A Probabilistic Approach to Language Understanding

Robert Prescott Goldman
Ph.D Dissertation

Department of Computer Science
Brown University
Providence, Rhode Island 02912

Technical Report No. CS-90-34
December 1990

A Probabilistic Approach to Language Understanding

Technical Report CS-90-34

Computer Science Department
Brown University
Providence, RI

by
Robert Prescott Goldman
B. A., Yale College, 1983
Sc. M., Brown University, 1987

For My Parents

Vita

Robert Prescott Goldman was born in Washington, DC on May 10, 1961. He graduated Magna Cum Laude with a B.A. in Philosophy from Yale University in New Haven, Connecticut in 1983. He received a Sc.M. in Computer Science from Brown University in 1988. He is currently an Assistant Professor in the Computer Science Department of Tulane University, New Orleans, Louisiana. Dr. Goldman's research interests are natural language processing and the application of probability theory to Artificial Intelligence. He is a member of AAAI and ACM.

Abstract

We discuss a new framework for text understanding. Three major design decisions characterize this approach. First, we take the problem of text understanding to be a particular case of the general problem of abductive inference: reasoning from effects to causes. Second, we use probability theory to handle the uncertainty which arises in abductive inference in general, and natural language understanding in particular. Finally, we treat all aspects of the text understanding problem in a unified way. All aspects of natural language processing are treated in the same framework, allowing us to integrate syntactic, semantic and pragmatic constraints. In order to apply probability theory to this problem, we have developed a probabilistic model of text understanding. To make it practical to use this model, we have devised a way of incrementally constructing and evaluating belief networks which is applicable to other abduction problems. We have written a program, Wimp3, to experiment with this framework.

Acknowledgments

I would like to thank my parents and my sister for help, both material and emotional, through the course of my graduate education.

My advisor, Eugene Charniak, has been the model of a thesis advisor. He has been supportive, pleasant and stimulating to work with. He has also been an excellent role model. I hope I can become as good, and as ethical, a computer scientist.

I am also grateful to my other two readers, Tom Dean and Judea Pearl, for wading through this staggering pile of paper, and seeing me through to my doctorate.

The Brown Computer Science department does a remarkable job of providing both a strong research environment, and a friendly, supportive atmosphere. I am both grateful and honored to have been a part of it. There are several people I would like to single out for particular appreciation: Trina Avery got me out of many financial scrapes, ensured that I received my stipend pretty much on time every month, and most of all, stood between me and the forces of bureaucracy in the University and the big bad world. Mary Andrade and Dawn Nicholaus, sitting behind the receptionist's desk, have done a myriad things to make my life more pleasant. Before they arrived, Mimi Tremblay filled the same role ably. Also special thanks to those other administrative staff with whom I've had most contact: Ginny Bunch, Jennet Kirschenbaum and Tina Cantor.

One of the nicest things for a student working in AI at Brown is that the AI graduate students in the Computer Science department have been able to form a group with a particular *esprit de corps*, without isolating themselves from the rest of the department. I've spent much of my time here with them, to my immense profit. Particularly important were Kate Sanders, Glenn Carroll (thanks for the marker-passer!), Ken Basye, Lynn Stein, Felix Yen and for the brief period of our overlap, Robert ("Bobbo") McCartney.

Three of my comrades deserve special mention: Keiji Kanazawa, Mark Boddy and Hope Wilson. In addition to being a colleague, Keiji is a roommate of long-standing and one of my closest friends in Providence. Mark and Hope are my other closest, and have provided me with more transportation than any five people have any right to ask.

Drew McDermott for his course, which first interested me in Artificial Intelligence, while I was an undergraduate at Yale University.

Elliot Soloway provided me with a haven at the Yale Artificial Intelligence Project my first summer out of college, making it possible for this interest to grow and take hold.

For five years at Brown University, Ken Duhamel provided excellent instruction in Tai ji, keeping me off the streets and in good health.

In my last two years, I also had the privilege of studying African Dance with Michelle Bach, who opened up further channels to sanity through strenuous body movement.

Thanks to my girlfriend, Jenny Jenkins, and my former landlord, Paul van Zuiden, for making it clear to me (in very different ways!) that I didn't want to be in graduate school any more.

Contents

List of Figures

Chapter 1

Introduction

NOTE: *This technical report is a revised version of the author's Ph. D. thesis, submitted in August, 1990. The original is available from University Microfilms, Inc. It contains full descriptions of Wimp3's database, but does not have the information included here as Chapter 9.*

This thesis discusses a new way of tackling the problems posed by uncertainty in text understanding: representing and reasoning about alternative interpretations of natural language texts. We are concerned with problems like pronoun reference, prepositional phrase attachment, and word-sense ambiguity. We are particularly interested in plan-recognition as it is needed in text understanding: understanding the meanings of stories by understanding the way the actions of characters in the story serve purposes in their plans. The work reported in this thesis is distinguished by three principles: first of all, we look on story understanding as a particular case of the general problem of abduction. Second, we have adopted a probabilistic approach to this abduction problem, unique in story understanding. Finally, unlike many language understanding programs, our program, Wimp3, tightly integrates syntactic, semantic and pragmatic processing.

We see text understanding as a particular case of the problem of abduction (reasoning from effects to causes), or diagnosis (see [19] or [56] for an outline of this position). That is, we take the text as an effect (a *symptom*), and look for a cause (a *diagnosis*) that would explain it. In this case, the kind of causes we consider are intentions to communicate some meaning.

The process of abduction is not a straightforward process like deduction; it is inherently non-monotonic, and uncertain. We have adopted probability theory as our framework because it is the most thoroughly-understood way of reasoning about uncertainty. Unlike other ways of treating uncertainty (e.g., default logic), probability provides a unified 'currency' in which we can weigh evidence from different sources. Furthermore, it is at least in some sense normative [29]. Finally, techniques for graphically representing probability distributions have made the task of drawing up and reasoning with probabilistic models much less onerous than it was once thought to be.

The third distinguishing feature of our approach is the extent to which it integrates all levels of processing. Previous work in text understanding has almost universally assumed that the text will be pre-processed by an intelligent parser (e.g., [30, 56, 85, 112]). We, on the other hand, read the text word-by-word through a parser that simply returns all possible parses of the input. Because the structure of the sentence is described in the same framework as its semantics and pragmatics, constraints from different levels can be assembled into a global interpretation. For example, in deciding the proper attachment of the prepositional phrase "with some poison" in "Janet killed the boy with some poison."

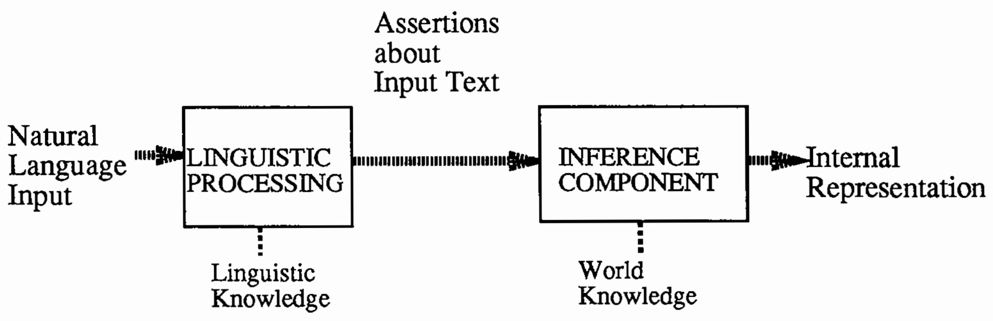


Figure 1.1: The idealized architecture of a story understanding program.

we can make use of our knowledge about murder, or about a particular poison-wielding boy that Janet dislikes.

Our current program, Wimp3, operates by turning problems of text interpretation into probability questions of a form like "What is the probability that this word has this sense, given the evidence?" (where the evidence is the text of the story so far). Wimp3 does this on a word-by-word basis. The probability questions are formulated in a language we have developed for this purpose. This formulation is given additional structure by a graphical representation, belief networks.

In this introductory chapter of the thesis, we will begin by discussing story understanding – its characteristics as an AI problem. We will then proceed to give a brief overview of the thesis and its contributions.

1.1 Story Understanding

The problem of story understanding is to take a story written in some natural language, and "understand" it in the sense that one's program could answer questions about it. It is generally believed that this involves translating the story into an internal representation which will support the inferences necessary to answer questions. This internal representation should be an unambiguous representation of the meaning of the text. In this translation process, the program will make use of knowledge about language and about the world the language is being used to describe. In this section we give a brief introduction to the problem of story understanding as an AI problem. To the best of our ability, we avoid giving any controversial details; any techniques special to our program will be discussed elsewhere.

An architecture for a story understanding program is given in Figure 1.1. First a linguistic processor applies grammatical and lexical knowledge to the natural language input to yield some assertions. These assertions are operated on by a component which applies knowledge about the world described, to yield a deeper understanding of the story. One of the most important kinds of reasoning done in this process of deeper understanding is plan recognition. This is inferring an agent's goals and plans from the actions he or she is described as taking. For example, in order to understand the actions of a person who is taking pots and pans out of a cupboard, taking food out of the refrigerator, consulting a cookbook and turning on the oven, we need to recognize that person's plan to prepare food.

Linguistic knowledge for story understanding is usually embodied in two databases. There is a lexicon which contains the information necessary to recognize words and to relate them to concepts in the internal representation. There is also a grammar which specifies the way words combine into sentences, and which usually specifies the information

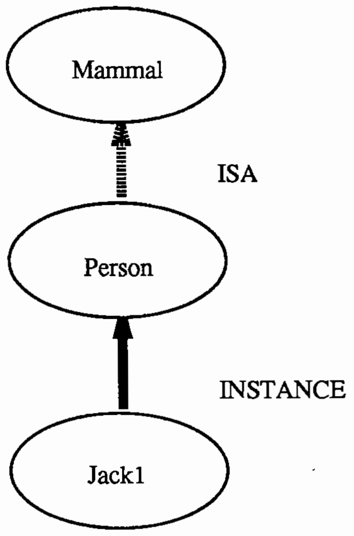


Figure 1.2: A fragment of the isa-hierarchy.

carried by syntactic relations (e.g., the subject of a verb is typically the agent of the action corresponding to the verb).[1]

Virtually all story understanding programs use a frame-based knowledge representation language, both to represent their knowledge, and to give the final translation. Such KR schemes have objects representing natural kinds: *e.g.,* person, chain, the action of walking somewhere. For historical reasons, we call these objects *frames* [83], although fundamentally the same objects have been called different things.[2] Frames are typically arranged in an inheritance hierarchy, or isa-hierarchy (a person *isa* mammal), in which sub-types inherit features of their super-types (people are warm-blooded). See Figure 1.2. Individual entities in the world are represented by *instances* of frames: Jack1 (the subscript is there to indicate a *specific* person named Jack) is an instance of the person frame.

Frames are hierarchical data objects. Frame instances have *slots* which can be filled by other instances. Slots are used to specify the sub-acts of complex plans, objects playing roles in plans (e.g., the instrument of an action), parts of complex objects, attributes of objects, and the roles objects fill in relations. Frame definitions specify the slots different frames will have, and restrictions on the kind of objects which can fill these slots. For example, Figure 1.3 shows the frame for shopping plans. A shopping plan *isa* action. From the definition of the action frame, the shopping frame inherits a slot for agent. The action frame definition specifies that the agent slot of an action must be filled by an animate-object. The shopping frame also has locally-defined slots for the store at which the shopping takes place, the object purchased, the go-step, and the pay-step (a thorough representation would, of course, have many more slots). The go-step is a step of the shopping plan: it is the go action which causes the agent of the shopping plan to be at the store at which he or she wishes to shop, and so forth.

Let us see what we would like our translations to look like. Take a simple story: "Jack went to the store. He paid for some milk." We would like this to match the frame for

[1]This is somewhat more controversial than the rest of our outline. Some approaches forego syntactic parsing in favor of largely lexicon-driven semantic parsing. Allen's book [2] contains a good discussion of the advantages and disadvantages of semantic parsing.

[2]*e.g.,* they are called 'concepts' in the languages in the KL-One lineage [5], schemas in the cognitive science literature, etc.

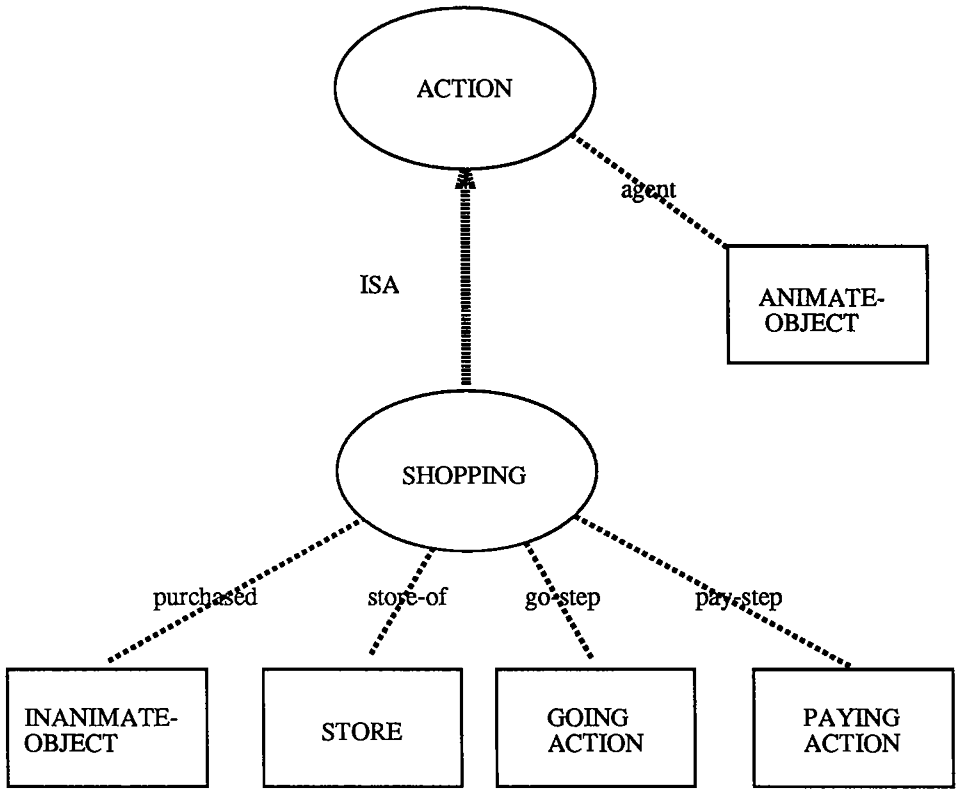


Figure 1.3: The frame for shopping plans. Frames are represented by ovals, and their slots by rectangles. Inside the rectangle for each slot is the type of object which fills that slot.

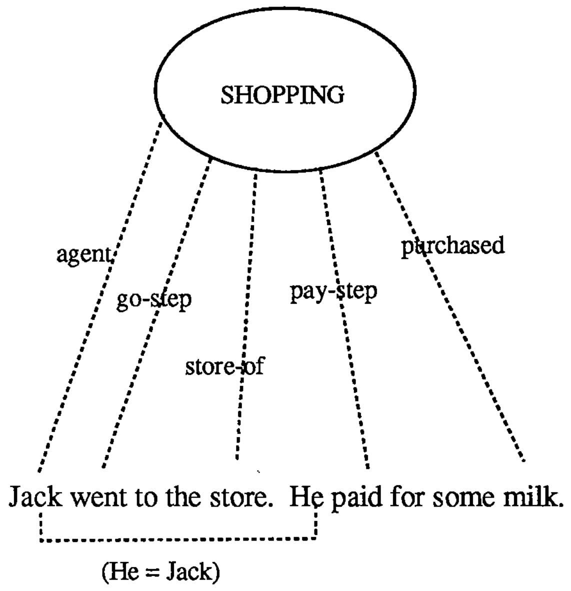


Figure 1.4: Matching the story "Jack went to the store. He paid for some milk." against the shopping frame.

```
;;;literal translation of the input
;;;'Jack went to the store.'
(inst jack1 boy)
(inst went2 go)
(inst store3 store)
(syn-rel subject jack1 went2)
(syn-rel to-pp store3 went2)
;;;literal translation of the input
;;;'He paid for some milk.'
(inst he4 boy)
(inst pay5 pay)
(inst milk6 milk)
(syn-rel agent he4 pay5)
(syn-rel for-pp milk6 pay5)
;;;Recognizing Jack's plan to go shopping.
(inst shop7 shopping)
(= (agent shop7) jack1)
(= (store-of shop7) store3)
(= (go-step shop7) went2)
(= he4 jack1)
(= (pay-step shop7) pay5)
(= (purchased shop7) milk6)
```

Figure 1.5: The translation of the story "Jack went to the store. He paid for some milk."

shopping that we've shown above, as shown in Figure 1.4. Figure 1.5 shows the internal representation as it might be stored in a database of statements. Note that slot-filler relationships are specified by equalities.

There are three aspects to the problem of story understanding: retrieval, confirmation or matching, and choice or competition [13]. A story understanding program must be able to retrieve the relevant frames against which to match the input. The matching process matches the input against the frames retrieved, confirming that they are acceptable matches, or rejecting them. Finally, if a number of different possible matches, or "understandings" result from the first two steps, the program must be able to choose the preferred match.

To summarize, then, the input to a story understanding program will be a story in some natural language. The story understanding program will apply its knowledge about language and the world it represents to generate a translation of this input. The translation will be in an unambiguous knowledge representation language. It will make explicit a deep understanding of the story. This understanding should include a recognition of the way the actions of agents in the story are organized into plans to achieve these agents' goals. A review of previous work in the area of story understanding is given in Chapter 2 of this thesis..

1.2 The Formalism

In this and the following sections of the introduction, we will give a brief overview of the salient points of the thesis. In this section, we outline the language we have developed to express linguistic problems in terms of random variables. Following that, we will show how these sets of random variables can be structured as belief networks, and how this structure helps us assess the quantities we need for our probabilistic models. Finally, we will show how Wimp3 actually constructs and solves these problems. The stochastic models Wimp3 reasons with in processing natural language stories are too large to be manageable by previously-existing techniques for probabilistic inference. In the process of constructing Wimp3, we have developed techniques for handling large problems of probabilistic inference which are applicable to other large problems.

In order to pursue a probabilistic approach to the problem of story understanding, our first problem was to come up with a clear understanding of what probabilities we were to manipulate. We have created a simplified formal language suited to expressing facts about a text's structure and meaning. We have devised a semantics for this language which allows us to treat its formulae as random variables. We show that this semantics gives us the guidance we need to devise a consistent set of probabilities for use by a story-understanding program. A complete account of this facet of the research, reviewed briefly here, is given in Chapter 4.

We will start by describing the notation of our language and showing how it is used to formulate the text understanding problem. For convenience's sake, we will proceed in a way which follows the traditional syntax–semantics–pragmatics distinction.[3] First we will show how the language can be used to describe the input text itself. Then we will go on to talk about how it can express semantic propositions. Finally, we will show the part of the language used for plan-recognition, and how this provides contextual information.

[3]The syntactic aspects of language are the structural aspects of language. Semantics is the way symbols of the language combine together to determine the meaning of a piece of discourse, independent of context. Pragmatics is concerned with the practical aspects of language use, in particular the effect of context on meaning. For the purposes of this thesis, we place problems of plan-recognition under the rubric of pragmatics. This is simply done for convenience's sake; it is not a theoretical claim.

The parser and morphological analyzer component of Wimp3 generate the statements that describe the natural-language input. These statements are of two types:

- specifying the words used (*word-inst*), and
- specifying relations between words used (*syn-rel*).

For example, the sentence

'Jack gave Mary the ball." (1)

would be translated into:

```
(word-inst word1 Jack)
(word-inst word2 give)
(syn-rel subject word1 word2)
(word-inst word3 Mary)
(syn-rel indirect-object word3 word2)
(word-inst word4 ball)
(syn-rel object word4 word2)
(syn-rel det word4 the)
```

In the case of syntactic ambiguity, the parser will generate and describe all possible parses of the input; semantic and pragmatic information will be used to choose between them. If the sentence were "Mary killed the boy with the poison." (2), the parser would give both (syn-rel with poison12 boy11) and (syn-rel with poison12 kill10).[4]

For semantic processing, we need to be able to express hypotheses about the denotations of the various words. Associated with each word instance is a constant representing the denotation of that word. We will write this as the same constant, but in boldface. We might more clearly write (**denotation word***n*), but this is too bulky. For example, if **ball27** was the constant representing the denotation of 'ball' in (1) above, two possible denotations might be expressed as:

```
(inst ball27 bouncing-ball)
(inst ball27 cotillion)
```

In order to express case relations, we have functions for the various cases. To return to example (2) ("Mary killed the boy with the poison."), the case relations corresponding to the different prepositional phrase attachments would be:

```
(== (accompaniment boy11) poison12)
(== (instrument kill10) poison12)
```

Moving into the gray area between semantics and pragmatics that Hobbs and Martin [54] call 'local pragmatics,' equality statements are used to express coreference. For example, part of the task of understanding "Jack gave Mary the ball. She liked it." (3) is to reason from

```
(inst mary24 girl-)
(inst she28 girl-)
```

[4]For ease of understanding we will use these more descriptive names, rather than word1, word2, etc.

to (== she28 mary24).

Finally, we need to be able to express a plan-based understanding of our texts, such that when reading a story like "Bill went to the liquor-store. He paid for some bourbon." (4) our program can understand the entire story as a description of a plan to buy liquor. Our database records what kinds of plans can explain each action or object.[5] E.g., a going action can be the **go-step** of a **liquor-shopping** the **go-step** of a **robbery**, etc. For this purpose we have another kind of proposition: the proposition that a given frame-instance (say **went2**) fills a slot in a given kind of plan (say **robbery**). This would be written

```
(and (inst robbery27 robbery)
     (== (go-step robbery27) went2)) (5)
```

The atom *robbery27* is written in a different typeface to indicate that it represents a different kind of constant. *robbery27* is a function of **went2**. Analogously to the the denotation constants, these constants might more clearly be written (**robbery-explanation-of went2**). Put differently, the formula given above as (5) is akin to an existentially quantified formula whose variable is *robbery27* and whose scope is contained within that of **went2**.

These propositions are virtually identical to those in our earlier logic for semantic interpretation [17], but their semantics is different. Following probability theory, the expressions of our language get their semantics from a sample space. This sample space is the set of all possible stories from some restricted universe. The word constants denote random variables whose domain is particular words. Likewise, the denotations of words are random variables whose domain is the set of entities (including events). Propositions are random variables whose domain is the set {true,false}. A more full discussion of the semantics of our language can be found in Chapter 4.

1.3 Network representations

In the past, it has been difficult to apply probability theory to reasoning problems because such problems did not have sufficient structure. It appeared that in order to apply probability theory, it would be necessary to have a gigantic joint probability table, giving the probability of all possible combinations of the propositions of interest. However, recently, there has been renewed interest in graphical representations for probability distributions, such as Markov networks, belief networks and influence diagrams. Belief networks have provided us with a way of structuring the problem of text understanding as a problem of probabilistic inference, and have simplified the problem of acquiring the requisite numbers.

Belief networks are a way of representing probabilistic dependency information in directed acyclic graphs. The nodes in a belief network represent random variables, and the edges represent direct dependence. There are three advantages to belief networks as representations for probability distributions. First of all, relations of conditional independence can be read off a belief network. Second, the probability distribution corresponding to a belief network may be represented locally. For each node, it suffices to provide a conditional probability distribution for each combination of values of its parent nodes. Finally, while in general the problem of determining the posterior distribution of a partially-instantiated belief network is NP-hard [25, 27], considerable attention has been devoted to finding efficient approaches to evaluating such networks. In Chapter 3, we give a brief introduction to the properties of belief networks, and other related graphical models.

We give a partial belief network representation of the analysis of example the sentence "Bill went to the liquor-store." (6) in Figure 1.6. This fraction of the network is intended to

[5]We have a frame-based database of events and objects.

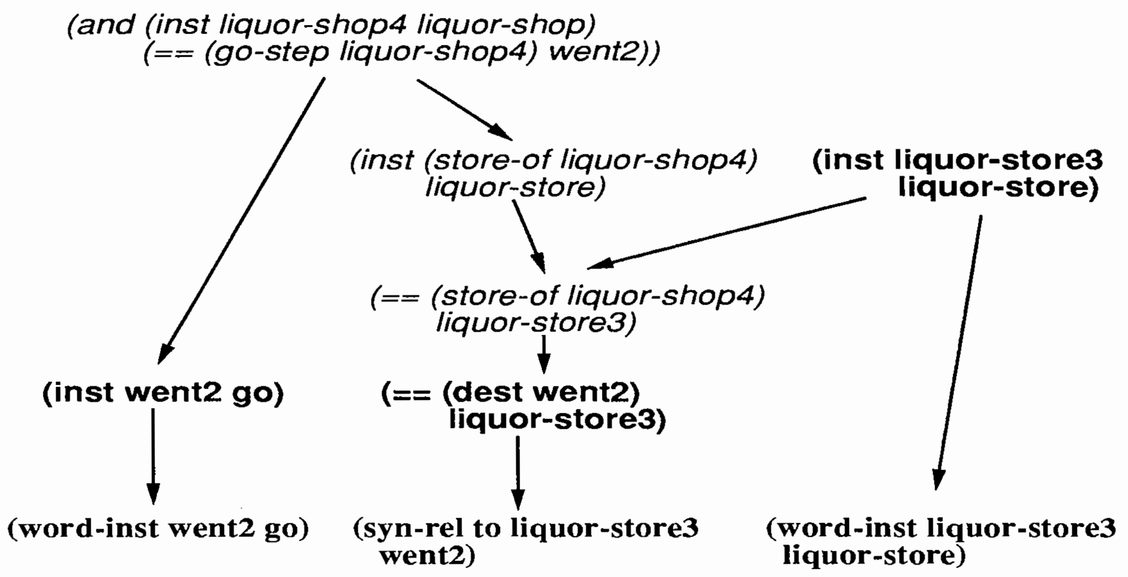

(and (inst liquor-shop4 liquor-shop)
(== (go-step liquor-shop4) went2))
(inst (store-of liquor-shop4) liquor-store)
(inst liquor-store3 liquor-store)
(== (store-of liquor-shop4) liquor-store3)
(inst went2 go)
(== (dest went2) liquor-store3)
(word-inst went2 go)
(syn-rel to liquor-store3 went2)
(word-inst liquor-store3 liquor-store)

Figure 1.6: A belief network representation of the story "Jack went to the liquor-store...."

capture the following relations: A liquor-shopping plan can cause there to be a go action. It will also cause there to be a liquor-store, the store-of the liquor-shopping. The liquor-store mentioned in the story, might be this liquor-store. The store-of the liquor-shopping will be the destination of the go-step of the liquor-shopping. These relations might be realized in the text sequence we have observed.

