973, 187-247.

108. Schank, R. C. and Abelson, R. P., "Scripts, plans and knowledge", *Advance papers, 4th IJCAI*, 1975, 151-157.

109. Schank, R., "SAM– A story understander", Yale University Computer Science Research Report #43, 1975.

110. Schank, R. C. and Abelson, R. P., *Scripts, Plans, Goals and Understanding*, Erlbaum, Hillsdale, N.J., 1977.

111. Schank, R. C. and Wilensky, R., "A goal-directed production system for story understanders", in *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth (editor), Academic Press, New York, 1978.

112. Schank, R. C., "Interestingness: Controlling Inference", *Artificial Intelligence 12*, 3 (1979), 273-298.

113. Schank, R. C., "Language and Memory", *Cognitive Science 4*, 3 (1980), 243-284.

114. Schank, R. C., "MOPs and Learning", *Proceedings of the Third Annual Conference of the Cognitive Science Society*, Berkeley, 1981, 166-169.

115. Schank, R. C., Collins, G. C., Davis, E., Johnson, P. N., Lytinen, S. and Reiser, B. J., "What's The Point", *Cognitive Science 6*, 3 (1982), 255-276.

116. Schank, R. C., "Reminding and Memory Organization: An Introduction to MOPs", in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982, 455-494.

117. Schmolze, J. G. and Lipkis, T. A., "Classification in the KL-ONE Knowledge Representation System", *Proceedings of the 8th IJCAI*, Karlsruhe, West Germany, 1983, 330-332.

118. Sidner, C., "Disambiguating References and Interpreting Sentence Purpose in Discourse", in *Artificial Intelligence: An MIT Perspective*, vol. I , P. H. Winston and R. H. Brown (editor), MIT Press, Cambridge, MA, 1982, 231-254.

119. Simmons, R. F., Klein, S. and McConlogue, K., "Indexing and dependency logic for answering English questions", *American Documentation 15* (1964), 196-204.

120. Simmons, R. F., "Answering English questions by computer", in *Automated Language Processing: The State of the Art*, H. Borko (editor), Wiley and Sons, New York, 1967, 15-30.

121. Simmons, R. F. and Slocum, J., "Generating English discourse from semantic networks", *Comm. of the ACM 15* (1972), 891-905.

122. Simon, H. A., *The Sciences of the Artificial*, MIT Press, Cambridge, MA, 1969.

123. Small, S. L., "Exploding Connections: Unchunking Schematic Knowledge", *Proceedings of the Fourth Annual Conference of the Cognitive Science Society*, Ann Arbor, 1982, 169-173.

124. Small, S., Cottrell, G. and Shastri, L., "Toward Connectionist Parsing", *Proceedings of AAAI-82*, Pittsburgh, 1982, 247-250.

125. Sondheimer, N. K., Weischedel, R. M. and Bobrow, R. J., "Semantic Interpretation Using KL-ONE", *Proceedings of Coling*, Stanford, CA, 1984.

126. Thorndyke, P. W., "Cognitive structures in comprehension and memory of narrative discourse.", *Cognitive Psychology*, 1977.

127. Thorndyke, P. W., "Pattern-directed processing of knowledge from texts", in *Pattern-Directed Inference Systems*, D. A. Waterman and F. Hayes-Roth (editor), Academic Press, New York, 1978.

128. Waltz, D. L. and Pollack, J. B., "Phenomenologically plausible parsing", *Proceedings of the National Conference on Artificial Intelligence*, Austin, TX, 1984, 335-339.

129. Waltz, D. L. and Pollack, J. B., "Massively parallel parsing: A strongly interactive model of natural language interpretation", Technical Report, Coordinated Science Laboritory, University of Illinois, Urbana, Ill, 1984.

130. Warren, D. H. D. and Pereira, F. C. N., "An efficient easily adaptable system for interpreting natural language queries", *American Journal of Copmputational Linguistics 8*, 3-4 (1982), 110.

131. Wilensky, R. W., *Understanding Goal-based Stories*, Yale University Computer Science Research Report, New Haven, CT, 1978.

132. Wilensky, R. W. and Black, J. B., "An Evaluation of Story Grammars", *Cognitive Science*, 1979.

133. Wilensky, R. and Arens, Y., "A Knowledge-based Approach to Natural Language Processing", Memorandum No. UCB/Electronics Research Lab./M80/34, UC Berkeley Electronics Research Lab., Berkeley, 1980.

134. Wilensky, R., Deering, M. and Faletti, J., "PEARL: An Efficient Language for Artificial Intelligence Programming", *Proceedings of the Seventh IJCAI*, Vancouver, BC, 1981.

135. Wilensky, R., "Points: A Theory of the Structure of Stories in Memory", in *Strategies for Natural Language Processing*, W. G. Lehnert and M. H. Ringle (editor), Lawrence Erlbaum Associates, Hillsdale, NJ, 1982, 345-374.

136. Wilensky, R., *Planning and Understanding*, Addison-Wesley, Reading, MA, 1983.

137. Wilensky, R., "Story grammars versus story points", *J. of The Behavioral and Brain Sciences 6*, 4 (1983), Cambridge Univ. Press.

138. Wilensky, R., "KODIAK: A Knowledge Representation Language", *Proceedings of the 6th National Conference of the Cognitive Science Society*, Boulder, CO, 1984.

139. Wilensky, R., Arens, Y. and Chin, D. N., "Talking to UNIX in English: An Overview of UC", *Communications of the ACM 27*, 6 (1984).

140. Wilensky, R., "Knowledge Representation: A Proposal and Critique", *First Annual Workshop on Theoretical Issues in Conceptual Information Processing*, Atlanta, Georgia, 1984, 148-159.

141. Wilensky, R., "Some Problems and Proposals for Knowledge Representation", Report No. UCB/Computer Science Dpt. 86/294, Computer Science Division, UC Berkeley, 1986.

142. Wilks, Y., "Understanding without proofs", *Proceedings of the Third IJCAI*, 1973.

143. Wilks, Y., "Preference semantics", in *The Formal Semantics of Natural Language*, E. L. Keenan (editor), Cambridge Univ. Press, Cambridge, 1975, 329-350.

144. Winston, P. H., "Learning by Creating and Justifying Transfer Frames", in *Artificial Intelligence: An MIT Perspective*, vol. I , P. H. Winston and R. H. Brown (editor), MIT Press, Cambridge, MA, 1982, 347-376.

145. Winston, P. H., ''Learning New Principles From Precedents and Examples'', *Artificial Intelligence 19*, 3 (1982), 321-350.

146. Wong, D., ''Language Comprehension in a Problem Solver'', *Proceedings of the Seventh IJCAI*, Vancouver, 1981, 7-12.

147. Woods, W. A., ''What's in a link?'', in *Representation and Understanding*, D. G. Bobrow and A. Collins (editor), Academic Press, New York, 1975, 35-82.

148. Woods, W. A., ''What's Important About Knowledge Representation?'', *Computer 16*, 10 (1983), 22-29.

149. Zadeh, L. A., ''A note on prototype theory and fuzzy sets'', *Cognition 12* (1982), 291-297.

150. Zwicky, A. M. and Sadock, J. M., ''Ambiguity Tests and How to Fail Them'', in *Syntax and Semantics*, vol. 4 , J. P. Kimball (editor), Academic Press, NY, NY, 1975.