Note that this is an immensely simplified version of the networks which are, in fact, used in our program. First of all, we have suppressed many of the nodes that would appear in the network for this interpretation (e.g., the nodes specifying Jack as the agent of the go). Second, the networks we construct are not restricted to contain only nodes which are part of *correct* interpretations of the text!

1.4 Quantifying the networks

The belief network representation makes it easier for us to specify the probability distributions we need. It permits us to express the distribution as a collection of conditional probabilities involving only a few nodes (no more than 3 in our current program). This representation is compact, and these low-order probabilities are easier to assess subjectively.[6] In this section we will give a brief discussion of the kinds of conditional probabilities we need. A full account is given in Chapter 4.

For the sake of clarity, our discussion here will proceed from the leaves of the belief networks to their roots, in parallel with the syntax–semantics–pragmatics distinction, and our treatment in section 1.2.

- At the leaf nodes, we have words, with possible word-senses as their parents. So we require P(word|denotation).

- Other leaf nodes are syntactic-relations between words, whose parents are relations between the objects the words denote. We require P(syntactic-rel|relation to be denoted).

[6]While in principle, we could collect statistics to find these distributions, this is not practical.

- Higher in the graph, we need probabilities for predications of object types.
- Finally, we have equality statements. These nodes have as their parents, type predications of their arguments. We require

$$P(entity_1 = entity_2 \mid (inst\ entity_1\ type), (inst\ entity_2\ type)).$$

The leaves of the network are the word-inst and syn-rel nodes. To make up the conditional probability matrices for the word-inst nodes, we need P(word|denotation) – the probability that this root form will be used given that the author wishes to denote *denotation*. For example, in Figure 1.6, we need P{(word-inst went2 go)| (**inst went2 go**)}, the probability that this word will be a form of the verb 'go,' given that the author wishes to denote a going event with this word.

In principle, these probabilities could be computed from a labelled corpus. In practice, we do not have the resources necessary to do so. Fortunately, it turns out not to be necessary. What we have done for the moment is to make an indifference assumption. For almost all words, we assume the same P(word|denotation) for all denotations that word can express.[7] This amounts to allowing the prior, P(denotation) to dictate the ranking of the different word-sense hypotheses.

We do not have a good theory of the conditional probabilities for the syn-rel nodes. These are conditioned on relations in the real world [P(syn-rel|relation in the real world)] – given that the author wishes to convey a given relation between the things denoted by two words, what is the probability that he or she will use this syntactic relation? For example, P((syn-rel to liquor-store3 went2))|(== (**destination went2) liquor-store3**)) is the probability that the author will write "go to the liquor-store," given that he or she wishes to express the fact that the destination of the going event denoted by this instance of the word 'went' (which we're calling **went2**) is the liquor-store denoted by 'liquor-store'(**liquor-store3**). We use a rough subjective estimate of how often a given construction will be used to convey some relation. E.g., the instrument relation is more often expressed by 'with' phrases than is the accompaniment relation (which is often expressed by conjunction).

The probabilities deeper in the networks – explanations (the liquor-shopping explanation in Figure 1.6) and equalities (the possibility that the store-of the liquor-shopping is the store mentioned in the text) – are derived from a distribution over the frames in our isa-hierarchy. In Chapter 4, we give the full details of this distribution. For the moment, let us just say that we have tried to tie these probabilities to the frequencies of events in the real world. We want to do this so that, e.g., when our program reads a sentence like "Mary went to the airport." it will favor explanations like 'Mary is going to meet someone on the plane,' or 'Mary will fly somewhere,' over 'Mary is going to hijack the plane.' This is because we're restricting our program to reading stories about events in the real world, written by an author who is, as near as possible, simply translating events into text. In theory, one could collect statistics for these networks by labelling a large corpus of stories, and counting nodes in the instantiated networks.

The equality statements are also based on this distribution over our isa-hierarchy. These probabilities are needed for coreference and for linking together explanations for different objects (we must recognize that the shopping plan that explains the presence of a liquor-store in a story is the same plan that explains the presence of a bottle of bourbon). These are difficult to assess; Chapter 4 and [18] discuss the problem of assessing such probabilities. In brief, these probabilities must be sensitive to the number of different objects of each type are likely to appear in a given story. For example, in simple stories like those Wimp3 reads,

[7] We will not discuss exceptions to this policy here. See Chapter 4.

```
(→(word-inst ?i ?word) :label ?A
   (→←(word-sense ?word ?frame) :label ?B
        (inst ?i ?frame) :label ?C)
   :prob ((?C →?A) ((t | t = ?B) (t | f = prior)) ))
```

Figure 1.7: One of the rules used to handle word-sense relationships.

$$P((== \text{person1 person2})|(\text{inst person1 person}), (\text{inst person2 person})) \\ < P((== \text{airport1 airport2})|(\text{inst airport1 airport}), (\text{inst airport2 airport}))$$

because it is very likely that there will be more than one person discussed in a single story, but unlikely to be more than one airport. We hope that discourse theory (e.g., [46, 109]) will shed some light on these quantities, and that notions like focus can be used as conditioning events.

1.5 The Program

In the preceding, we have introduced our probabilistic model of stories. Now we will outline how we use this model to do story understanding. Because the network models we have outlined here are impractically large, we proceed by incrementally constructing and evaluating relevant portions of the full network. These partial networks are constructed by network-building rules which are similar to forward-chaining rules. We have experimented with evaluating them using a number of different belief network algorithms.

Wimp3 works as follows: an all-paths parser reads the English-language input, one word at a time, and yields statements about the input (the *syn-rel* and *word-inst* statements). These statements are given to the network-construction component. This component extends the belief network corresponding to the input. Then the network evaluation component computes the posterior distribution of the network. If any hypotheses seem particularly good or bad, they may be accepted or rejected categorically. At this point, control is returned to the parser, which reads the next word.

We have developed network building rules for constructing belief networks which correspond to story understanding problems. These rules are similar to the forward-chaining rules in a conventional TMS. They differ in that rules are not restricted to adding justifications to some derived statement. In fact, since we are trying to build networks for a diagnostic problem, we will usually be extending our networks *above* the triggering node: i.e., adding possible causes for this statement. To put it differently, our rules will typically be triggered by the heads of arcs they add, rather than by the tails. Our network-building rules also provide more information than simple connectivity: they contain information used to compute the conditional probability matrices of the nodes in the network.

A sample network-building rule is given in Figure 1.7. This rule handles part of the problem of finding an appropriate referent for a word (the rules for plan-recognition are similar, but more complex). The meaning of the rule is:

> If a node is added to the network describing a new word-token, and if there is a statement in the database specifying that one of the senses of this word is ?frame, add a node for an instance of the type ?frame. Draw an arc between this newly-added node (?C), and the word-inst node (?A). Construct the conditional

probability matrix for the word-inst node using information from the word-sense statement (?B).

For example, if we were to learn of a use of the word 'bank,' (say bank1) and knew of two meanings: financial institution and river bank, this rule would tell us to add two nodes to our network: a node specifying that bank1 referred to a financial institution, and a node specifying that bank1 referred to a river bank. These two nodes would have arcs pointing to the node for (word-inst bank1 bank). The conditional probability matrix for the latter would be drawn up to reflect that the two possible senses of the word are related as exclusive causes. I.e., the probability that the word will be used given that one wishes to describe a financial institution or a river bank is some value determined by the word-sense rule; the probability that the word is used given one wishes to express both senses simultaneously is zero, and the probability that the word will be used given neither of these senses is meant is some default value.

We have similar rules for syntactic relations, reference resolution, etc. Initially we used this kind of 'forward-chaining' for plan-recognition, as well. However, this led to an explosive growth in the belief networks. The reason for this is simple: consider how many possible reasons there are for 'going,' for example. To provide more direction to the way the networks are expanded, another researcher at Brown, Glenn Carroll, has developed a marker-passing search algorithm. For a discussion of the marker-passer, see [9].

The posterior distribution for the belief network constitutes the interpretation of the input text. To see why this is the case, refer to the sample belief network given as Figure 1.6. What we want to assess is the probability that Jack is going to buy liquor, given that we have been told "Jack went to the liquor-store." We can read this information off the network, once we have evaluated it so that it reflects the posterior distribution – the probability of each node given the evidence in the story. It should be noted that our representation does not tie us to the idea of solution by computing posterior distributions: we could compute a maximum a posteriori instantiation of the networks to get a best global interpretation.

We have been experimenting with a number of different algorithms for computing the posterior distribution of these networks. We have been using the IDEAL system[106, 107], an environment in which one can define a belief net or influence diagram and apply to it a number of different evaluation algorithms. We find that a variant of the algorithm of Lauritzen and Spiegelhalter[72] developed by Jensen, et. al.[59], gives the best results on networks like ours.

Wimp3 is has been fully implemented. It is written in Common lisp, and runs on Symbolics Lisp Machines and Sun SPARCstations.

1.6 Summary

There are two important aspects to the work described in this thesis. First of all, we have constructed a probabilistic model of story comprehension. This makes it possible to address the problem of text understanding within axiomatic probability theory. Second, we have developed a way of constructing and evaluating probabilistic models on an as-needed basis. This makes it possible to apply probability theory to problems for which a precompiled model is either not available, or impractically large.

In the next chapter, we will review previous work in story understanding, shaping our account to show how this earlier work has influenced our own. In particular, we will try to explain why we have chosen to work with probability theory. Having done so, we move on to a brief discussion of various graphical models for probability distributions, in the following chapter. We will show how these models help us manage probabilistic reasoning, and explain why we have chosen to use belief networks, rather than some other graph representation.

These first three chapters give the necessary foundation to discuss the original work done in this thesis. The next two chapters discuss our foundational work. Chapter 4 details the probabilistic model of stories Wimp3 uses in making its interpretations. As mentioned above, this model is far too large for a straightforward application. Chapter 5 discusses the dynamic construction and evaluation of belief net models.

The remaining chapters of the thesis detail the construction of Wimp3. Chapter 6 gives an outline of its architecture. In Chapter 7, we discuss the knowledge Wimp3 draws on, paying particular attention to the network construction rules it uses. Then we give a detailed example of Wimp3's processing. We close with a summary, and discussion of the research issues raised by Wimp3.

Chapter 2

Previous work in story understanding

In this chapter of my thesis, I will review previous work in story understanding. In doing so, I will try to show some of the problems left open by previous approaches, and how we have tried to solve this problems in Wimp3. In order to make comparisons more understandable, I will discuss these programs in terms of the three sub-problems isolated earlier (see page 6): frame retrieval, matching and competition between explanations. Since the contributions of this thesis are primarily in the competition sub-problem, I will concentrate on this aspect of previous work.

2.1 SAM

One of the first story understanding programs was Cullingford's Script Applier Mechanism (SAM) [30, 31]. SAM read newspaper stories about stereotyped situations like official visits, earthquakes and automobile accidents. As its name suggests, SAM was concerned only with the problem of matching input with scripts. The problems of retrieval and competition between competing interpretations were beyond its scope.

SAM was based on the theories of Schank and Abelson[101]. Schank and Abelson hypothesized that people store bundles of information about stereotyped situations in scripts. The scripts SAM used are data structures similar to our frames, but less powerful.

SAM had three modules: a conceptual analyzer, PP-memory and the script applier itself. The conceptual analyzer (ELI) was a semantic parser, which translated natural language input into Schank's Conceptual Dependency language.PP-memory identified known entities in the input (e.g., places, heads-of-state).

The script applier matched the input against the various scripts SAM knew. When it found a match, it would flesh out the story by inferring events not explicitly mentioned, and use its scriptal knowledge to disambiguate references. for example, if it read "Jack went to a restaurant. The waiter handed him a menu." SAM was able to recognize 'him' as referring to 'Jack,' because it knew that the customer in the restaurant script is the one who consults the menu.

This division of labor was carried over into the later story understanding programs from the Yale AI project (PAM, ARTHUR, etc.). All used a smart parser to yield a semantic representation of the natural language text, and took that semantic representation as their input. Some similar assumption is made by all of the other programs discussed in this section, with the exception of the Wimp series.

SAM made no attempt to attack the problems of retrieval or competition. SAM's database contained only a handful of scripts, and it would simply compare each, in turn, with the input CD representation. Similarly, SAM didn't handle competition between scripts. It just picked the first one which matched. SAM's scripts covered a small set of very different situations (automobile accidents, transit trips, eating in restaurants, etc.) so there was little danger of confusion. Few actions are shared between automobile accidents and dining in restaurants.

A later program in the same vein was DeJong's FRUMP (Fast Reading Understanding and Memory Program)[36]. This differed from SAM only in that it was designed to skim its input, getting only a sketchy understanding of the story, but with greater tolerance for noise.

SAM was limited to simple stories by the nature of its knowledge representation. Unlike frames, scripts cannot nest hierarchically, so each script must describe an event in terms of primitive actions. For example, if one wanted to include going to the restaurant as one of the scenes of the restaurant script, one would have to multiply out the different scripts into $RESTAURANT-BY-BUS,$RESTAURANT-BY-CAB, $RESTAURANT-BY-CAR, etc. Cullingford finessed this problem to some extent (e.g., by specifying allowable transitions between scripts), and it was not a practical problem for SAM because of the simple kinds of story SAM was designed to understand.

2.2 Ms. Malaprop

Another of the earliest story understanding programs was Charniak's Ms. Malaprop [11]. Charniak's work differed from Cullingford's in placing much greater emphasis on the software engineering aspects of story understanding. His knowledge representation scheme was a more modern frame language, paying attention to issues of modularity and information hiding. He used a Truth Maintenance System (TMS) which allowed Ms. Malaprop to detect and remove incorrect assumptions. The program was more a test bed for Charniak's representation scheme than anything else, and so was less fully fleshed-out than SAM. Finally, developments in Charniak's knowledge representation scheme outstripped the state of the art in search for story understanding, so in some ways it was a less capable program than SAM.

Charniak's knowledge representation language was more carefully-designed than Cullingford's. He used a full-fledged frame language, in which frames were used to represent plans, events, states, objects, etc. In contrast, SAM had different memory structures for events, scripts and objects. The frames could be nested, unlike SAM's scripts. Furthermore, plans were broken down in terms of subgoals, rather than sub-steps. For example, the final step of the painting plan was to make sure that there was no more paint on the instrument, not 'wash the paintbrush.' This kind of modularity would allow adding, *e.g.,* knowledge about new kinds of painting implements, without having to revise the painting frame.

Ms. Malaprop only understood stories about painting, but it understood this domain more deeply than SAM understood stories any of its set of domains. Ms. Malaprop's database included a large number of the cause and effect relationships necessary for understanding the way one paints an object. *e.g.,*the reason one washes the brush after painting is so that paint does not dry on it, rendering it 'unabsorbent.'. While SAM had scripts for a wider variety of domains, it had no knowledge of why an action occurred in a script. *e.g.,*in the restaurant script, the waiter brings the customer a menu just because it is always done that way, not in order that the customer can choose his or her meal.

Ms. Malaprop used a TMS [40, 41, 78] to record the assumptions on which her decisions depended. If incorrect assumptions were made when Ms. Malaprop was matching her input against a stored frame, leading to a later contradiction, these assumptions would be revoked, and new inferences made.

Ms. Malaprop was not as full-blown a program as SAM. In particular, it did not have a natural-language front end. Instead, Charniak gave it a semantic representation of text as input. Furthermore, Charniak made no attempt to tackle the problem of frame retrieval in Ms. Malaprop. Unlike SAM, Ms. Malaprop would not match actions against possible frames in search of an explanation – she would only match against frames that were specifically activated. So Ms. Malaprop would only understand stories like "Jack wanted to paint a chair. He put a brush in some paint." It would *not* understand stories like "The waiter brought Jack the menu. He decided on a hamburger." because it doesn't specifically mention the frame which needs to be activated.

Even when restricted to top-down search, Ms. Malaprop's knowledge representation outstripped her ability to search. Charniak offers as an example the story "Jack went to a restaurant. The menu was in Chinese." Ms. Malaprop would not be able to notice Jack's problem, because the second sentence wouldn't match anything in the restaurant frame – only in the frame for know-language, which would be nested inside the read-object frame, which would be inside restaurant. This was not a problem for SAM, because of its flat data structures; all information was represented where it was needed, at the expense of redundancy. What is needed in order to handle this problem is a search regime which proceeds bottom-up, as well as top-down. Such a regime was introduced as part of PAM (see Section 2.3, below).

The software engineering kind of concerns with knowledge representation that first appeared in Ms. Malaprop will reappear in other work done at Brown University. We will see below a program, BRUIN, done by Doug Wong, under the supervision of Eugene Charniak, that is very similar to Wilensky's PAM, but which is motivated by concerns with readability and modularity of knowledge representation. In general, work at Brown has been characterized by an interest in applying general-purpose knowledge representation schemes to particular inference problems. This work is situated somewhere between work like that done at Yale, in which it is considered important to tailor data structures for particular inference problems, and more formal work (e.g., [50, 49, 48, 69]), in which it is considered of the essence to concentrate entirely on representing knowledge in a language with clean semantics, disregarding the issue of the use of that knowledge. For this reason, and because we are interested in the kind of (necessarily) poorly-understood issues which interest the 'scruffies,' we think of work in the 'Brown tradition' as being on the neat fringe of scruffy AI.[1]

2.3 PAM and BRUIN

Wilensky's PAM (Plan Applier Mechanism) [110, 111] was an attempt to implement the aspects of Schank and Abelson's theories that had to do with 'higher level' processing: employing novel, unpredictable plans to achieve goals. PAM was also able to understand stories involving interactions between goals.

[1]Kautz writes:

> Workers in Artificial Intelligence are often divided into the neat and scruffy camps, with the neats trying to create formal theories which systematize the heuristics uncovered by the intuition-driven scruffies.

In practice, the two camps often degenerate into unrealizable logicism or unprincipled (and unreproduceable) hackery.

PAM was able to recognize novel plans because PAM's plans were hierarchical data structures. They were similar to Ms. Malaprop's frames, but with less concern for questions of readability and modularity. PAM's plans were defined in terms of abstractly-specified subgoals, rather than sequences of primitive actions. For example, PAM might have a get-rich plan. A couple of ways of getting rich are to win the lottery and to kill your rich uncle. In turn, PAM might know of 4 ways someone might be murdered. This would allow for a spectrum of $2 \times 4 = 8$ possible get-rich-quick plans, that could be described with only 5 data structures, rather than 8. In addition to more efficiently organizing the database, hierarchical plans allowed PAM to recognize novel plans. Someone setting up the database would not have to foresee the application to getting rich when describing recipes for murder.

PAM tried to explain stories by linking characters' actions with their goals. These goals could be explicitly mentioned in the story ("Jack wanted to get gasoline."), or recognized bottom-up ("Jack went to the gas-station"). PAM searched from actions to find such goals.

One problem with PAM was its knowledge representation (KR) scheme. PAM's representation for its scripts, plans and goals was very difficult to read and write. Further, these data structures were specifically tailored to the problem of story understanding. This knowledge was, in turn, applied to this problem by a slew of production rules for recognizing the different plans, etc. and for understanding their interactions.

Wong's BRUIN work addressed the problem of KR design for story understanding [113]. BRUIN was a program which attempted to duplicate the accomplishments of PAM, but in a tidier way. Wong used FRAIL (FRame-based AI Language) to describe plans, objects, etc. FRAIL was a more modern KR scheme (indeed, we still use a variant of FRAIL today), with inheritance [14]. It allowed more compact, easily understandable, knowledge representation. Furthermore, knowledge expressed in FRAIL could be used for planning, as well as plan-recognition. In contrast, PAM's ability to understand a story about going to the post office to buy stamps didn't entail any ability to originate a plan to buy stamps, nor could its knowledge of such plans be used by a different program that *was* able to generate plans.

BRUIN's way of applying its knowledge was also easier to understand than the the procedural aspects of PAM. Rather than having production rules tied to specific data structures, BRUIN had a content-independent algorithm for searching its library of plans. This algorithm worked identically in its three domains (blocks world, children's stories, factory inventory). This routine was more limited than PAM's in the class of stories it could understand, however. BRUIN did not treat goal interactions.

Both PAM and BRUIN had troubles with the problems of retrieval and competition between explanations. Both programs tried to search bottom-up from actions described in their input to active plans or goals. This approach worked well with input that started by specifying a plan or goal, and then elaborated on it. E.g., "Jack wanted to kill himself. He took a lot of sleeping pills." It didn't work well when the stories described a set of actions that could play parts in many different plans. In this case, the programs could 'get lost' searching bottom-up. For example, processing a story like "Jack got a rope. He wanted to kill his rich uncle." PAM and BRUIN might be offered the problem of choosing between the jump-rope explanation for getting the rope and the killing (via hanging) explanation. If the program chose the jump-rope interpretation, it would be unable later to recognize that getting the rope was part of Jack's plan to kill his rich uncle. See Figure 2.1.

Neither PAM nor BRUIN handled the problem of competition particularly well. It is difficult to tell exactly how PAM handled this problem, since there was not a clear distinction between retrieval, matching and competition in PAM: all were tied up in the operation of PAM's production rules. Wilensky assumed that all but one interpretation would eventually

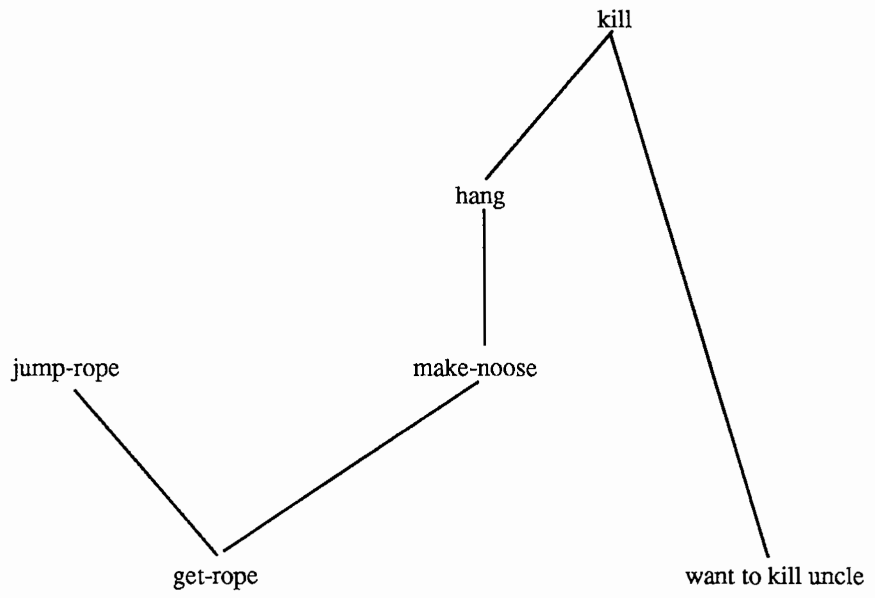


Figure 2.1: A simple example of difficulties with bottom-up search for PAM and BRUIN

be eliminated [110, p. 37].

BRUIN had a simple measure for choosing among different possible explanations: it chose the explanation which required the fewest assumptions. By assumptions we mean statements added to the database. The problem with this criterion is that semantically identical explanations could be described in more than one way, possibly making this measure inconsistent. It also makes it impossible to account for differences in a priori likelihood of explanations. For example, using this measure, a plan to blow-up an airplane seems just as good an explanation for going to the airport as does travelling by air. Unless we know some particular reason for believing that someone is a hijacker, though, we are going to assume the latter explanation.

2.4 ARTHUR

Granger's ARTHUR (A Reader THat Understands Reflectively) [95] was a program from the Yale Artificial Intelligence project that was concerned with the process we've described as explanation competition. It was a successor to PAM that was concerned with achieving more efficient memory use, and with detecting and correcting incorrect inferences made in story understanding.

To interpret its input, ARTHUR used a bi-directional search method Granger called "specification search." Input to ARTHUR was categorized as either describing goals or actions. Information about goals or plans would cause ARTHUR to retrieve possible ways of meeting these ends (predictions). In response to information about actions, ARTHUR would retrieve plans, goals or scripts that might be served by these actions (explanations). In order to choose an interpretation, ARTHUR took the intersection of the predictions and explanations.

ARTHUR's memory was extensively indexed for the retrieval process. Associated with each action in ARTHUR's memory was the set of plans that could use that action. Likewise, each plan or goal had associated with it the set of ways it could be realized. These lists were further indexed by slot-fillers. E.g., the set of getting events might be further divided into getting events whose patient is food, getting events whose patient is money, etc.

Consistency checking was done during the retrieval process. When ARTHUR retrieved a plan or a prediction, it would be matched against the input which caused it to be retrieved. This would cause slots to be filled, further fleshing out the explanation or prediction, and possibly causing it to be rejected as inconsistent. Further matching would be done when the explanations and predictions from different parts of the input were intersected.

Specification search solved some of the problems of bottom-up search that PAM had. Recall that PAM could 'get lost' searching bottom-up from actions that could play a part in more than one kind of plan (see Figure 2.1). This wouldn't happen to ARTHUR, because it retained all possible bottom-up inferences from the input.

On the other hand, ARTHUR didn't handle the problem of competition, as we understand it. Granger still assumed that after reading an entire story, there would only be one interpretation consistent with the input. There was no concept of what to do if there was more than one element to this intersection.

Conversely, ARTHUR had no way of handling stories where the specification search resulted in an empty intersection. This would happen if a story had to be understood as describing more than one plan, e.g., "Jack went to the movies. Then he went to the supermarket."

Furthermore, ARTHUR's indexing scheme was not very efficient, because it did not have any concept like inheritance which would allow for representing information about similar entities together. For example, ARTHUR's memory entry for the *New Yorker* magazine contained a pointer to the `$RESTAURANT` script (because the listings section of this magazine can be used to find restaurants). It also contained a pointer to the plan to read for pleasure, etc. If ARTHUR also knew about the *Village Voice*, it would have a memory entry that duplicated that for the *New Yorker*. An obvious optimization would be to have a memory entry that would package together everything known about such reading-matter. The marker-passing systems discussed in the next section provide just this kind of capability.

It is also difficult to see how to cache complicated plan-recognition inferences in flat list indices like ARTHUR's. Consider a search problem like that sketched out in Figure 2.2. What would be the correct way to index it such that it could find *Plan* searching from A and B by simple specification search? At the very least, it would seem necessary to include the bottom-up indices from C and D in the bottom-up indices for A and B, respectively. This could clearly lead to an explosion in the size of indices.

Finally, ARTHUR relies on being able to decide a priori whether one will need to search bottom-up, or top-down from a given piece of input. That is, it is necessary to be able to decide when constructing the database, whether a given observation should be classified as a plan/goal, or as an action. If it is the former, it should give rise to predictions; if the latter, explanations. But what about a sentence like "Jack got on the bus"? We need to be able to infer both top-down and bottom-up from this sentence: we need to be able to understand it when followed by "He gave the bus-driver a token." (this input should be predicted by activating the frame for bus-trips) and when followed by "He got off at the airport." (we should be able to activate the frame for air-travel).

2.5 Marker-passing systems

Marker-passing, or spreading activation, is a cheap form of bi-directional search. It was first developed, and applied to the problem of text understanding by Quillian [93]. Later programs like Wimp [15] and FAUSTUS [85] retrieved plans by passing marks through a semantic network of plans. Riesbeck and Martin [96] used marker-passing in a story

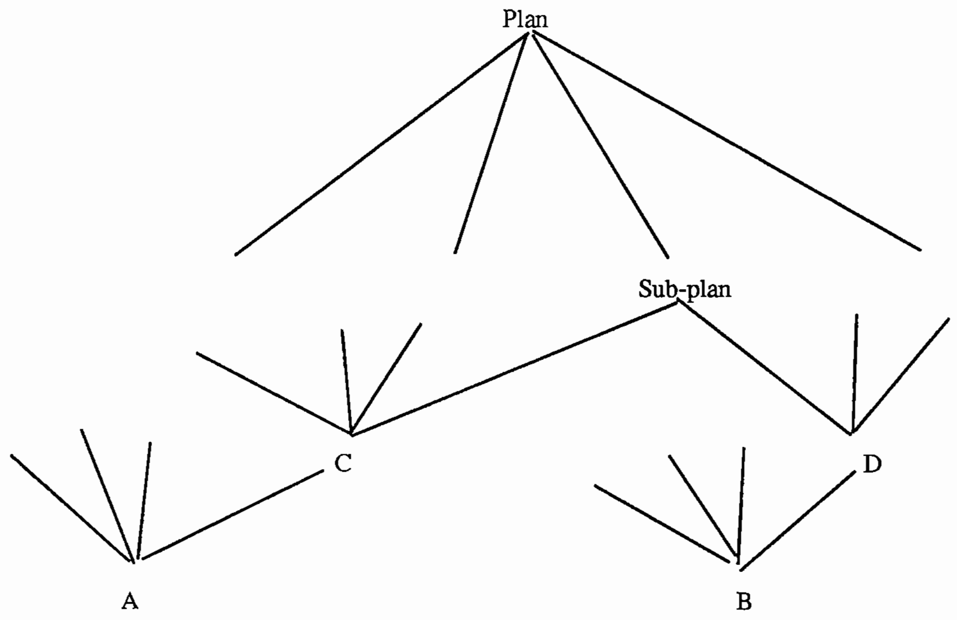


Figure 2.2: A search problem that would be difficult to cache.

understanding system that emphasized the retrieval process at the expense of matching and competition.

Marker-passing is not a well-defined concept, but some features common to the different marker-passing systems can be listed.[2] Marker-passing is a search technique to be applied to problems where search cannot be easily directed. What is done, then, is to make the operations of search very cheap, so that search can radiate out in all directions. Marker-passing is bi-directional: search proceeds from two or more points in the search space, and an answer to the search problem is either a path between some points in the space, or simply a response that such a path exists.

Procedurally, marker-passing works as follows: Starting at one endpoint of the search, mark the node and all of its neighbors. Then mark all of the neighbors' neighbors, and so on. After this sweep has been carried out, start marking from another endpoint of the search. In this sweep, when you encounter a node marked in the first pass, collect it as an intersection point. See Figure 2.3 for an example.

There are a number of variations on the theme of marker-passing, designed either to limit its search, or to make it yield particular kinds of response. To limit the search, one might set a horizon – a limit of how far from the origin marks will be passed. The approach sketched in the previous paragraph is sufficient only to detect the existence of a path between two nodes. It is common to mark the node with information about the source of its activation. This makes it possible to return paths between the endpoints.

Story understanding presents us with a very poorly-constrained search problem, suited for the application of marker-passing. As we have seen in the previous discussion, we would like to be able to combine information from many levels of a plan network – mixing bottom-up and top-down inference – in the search process. Furthermore, it is appropriate to have a search method that involves inexpensive operations, to be followed by a more expensive matching process. Consider again the diagram given as Figure 2.2. It is better to find the interpretation of actions *A* and *B* in terms of *Plan* before considering in depth their

[2]See [51] for a thorough discussion of marker-passing and review of its applications.

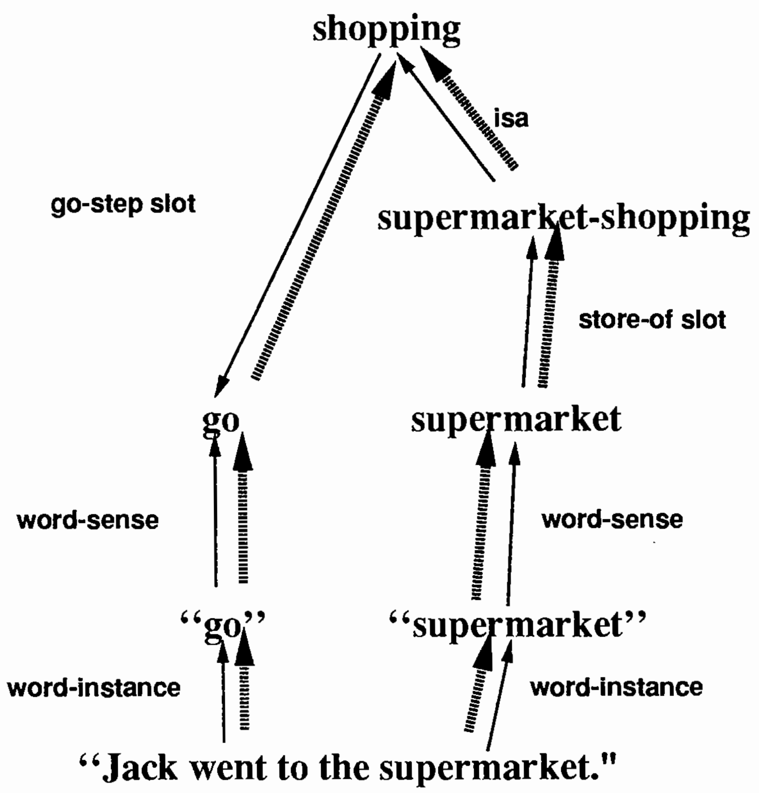


Figure 2.3: This figure shows how marks might be passed through a associative network of frames, from nodes for the words "went" and "supermarket," to retrieve the supermarket-shopping frame. In this example the marks (large dashed lines) meet at the shopping frame. The smaller, solid arrows, are the links in the associative network, labelled in small type (*e.g.,*the word "went" is a form of the verb "go," one sense of which is the frame go, one of whose uses is in the shopping plan.)

explanation in terms of all of their immediate parents.

Charniak's Wimp [15] and Norvig's FAUSTUS [85, 86] are story understanding programs that used marker-passing as their search mechanism. The basic thrust of these two projects was similar, but they emphasized different aspects of the problem. Charniak's research was concerned with giving a clear interpretation of the paths resulting from marker-passing, and also with integrating constraints from different levels of analysis (syntactic, semantic and pragmatic). Norvig was more concerned with controlling the marker-passer: he developed a taxonomy of acceptable paths, and a cheap way of filtering paths.

Wimp and FAUSTUS retrieved plans by passing marks through a semantic network of plans. A path of marks between two story entities (actions or objects) corresponded to a set of assertions which constituted recognizing a plan. For example, given the input "Jack went to the supermarket. He found some milk on the shelf." a marker-passing program would recognize the 'supermarket-shopping' plan by passing marks from the supermarket frame to supermarket-shopping, and from milk to food to supermarket-shopping. Only after such a path was found would the program attempt to check an interpretation for consistency (by matching the input against the stored frames).

One important development of the work on the Wimp program was an interpretation of the paths produced by the marker-passer. Charniak showed that they could be viewed as 'backbones' of abductive proofs. On at least one account, the process of abduction is to find a set of facts that entails the explanand. Charniak showed that the path returned by the marker-passer, together with some "abductive assumptions" would logically entail the input. Furthermore, he showed that the abductive assumptions were just the kind of assumptions made during the process of script/frame matching in earlier story understanding programs.

Wimp was also distinctive in presenting an integrated approach to text understanding. Earlier story understanding programs typically assumed a front-end that produced a semantic representation of the natural language input. Syntactic and semantic ambiguities were to be resolved before activating the plan-recognition component. Conversely, the plan-recognition component was unable to use syntactic and semantic cues in its own inference process (e.g., when resolving problems of pronoun reference).

Wimp allowed its parser (a conventional, top-down atn parser) to constrain its plan-recognition component, and vice versa. Wimp's parser generated a set of assertions describing lexical items and syntactic relations. The first step in Wimp's marker-passing was the step from the assertions about the instances of words to the things they might denote. The syntactic relations supplied constraints to the path-checker. E.g., if Wimp read a sentence like "Jack drove home from the airport." one path it would retrieve would be the one from drive through air-travel to airport, representing the possibility that Jack was going to the airport to fly somewhere. This interpretation would be ruled out by the fact that Jack was driving *from*, rather than *to*, the airport; a piece of information that comes from the syntax of the sentence.

Wimp's plan-recognition also provided guidance to its parser. When an ambiguous input was encountered, the parser presented all possible parse paths to Wimp. When Wimp selected an interpretation it preferred, it would inform the parser which parse lead to this interpretation.

The most distinctive feature of Norvig's FAUSTUS was the way it filtered the paths it got from marker-passing. Typically, because of its lack of direction, marker-passing can yield a high ratio of bad paths to good paths. This was observed by both Charniak and Norvig, as was the fact that some of the bad paths could be detected simply on the basis of their form. For example, a good path shape would be like the one in Figure 2.4. It represents the fact that two objects can be meaningfully related by filling slots in the same

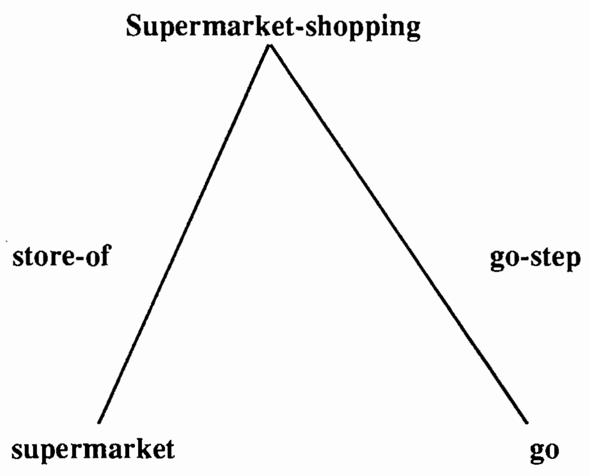


Figure 2.4: An acceptable path.

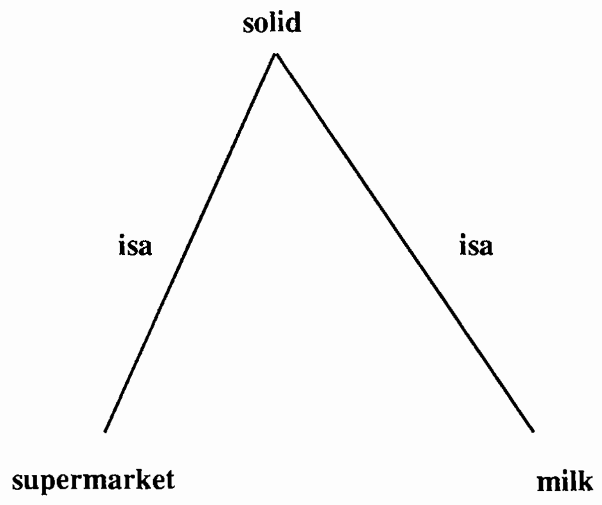


Figure 2.5: A bad path.

plan – the supermarket can be the store-of a shopping plan, and the go fills the go-step of the plan. On the other hand, Figure 2.5 shows a bad path. No useful information is conveyed by detecting that two different objects are sub-types of the same type. It is not of any use to us to notice that the milk and the supermarket are both solids. Norvig systematized this information by developing a regular expression grammar of good paths. This grammar described how links could be traversed to give useful information. Before carrying out the matching operations for any path, FAUSTUS would check to see that the path corresponded to its grammar.

Even with these filters, marker-passing typically yields more paths than strictly necessary for generating an interpretation. Some of these surplus paths are simply incorrect (they are rejected because of constraints not available to the marker-passer), some are redundant (they represent identical, or weaker, inferences), and some represent genuine ambiguity in the text. This causes problems for programs which use conventional deductive databases. Such databases are only able to represent a single state of affairs, which must be logically consistent. If there are several possible paths, it is expensive to test them for consistency, since each test requires resetting the database.

Both of these programs had trouble with the competition phase. In the following, I'm going to sketch how Wimp handled this problem; FAUSTUS used techniques that were broadly similar. When forced to choose between different interpretations, Wimp had to rely on a fairly ad hoc criterion of predictive power. Roughly speaking, this was a function of the number of true statements explained (entailed) by the explanation, in comparison with the number of statements which had to be assumed in order to instantiate the explanation. The obvious problem is that this criterion is crucially dependent on the syntactic form of the database.

Furthermore, in cases where there is genuine ambiguity (when all of the explanations have the same explanatory power) such programs have no alternative but to defer the decision. This entails a great deal of wasted effort, since the work of checking the various paths for consistency must be thrown away, and later redone, when more information becomes available. The next two sections will outline two different approaches to this problem. Both hinge around representing disjunctions for language interpretation.

The reader should note that this is a very partial review of marker-passing for language understanding. There is another 'tradition' of marker-passing work in text understanding, that starts with the original work of Quillian [93] and continues in the work of Riesbeck and Martin [96]. This is a tradition in which the *only* inferences necessary come from the marker-passer. An approach like this necessarily concentrates on getting the knowledge representation engineered so that only the correct structures will be retrieved. As such, it has little to say to us directly, since we are concerned with the problem of choosing between different alternatives that are retrieved. However, indirectly, we can benefit a great deal, since work like this pushes the state of the art of knowledge representation for text understanding.

2.6 Kautz and Allen

Kautz and Allen proposed a formal, deductive approach to plan-recognition [64], which was further developed in Kautz' Ph.D. thesis [63]. Kautz and Allen's approach was to augment their database of plans with axioms that explicitly represented various assumptions. In conjunction with this augmented database, one can recognize plans by deduction from the observations.

The axioms that Kautz and Allen add to the plan database make explicit two assumptions. The first assumption is that the plan hierarchy is closed and complete. This has the effect of allowing them to deduce from an observed action, the disjunction of possible plans using that action.

The second assumption is that observations should be explained by postulating a minimum number of plans. In order to formalize this assumption it is necessary to specify a restricted set of plans as 'top level' plans, and define the plan-recognition problem as identifying the top level plan or plans which account for a set of observations. Then this assumption may be expressed by minimizing the number of top-level plans.

These assumptions are formally expressed using circumscription. Because of their special nature, the second-order circumscriptive axioms can be 'cashed out' into first-order expressions for any specific plan-recognition problem. The first assumption may be compiled into the database ahead of time. E.g., their axiomatization of the cooking world domain contained the following axiom:

$$\begin{aligned}&\forall\, x\ .\ \mathrm{MakeNoodle}(x) \supset \\ &\qquad \exists\, y\ .\ \mathrm{MakePastaDish}(y) \wedge x = \mathrm{step1}(y)\end{aligned}$$

This states that any action which is an action of the type MakeNoodle is done as part of a plan to make a pasta dish.

The second assumption is handled in a more cumbersome way: initially, an axiom is added to the database specifying that the story must be understood in terms of one top level plan. If that fails, this assumption may be revoked, and the process started over, with the assumption that there are two top level plans, and so forth.

The Kautz and Allen paper showed how plan-recognition could be formalized in deductive terms. Kautz' thesis described a graph-based implementation of the theory. In brief, the approach is to construct for each observed action, a path from that action to the top-level plan in the plan hierarchy. This corresponds to finding all ways this action can be understood as playing a part in a top-level plan. After this is done for each action, the various graphs are collapsed together. This is done to prefer explanations which are coherent (*i.e.,* in which all actions are done for some common purpose or purposes). Kautz offers several heuristics for grouping observations into top-level plans for the troublesome cases where a set of observations must be understood in terms of more than one plan.

One of the most profound contributions of this work is that it gives a way of efficiently representing disjunctions. The graphs representing paths from observations to explanations compactly represent all possible interpretations of the observations. Representing such disjunctions was beyond the capability of early deductive databases, which concentrated on representing single consistent states of the world.

Kautz and Allen's research is one of the finest examples of the 'neat' approach to AI. They have given a clear formalization of the problem of plan-recognition. This formalization is expressed in conventional predicate logic, augmented only by formal expression of some straightforward assumptions, so it is easily understandable and intuitively appealing. In turn, rather than leaving this formalism as an amusing exercise in logic, Kautz has translated it into an efficient procedure, and described the procedure with laudable clarity.[3]

That said, a number of problems arise in fitting this neat approach to plan-recognition to the problem of plan-recognition as encountered in story understanding. One of the problems is the requirement that one specify ahead of time the class of top-level plans to be identified. To understand why this is a problem, consider understanding the sentence "Jack got on a bus." Taken by itself, one would like to interpret it as being part of a plan to take a bus-trip. On the other hand, taken as part of the story "Jack got on a bus. He got off at the airport. He bought a ticket to Chicago." we would like to understand it as just a step of Jack's plan to fly to Chicago. So sometimes we would like to consider bus-trip as a top-level plan, and sometimes not. Furthermore, in the cases when we would like to consider it as a top-level plan, we do not want to consider all the other top-level plans in which it could play a role. That is, on reading "Jack got on a bus." we don't want to consider air-trips, visits to museums, eating out at a restaurant, etc. This is a problem for an approach like Kautz' which relies on our being able to construct explanations by explaining every event in isolation, and then combining these individual explanations into a global one. The example given above indicates that we will have to take the global context into account when explaining individual events.

Another problem is that Kautz' approach requires an acyclic plan hierarchy ("one that can be exhaustively searched in finite time" [63, p. 32]). This forbids us from having a plan hierarchy with entries like:

A bus-trip isa go.

[3]Unlike any of the other methods described here, or my own method, I would feel confident in my ability to re-implement Kautz' approach in short order.

One step of the bus-trip plan is to go to the bus depot.

In fact, we have just these facts in our database, and it is difficult to argue against them on knowledge-engineering grounds. Furthermore, it is hard to see how a program could understand both stories like "Jack got on the bus. He went to the supermarket." and "Jack took a taxi to the bus-station."[4] without knowing both that a bus-trip is a kind of going, and that one of the steps in it is to go to the bus-station.

Another problem is the problem of competition between explanations. Kautz and Allen have little to say about choosing between explanations involving the same number of top-level plans. This is one of the problems which we have attempted to address, both in the work described in the following section, and the work discussed in this thesis.

2.7 Wimp2

In the program Wimp2, we took an approach broadly similar to that of Kautz and Allen (though less well-grounded formally) [17, 43] Our approach differed in using a multiple-context TMS for representing the disjunctions resulting in plan-recognition. The multiple-context TMS allowed us to cache the results of matching various different frames with the input text. We associated probabilities with the different TMS contexts in order to control our search and help us choose between different explanations.

Wimp2's multiple-context database was based on deKleer's ATMS (Assumption-based Truth Maintenance System) [37, 38, 39]. The ATMS allows its user to represent disjunctions by splitting the database into a set of "possible worlds". This is done by adding distinguished nodes called assumptions to the data dependency records of a standard TMS. Sets of assumptions determine an environment, or *context.* Each ATMS formula has associated with it a set of contexts, its *label*, which specifies the minimal set of assumptions sufficient to derive the formula.

For example, let us assume we have the rule:

```
(-> (inst ?x bird) (or (can-fly ?x) (flightless ?x)))
```

and we learn

```
(inst tweety bird)
```

After forward-chaining, our database would be as illustrated in Figure 2.6.

Nogoods and *chooses* impose constraints on the labels of the ATMS. Nogoods are contexts which are inconsistent. They are detected by forward-chaining. To return to the example of Figure 2.6, if we also had the forward-chaining rule

```
(-> (and (can-fly ?x) (flightless ?x)) FALSE)
```

we would derive the nogood (1 2). Chooses are disjunctions of assumptions. In order for the representation of tweety above to be correct, we would also have to add the choose (1 2). If we were later to learn that tweety was a penguin and knew that

```
(-> (and (inst ?x penguin) (can-fly ?x)) FALSE)
```

we would derive the nogood (1), and from that and the choose (1 2), we would deduce that assumption 2 was true.

[4]Wimp3 can, in fact, understand both of these stories.

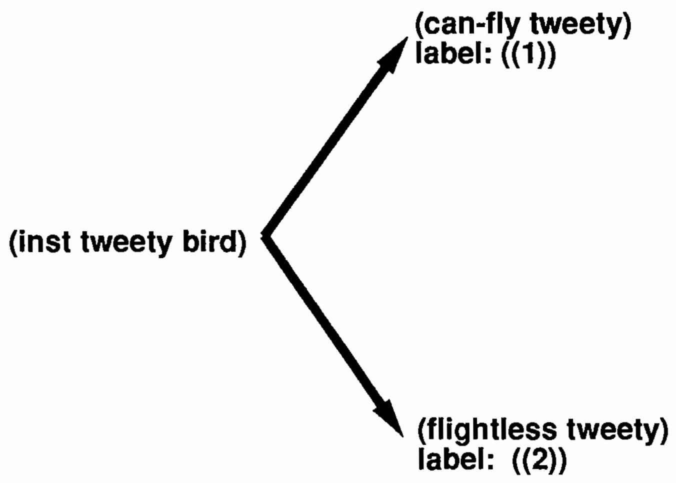


Figure 2.6: Representing disjunctions in the ATMS

An ATMS is suited to AI problems of two types. First of all, it is more efficient than a conventional TMS when one is interested in finding all possible answers to a query, because it can take advantage of structures shared between different answers. Second, it is useful in problems where there is a lot of 'unouting.' 'Unouting' occurs when one changes between contexts in a TMS. One needs to find a new set of assumptions to reason with, and then has to propagate them, as a set of constraints, through the database. We argue that story understanding fits this classification, because in story understanding one is interested in finding a number of possible interpretations, and choosing the best from among them.

We used probabilities to focus our search and to compare different explanations. These probabilities served to capture three intuitions:

- We prefer to explain things mentioned in a story as references to known objects, rather than as new entities, whenever possible.

- Given alternative explanations for an object, we prefer one which eventually leads to a reference to a previously-known object. For example, consider the story "Jack went to the supermarket. He found some milk on the shelf." Wimp2 knows of two slots that foods can fill: the `purchased` of a `supermarket-shopping`, or of a `restauranting`. Wimp2 will prefer the `supermarket-shopping` explanation, because when it postulates a `supermarket-shopping`, it will find that it might refer to the `supermarket-shopping` which it hypothesized earlier to explain going to the supermarket, and it will not find any previous `restauranting`.

- In the absence of other evidence, we prefer to explain a thing as filling the slot that best fits that thing. E.g., if Wimp2 heard about an airport and knew of two slots it could fill, the `start` of an `airline-travel`, or the `location` of a `meeting`, it would prefer the `airline-travel` as an explanation, because it requires an airport to fill its slot, and the `meeting` only requires a location.

In general, our use of probabilities served our limited purposes, but there were a number of problems with this approach, having to do with the marriage of logic and probability. It was redundant to have both logical and probabilistic constraints. The probabilistic updating mechanism was ad hoc. Finally, the logical framework was not expressive enough for our kind of diagnostic inference.

Taking these two problems in order, we found that Wimp2 was often doing the same inference more than once, since it was propagating probabilistic and logical constraints. That is, the database statements were first labelled with the set of consistent contexts, and then were weighted for probability. It would be more efficient to just weight the statements with probabilities, taking into account the relations of consistency at that time.

We also found it difficult to correctly compute the probabilities of the contexts. At roughly the same time as our work, Laskey and Lehner [70, 71] and d'Ambrosio [32, 33] developed probabilistic interpretations for the ATMS that were cleaner than our own. However, to apply these methods to diagnostic inference essentially required that one reason forward exhaustively from a set of possible diagnoses, and then compare the resulting inferences with the actual state of the world. This kind of first-principles diagnosis is not appropriate for language understanding: it amounts to reasoning top-down from all possible interpretations, and we have seen earlier in this chapter the need to mix top-down and bottom-up inference.

This problem is related to the representational limitations of the ATMS in particular, and logic in general. Logic is incapable of distinguishing evidential and causal support for propositions. When a new justification is added to a proposition, one cannot tell whether it represents evidence for that proposition, in which case it would confirm hypotheses which explain that proposition; or if it represents an alternative explanation, in which case it should compete with other explanations.[5]

The problems with our hybrid architecture led us to discard the ATMS in favor of a wholly probabilistic approach. We also wanted to improve on Wimp2 in some other ways. First of all, we wanted to get a clear sense of the semantics of the probabilities. One of the reasons we wanted this was to use the probabilities to represent more information than just the intuitions listed above. In particular, we wanted to take into account the different a priori probabilities of different interpretations. Consider the sentence "Jack went to the airport." We argue that, in the absence of other information, the correct interpretation is that Jack is probably going to fly somewhere, or perhaps he is going to bring someone home from the airport. On the other hand, the hijacking and airplane-bombing explanations are *not* good interpretations, although they could become good interpretations if we learn that Jack has a bomb in his suitcase, or that Jack is a member of a terrorist organization.

2.8 Summary

In the above sections, we have tried to give a sense of the state of the art in story understanding, and the way its development has guided our approach. As we've shown immediately above, our experience with Wimp2 has led us to attempt a wholly probabilistic treatment of this problem. Our approach will be to treat the problem of text understanding as the composition of a number of probabilistic inference problems, such as 'What is the most likely structure of this sentence?' 'What is the most likely interpretation of this word, given this context?' 'If these are the actions taken by this character in this story, what is the plan he or she is most likely to be pursuing?' We will be using marker-passing to help us construct the set of hypotheses, because it allows us to integrate top-down and bottom-up inference. Finally, as with Wimp and Wimp2, we will be taking an integrated approach. That is, we will be posing the above questions simultaneously, so that all processes, syntactic, semantic and pragmatic, can constrain each other.

[5]Judea Pearl has done a great service to AI by presenting this fact clearly [89].

2.9 Other Abductive Approaches to NLP

In the previous section I have reviewed the literature of story understanding and plan-recognition, and showed how it pointed us in the directions we have taken in this research. In this section, I will shift gears a bit and talk about other programs which take seriously the notion of text understanding as abduction. These programs tackle some very different problems than our own, but are worthy of comparison.

2.10 TACITUS

Perhaps the work most similar to our own is that being done at SRI as part of the TACITUS project [54, 56, 55]. In this approach, the problem of text understanding is viewed as the problem of finding the best abductive proof of the input. They use this approach to find semantic and 'local pragmatic' interpretations, but argue that it could be extended to include syntactic analysis and the kind of schema recognition that Wimp does. They use a system of costs to control their search for abductive proofs.

Like us, the TACITUS group believes that the process of interpretation can be viewed as giving a proof of the input. Their approach differs from ours in that they use a Prolog-like architecture to construct these proofs [108]. So the semantic representation of a sentence is given to the theorem-prover, which constructs a proof. *e.g.,*to interpret the sentence "The car is red." the theorem prover would be asked to prove

$$\exists x.car(x) \land red(x)$$

If it knew about a car, say $car54$, but did not have any information about its color, it might construct the proof by retrieving $car(car54)$ and assuming $red(car54)$.

Hobbs et. al. show that this approach can be extended to handle more difficult problems of interpretation. The problem of metonymy (violated selection restrictions) can be handled by additional assumptions. For example, in order to correctly interpret "I got a call from the office." it is necessary to assume that there is a person calling me, and that this person works at the office. TACITUS also handles such problems as compound nominals, and some problems of syntactic ambiguity.

Since this approach is allowed to relax semantic constraints, it is important to control its search. Otherwise it would be possible to miss a literal interpretation of a sentence in favor of a far-fetched figurative one. Or consider our earlier example – "The car is red." TACITUS might try to understand this by just finding some red object it knows about, and assuming that it is a car, rather than vice versa.

To control this inference process, Hobbs et. al. associate assumability costs to rules and literals. For example, to assume a literal corresponding to an indefinite reference – the car in "A car is on the corner." – would be cheap, but a definite reference – "My car is on the corner" – would be expensive because we would like TACITUS to find the referent in its memory. Rules specify relative assumability costs:

$$red(x)^{.1} \land car(x)^{.9} \supset red\text{-}car(x)$$

would dictate that most of the cost of assuming $red - car$ comes from assuming car, so if TACITUS already knows about a car, it is better to assume that car is red, than to assume a whole new red-car from scratch, or to assume that some red thing is also a car.

If one thinks about TACITUS as searching all possible proof trees for the lowest-cost one, and Wimp3 as searching for the most probable one, it can be seen that our approaches are quite similar. Indeed, Shimony and Charniak have proven that, if one understands TACITUS' costs as log-probabilities, it is also searching for a most probable explanation [21]. We argue that in this case, the probabilistic interpretation provides more useful intuitions into at least the relative values to be assigned. *e.g.,* In allocating the weights for the rule

$$red(x)^{.1} \wedge car(x)^{.9} \supset red\text{-}car(x)$$

we draw on the knowledge that red things are common and cars are not.

2.11 Parsimonious Covering and NLP

Another abductive approach to language understanding is that of Dasigi and Reggia [34, 35]. Their approach is based on the parsimonious covering theory of diagnosis developed by Peng and Reggia [90]. This work is not aimed at the kind of deep understanding we have discussed in Section 2, but at interpreting natural-language descriptions of sets of findings for medical diagnosis problems. As such their program is more a semantic parser than anything else.

Parsimonious covering is a qualitative approach to abductive inference. In this approach, a class of diagnostic problems is described by a set of Disorders, a set of possible symptoms, and a set causal connections of the form evokes(disease, symptom). A particular diagnostic problem is posed by presenting a set of symptoms. A parsimonious cover of a set of symptoms is a minimal set of disorders which covers (evokes) all of the symptoms. Different criteria for minimality may be used. Dasigi and Reggia [35] use *irredundancy*, or subset-minimality: a set of disorders is minimal if no proper subset is also a cover.[6]

Dasigi and Reggia present a technique for constructing natural language interfaces to diagnostic expert systems, based on parsimonious covering. These interfaces are designed to take assertions like "Visual acuity is blind on the left and is normal on the right." and translate them into attribute-value assertions which can be used by a medical expert system. The approach is general in that it is built into a program which generates such interfaces, given a lexicon and a semantic network for the domain of the expert system.

Such interfaces use a "dual-route parsimonious covering" algorithm to interpret a text. In this method, partial story templates are matched against the input text, suggesting possible structural analyses. *e.g.,*the sentence "Visual acuity is blind." may be covered by the categories **pattr asg-verb pval** – the sentence is an assertion of the form 'attribute is at value.' The set of possible categories for each word is found in the lexicon, as it would be in a semantic parser. Also associated with the words are senses from the semantic network for the domain. Possible sentence structures ("syntactic covers") are then searched for semantic interpretations ("semantic covers") by finding sub-graphs of the semantic network which cover them. *e.g.,*after finding the structure of the sentence above, this method would find its interpretation by covering it with the part of the semantic net which indicates that visual acuity is a domain attribute, one of whose possible values is blind. What this approach does, in effect, is use an abductive method to augment a semantic parser to take into account coherence over larger units of text.

2.12 Conclusion

In this chapter, we have tried to make clear the reasons behind the directions taken in our work by reviewing previous work in story understanding. We have outlined problems

[6]A number of different minimality criteria are described and contrasted in [90].

previous story understanding programs have had when dealing with uncertainty. Finally, we've described some related work that sheds some light on the idea of viewing language understanding as abductive inference.

Chapter 3

Graphical Models of Probability Distributions

In the previous chapter, we have tried to give a sense of previous work in story understanding. One of the themes of that chapter was the way our attempts to deal with the uncertainty in story understanding led us to the use of probability theory. Unfortunately, until recently the application of probability theory has confronted us with great practical difficulties: both computational difficulties and difficulties in formulating models with which to work. It has seemed that the only two options were to try to manipulate gigantic tables of joint probabilities, or to try to associate quantities to conventional rule systems.

Graphical models of probability distributions are ways to answer the problem of *structuring* the use of probability theory. They all involve specifying a distribution in terms of interactions among small sets of random variables. *e.g.,* a model of a medical diagnosis problem could be decomposed into sets of diseases, pathologies (intervening conditions) and symptoms. Such a model would be described in terms of disease $\rightarrow$ pathology and pathology $\rightarrow$ symptom interactions, rather than having to give a joint probability for every possible combination of diseases, pathologies and symptoms. The ability to structure problems simplifies not just the use of probabilities, but their collection as well.

In this chapter, we will review three types of graphical representation for probability distributions: Markov networks, belief nets and influence diagrams. Markov nets are undirected graph representations and Belief networks are directed acyclic graphs. The directions on the edges of these graphs allow them to represent different independence relations. Influence diagrams are generalizations of belief networks for the purposes of decision analysis. They allow addition of special nodes to represent decisions and utilities, as well as random variables.

As AI practitioners, we are interested in using the topological structure of networks to express our knowledge of the *structure* of a problem. This knowledge about the structure of a problem will guide us in acquiring the quantitative information we need to formulate a question in probabilistic terms, whether we assess the quantities subjectively, or collect statistics to estimate them. Our interests differ from those of others who are interested in network representations. For example, a statistician interested in graphical representations, might be interested in finding a good graphical model for some known distributions (see *e.g.,* [73]). This is not what interests us, however, and we will not spend time discussing it.

3.1 Markov networks

Markov networks are undirected graph models of probability distributions. In a Markov network, the nodes represent random variables, and edges represent direct influences. The mathematical foundations of Markov networks, Markov field theory, is a generalization of the theory of Markov chains. The discussion in this section is based on [66, 73] and on chapter 3 of Pearl's book [89].

Formally, a Markov net is a graph, described as usual as a set V of vertices, and a set E of edges (unordered pairs of vertices). Associated with each vertex in V is a random variable, v, whose state space is ω_v[1] There is a strictly positive probability[2] distribution over the state space

$$\Omega = \omega_{v_0} \times \omega_{v_1} \times \cdots$$

This probability distribution satisfies the condition that for all v in V, $v \perp V - \{v\} \mid bd(v)$. $x \perp y \mid z$ should be read as "the random variable x is conditionally independent of y given z." By conditionally independent, we mean the conventional definition:

$$x \perp y \mid z \equiv P(x, y \mid z) = P(x \mid z)P(y \mid z)$$

Note that this relationship must hold for all elements of the sample spaces of x,y and z. $bd(v)$ is the set of all vertices adjacent to v (the "boundary" of v): $\{n \in V \mid \{n, v\} \in E\}$. In English, the condition is that the random variable corresponding to a node is independent of the values of all other random variables in the graph, once we know the values of its neighbors – all influences on the node are summarized in the values of the neighboring nodes.

Markov networks are being used in several kinds of Bayesian inference problems. They have been applied to problems of pattern-recognition [42, 44] and to optimization problems like the Travelling Salesman Problem [67]. Markov networks also form the foundation of Boltzmann machine models of neural networks [53]. Most of these applications rely on using simulation approaches to solving the networks: they select points in the sample space randomly, according to the distribution modeled.

We do not find Markov networks suitable to our application, for two reasons. First of all, it is difficult to quantify the networks. By this we mean it is difficult to take a problem for which we have worked out a qualitative model (a set of variables and the dependencies between them), and add the quantitative information which allows us to make decisions based on the model. In order to quantify a Markov network, we must specify for each node, a table of probabilities for its values, conditioned on the values of its neighbors. *e.g.,* for a Markov network in which a is adjacent to two nodes, b and c (see Figure 3.1), each of which

[1]Where there is no danger of confusion, we will refer to vertices and the associated random variables interchangeably. We will also occasionally use the name of a random variable to refer to an element of its state space. *e.g.,*$P(x)$ for $P\{X = x_i\}$ (where $x_i \in \omega_x$).

[2]The requirement that the distribution be strictly positive may be relaxed, but if it is, we can no longer guarantee that the Markov network will be a good model of the probability distribution. In particular, it may be necessary to add edges to the graph which represent spurious dependencies.

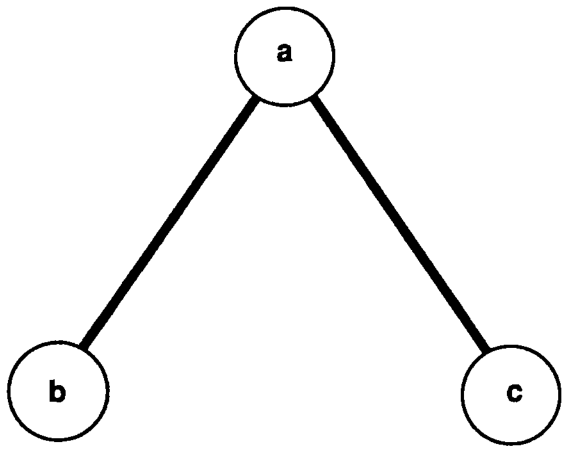


Figure 3.1: A simple Markov network. a's neighbors are b and c.

can take on the values $\{0,1\}$, we need to supply

$$\begin{array}{ll} P(a=1 \mid b=1,c=1) & P(a=0 \mid b=1,c=1) \\ P(a=1 \mid b=0,c=1) & P(a=0 \mid b=0,c=1) \\ P(a=1 \mid b=1,c=0) & P(a=0 \mid b=1,c=0) \\ P(a=1 \mid b=0,c=0) & P(a=0 \mid b=0,c=0) \end{array}$$

and analogous distributions for the nodes b and c.

One difficulty arises in assigning such probabilities *consistently*. We must make certain that, when assigning probabilities for nodes conditioned on their neighbor sets we do not overconstrain the distribution. Conversely, we must avoid underconstraining the distribution – failing to specify a unique probability distribution. A final problem is ensuring that these assignments honor the Markov properties we have encoded in our graph.

There is a procedure for consistently generating a probability distribution for a given Markov network, but unfortunately it is difficult to relate the process to intuitive judgments. The procedure is assign a compatibility function to each clique of the Markov network. A compatibility function is a function from the state space of the clique to the nonnegative real numbers. As its name suggests, it is meant to capture a sense of how compatible a set of values is.

To return to the example in Figure 3.1, we would have to provide compatibility functions $f_1(\omega_a, \omega_b)$ and $f_2(\omega_a, \omega_c)$. If the random variables represented *e.g.,*spins in a magnetic material, and the material was attractive, then we might specify a compatibility function, $f = f_1 = f_2$ such that $f(1,1)$ and $f(0,0)$ are greater than $f(0,1)$ and $f(1,0)$. This will make states in which the values of a, b and c are equal more likely than other states.

This procedure is interesting for many purposes, but not useful to us in constructing models. First of all, it is a very difficult problem to go from statistics to compatibility functions. A closely related problem, and one more acute for most AI practitioners is that the compatibility functions don't have any semantics taken apart from the network as a whole. When drawing up a model with the aid of experts, low order conditional probabilities, which belief nets allow us to use, are more appealing. It is easier for us to assess subjectively, or collect data for finding probabilities like the probability of a symptom given a disease, than to devise compatibility functions.[3]

A closely related problem is that Markov networks don't allow us to represent the kinds of dependencies and independencies which we would like to use in reasoning. Markov networks do not represent well distributions which have probabilities of zero in them. These extremal distributions require that superfluous edges be added to Markov networks which model them. This means that they require more parameters, and because some random variables are not correctly recognized as conditionally independent, the reasoning process is more expensive. As we will see in the next section, belief networks offer us attractive solutions to these representational problems.

However, the reader should be aware that the divide between Markov networks and belief networks is not as absolute as we have made it appear here. In fact, as will be seen later, our program makes use of a procedure for translating a belief network into an equivalent Markov network [72], because belief networks are easier for us to work with, as designers, and Markov networks are easier for us to compute with.

3.2 Belief nets

Belief networks are a directed acyclic graph representation, similar to Markov networks.[4] By adding direction to the edges of our graphical representations, we gain a very different kind of expressive power. Our discussion of belief networks is drawn from Judea Pearl's book [89] and the monograph by Lauritzen et. al [74].

Belief networks are directed acyclic graphs: pairs $< V, E >$, where V is a vertex set, as before, but the edges in E are *ordered* pairs of vertices, (v_i, v_j). As in Markov networks, the nodes of a belief network correspond to a set of random variables, $v_0, v_1, \ldots$ Each such random variable has a state space ω_v, and there is a probability distribution over the configuration space

$$\Omega = \omega_{v_0} \times \omega_{v_1} \times \cdots$$

Belief networks have a more complicated Markov property than their undirected graph equivalents. In Markov nets there is a straightforward relationship between graph separation and independence: if we want to determine whether a random variable A is independent of B given some set of conditions C, we need only check to see if every path from A to B contains (is blocked by) a node in C. In directed graphs, however, by adding conditions we can not only block dependencies, but actually *create* them.

One simple Markov property is that each random variable is independent of its non-descendents, given its parents. In the notation we have used above,

$$v \perp \text{non-descendents}(v) \backslash parents(v) \mid parents(v)$$

($\backslash$ is used to represent the set difference function). For example, in the diagram given as Figure 3.2, Burglar Alarm is independent of Broken Crockery given Earthquake and Burglary.

The separation criterion becomes more complicated when descendent nodes are included in the conditioning set. For example, in Figure 3.2, Earthquake and Burglary are initially independent, but are no longer dependent when conditioning on the value of Burglar Alarm. This is reasonable, because (by the assumptions drawn upon in making up the diagram)

[3]However, the reverse translation *is* possible, and is made use of by the algorithm of Lauritzen and Spiegelhalter [72], as we will see later on.

[4]Belief nets have been known by many other names. Judea Pearl [89] suggests "Bayesian networks." They are also called "causal probabilistic networks" [72] and "directed Markov fields" [74].

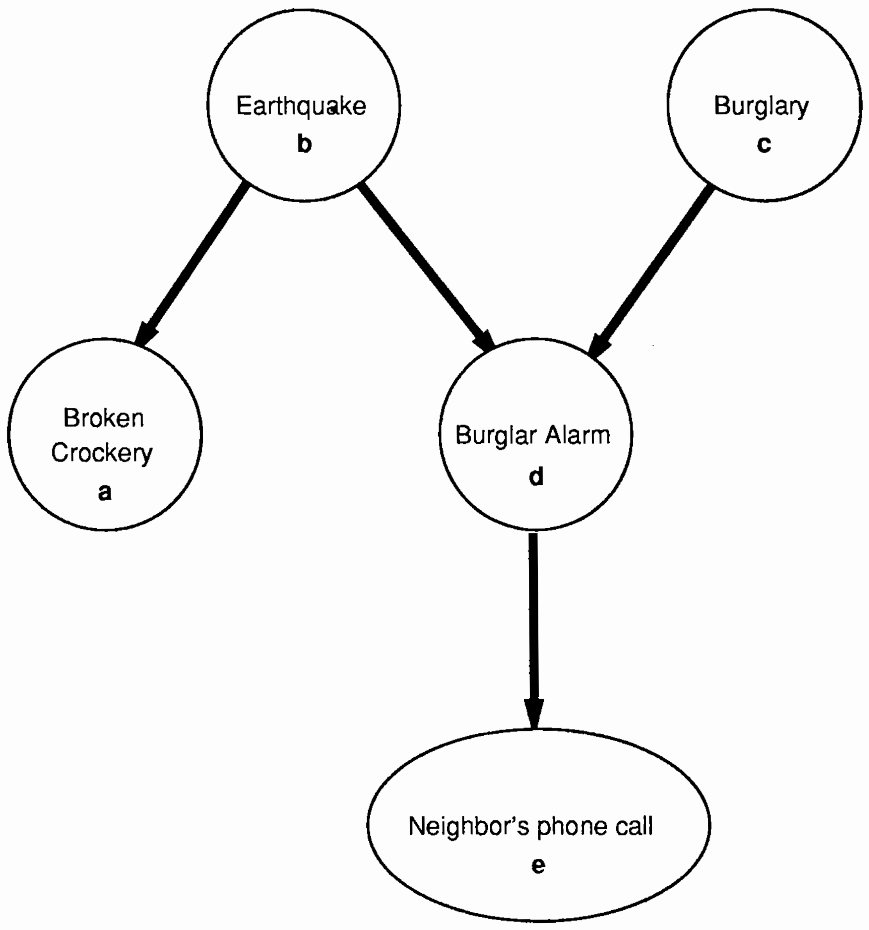


Figure 3.2: A simple belief net. Nodes are given letters as well as names so that we may conveniently refer to them. This example is adapted from one in [65].

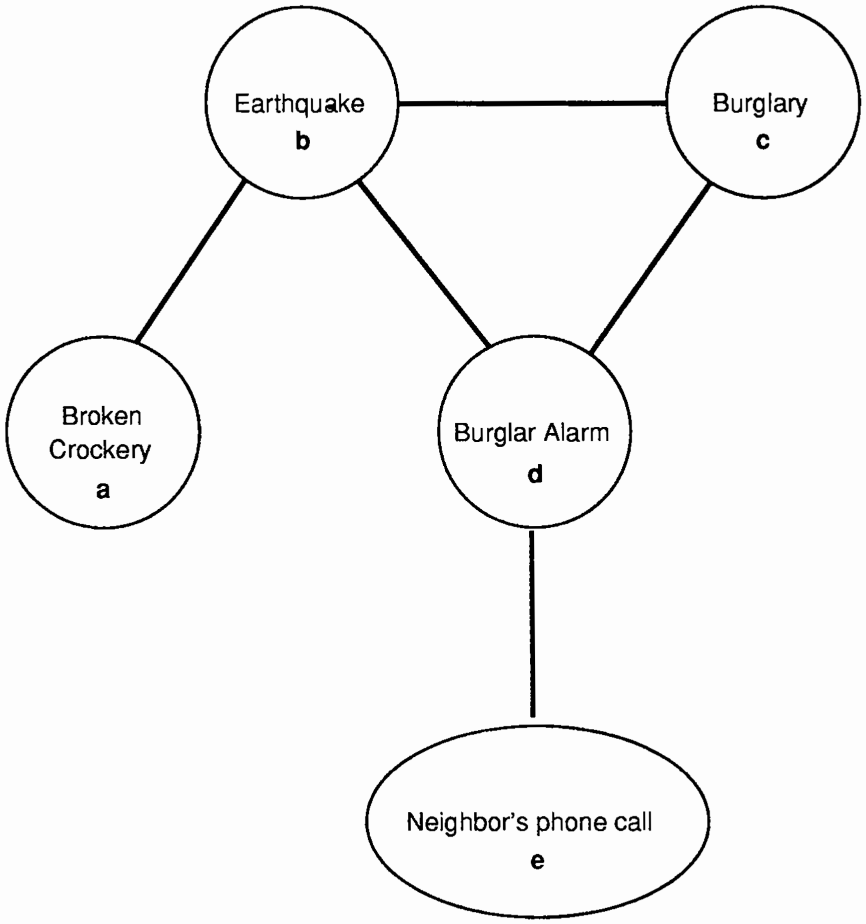


Figure 3.3: The moral graph corresponding to Figure 3.2. Note the edge that has been added between nodes b and c.

whether one is going to be burglarized is independent of whether there will be an earthquake. However, one knows that one's burglar alarm has sounded, and knows that it only sounds during a burglary or an earthquake, then one can conclude from the absence of an earthquake, that the alarm is caused by a burglary. Therefore, conditional on burglar alarm, earthquake and burglary are not independent.

Lauritzen et. al. have developed an easy-to-use graphical criterion for conditional independence [74]. To test whether A and B are independent given some conditioning set C, find the minimal *ancestral set* containing $\{A, B\} \cup C$. Construct the corresponding *moral graph*. Then check to see if A and B are separated by C in the moral graph. The ancestral set of a set of nodes is the made up of the set itself and all of its parents, and all of their parents, and so on: the reflexive, transitive closure of the parent relation. For example, the ancestral set of $\{e\}$ in Figure 3.2 is $\{b, c, d, e\}$. The moral graph is formed by 'marrying' the parents of all nodes in a belief network, and dropping the direction on the arcs. The moral graph corresponding to Figure 3.2 is given as Figure 3.3.

To see how to apply this criterion, consider the problem of determining if a (Broken Crockery) is independent of c (Burglary), given d (Burglary), in Figure 3.2. First, we take the ancestral graph of $\{a, c, d\}$, given as Figure 3.4. Then we transform it into its moral graph, Figure 3.5. From this it can be seen that a and c are *not* independent, because there is a path between them, $a \leftrightarrow b \leftrightarrow c$, which is not blocked by d.

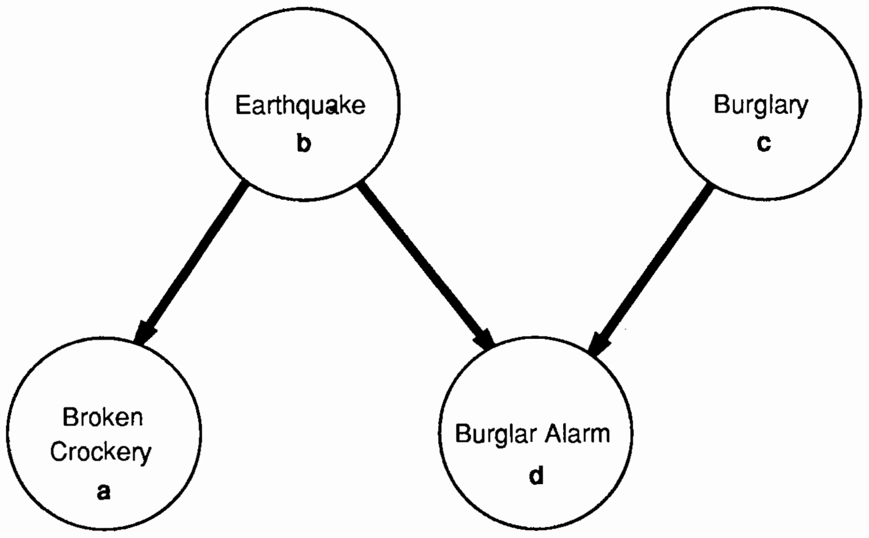


Figure 3.4: The ancestral graph of $\{a, c, d\}$.

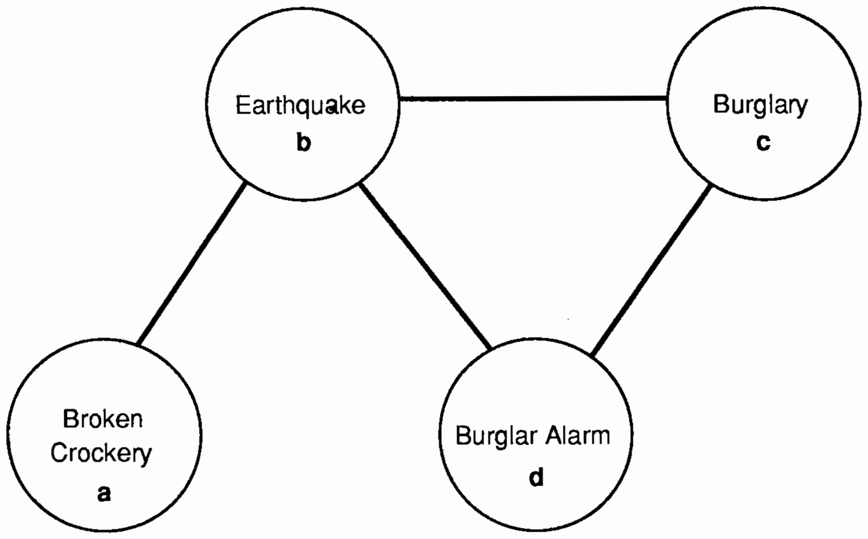


Figure 3.5: The moral graph corresponding to the ancestral graph of $\{a, c, d\}$.

Pearl has an equivalent separation criterion which he calls *d-separation* [89]. It is more convenient in that it applies to belief networks themselves. However, it is more difficult to grasp and to explain, because it does not build on our notions of graph separation in a simple way.

As mentioned in the previous section, belief networks are easier to quantify than Markov networks. In order to specify the probability distribution corresponding to a given belief network, it is sufficient to give, for each random variable, a conditional probability for every possible combination of values of its parent or parents. For example to quantify the diagram in Figure 3.2, we would need the following quantities (assuming all of the random variables are boolean, *i.e.,* take on values x or $\overline{x}$):

$$\begin{array}{ll}
P(b) & \\
P(c) & \\
P(a \mid b) & P(a \mid \overline{b}) \\
P(d \mid b, c) & P(d \mid b, \overline{c}) \\
P(d \mid \overline{b}, c) & P(d \mid \overline{b}, \overline{c}) \\
P(e \mid d) & P(e \mid \overline{d})
\end{array}$$

Such low-order conditional probabilities, in addition to being more efficient to store, are easier to assess, either subjectively or statistically, than the compatibility functions of Markov networks. Furthermore, since we are always specifying distributions of children conditioned on parents, it is simple to specify a consistent probability distribution. We are always adding constraints *down* the network; there are no constraints to be added from the bottom-up that might cause conflicts. In this example, we never need specify a $P(c \mid d)$ that might conflict with the $P(c)$ and $P(d \mid c)$.

One problem which *does* arise in quantifying belief networks is that of filling out conditional probability distributions which must combine influences. Consider again Figure 3.2. In order to specify a distribution corresponding to this network, we must specify a probability that the burglar alarm will sound given a burglary during an earthquake ($P(d \mid b, c)$). It is quite unlikely that this quantity will be available to us. Whether we quantify our diagrams based on expert judgments or by collecting statistics, we will probably only have quantities like P(burglar alarm|earthquake) and P(burglar alarm|burglary) at our disposal.

Pearl suggests that we assume stereotyped patterns of causal combination to allow us to complete the conditional probability tables [89, chapter 4.3.2]. He further suggests that we can think of there as being 'gates' which combine causal influences. For simple cases like the Burglar alarm, he suggests that causes combine as a 'noisy-OR' gate; that is to say that *either* of the two causes will result in the effect, and that if both causes occur simultaneously, neither will block the other's effect. Other gates are needed for more complicated interactions – *e.g.,* AND-gates for situations where one state of affairs enables another to exert an effect, etc.

The belief network representation, in addition to making it easier to create and store probability distributions, helps us to reason about them. A particularly important kind of reasoning is finding posterior distributions: given some evidence about the state of some random variables, find the probability distribution of the remaining variables. For example, I might want to know what is the probability that a burglary has occurred, given that my neighbor has called me at work and told me that my burglar alarm has gone off.

Cooper has shown that the general problem of finding a posterior distribution is NP-hard [25, 27]. Pearl and Kim [65] have found a 'singly-connected' special case for which there is a polynomial algorithm. Unfortunately, the singly-connected condition – which is that the the belief network treated as an *undirected* graph must form a tree – is so restrictive

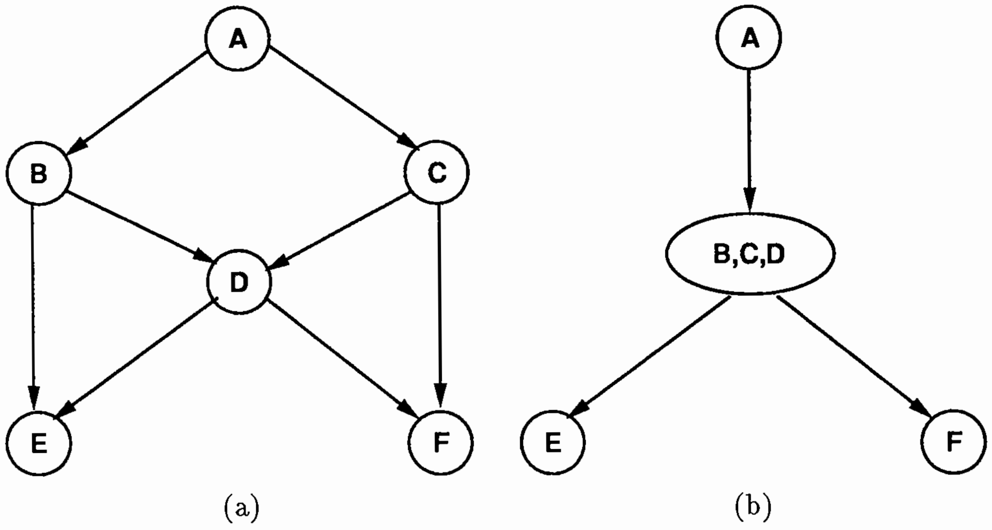


Figure 3.6: Example of Clustering, from [10].

that this result is of little immediate use. However, many algorithms for the general case build on this algorithm, as we will see below.

There are several different approaches to the general problem of belief network evaluation. We will just mention three of the most prominent here: clustering, conditioning and simulation. In clustering approaches [72, 59, 60, 10, 89] one creates a version of the input network which is singly-connected by combining ('clustering') nodes into macro nodes. These random variables corresponding to these macro nodes have sample spaces which are the cartesian product of the sample spaces of their components (*e.g.*,we might combine two boolean variables a and b into a macro node whose states are $ab, a\bar{b}, \bar{a}b, \overline{ab}$). Then some singly-connected inference algorithm is applied to the network of macro-nodes. For example, a clustering algorithm might transform the graph in Figure 3.6(a) into the singly-connected graph of macro-nodes shown in Figure 3.6(b).

Conditioning approaches [57, 89] are based on the process of reasoning by cases. What one does is to find a cutset which separates the multiply-connected belief network into a number of singly-connected subnetworks. *e.g.*, $\{B, C\}$ is a cutset for the graph in Figure 3.6(a). Then one finds a distribution for each combination of values of the nodes in the cutset. For each possible instantiation of B and C find the probability of A, D, E and F. Finally, one combines the values from each distribution, based on the joint probability of the cutset instantiation.

Simulation approaches [22, 89, 52, 100] are based on simulating the distribution. What one does in a simulation approach is to use a random number generator to choose a point in the sample space, in accordance with the distribution. This can be done by 'visiting' each node in the network, according to some schedule, and choosing a value for that node randomly, according to its probability conditioned on the values of the nodes in its 'markov blanket' (the set of nodes which make it independent of the remainder of the diagram). One samples from the distribution many times, and collects statistics, which are used to approximate the posterior distribution. Unfortunately, as Cooper and Chin have shown [23], the rate at which a simulation algorithm will converge on the true distribution is inversely proportional to the smallest probability in the network. In practice, networks with many extreme probabilities will get stuck in particular sink states and take impractically long to converge. In the limiting case (conditional probabilities of 1 or 0), simulation approaches cannot be used, because they may *never* converge on the correct distributions.

There are other kinds of computation that one might wish to do with belief networks, which we do not have the space to discuss here. For example, rather than trying to find a posterior distribution over the network, one might want to find which state of the whole network is most likely, conditioned on the evidence (MAP estimation) [89, 104]. Cooper suggests finding a best conditional joint probability of distinguished nodes in a network, for diagnosis applications. [28]

In his influential book [89], and in many articles, Pearl suggests that when designing a belief network model, one direct the edges of the graph causally: nodes at the tails of edges should be direct causal influences on the nodes at their heads. Doing so simplifies the collection of probabilistic judgments from experts. It also provides a simple way of consistently ordering a graph. As much as possible, we have tried to adopt this approach in our stochastic model of story understanding. But it is not necessary for the use of belief networks, and indeed some network-processing algorithms (see, *e.g.,* [99]) capitalize on the arbitrariness of arc directions.

3.3 Influence diagrams

Influence diagrams are generalizations of belief networks. They were developed by Miller, Merkover and Howard [82, cited by Schachter] and by Howard and Matheson [58], as ways of efficiently representing decision-theoretic problems – particularly as more-efficient alternatives to decision trees. Later, Schachter developed algorithms for directly processing influence diagrams, bypassing the need to translate them into decision trees [97]. Our discussion in this section follows [97, 98].

Influence diagrams are like belief networks, but may have nodes of two additional types: decision nodes and value nodes. *Decision nodes* (drawn as rectangles) represent decisions that an agent may take. Arcs from random variables, referred to as *chance nodes* and drawn as circles, represent information available to the decision-maker at the time of the decision. Finally, it is possible to have a distinct *value node*, drawn as a rounded rectangle. The value node represents the utility of the final situation, after all decisions have been made, and their effects felt. Arcs into the value node represent the variables on which the utility function depends.

A sample influence diagram is given as Figure 3.7. This example, taken from [97], represents a decision problem in oil exploration. This diagram represents the fact that in an oil exploration problem, one is faced first with the decision of whether or not to test. Then, based on the test results (if any), one decides whether to drill. Finally, one's utility depends is a function of profit, which in turn depends on the cost of drilling, whether or not one drills, the amount of oil, and whether or not one has conducted tests (presumably there is some cost for testing).

Algorithms for solving influence diagrams [1, 97, 98] aim at finding the optimal decision policy, the one which maximizes expected utility, and as a side effect, also report the resulting expected utility. This decision policy is made up of the best decision for each decision node, for every combination of values of its parents. E.g., in the diagram of Figure 3.7, the decision policy would specify whether or not to test, and then, for each possible test result (including 'test not made'), whether or not to drill. Influence diagram algorithms are more directed in that, rather than finding a distribution for each node, they only consider chance nodes which influence the value of the decision process.

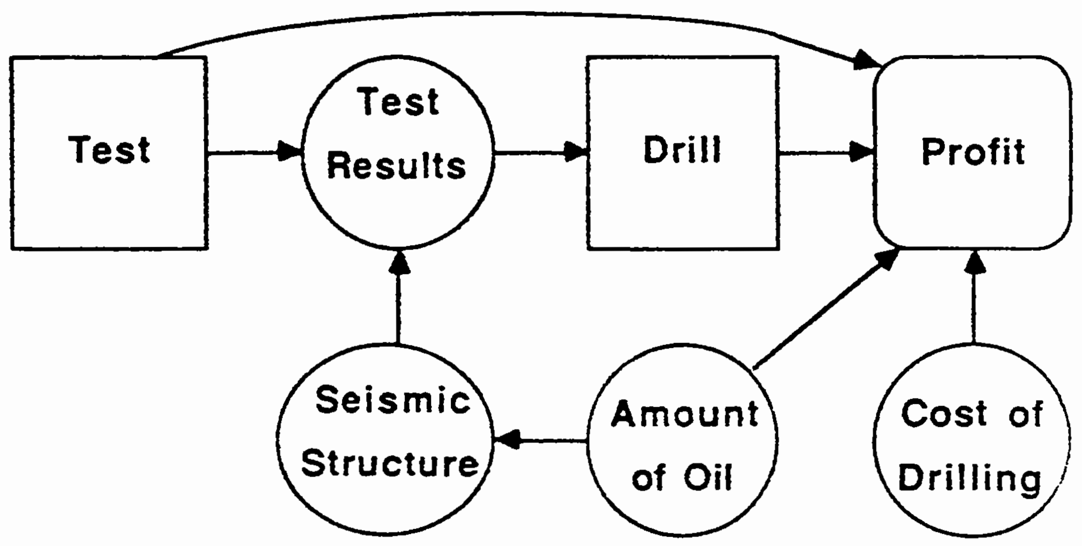


Figure 3.7: Sample influence diagram.

3.4 Conclusion

In this chapter, we have reviewed important features of three kinds of graphical models of probability distributions: Markov networks, belief networks and influence diagrams. Markov networks are graph models, belief networks are DAG models, and influence diagrams are like belief networks, but with extra representational capabilities for decision-theoretic applications. For the problems of language understanding discussed in this thesis, we will be using belief networks. We have chosen belief networks over Markov networks because it is easier to specify the probability distribution for a belief network, and because of the kinds of conditional independences that a belief network can express. Our choice of belief networks over influence diagrams reflects the kind of domain we are working in. For us, the problem of story understanding is purely a problem of interpretation. In the research discussed here, we are not concerned with the interpretation of texts in service of some other task (*e.g.,*responding to queries). If we were doing so, then we could make use of influence diagrams, and the concept of utility.

Belief networks, then, are the tool we will use to structure probabilistic inference, in applying probability theory to the problem of story understanding. The next chapters of this thesis will discuss the way we will pose the problem as a problem of probabilistic inference.

Chapter 4

A Stochastic Model for Story Understanding

In order for us to apply probability theory to story understanding, we need to develop a probabilistic model of the story understanding process. If we were to proceed along more conventional, logic-based lines, we would need to isolate the interesting entities in the domain, and describe the relations between them in logical terms. This would allow our program to use deduction to reason in this domain. Similarly, when taking a probabilistic approach, we need to find a set of entities in terms of which to describe the problem. Instead of describing the relations between them in terms of implication, we will use the relation of conditional probability.[1]

In this chapter we discuss a probabilistic model of story understanding, which will be used by the program Wimp3 in understanding simple texts. The previous two chapters have provided the necessary groundwork to discuss the work done in the course of this thesis. We have reviewed the shortcomings of earlier story understanding programs, and in particular the factors which led us to abandon conventional rule- and deduction-based approaches in favor of a probabilistic approach. We have introduced belief networks, which we will use to structure our probabilistic model.

For the purposes of this thesis, we are simplifying the story understanding problem by considering only written, expository text describing events and objects in the real world. Modal verbs, such as "want" or "will" are not allowed.[2] This allows us to view the language user as a transducer. The language user observes some thing (event or object), and translates this thing into language. Our task is to reason from the language to the intentions of the language user and thence to the thing described.

We need to describe our model's random variables in a formal language. This is necessary in order that our program, Wimp3, be able to reason about them in a rule-based way. So among other things, we will be concerned in this chapter with giving a formal semantics for this language. In the discussion, we will see that this formal semantics helps guide us in the subjective assessment of probabilities for this model.

Discussion in this chapter will center on the properties of the model itself. We will try to skirt issues of model construction and manipulation as much as possible. However, this will not be entirely possible – these issues will overlap when we talk about using network representations to structure the model. Greater structure allows us to assess the probabilities more easily, and makes reasoning with the model more efficient (since we will

[1]See Nilsson [84] or Pearl [89] for a discussion of the differences between implication and conditioning.

[2]Because of the limitations of our semantics, we also exclude statements about groups of objects.

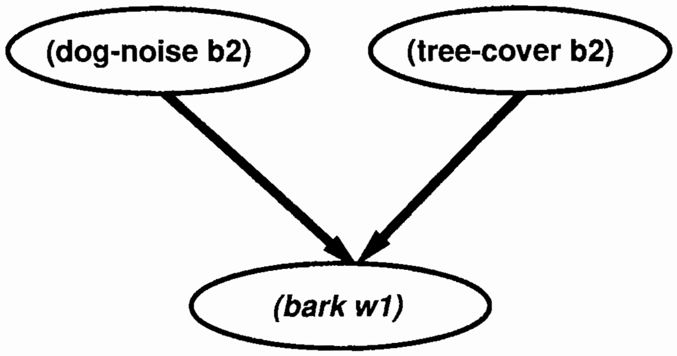


Figure 4.1: Ambiguity of "bark"

be able to take advantage of more independences and conditional independences). However, the model itself does not depend on the network representation.

In order to motivate our discussion, let us consider a particular problem, word-sense disambiguation, in a probabilistic light. Figure 4.1 shows a belief network designed to capture (part of) the situation when an author uses an ambiguous word like "bark." We have adopted here a convention, to be used throughout the paper, of using bold face for entities in the world, and predicates on them, and italics for words and predicates on them. So (**dog-noise b2**) says that the entity **b2** is the noise that a dog makes, while *(bark w1)* says that *w1* is a token of the English word "bark."

In this diagram, *w1* is an instance of the word "bark," and **b2** is a token representing the denotation of *w1*. Looking at the top-leftmost node, its connection to the bottom node is designed to capture an influence on the decision to use *w1*, a token of the word "bark." In this case the influence is that an author is likely to use the word "bark" if the object she wishes to refer to is of the type **dog-noise**. If the entity she wanted to talk about were a radish, she obviously would not have used the word "bark."

Ideally we would have here a probabilistic description of word-choice in English, but the nice thing about probabilistic models is that even very incomplete models can do some good, and here we have reduced the problem of word choice to that of matching the kind of object to the words. At any rate, given this formalism we calculate the probability that the word means, say, **dog-noise** using Bayes' Theorem.

$$\begin{aligned} &\mathrm{P}((\textbf{dog-noise b2}) \mid (\textit{bark w1})) \\ &\qquad = \mathrm{P}((\textbf{dog-noise b2}))\ \frac{P((\textit{bark w1})|(\textbf{dog-noise b2}))}{P((\textit{bark w1}))} \end{aligned}$$

A number like the probability of using the word "bark" given that the entity is the corresponding noise is easy to estimate. It is certainly high, say .9. But the other probabilities required here are harder. What, for example, is the prior of (**dog-noise b2**)? Typically we think of terms in our language as adhering to entities in the world, like a sound emanating from my backyard last night. If so, then since I thought that it was most likely a dog barking, I might say that the probability is .7. On the other hand, **b2** is an arbitrary symbol, created by my language-comprehension system to denote whatever the writer was referring to. What is the probability that an "arbitrary symbol" denotes a bark? This must be astronomically small, assuming we can understand the notion at all. Or again, given that **b2** is arbitrary, perhaps we should interpret the formula (**dog-noise b2**) as the skolemized version of the formula **exists(x)(dog-noise x)**. Interpreted in this light the probability is 1, since, of course, barking sounds do exist.

As we have seen here, the problem is not really assigning the probability, per se, but rather deciding what the formula **(dog-noise b2)** *means.* Nor is **(dog-noise b2)** the only kind of formula we will have problems with. In a sentence like "Janet killed the boy with some poison." there is case ambiguity in that the word "with" can indicate that the "poison" is in the instrumental case **instr**, or the accompaniment case **acc**. That is, did Janet use the poison, or just take it along for the ride (as in "Janet went to the movies with Bill.")? Here we need the prior probability of a formula like **(instr k1) = p1**. All the same problems arise, and more.

4.1 The Entities of Interest

In our model, we consider the space of simple, declarative stories from a restricted vocabulary. The primary entity in the decomposition of this model is the word token. This is a particular *use* of a word – complete with its denotation, relations to other words in the text, etc.

We define a function, **denotation**, that applies to word tokens. For words which have a denotation, this function evaluates to that denotation. For function words, this function evaluates to a distinguished, uninteresting element, $\perp$.

In the deeper phases of analyzing a text, we are interested in explaining the actions and objects in stories as playing parts in the plans of the stories' characters. To this end, for each action, for each plan, say plan_i, we define a function $\textbf{plan}_i\textbf{-explanation}$. When applied to an object or action, this function represents the plan of type plan_i, in which the object or action fills a slot. If there is no such plan, it evaluates to $\perp$.

Let us consider a simple example, the sentence "Jack went to the supermarket." Call this particular use of the word 'went', *went256*. There will be some going event which this word denotes: denotation(*went256*). Let us call this going event go29. So

$$\text{denotation}(\textit{went256}) = \text{go29}$$

Now, assuming that this story describes Jack's going shopping, there will be a shopping plan, shopping-explanation(go29). Furthermore, unless Jack has more nefarious purposes in mind, in addition to stocking his larder,

$$\text{robbery-explanation(go29)} = \perp$$

Note that not all entities mentioned, only the actions of characters, have $\textbf{plan}_i\textbf{-explanations}$. This avoids having objects call into consideration every plan in which they could play a role. For example, we don't consider a shopping explanation when we read "Jack bombed the supermarket." Also, typically the shopping-explanation of a supermarket is less interesting than that of a particular going event, because a given supermarket will fill the store-of slot of a myriad shopping events.

We have corresponding functions, only for objects, which handle part-whole explanations. For example, if we had frames for elephants and their trunks, we would have **elephant-trunk-explanations** for elephant-trunks.

These, then, are the entities which we will use to make up our model of natural language stories. The space of such stories is the space of ordered tuples of word uses of bounded length (the precise bound need not concern us here). Stories of length less than the bound are considered to be padded by $\perp$'s at the end.

Now that we have defined the entities in our model, we will continue by discussing the language which Wimp3 will use to talk about these entities. We will use this language to formulate propositions which we will treat as random variables.

4.2 The language

The syntax of our language is a restriction of the language of first-order predicate calculus. We have the customary constants, functions, predicates and connectives. However, we do not allow quantifiers or variables.

Having said this, it is important to emphasize that we are *not* providing a logical calculus. Our calculus is probability theory. We are merely providing a language of propositions we will reason about using probability theory.

4.3 Semantics

1. We define two disjoint sets of primitive events, and probability distributions over them:
   (a) The set of all words, W.
   (b) The set of all individual things, T. 'Thing' is defined as per the isa hierarchy: events, objects, persons, concepts, etc.
2. We define the overall sample space, $\Omega = \{W \cup T\}$.
3. There is a distinguished element, $\perp$, which is not contained in Ω.
4. Constants represent the outcomes of trials. For example, in Figure 4.1, *w1* and **b2** are random variables with values from W and T, respectively.
5. Functions of arity n are functions $\Omega^n \rightarrow \Omega \cup \perp$. For example, **rope-of** is a function which maps hanging events to the ropes used in them (if there is one) and maps other entities to $\perp$.
6. A predicate P^n is a function $\Omega^n \rightarrow \{0, 1\}$.
7. The Boolean operators, or, and, and not allow us to compose predicates, in the customary way. Formally boolean operators are functions from predicates, or pairs of predicates, to predicates.

The constants which are of particular interest to us are the word tokens. Because we do not have access to these objects in all of their splendor, we will be working with functions of them. Particularly important functions are word-type and denotation. For a word-use, the word-type function picks out the word of which this word-use is an instance.

Let us consider again the sentence: "Jack went to the supermarket." If this use of the word 'went' is *went256*, then word-type(*went256*) = the verb 'go'. In the future, we will make use of the two-place predicate **word-inst**, and write the foregoing as (word-inst *went256* go). Our language also contains the predicate **inst**, which applies to an object and a type-name, and is true if the object is an instance of the type. So the following predicate holds: (inst (denotation *went256*) go-).[3]

Now, as we mentioned above, we do not have access to the word tokens in all their glory. In fact, the evidence we have is typically restricted to knowing the word-type of the word tokens in a story, and knowing some relations between the words. So, for example, when Wimp3 reads the sentence "Jack went to the supermarket." all it 'knows' are the facts:

[3]We will follow the convention of adding a hyphen to the end of a type to indicate the thing, rather than the word. So going events are instances of go-, "went" is an instance of go.

```
(word-inst w1 jack)
(word-inst w2 go)
(word-inst w3 to)
(word-inst w4 the)
(word-inst w5 supermarket)
(subject w2 w1)
(syn-rel to pp w5 w2)
```

From Wimp3's point of view, w1,w2,w3,w4 and w5 are random variables over the space of word-tokens, and when it is reasoning about the sentence it has read, the above are the conditioning events.

A brief notational aside: The notation we've laid out above, while clear, is too cumbersome for regular use. Accordingly, we have adopted some conventions to make our language more readable. In particular, when naming the word-tokens, we use names which are the name of the word, followed by an index to indicate that we are referring to a particular instance. For example, when talking about the sentence above, we might refer to the word-tokens as *jack23, went26, to45, the98,* and *supermarket476*. Because it is so cumbersome to write denotation(*jack23*), we will indicate the denotation of a word-instance by a constant which is simply the word's name in bold face, rather than italics, *e.g.,* **jack23**. We will do likewise for the explanation functions, taking the names this time from the name of the plan in question. So, for example, we might call shopping-explanation(go29), **shopping321**.

4.4 Belief network representations

As we've discussed in Chapter 3, belief networks help us structure probabilistic inference. In this section, we will describe a network topology for our model of story understanding. The topology we suggest will benefit us by restricting the probabilities we need to obtain in order to specify the distributions we use for reasoning (as will be seen in following sections).

As mentioned above, we are interested in analyzing natural language stories in terms of the words used, their denotations, and the plans which explain the entities in the stories. We take the words in the text to be evidence about the denotations of the words. In turn, we take the actions and events as evidence about the plans of the characters in the story. Because we allow for hierarchical plans, these plans may in turn be evidence for other plans. *E.g.*, in the story "Jack got on the bus. He got off at the airport." mounting, dismounting and the bus are evidence for an bus-trip plan. In turn, the bus-trip to the airport is evidence for an air-trip plan. See Figure 4.2.

As we explained in Chapter 1, we wish to translate the input text into a conventional frame-based knowledge representation language with an isa-hierarchy. In this language, slots, sub-acts, and roles in frames are represented by functions (e.g., the patient of a get action, **g1**, is represented as (**patient g1**)). The relations of interest between entities are represented by equality statements. For example, in order to represent that a going action (**g2**) is part of a plan to go shopping at a supermarket (**plan1**), we write (**go-step plan1**) = **g2**. The only predications other than equality we will use specify the types of objects (*e.g.,* (**inst r1 rope**)), types of words (*e.g., (word-inst r1 rope)*) or syntactic relations between words (e.g., *(object w1 w2)*).

Because we are limiting ourselves to describing stories using this language, in which every entity is described by naming its type and the entities which fill slots in it, our inference problems roughly follow the pattern depicted in Figure 4.3.

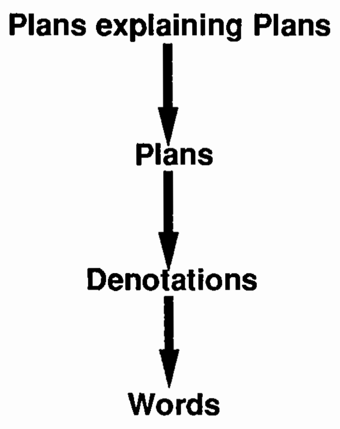


Figure 4.2: A skeletal belief network showing the basis of our inference problem.

A sample belief network based on this model is given as Figure 4.4. This belief network is a small fragment of the one Wimp3 creates when reading the story "Jack got a rope. He killed himself." We have made a number of abbreviations. In order that the figure be more readable we have written **(type entity)** for **(inst entity type)** and *(word-type word)* for *(word-inst word word-type)*.

Note that we have removed irrelevant possibilities from the network. For example, we don't have a node for **(kill r2)**. Wimp3 knows when it draws up the network that **r2** is the denotation of a word whose word-type is *rope* so, without loss of generality, it omits types that are not consistent with this piece of evidence. Since P(**(kill r2)**|*(rope r2)*) = 0 this shorthand does not have any effect on the probabilities when interpreting this story. Similarly, we do not represent all possible equality statements. Rather, we will only include equalities which are suggested by some evidence in the story (*i.e.,* equalities which have as descendents some node with evidence). This may be done because equalities without such evidence will not have any effect on the posterior probabilities at any other node: when updating networks, no information flows out of such nodes. Further, we are not interested in the probabilities of such equality statements (if we became interested in them, we could easily compute their probabilities).

This, then, is the outline of the belief networks that will be used to represent our model of story understanding. We have omitted some details for clarity of presentation (*e.g.,* as will be seen in Chapter 7, we add a number of intervening random variables which do not change the distribution represented, but which make the networks easier to evaluate), but this skeleton is followed, and provides the conceptual underpinnings.

4.5 Where do the numbers come from?

Given this model of the domain, and the network architecture outlined in the previous section, we need only a limited family of probabilities to specify our distribution. Consider again Figures 4.3 and 4.4. There are certain regularities. First of all, nodes with predications about the types of objects are always at the root of the diagrams. Other patterns of connection are that equality nodes have as their parents type predications. For example, **(get-step k1) = 1** has as its parents **(get g3)** and **(get (get-step k1))**. Finally, syntactic-relation nodes (*e.g., (object-of r2 g3)*) have as their parents relations in the world.

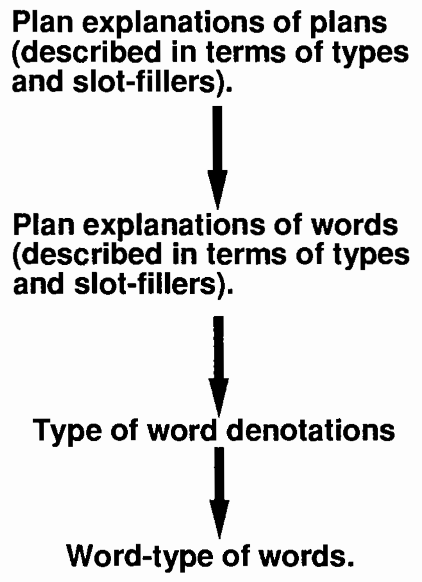

Plan explanations of plans (described in terms of types and slot-fillers).
Plan explanations of words (described in terms of types and slot-fillers).
Type of word denotations
Word-type of words.

Figure 4.3: A skeletal belief network showing the basis of our inference problem, as it looks using our knowledge representation scheme.

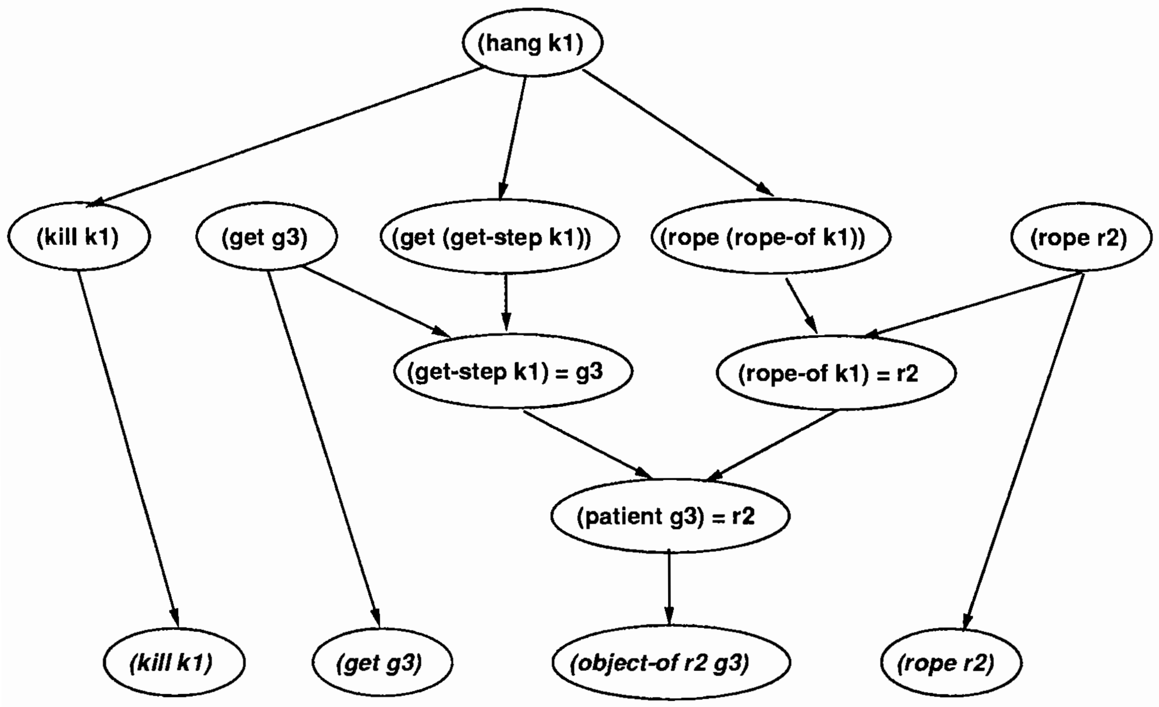


Figure 4.4: The belief network for “Jack got a rope. He killed himself.”

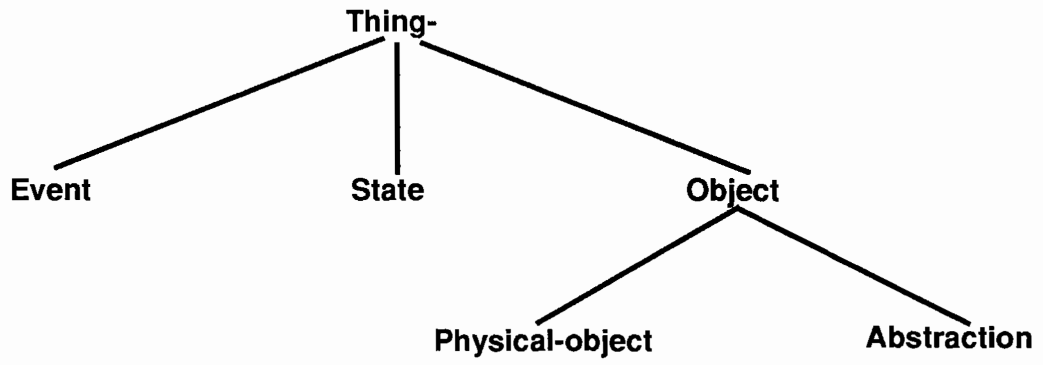


Figure 4.5: A fragment of the isa-hierarchy.

To summarize, the probabilities we need to quantify these belief networks are the following:

- Prior probabilities for propositions that an entity is of a given type.
- Probabilities of equality statements, conditioned on the types of the arguments.
- The probability that a word will be used given that the thing it denotes is of a particular type.
- The probability of a syntactic relation existing, given that a particular relation exists in the world discussed.

In the following sections, we will discuss each of these quantities in turn. We will discuss the import of these statements, in the light of our model, and we will explain the assumptions we have used in assessing the relevant quantities.

4.5.1 Priors

We require prior probabilities for propositions that a random element of the set T (the set of things in the world, see 48) is of a given type. This is used in reasoning about word denotations, and in reasoning about the various explanations of entities. In this section we give a principled way of getting these priors from our isa-hierarchy and show that this method will provide consistent probabilities. We then explain the two ways these quantities are used in our model.

The approach we take is to use our isa hierarchy, treated as a belief network, to define a distribution over the types of interest. Consider the fragment of the isa-hierarchy given as Figure 4.5. Pearl [89] shows that it is possible to assign a consistent probability distribution to any belief network(see discussion in Chapter 3), working down from the root, estimating the probabilities of children, relative to their parents.

In creating a distribution over the types in the isa-hierarchy, we simply specify for each type the proportion of objects of its super-type, which are of that type. So, for example, we specify P(Event|Thing-), P(State|Thing-), P(Object|Thing-). Given these conditional probabilities, and given that we know the prior of Thing- (1.0), we may compute a probability for each type.

Now, given that we can define such priors, how are we going to use them? First of all, we are going to use these quantities in the portion of the network pertaining to the denotations of words. In the absence of all other knowledge, the denotation of a word is simply a random variable chosen from T. Ergo, a priori the probability of it being of a given type comes from the distribution given above.

What we have given above is only true given that we are talking about a word which *has* a denotation. Function words do not have a denotation. So if we, in fact, knew nothing about a word, and were considering its denotation, we would have to use a distribution over the sample space $T \cup \perp$, rather than T. Fortunately, this state of affairs does not arise.

We are never, in practice, interested in reasoning about the denotation of a word, given that we know only that it is an open-class word, either. When Wimp3 is operating, it is always concerned with only the denotation types that are relevant. For example, if Wimp3 read the word 'bank', it would be concerned only with the two possibilities *river-bank* and *financial-institution*. If Wimp3 wasn't able to limit its decisions in this way, it would not be able to operate efficiently.

Note that this only tells us about the prior probabilities of the types of word-denotations. In order to reason about the posteriors (which is, of course, what we are interested in), Wimp3 also needs to know P(*word-type* | *denotation-type*). We will discuss this later on.

4.5.2 Explanation probabilities

We also use the distribution over T in reasoning about the plan-explanations of actions. In doing so, we make two assumptions. First of all, we assume that for each action, we know the possible plans that action could play a part in. This is a conventional assumption in plan-recognition (see, *e.g.,* [64] or [63]). In fact, our version of the assumption is weaker than usual, because we allow for actions to go unexplained. The second assumption is that no slots of a frame are optional. That is, P(slot-filler|super-frame) = 1. *e.g.,*, if the shopping plan includes a slot for going to the supermarket, that slot is always filled.

Making these assumptions, we find the probability of an explanation given an action to be

$$\frac{P(action)}{P(plan)}$$

This can be represented by a chunk of belief networkthat looks like Figure 4.6. The node labelled 'plan explanation' in the figure will be of the form **(inst** *skolem-name* **plan)**, where *skolem-name* is a unique, name created for the **plan-explanation** of **action**. The prior of this node being true will be the P(plan) we have defined above. The probability of the action-node being true given that the plan-explanation is true is 1. The probability of its being true given plan-explanation is false will be P(action) - P(plan). This latter is only an approximation – it amounts to assuming that actions do not fill slots in more than one plan – but it is a reasonable one for many domains, including our own.[4]

4.5.3 Other conditional probabilities

For story understanding in our formalization, we need conditional probabilities of three sorts. We need the probability of an equality relation between two objects of a given type. We need the probability of a particular word being used, given that thing it denotes is of a given type. Finally, we need the probability of a syntactic relation existing, given that a particular relation exists among the things discussed in the text.

[4]Note that the fact that we have made this approximation in figuring some of the distributions does *not* mean that our program is unable to understand stories where actions are used for more than one purpose. Wimp3 is capable of doing so.

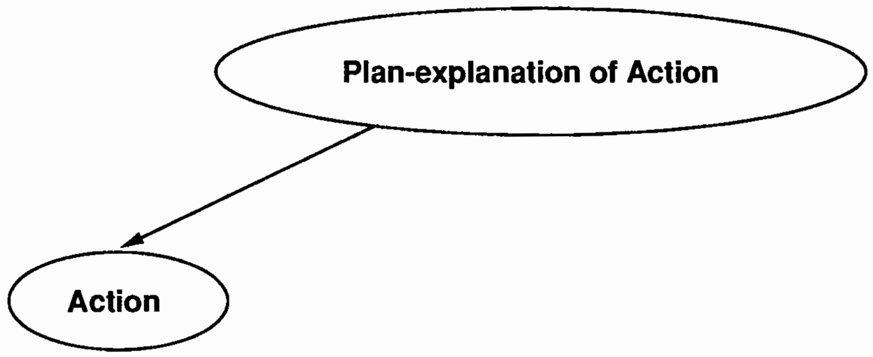


Figure 4.6: The fragment of the belief network which corresponds to the plan-explanation of an action.

- Probabilities of equality statements: We require the probability of equality statements, $x = y$, conditional upon x and y being objects of the same type, t. For example, to use Figure 4.4, we need P((**get-step k1**) = **g3** | (**get g3**), (**get** (**get-step k1**))). Given that we know the size of T, and a prior for $t(x)$, and assuming T is finite and all of its elements equiprobable, $\mathrm{P}(x = y|t(x), t(y))$ is seen to be

$$\frac{1}{\mathrm{P}(t(z)) \times |T|}$$

- P(word|denotation): As we've seen in the previous section, we take the probabilities of the **inst** statements about the denotation variables from the distribution over the sample space T. This leaves us needing the probabilities of word statements given denotations. In principle, these probabilities could be computed from the lexicon, and could absorb information about relative frequency of different words. In practice, we do not have the resources to label a large corpus, and collect the relevant statistics (in addition, features of the domain make the collection of linguistic statistics very difficult). What we actually do is to make an indifference assumption – that for all words, w, and denotations, d_i and d_j, as long as d is a possible meaning for w., $\mathrm{P}(w \mid d_i) = \mathrm{P}(w \mid d_j)$.[5] This means that the relative priors of the denotations will be the factor which determines the probabilities of the different hypotheses. For cases of very rare meanings we tag our lexicon entries with conditional probabilities which reflect a crude sense of relative frequencies. *e.g.,* we allow two meanings for the word 'went': **go-**, and the metaphoric sense **die-** ("the patient went at 8 P.M. last night."). In this case, we take advantage of the lexicon-tagging facility to weight the former sense heavily versus the latter.

- Syntactic relations: We have to provide the probability of syntactic relations, given some relationship between the entities denoted by words entering into the syntactic relations. For example, in Figure 4.4, we need P(*(object-of r2 g3)* | (**patient g3**) = **r2**), where **g3** is the action referred to by *g3*, an instance of the verb 'went'; and **r2** is the entity referred to by , *r2*, an instance of the noun 'rope.' *i.e.,* we need the probability of an author expressing a patient relation between two entities by means of the direct-object relation between the words which denote these entities.

[5]The precise value of $\mathrm{P}(w \mid d)$ is of no importance, since we are evaluating these different alternatives vis a vis each other.

Proposition	Conditions	Probability
(hang k1)		10^{-17}
(kill k1)	**(hang k1)**	.9
(get g3)		10^{-5}
(rope r2)		10^{-11}
(kill k1)	**(kill k1)**	.8
(kill k1)	**¬(kill k1)**	0
(get g3)	**(get g3)**	.8
(get g3)	**¬(get g3)**	0
(rope r2)	**(rope r2)**	.8
(rope r2)	**¬(rope r2)**	0
(object r2 g3)	**(patient g3) = r2**	.8
(object r2 g3)	**¬(patient g3) = r2**	0
(get-step k1) = g3	**(get g3), (get (get-step k1))**	10^{-6}
(get-step k1) = g3	**¬(get g3), ¬(get (get-step k1))**	10^{-20}
(rope-of k1) = r2	**(rope r2), (rope (rope-of k1))**	10^{-9}
(rope-of k1) = r2	**¬(rope r2), ¬(rope (rope-of k1))**	10^{-20}
(patient g3) = r2	**(get-step k1) = g3,(rope-of k1) = r2, (rope r2),(get g3)**	1
(patient g3) = r2	**¬(get-step k1) = g3,¬(rope-of k1) = r2, (rope r2),(get g3)**	10^{-7}

Figure 4.7: Probabilities for the belief networkin Figure 4.4.

Actually there is a complication in these last two probabilities. Really the conditional probability is the product of the probabilities we have outlined above *times the probability that a particular object (or relation) would be realized in the sentence at all.* When this is factored in we get a very low number, but it will be large relative to the probability that the author would use the word (or syntactic relation) given that the proper facts in the world did not exist. This is all we need.

4.6 Getting the model to produce reasonable results

We have defined our model and shown that it gives us some guidance in assigning probabilities we need. In this section we show that the guidance is not sufficient. We give an example which shows that the model we have outlined so far does not see stories as coherent wholes. We show that this is because the model does not take into account enough conditioning information, and show how to fix it.

Let us consider the example "Jack got a rope. He killed himself." treated in Figure 4.4. Intuitively we would say that the probability that Jack hanged himself was quite high. Certainly greater than .1.

Figure 4.4 depicts a fragment of a belief network for understanding this sentence. Put in words, the network expresses the facts that a hanging is also a killing, and that the slots in the hanging frame, **rope-of** and **get-step**, must be filled by ropes and getting events respectively, with the latter being the event in which **(rope-of k1)** is obtained. The equality statements express the idea that the "get" and "rope" mentioned are, in fact, the ones which fill the appropriate slots. It follows from these facts that **r2** must be the thing fetched in **g3**. This is captured in the connections to the node **(patient g3) = r2**.

We evaluated this netword using the probabilities given in Figure 4.7. For example, the

10^{-11} above **(rope r2)** indicates the prior probability of an arbitrary entity being a rope. Examining the link between **(rope r2)** and *(rope w2)*, we see that the probability of using the word 'rope', given that the referent of that word is a rope, is 0.8. The probability of **r2** filling the **rope-of** slot of **k1**, given that both the **(rope-of k1)** and **r2** are ropes, is 10^{-9}, if one is a rope and one is not, the probability is 0, and if neither is a rope, the probability is $\frac{1}{|T|}$.

Since the network as given does not contain the information about Jack being both the person who does the killing, and who obtains the rope, the probability on the equality of **get-step** and **g3** has been modified to reflect this information. The numbers used are calculated as discussed earlier, with $| T |= 10^{20}$, 10^9 ropes, 10^9 people, 10^{15} get events, 10^3 hangings, and about 10^6 killings.

When one evaluates this network[6] one gets a probability of hang very close to 10^{-3}. That is, the information about getting a rope has made no difference, since the probability of hang given kill is about 10^{-3}. The problem turns out to be the probabilities assigned to the equality statements.

To see this, it is necessary to get some feel for the flow of probabilities in this network **(rope r2)**, **(get g3)**, and **(kill k1)** have probability one, **(hang k1)** is about 10^{-3}, but the equality nodes for **get-step** and **rope-of** have very low probabilities because their posteriors given type equality are $\frac{1}{|ropes|}$ and $\frac{1}{|gets|}$ which are very small indeed. Changing our belief in **(patient g3) = r2** to 1 raises the probability of hanging, but not by much. The reason is that the current belief in **get-step** and **rope-of** equalities are *so* low that it is more likely that the rope was the patient of get "simply by accident," rather than seeing it as a consequence of the **get-step** and **rope-of** equalities. Thus the low probabilities on these two equality statements are the reason for this counter-intuitive result.

The solution is clear from an analysis of why the **rope-of** and **get-step** posterior probabilities are so low. Looking at the first one, it is asking the question, picking an arbitrary hanging event, and an arbitrary rope, what is the probability that the rope so chosen will be the one used for the hanging event? Obviously it is quite low, since we might be picking a rope which exists thousands of miles away from the hanging, or which existed a few hundred years ago (if we are including historical objects and events in Ω.) So *given this meaning for the number*, the number we gave is in the right ballpark. But there is other information which we did not bring to bear. Obviously in a story like "Jack got a rope. He killed himself." it is simply not likely that the rope and the killing are separated by thousands of miles, or hundreds of years. In other words, stories, and for that matter, observations in the world, are typically constrained by temporal and spatial locality. Our semantics, with random experiments over Ω, has no such constraint. We need to include it.

There are two ways to go. One would be to change the semantics to include some recognition of spatial and temporal locality of objects and events. The other is to keep the same semantics and simply include the assumption of spatio-temporal locality as a conditioning event. For the purposes of Wimp3, we do the former. Later in this section, we will see why changing the semantics is probably a bad idea in general, and mention some of the more sophisticated ways we think we can treat the hypothesis of spatio-temporal locality.

Figure 4.8 shows the **rope-of** equality statement from Figure 4.4, but now with it conditioned on spatio-temporal locality.[7] Such an **stl** predication would also appear as a condition on the **get-step** equality statement. Figure 4.8 gives the probability for **rope-of** being a particular rope, assuming locality. Note the difference in probabilities. Before it

[6]Actually we evaluated a singly-connected version using the algorithm of Pearl and Kim. [89]

[7]The statement (stl k1 r2) indicates that k1 and r2 are in the same locality.

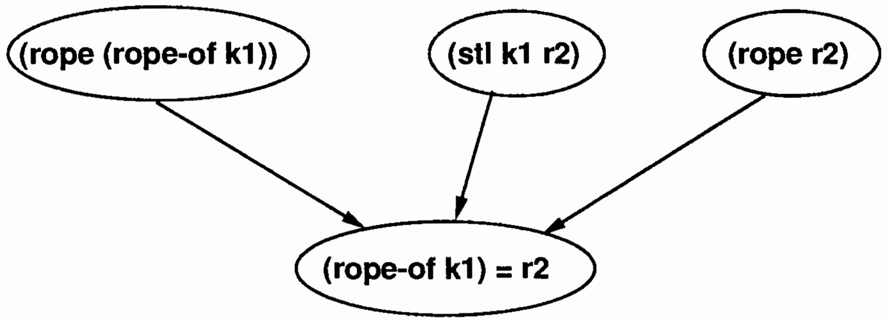


Figure 4.8: Network with spatio-temporal locality

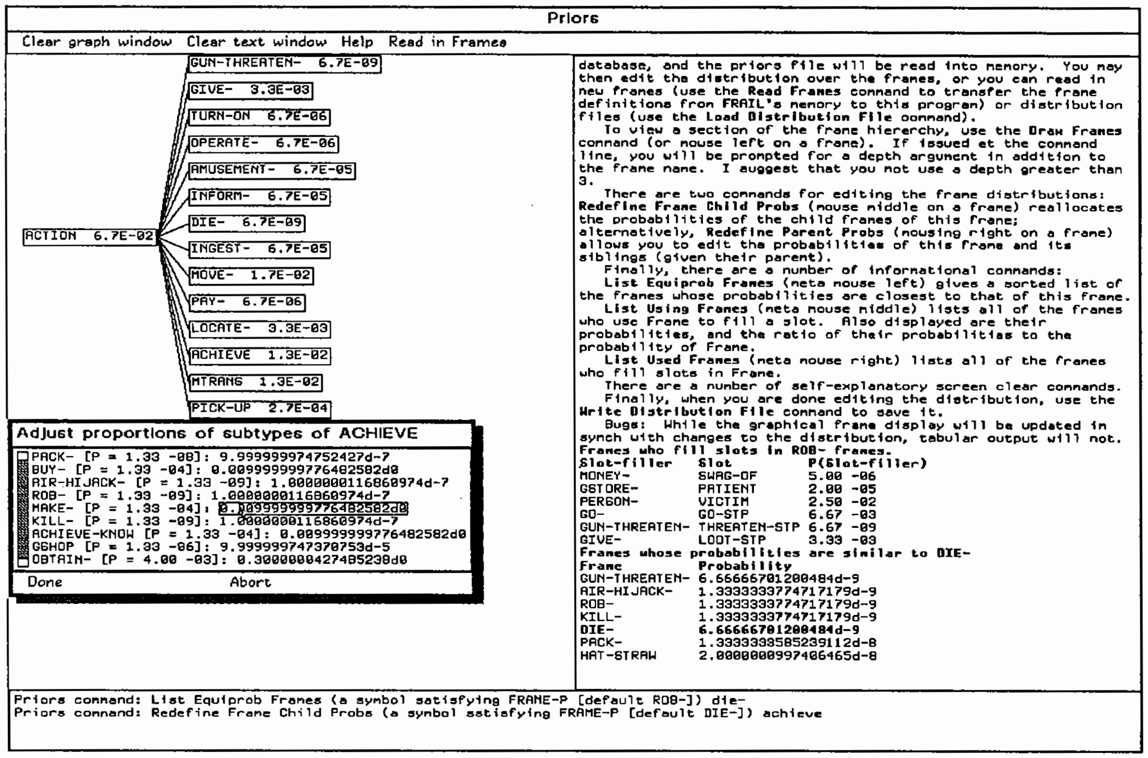
was 10^{-9}, since only one of the 10^9 ropes would be the right one. Now, however, with spatio-temporal locality, we are asking a very different question. Given that a hanging, and a rope, are found in a small part of space-time, what is the probability that the rope was used in the hanging? Obviously this depends on the size of the "part of space-time," and a more sophisticated analysis should make **stl** take this into account. However, for current purposes, suppose we envision it as a city block. Now, rather than 10^9 ropes we are talking about 10, or perhaps 100. Figure 4.8 adopts 100, so the probability that any one of these is the rope in question is 10^{-2}. Similarly, the **get-step** equality changes from 10^{-6} to 10^{-2}. Naturally, the probability of **hang** now goes up precipitously, to .3.

Now if **stl** statements were always true our analysis would be quite simple. But they are not. In stories neighboring sentences, or even different clauses in the same sentence, can be about different parts of space and time. Normal perception, it is true, sticks to local things, but if we watch TV we can get widely dispersed images as well. Thus we need some theory of under what circumstances **stl** statements are true. Building **stl** into the semantics would, presumably, mean building this theory into it as well, and since this promises to be a substantive theory, it seems a bad idea to include it in the semantical definitions.

Interestingly enough, some parts of an **stl** theory are already in place, albeit not under this name. Within the AI/Natural Language Understanding community there is a sizable body of work on "discourse structure." [46, 109] To take a typical example (this one from, [2] which provides a good overview)

1 Jack and Sue went to a hardware store to buy a new lawnmower
2 since their old one had been stolen.
3 Sue had seen the men who took it
4 and had chased them down the street,
5 but they'd driven away in a truck.
6 After looking in the store, they realized that they couldn't afford a new one

Note that lines 1 and 6 are about the same part of space-time, as are lines 2-5, but there is no commonality between the two. Discourse structure theorists would say that there are two discourse segments in this example, a major segment consisting of 1 and 6, and a sub-segment consisting of 2-5. It is obvious in this example that this analysis, and the one necessary to determine spatio-temporal locality, exactly overlap. Furthermore, when one looks at the clues that discourse theorists suggest for determining this structure, (change of time, no referents for pronouns, certain key phrases, such as "by the way") it is easy to see how they would be equally useful in determining the truth of a spatio-temporal locality hypothesis. We have done some preliminary work on exploring more discourse-oriented conditioning events than simple spatio-temporal locality, see [18].



Adjust proportions of subtypes of ACHIEVE

```
PACK- [P = 1.33 -08]: 9.999999974752427d-7
BUY- [P = 1.33 -04]: 0.009999999776482582d0
AIR-HIJACK- [P = 1.33 -09]: 1.0000000116860974d-7
ROB- [P = 1.33 -09]: 1.0000000116860974d-7
MAKE- [P = 1.33 -04]: 0.009999999776482582d0
KILL- [P = 1.33 -09]: 1.0000000116860974d-7
ACHIEVE-KNOW [P = 1.33 -04]: 0.009999999776482582d0
GSHOP [P = 1.33 -06]: 9.99999974737075 3d-5
OBTAIN- [P = 4.00 -03]: 0.30000000427485238d0
```

```
database, and the priors file will be read into memory.  You may
then edit the distribution over the frames, or you can read in
new frames (use the Read Frames command to transfer the frame
definitions from FRAIL's memory to this program) or distribution
files (use the Load Distribution File command).
   To view a section of the frame hierarchy, use the Draw Frames
command (or mouse left on a frame).  If issued at the command
line, you will be prompted for a depth argument in addition to
the frame name.  I suggest that you not use a depth greater than
3.
   There are two commands for editing the frame distributions:
Redefine Frame Child Probs (mouse middle on a frame) reallocates
the probabilities of the child frames of this frame;
alternatively, Redefine Parent Probs (mousing right on a frame)
allows you to edit the probabilities of this frame and its
siblings (given their parent).
   Finally, there are a number of informational commands:
   List Equiprob Frames (meta mouse left) gives a sorted list of
the frames whose probabilities are closest to that of this frame.
   List Using Frames (meta mouse middle) lists all of the frames
who use Frame to fill a slot.  Also displayed are their
probabilities, and the ratio of their probabilities to the
probability of Frame.
   List Used Frames (meta mouse right) lists all of the frames
who fill slots in Frame.
   There are a number of self-explanatory screen clear commands.
   Finally, when you are done editing the distribution, use the
Write Distribution File command to save it.
   Bugs:  While the graphical frame display will be updated in
synch with changes to the distribution, tabular output will not.
Frames who fill slots in ROB- frames.
Slot-filler    Slot           P(Slot-filler)
MONEY-         SWAG-OF        5.00 -06
GSTORE-        PATIENT        2.00 -05
PERSON-        VICTIM         2.50 -02
GO-            GO-STP         6.67 -03
GUN-THREATEN-  THREATEN-STP   6.67 -09
GIVE-          LOOT-STP       3.33 -03
Frames whose probabilities are similar to DIE-
Frame          Probability
GUN-THREATEN-  6.66666701200484d-9
AIR-HIJACK-    1.333333377471717 9d-9
ROB-           1.3333333774717179d-9
KILL-          1.3333333774717179d-9
DIE-           6.66666701200484d-9
PACK-          1.3333333585239112d-8
HAT-STRAW      2.0000000997406465d-8
```

```
Priors command: List Equiprob Frames (a symbol satisfying FRAME-P [default ROB-]) die-
Priors command: Redefine Frame Child Probs (a symbol satisfying FRAME-P [default DIE-]) achieve
```

Figure 4.9: A session with the distribution editor.

4.7 Some practical considerations

In this section, we discuss the actual procedure by which we created the distribution over T. We have developed a small program, with graphical interface, for drawing up the distribution over T. As mentioned in section 4.5.1, we are able to treat our isa-hierarchy as a belief network, and work our way down the trees, weighting each branching point. However, while this does guarantee us a consistent distribution, it forces us to make choices about which we have no intuitions: *e.g.*, what are the relative probabilities of **state**, **event** and **object**. Nor is it easy to understand the ramifications of these choices. In fact, unless the relative probabilities in this example are wildly skewed, this choice is of very little importance, because Wimp3 is not faced with the need to decide between alternatives, one of which is an event and one of which is an object (we typically know whether a word is a noun or a verb).

What we would like, then, is to be able to use information about the choices Wimp3 will actually face, as well as this framework (the isa-hierarchy as belief network) which helps us ensure consistency. We've written a program, with graphical interface, to do so. With this program, we may weight different branches of the isa hierarchy, while it is displayed on the screen. At the same time, we may access information about the choices Wimp3 must make: in particular, while examining a frame, we can find all other frames with roughly the same priors. We can also find all frames which fill slots in the given frame (all frames this one can explain), and we can find all frames in which this frame can fill a slot (all explanations for this frame). This makes it possible for us to focus on the important frame-frame relations. Figure 4.9 gives an illustration of this distribution editor, running on a Symbolics lisp machine.

4.8 Probabilistic logics

Given that we would like a formal language to describe a probabilistic model, a reasonable question to ask is why we don't use a probabilistic logic like that of Nilsson [84] or Bundy [8]. There are two reasons.

First, Nilsson's logic (Bundy's approach is similar) allows the user to specify the probabilities of statements and uses this information to bound a sample space of 'possible worlds' or truth-assignments. It is not possible to use conditional probabilities in this framework, since they specify ratios of probabilities of statements, rather than probabilities of statements in isolation. This is appropriate for Nilsson's problem which is to explain probabilistic entailment, but we are more concerned with conventional probabilistic inference, than with probabilistic analogs to material implication.

Second, Nilsson's Probabilistic Logic is only a propositional language. However, as the formulas of our language contain neither quantifiers nor variables, we could try to deal with them as propositional constants. If we do this, however, we sacrifice information about the relation between propositions. For example, suppose that we have a probability distribution such that **(rope r1)** is .5, and in the probabilistic logic we assign probabilities to the possible worlds to make the probability of **(rope r1)** .5 as well. Now let us ask, what is the probability of, say, **(rope r19)**. In the absence of any other information, for us this must be .5 as well. Probabilistic logic makes no such commitment. It is a propositional logic, so the probability of the two propositions **(rope r1)** and **(rope r19)** can be varied independently. Thus, our semantics is more restrictive. Furthermore, this extra restriction has the effect of placing tight bounds on the probabilities we can assign to basic type predications like **(rope r1)**. Since it will specify that any outcome of a basic experiment is a rope 1/2 the time, a probability of .5 commits us to the belief that half the entities we deal with will be ropes; not a very plausible assumption. Bundy's Incidence Calculus is a full first-order language, but he does not provide any guidance in interpreting the constants of the language, so the Incidence Calculus is no more restrictive than Probabilistic Logic.

After completing the work described in this chapter, we discovered the work of Bacchus [3], which is in many ways similar. His logic, **Lp**, is similar to our language in that the distributions he uses are over the domain of discourse, rather than over interpretations. Like our language, **Lp** makes it possible to refer to random variables. His language differs from ours in two ways. First of all, it is intended to support theorem-proving, whereas ours is designed to give a clear semantics to statements about random variables. Second, Bacchus' language is designed to support reasoning about probabilistic judgments. Statements about the probability of given events can be expressed in **Lp**, whereas in our language they are metalogical.

4.9 Conclusion

In this chapter, we have presented our model for story understanding. In doing so, we have presented a semantics for interpreting probabilistic statements expressed in a first-order quantifier-free language. We have shown how this semantics constrains the probabilities which can be associated with the propositions. Finally, we saw that while the semantics dictates very low prior probabilities for many of the statements we needed, once they are adequately conditioned, in particular with spatio-temporal locality, the probabilities become more "reasonable." We suggested that our notion of spatio-temporal locality, and the notion of discourse segment found in current AI natural language work are at least very close, and may be identifiable. In our estimation this possibility sheds some interesting light on the notion of discourse segments, since it allows for their computation in a probabilistic way. Those familiar with the work in the area will be aware of how hard it has proven to give deterministic, non-circular rules about when such segments are to be created, and what can be determined from their creation. In the chapters to come, we will show how this model can be used by a story understanding program.

Chapter 5

Dynamic Construction and Evaluation of Belief Networks

5.1 Introduction

The preceding chapter has introduced our stochastic model for text understanding. In order to practically apply this model (with its impractically large network) to a story understanding problem, we must be able to reason with only the relevant portions. To this end, we have developed a technique for incrementally constructing and evaluating belief networks, using network construction rules.

We have developed a network-construction language similar to a forward-chaining language using data dependencies, but with additional features for specifying distributions. Using this language, we can define parameterized classes of probabilistic models. These parameterized models make it possible to apply probabilistic reasoning to problems for which it is impractical to have a single large, static model.

While this technique was developed for our particular application, it is of more general use. There are a number of applications where such a facility would be useful. An expert system for genetic counseling would be able to quickly draw up and evaluate a pedigree network for a given family. Our language provides a convenient facility for combining general knowledge from a domain theory (in this case, genetics) with facts about a specific case (an individual pedigree). One could also diagnose machine faults in a large class of machines, all built out of a given set of components. For example, one could define a class of models for diagnosing mechanical problems in motorcycles. This class of models would be defined in terms of the operation of components. To create a instance of this class of models for a particular motorcycle, one would specify the values of parameters like fuel supply (carburetion or fuel-ignition), transmission (chain or shaft), number of cylinders, etc. For visual pattern recognition, this approach would make it more convenient to experiment with different topologies and bond functions. A standard set of rules for recognizing edges in camera images could be combined with different libraries of shape descriptions.[1]

In this chapter we give a general discussion of our network construction language. In order to stress its generality, we will offer a motivating example from a domain other than language understanding (genetics). In later chapters, we will discuss the way Wimp uses incrementally-constructed belief networks.

[1]An observation for which we are indebted to Ulf Grenander.

5.2 Our network-construction language

This section describes how rules are written in our network-construction language, FRAIL3. FRAIL3 has the following features:

- It allows us to create propositions, random variables in a belief network, which are also indexed as statements in a first-order logic database.
- In order that these belief networks be solved, FRAIL3 makes it convenient to specify the conditional probability matrices of these propositions, using declarative knowledge stored in the logical database.

Our probabilistic database has a great deal in common with a deductive database with truth maintenance (see [19] for an introduction, or [41, 78, 37]). Such a system combines a logic-programming language with forward- and backward-chaining (bottom-up and goal-directed inference, respectively) with a data dependency, or justification, network. When statements are added to the database, they are given a justification, which records the antecedent statements necessary to justify belief in the consequent.

Our database language is similar to a TMS language in having rules for forward and backward reasoning. It differs in allowing greater flexibility in specifying the way arcs connecting database statements are added. In a TMS, one always adds arcs (justifications) from antecedent statements to consequents. However, in evidential reasoning, we often want to chain from observed statements to possible causes, and direct arcs from the causes to the observation. Our rules allow us to specify independently the statements to be added to the database, and the arcs to be drawn between them.

The data structures corresponding to the arcs are also more expressive. Rather than having justifications, we attach *pforms* to the statements in the database. Pforms contain information which specifies the probabilistic dependency of the node at the head of an arc set on its direct ancestors. More formally, each pform is a hyperedge from a set of statements to one statement. This hyperedge corresponds to a set of edges in a belief-network. See Figure 5.1

In the next several paragraphs, we will discuss our database language. We interact with FRAIL3 using the functions **assert**, **retrieve** and **index**, and the declarations **defpredvals**, **defpreddist**, and **defauxpred**. Assert, retrieve and index make use of the distinguished predicates →, ←, and →←.

5.2.1 Predicates

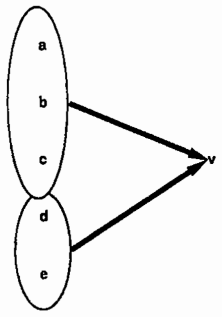
(← *query antecedent(s)*) To answer a request for *query*, retrieve *antecedent(s)*. A conventional backward-chaining rule.

(→ *trigger consequent* :prob *pforms*) → is the basic network-constructing operator. When a statement unifying with trigger is added to the database, apply the bindings to consequent, and add it to the database. The pforms control the way arcs are added to the network, and give information about the way the conditional probability matrices are set up at the tails of the arcs. This information will be used by the distribution functions discussed below. A sample rule:

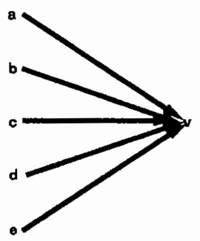
Rule 1
```
(→(condition ?x) :label ?caused
   (causal-event ?x) :label ?cause
   :prob ((?cause →?caused) ((t |t) .9) ((t |f) :p)))
```

```
:parents (a, b, c) :child v
:parents (d, e) :child v
```

(a) pforms



(b) corresponding hypergraph



(c) corresponding belief network

Figure 5.1: The interpretation of pforms.

This states that whenever we predicate condition of some entity, we should also postulate a corresponding causal-event. The first part of the :prob argument specifies that we should draw an arc from the causal-event node to the condition node. In order that we be able to specify the addition of arbitrary arcs, we need to be able to refer to statements by name. The :label operator provides this capability. In this rule, ?caused will be bound to a pointer to the triggering statement, and ?cause will be bound to a pointer to the newly-added statement. It should be emphasized that, unlike in a conventional TMS language, which automatically adds arcs from antecedents to consequents, we do *not* add any arcs other than those explicitly designated by :prob arguments to rules.

The second part of the :prob argument indicates that when we draw up the conditional probability matrix for the condition node, its probability given that causal-event is true is .9. Its probability given that causal-event is false should default to the prior probability (the way this defaulting can be done will be discussed below when we discuss defpreddist).

(→← *additional-antecedents consequent pforms*) If FRAIL3 is told to add a statement of this form to its database, it will try to retrieve *additional-antecedents*, and apply their variable bindings to assert *consequent*. This operator is often used to retrieve additional facts from our domain knowledge. For example, the following is a rule used in our language-understanding application, for reasoning about word-senses:

Rule 2

```
(→(word-inst ?i ?word) :label ?A
   (→←(word-sense ?word ?frame ?prob)
        (inst ?i ?frame) :label ?C)
   :prob ((?C →?A) ((t | t) ?prob) ((t | f) (/ :p 100))))
```

This rule may be read as follows: If a node is added to the network describing a new word-token (an instance of a word), and if there is a statement in the database specifying that one of the senses of this word is ?frame, add a node for an instance of the type ?frame. Draw an arc from this newly-added node (?C), to the word-inst node (?A). When drawing up the probability matrix for the word-inst node, use information about the frequency with which it is used to express the concept frame, for the case where the node for frame is true, else use a default value of $\frac{1}{100}$th of the prior probability.

Another common use of →← is to collect additional parents for a node. For example, one of the tasks for our natural language system is to find referents for pronouns. A desire to refer to an already-discussed person or thing can be the explanation for using a pronoun. So when we see a pronoun, we need to find all the previously known entities that are of the same gender as the pronoun, and build arcs from them to the newly-added statement. This requires the kind of combined forward- and backward-chaining that →←provides. A simplified version of our pronoun rule follows:

Rule 3

```
(→ (word-inst ?i ?pronoun) :label ?PRONOUN
   (→← (and (inst ?x ?frame)
             (compatible-gender ?pronoun ?frame))
        (and (== ?i ?x) :label ?EQ
             (has-referent ?i) :label ?HAS-REF))
   :prob (((?EQ → ?HAS-REF)
        ((t |t) 1)
```

```
    ((t |f) 0))
   ((?HAS-REF → ?PRONOUN)
    ((t |t) 1) ((t |f) .00001))))
```

We may gloss this rule as follows: If you see a pronoun (the trigger statement) , check to see if there is an already-existing entity with compatible gender (this is an additional condition, added by the →←predicate), this rule will fire. When this rule fires, it asserts that there is a possible coreference relation between the pronoun (?i) and the previously-known entity (?x): this is the statement we have labelled ?EQ. It also asserts that there is a referent for the pronoun: this assertion is given by the statement ?HAS-REF.

5.2.2 Declarations

(**defauxpred** *predicate*) Declares that *predicate* is an auxiliary predicate. Predications using auxiliary predicates are not included in the belief network, but are indexed in the database, and trigger chaining.

(**defpredvals** *predicate value-list*) Declares that the state space of statements/random variables with this predicate is *value-list*. We will see the use of this declaration in the genetics example in section 5.3 (rule 8), below.

(**defpreddist** *predicate prior-fn posterior-fn*) These declarations tell us what functions to use to set up the conditional probability matrices for statements with predicate *predicate*. If such a statement is a root node of the graph, *prior-fn* will be called with the statement as an argument. Otherwise, FRAIL3 will apply *posterior-fn* to the pforms attached to the statement. The function of the posterior-fn is to combine the information from the various pforms.

We need **defpreddist** because our other rules are not sufficient to quantify belief networks. There are two cases where we fall short:

1. They do not allow us to specify distributions for nodes without parents, and
2. they do not dictate how to fill out the conditional probability distributions for nodes which have more than one pform.

Problem (2), filling out the conditional probability distributions, arises in hand-construction of belief networks, as well. It occurs because, typically, rather than being able to determine the probability of a random variable conditioned on all direct influences, one only has lower-order conditional probabilities.

Functions which we use as posterior-fns in our language-processing application are noisy versions of various logic gates: noisy-or, noisy-xor and noisy-and. To see how these are used, consider the rule for word-senses, given above as rule 2. The posterior-fn we use for word-inst statements is xor-dist. Typically, words are ambiguous, and have a number of senses. Each such sense will give rise to a different pform. What we want our posterior-fn to dictate is that each one of these senses is a possible explanation for the use of the given word, and that the senses are mutually exclusive. Pseudo-code for xor-dist is given as Figure 5.2. This is a simple procedure which specifies that the probability of a node given a state of its parents is 1 if that conditioning-case satisfies one and only one pform, and 0 otherwise.[2]

[2]A conditioning-case is a list of (node . state) pairs, and a node-case is (node . state). This terminology is borrowed from IDEAL[106].

```
procedure xor-dist (node pforms)
/*this loops through all possible instantiations of the predecessors
  of node */
for conditioning-case = all-states(node-predecessors node)
    /* this loops through all possible assignments to node --
       in this case, since node is boolean, :true and :false */
    for node-case = all-states(node)
       number-true = number of pforms satisfied by conditioning-case;
       if state-name(node-case) = :true then
          if number-true = 1 then
                prob[node-state,conditioning-case] = 1;
          else
                prob[node-state,conditioning-case] = 0;
       else %the state of the node is :false
          if number-true = 1 then
                prob[node-state,conditioning-case] = 0;
          else
                prob[node-state,conditioning-case] = 1;
    end for node-case;
end for conditioning-case;
end procedure.
```

Figure 5.2: Pseudo-code for the xor-dist posterior-fn.

5.2.3 Functions

(**index** *statement*) Adds statements to the first-order logic database that are *not* also linked into the belief network. Statements which are **indexed**, rather than asserted, do not trigger chaining. Rules, and other non-ground formulae are indexed.

(**assert** *statement*) Add *statement* to FRAIL's database. Trigger relevant chaining rules.

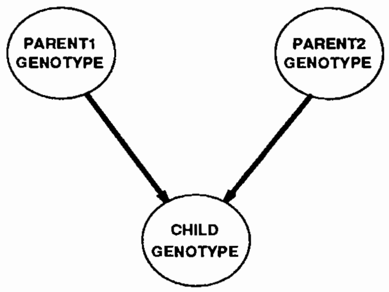
(**assert-with-evidence** *statement state*) Like **assert**, above, but also add evidence that the state of *statement* is *state*.

(**retrieve** *statement*) Retrieve statement from FRAIL's database, using appropriate backward-chaining rules. Returns a list of binding lists.

5.3 An example from genetics

To give an idea of how our system can be used, we give our treatment of an example in genetics. We have adapted this example from [60] and [105]. This example has been chosen solely for its convenience and clarity. Any errors should be attributed to our ignorance about the domain.

We consider a gene with two alleles (or values), a1 and a2. Every individual's **genotype** consists of two alleles, one inherited randomly from each parent. Because we have no way, in general, of knowing from which parent a given allele has been inherited, we represent genotypes as unordered pairs: a1a1, a1a2 or a2a2. a1 is recessive and gives rise to a detectable



	a1a1	a1a2	a2a2
a1a1	(1, 0, 0)	(0.5, 0.5, 0)	(0, 1, 0)
a1a2	(0.5, 0.5, 0)	(0.25, 0.5, 0.25)	(0, 0.5, 0.5)
a2a2	(0, 1, 0)	(0, 0.5, 0.5)	(0, 0, 1)

Figure 5.3: Conditional Probability Matrix for Transmission Probabilities. The three entries in each cell are P(a1a1|*conditioning-case*), P(a1a2|*conditioning-case*), and P(a2a2|*conditioning-case*), respectively. Genotypes of parents are the horizontal and vertical indices.

condition, a2 is dominant and does not. Therefore, the **phenotypes** (the detectable conditions) are **present** corresponding to genotype a1a1, and **absent** corresponding to genotypes a1a2 and a2a2.

In order to model the situation resulting in populations with these genotypes, we need to know probabilities determining the transmission of the genes from parents to children. These are given in Figure 5.3. In order to avoid infinite regress, we must make some assumptions about the genotypes of individuals about whose ancestry we are ignorant (**founders**). We assume they are members of a population in Hardy-Weinberg equilibrium. This yields the following table of priors:

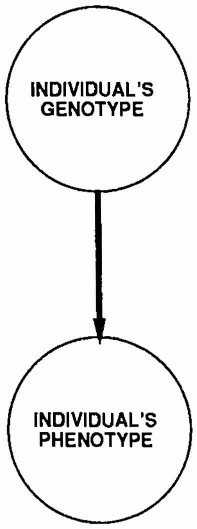
$$\begin{aligned} p &= p(a1) \\ q &= p(a2) = 1-p \\ p(a1a1) &= p^2 \\ p(a1a2) &= 2pq \\ p(a2a2) &= q^2 \end{aligned}$$

Finally, we need the **penetrance probabilities**: the distribution of the phenotype given a genotype. A very simple one is given in Figure 5.4.

The rules necessary to describe this situation are:

Rule 4 *Transmission of genotypes:*

```
(→(child ?c ?p1 ?p2)
   (and (genotype ?c) :label ?CHILD-NODE
        (genotype ?p1) :label ?PARENT-1
        (genotype ?p2) :label ?PARENT-2)
   :prob((?PARENT1 ?PARENT-2 →?CHILD-NODE
          ((a1a1 |a1a1 a1a1) 1)
          ((a1a1 |a1a1 a1a2) 0.5) ((a1a2 |a1a1 a1a2) 0.5)
          ((a1a2 |a1a1 a2a2) 1)
          ((a1a1 |a1a2 a1a1) 0.5) ((a1a2 |a1a2 a1a1) 0.5)
```



	Genotype		
Genotype	a1a1	a1a2	a2a2
:present	1	0	0
:absent	0	1	1

Figure 5.4: Conditional Probability Matrix for Penetrance Probabilities.

```
((a1a1 |a1a2 a1a2) 0.25) ((a1a2 |a1a2 a1a2) 0.5)
((a1a2 |a1a2 a2a2) 0.5) ((a2a2 |a1a2 a2a2) 0.5)
((a1a2 |a2a2 a1a1) 1)
((a1a2 |a2a2 a1a2) 0.5) ((a2a2 |a2a2 a1a2) 0.5)
((a2a2 |a2a2 a2a2) 1))))
```

Rule 5 *Penetrance:*

```
(→(phenotype ?NAME)
  (→←(penetrance-prob ?PROB)
       (genotype ?NAME) :label ?GENOTYPE
  :prob ((?GENOTYPE →?PHENOTYPE)
        (((present |a1a1) ?PROB)
        ((absent |a1a2) 1) ((absent |a2a2) 1)))))
```

The instantiate-observation statement above will not be a real statement. We specify an assert function for this predicate, so that when FRAIL3 is told to assert such a statement, it will instead call the function add-evidence on the statement to which ?PHENOTYPE is bound.

Rule 6 *Assert function for* instantiate-observation*:*

```
(defun assert-observation (pattern)
        (add-evidence (second pattern)
```

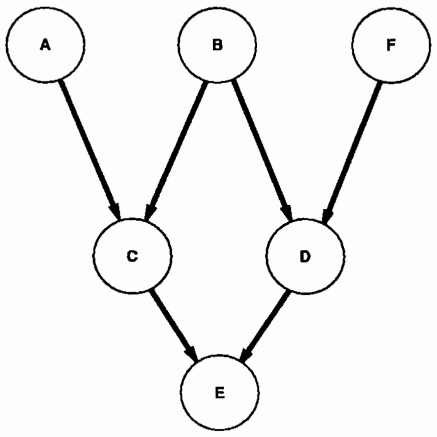


Figure 5.5: Belief network/pedigree corresponding to a case of incest.

```
                    (third pattern)))

(setf (assert-function instantiate-observation)
      assert-observation)
```

Rule 7 *Penetrance probability:*
(penetrance-prob 1)

Rule 8 *Predicate definitions:*

```
(defpredvals genotype :a1a1 :a1a2 :a2a2)
(defpreddist genotype hardy-weinberg transmission)

(defpredvals phenotype :present :absent)
(defpreddist phenotype nil simple-pform)

(defauxpred child)
```

We should note a couple of features of the above rules. First of all, note that we can use the same rules for many different values of p and q, simply by changing the prior-function for genotype nodes. Furthermore, we can change the penetrance probability just by changing rule 7. Finally, note that this scheme avoids adding unnecessary phenotype nodes. We will only have phenotype nodes which correspond to actual observations.[3]

In [60], Jensen et. al. give an example of a belief network for an example of incest. It is given as Figure 5.5. Individual E is known to have the condition. We would create a diagram corresponding to this situation with the following commands to FRAIL3:

```
(assert (child C A B))
(assert (child D B F))
(assert (child E C D))
(assert-with-evidence (observed-phenotype E) :present)
```

5.4 Implementation details

[3]If we so desired, we could also have phenotype nodes for individuals about whom we would like to query.

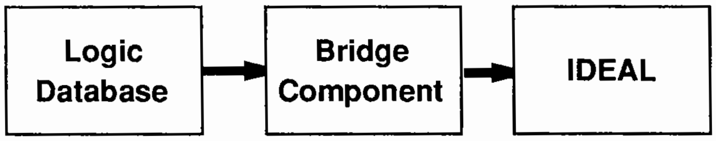


Figure 5.6: The three components of FRAIL3.

In this section, we describe the actual structure of FRAIL3. FRAIL has three components: its logic database, a network evaluation component, and a bridge component (see Figure 5.6). The logic database, built on existing software, provides the facilities for indexing statements, unification, performing pattern-directed inference, etc. The network evaluation component, for which we use the IDEAL system, allows us to apply a number of network-evaluation algorithms to belief networks. The bridge component translates the data structures of the logic-database, constructed by network-construction rules, into structures suitable for processing by IDEAL. Both the logic database and the network evaluation component are straightforward adaptations of well-understood software, so we will discuss them only briefly, and concentrate on the bridge component.

The logic database is an adaptation of the FRAIL (FRame-based AI Language) of long standing at Brown University [14, 16]. FRAIL provides the conventional forward- and backward-chaining rule facilities of an AI language (see, for example [19, 20]). The distinguishing feature of FRAIL is that its statements are indexed both as statements in first-order logic (as in, *e.g.,* a Prolog interpreter [24]), and as statements in a semantic network. However, this feature, while useful to us in our natural-language processing application, is not relevant to our discussion here. From this point of view, any such AI language would be an equally useful foundation, provided it allowed some way to associate auxiliary data structures with statements in the database.

We use the IDEAL[4] system [106, 107], written at Rockwell's Palo Alto Science Lab, to evaluate the networks. IDEAL is a program allowing the specification of belief networks and influence diagrams in LISP. It provides a framework within which a number of network-solution algorithms can be applied to such diagrams, to date it includes two simulation algorithms, two conditioning algorithms, two clustering algorithms and two algorithms for evaluating influence diagrams. All of these algorithms operate on influence diagrams specified as particular Common Lisp structures. Our bridge component takes networks specified as networks of statements in the logic database component, and translates them into IDEAL structures. The IDEAL structures are then associated with the statements indexed in the database, so that the probabilities of statements are accessible to the database routines.

The bridge component, then, is at the heart of the work described here. There are two parts to the bridge component. The first is straightforward. This part takes a list of statements, and constructs for each statement, an IDEAL structure (or node), and links these IDEAL nodes together according to the arcs specified in the pforms of a statement. The second, and more complicated part of the bridge software constructs the distribution arrays for each IDEAL node. It is this component which applies the functions declared in **defpreddist** forms (see page 65). For each IDEAL node, we find a distribution function from the predicate of the corresponding statement, and call that function. This is the stage at which the belief network's distribution is fully specified: the root nodes get prior distributions, and nodes with several pforms get complete posterior distributions. A pseudo-code description of the bridge component is given as Figure 5.7.

[4]Influence Diagram Evaluation and Analysis in Lisp.

```
procedure bridge-component (statement-list);
var ideal-network = NIL;
    node;
    distribution-function;
begin
    foreach statement in statement-list do
        stm-node(statement) := add-ideal-node(statement,ideal-network);
    foreach statement in statement-list do
        begin
            node := stm-node(statement);
            /* attach node to its predecessors */
            add-ideal-arcs(statement, node, ideal-network);
        end;
    foreach statement in statement-list do
        begin
            node := stm-node(statement);
            distribution-function :=
                distribution-function(stm-predicate(statement));
            apply(distribution-function,stm-pforms(statement),node);
        end;
end procedure.
```

Figure 5.7: Pseudo-code for the routine which creates IDEAL data structures corresponding to FRAIL statements.

5.5 Related Work

There has been a great deal of work on probabilistic ATMS's for dynamically constructing probabilistic models. An ATMS[37] is a multiple-context truth maintenance system. Laskey and Lehner [70, 71] have shown that Dempster-Shafer theory can be mapped onto the ATMS in a straightforward way. D'Ambrosio developed a probabilistic ATMS[32, 33], and we developed a probabilistic ATMS tailored for language understanding[43].

In our experience, these logical-probabilistic hybrids are unsatisfactory for a number of reasons. Probabilistic inference and logical inference do not correspond perfectly. The operation of conditioning is not natural in logic. Logical inference does not permit us to distinguish between evidential and causal support for a proposition. Pragmatically, it seemed wasteful to combine two expensive labelling procedures – ATMS context labelling and probabilities.

We are greatly indebted to Judea Pearl for his idea of combining causal influences in stereotyped ways like the noisy-or gate [89]. Our posterior-functions are a generalization of this idea.

In work done simultaneously with ours, Jack Breese has developed very similar techniques for rule-based construction of probabilistic models[7]. The differences between his work and ours arise from the different problems we are trying to solve. Breese's method is aimed at creating influence diagrams in order to reply to queries. Our approach, on the other hand, has been motivated by an interpretation problem made up of a myriad micro-problems (each of the syntactic, semantic and pragmatic interpretation processes poses many such questions). Furthermore, we want Wimp3 to build its interpretations incrementally: at all times to have a representation of the text it has seen so far. These features motivate us to take a more forward-reasoning, less goal-directed approach. In fact, influence diagrams are a more goal-directed representation, in that effort is focused on evaluating the value node of the diagram (and in the process, discovering the optimal decision policy).

Another difference is that Breese does not have anything corresponding to our posterior-functions. Associated with his network-constructing rules are fully-fleshed-out conditional probability distributions. When more than one such distribution applies, his system chooses which to apply according to some simple heuristics. We need to be able to specify in greater detail how to compare different probabilistic dependencies, because our models are too big to describe more explicitly, and because nodes and arcs are added to and deleted from our networks.

Two examples will clarify this important difference: First of all, consider our rule for finding word-senses (rule 2). If each word has a large number of possible senses, it is clearly impossible for us to store a full conditional probability matrix for each word, given all its senses. Impossible, and unnecessary as well, because the different causes interact in a stereotyped way.

The second reason for having prior and posterior functions is the incremental nature of our interpretation problem. We make hypotheses about actions and events to explain a text, and then consider the plans and goals of characters in the story to explain these actions and events. In order to do so, we gradually deepen our networks; nodes which were at the root of our networks become interior nodes as we 'unpack' their prior probabilities into probabilities conditioned on more remote causes.

Before we conclude, we would like to point out this approach to network construction could straightforwardly be adapted to constructing other kinds of network representations, like Markov networks or influence diagrams. For Markov networks, compatibility functions could be specified analogously to the way we specify conditional probabilities. For influence diagrams, we would simply extend our existing facilities to allow specification of utility functions and decisions.

5.6 Summary

In this chapter we have outlined a language for constructing belief networks on an as-needed basis. This language of rules makes it convenient to define parameterized classes of probabilistic models. Such parameterized models are more convenient to use, and more efficient to represent than full-blown belief networks. We have demonstrated the usefulness of this technique with a simple example from genetics.

Chapter 6

The Architecture of Wimp3

In the previous chapters we have laid the foundations for our approach to story understanding. We have discussed the problem and the probabilistic model, and we've introduced a technique for probabilistic reasoning which makes it possible for us to use this model. In this chapter we will outline our program, Wimp3, and how it applies this work. We will start by giving an overview of Wimp's functioning, and then move on to detail the different components of the program.

Wimp3 works as follows:

1. A parser reads a single word of the English text. It produces statements which describe the words in the story and the syntactic relations between them.

2. The output of the parser is used by the network construction component. This component contains rules for language abduction. It builds a net, or extends the current net if some input has already been received. The plan-recognition parts of network-construction are carried out under the direction of a marker-passer.

3. We collect the portion of the belief network which may be affected by the most-recent additions to the belief network. This sub-graph is evaluated by a network-evaluation component.

4. If certain conclusions are overwhelmingly favored, they may be accepted as true to simplify further computation. Statements not rejected as too improbable will trigger further network-expansion. If nodes are added to the network, go to step 3, else, return to step 1.

Let us consider a simplified account of Wimp's processing of the sentence "Jack went to the liquor-store.". When the parser reads the word 'went', it will call on the morphological analyzer, which recognizes 'went' as a form of the verb 'go'. It will generate the statement (word-inst went2 go). This will trigger a network-construction rule, which looks for possible

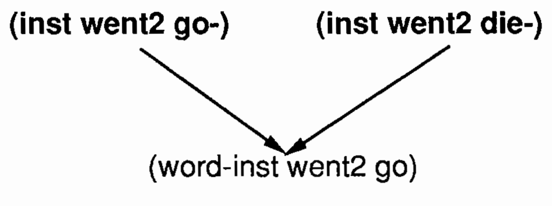


Figure 6.1: A fragment of the belief network representation of "Jack went"

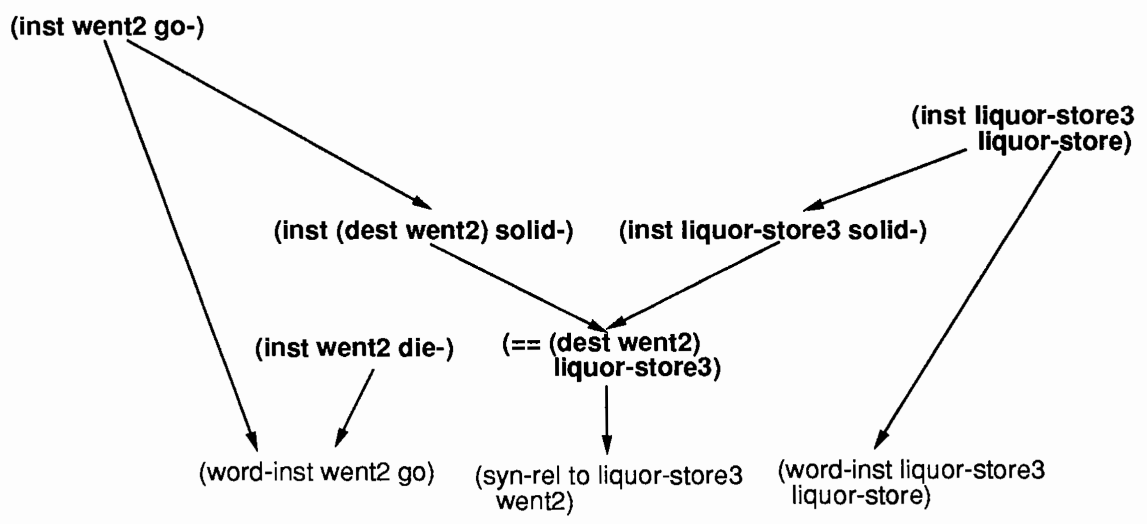

(inst went2 go-)
(inst liquor-store3 liquor-store)
(inst (dest went2) solid-)
(inst liquor-store3 solid-)
(inst went2 die-)
(== (dest went2) liquor-store3)
(word-inst went2 go)
(syn-rel to liquor-store3 went2)
(word-inst liquor-store3 liquor-store)

Figure 6.2: The belief network after reading the sentence "Jack went to the liquor-store."

senses of the word 'go'. It finds two: the first being to move oneself – **(inst went2 go-)** and the second, unusual meaning, 'died', as in "Francisco Franco went at 10 o'clock last night." – **(inst went2 die-)**. A fraction of the resulting network[1] is given as Figure 6.1.

The next processing of interest is done after Wimp3 has read the rest of the sentence: "...to the liquor-store." See Figure 6.2. When the parser reads this expression, it generates the statements (word-inst liquor-store3 liquor-store) and (syn-rel to liquor-store3 went2). Working as before, our word-sense rule generates the statement **(inst liquor-store3 liquor-store)**. The rule for cases adds the hypothesis that **liquor-store3** is the **destination** of **went2**, which relies on the assumption that **go-** is the correct sense of the word 'went' (went2).[2]

After updating the belief network, we prepare for another round of network-extension. At this point we pass marks from **liquor-store3**.[3] Marks from **liquor-store3** and **went2** meet at the plan **liquor-shop**: a plan to buy liquor. The marker-passer checks the path against its filters, and accepts it. This causes the addition of a liquor-shop hypothesis (**liquor-shop4**), resulting in the network shown as Figure 6.3. Note that the new hypothesis is tied to **went2**, rather than to **liquor-store3**, because the former is an action and the latter an object (see the statement in the upper left corner of the network).[4] Note also, that this hypothesis has given us another reason to believe in **(== (dest went2) liquor-store3)**: if **went2** is the **go-step** of **liquor-shop4** and **liquor-store3** is the **store-of liquor-shop4**, then this equality follows necessarily (viz. the new edge into the statement **(== (dest went2) liquor-store3)**).

Updating this network results in a high degree of belief in **liquor-shop4**. This in turn supports the interpretation of 'went' as meaning 'go', rather than 'die.' No further chaining is made, and Wimp3 returns to the parser to wait for further input.

Wimp3 has three main components: the parser, the network construction component, and the network evaluation component. In the sections of this chapter, we will explain

[1] We omit statements about Jack's being the agent of went2.

[2] As before, we have suppressed some of the statements, *e.g.*,(syn-rel det liquor-store3 the), for the sake of clarity.

[3] See p. 21 for a discussion of marker-passing.

[4] The reasons for associating explanations with actions rather than objects are outlined in Chapter 4, see p. 47.

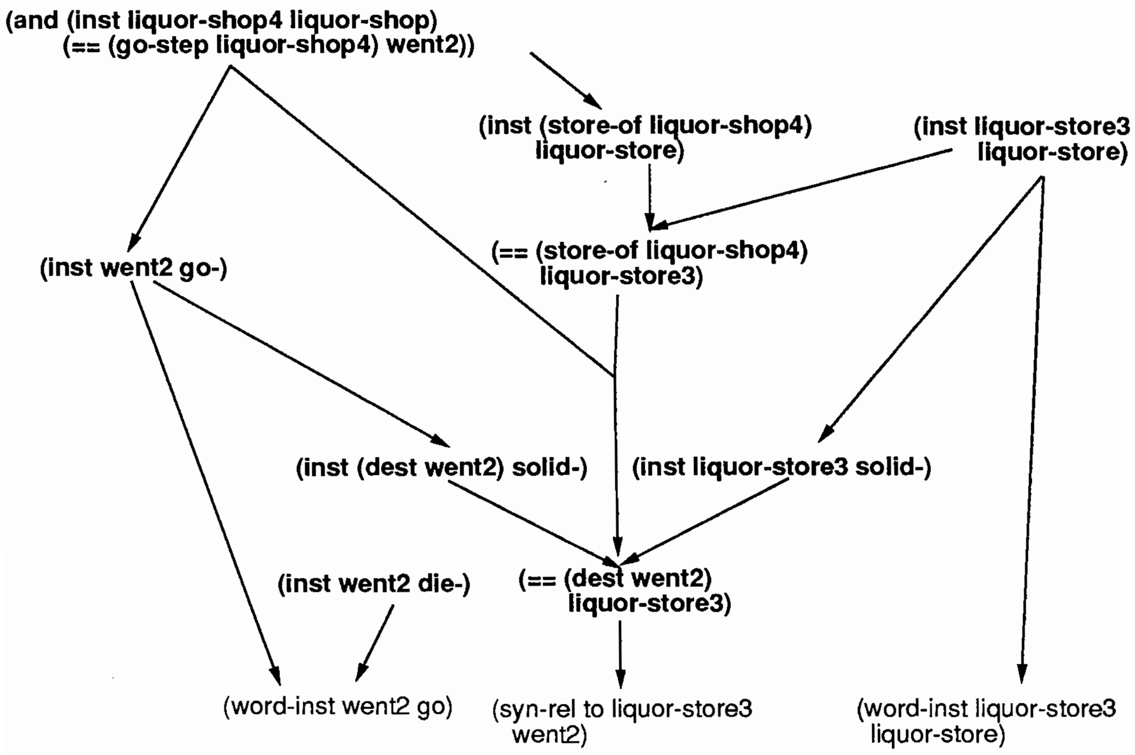

(and (inst liquor-shop4 liquor-shop)
(== (go-step liquor-shop4) went2))
(inst (store-of liquor-shop4)
liquor-store)
(inst liquor-store3
liquor-store)
(inst went2 go-)
(== (store-of liquor-shop4)
liquor-store3)
(inst (dest went2) solid-)
(inst liquor-store3 solid-)
(inst went2 die-)
(== (dest went2)
liquor-store3)
(word-inst went2 go)
(syn-rel to liquor-store3
went2)
(word-inst liquor-store3
liquor-store)

Figure 6.3: The belief network after marker-passing inferences.

the structure and operation of these modules. We will have relatively little to say about the parser. Its construction builds on well-understood principles, and the details of the grammar are not relevant to any theoretical claims we make. We have discussed the network construction component somewhat in Chapter 5; in this chapter we will discuss the strategies we have used to manage the construction process specifically for Wimp3. Later, in Chapter 7, we will discuss the particular network-construction rules used in Wimp3. As far as the network-evaluation component is concerned, we will discuss the posterior distribution algorithms we have experimented with, and why we have chosen the one we use now.

6.1 The Parser

The parser used by Wimp3 is an incremental, all-paths augmented transition network (ATN) parser. An ATN parser is a transition-network parser for a context-free grammar. It is augmented by rules which allow the setting of variables, etc., in order that it can parse languages which are not truly context-free, and in order to do semantic processing. For an introduction to the principles of ATN parsing, see [2] or [19].

Wimp3's parser is unusual in being both an all-paths parser and an incremental parser. By 'all-paths parser' we mean that Wimp3's parser yields all possible parses of the input. By 'incremental parser' we mean that Wimp3's parser operates on the input left-to-right, one word at a time, yielding partial results as it goes, rather than only after parsing an entire sentence.

A conventional ATN parser works as follows: For every rule in the grammar, we construct a transition network. A transition network is a push-down automaton(PDA). A sample set of transition networks is given as Figure 6.4. When parsing, a conventional transition

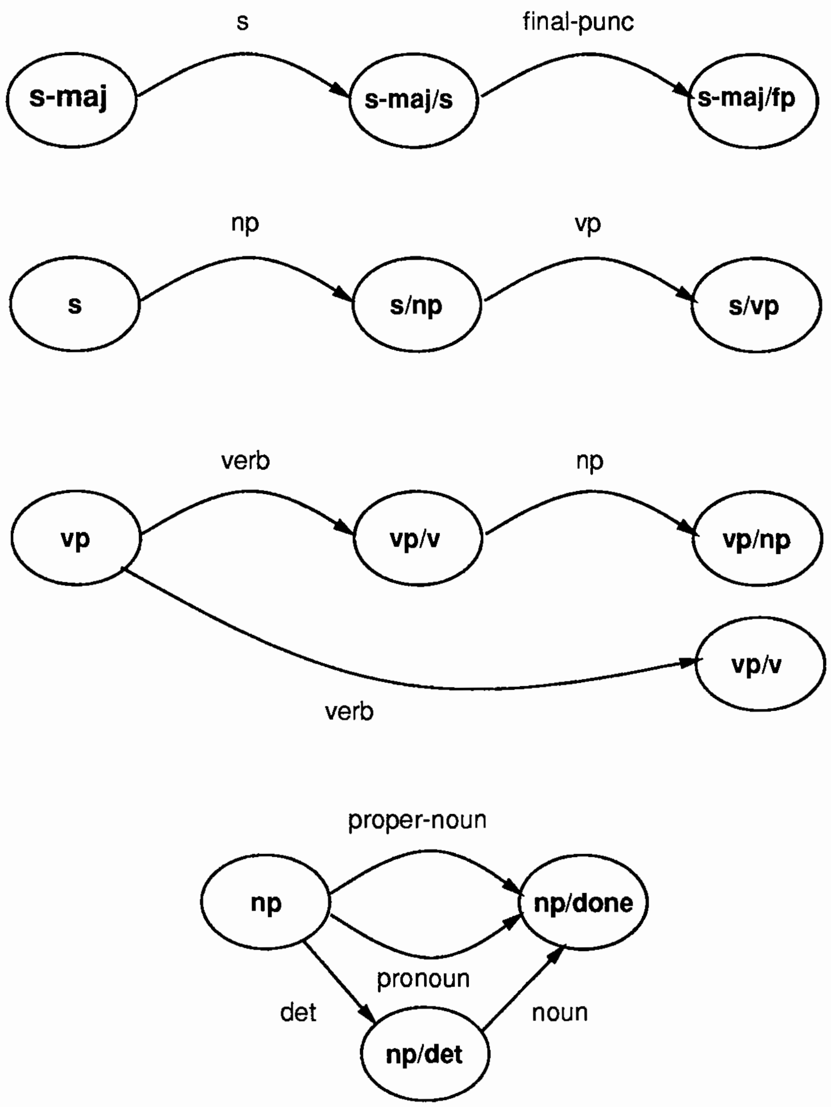


Figure 6.4: A simple ATN, adapted from example in [19].

network parser keeps its state represented on a stack. Entries are pushed onto the stack when traversing subnetworks. *e.g.,*, when trying to parse a **s-maj**, the first action would be to push the network for **s-maj** onto the stack (the first network in Figure 6.4). Looking at the arc-label, the parser will then push the network for **s** (the second network in Figure 6.4), onto the stack. When the **s** is successfully parsed, we pop the stack and move on to look for a period or question mark (final-punc). The state of a transition network is specified by a position in the network and the contents of the stack. An augmented transition network adds some more memory to handle constructions which are not context-free. *e.g.,*, one might add a 'hold' register for handling movement constructions in transformational grammar.

In order to be an all-paths parser, Wimp3's parser duplicates its state whenever it must make a choice. *e.g.,*, if it was parsing a **vp**, it would split into two states, corresponding to the two possible next actions: parse a **verb** and finish, and parse a **verb**, and continue to parse a **np**. Put differently, the parser splits its state in response to syntactic ambiguity.

When the parser splits its state like this, it creates a choice node to be placed in the belief network. It also stores this choice node in the state record. This allows it to connect all of the dependent inferences to the syntactic choice, which allows Wimp's syntactic processing to interact with its semantics and pragmatics. For example, when parsing the sentence "Janet killed the boy with some poison." Wimp3 must consider two possible pp attachments: first, that "with some poison" modifies 'killed', and second, that "with some poison" modifies 'boy'. It creates a choice node, assuming for that the two alternatives are a priori equally likely. Later, when reasoning from the two alternatives, it will find the two semantic alternatives, which are that the poison was the instrument of the killing, or that the poison was the accompaniment of the boy. Reasoning further, it will find that it has no reason to predict that there will be a boy accompanied by some poison (although in some situations, it might). On the other hand, if it assumes that the killing mentioned in the story is a poisoning, the poison will be explained. The result is that Wimp will tend to prefer to attach the pp to the verb, rather than to the direct object, "the boy."

The parser conducts these operations word-at-a-time. For each stack state in its agenda, it carries out every possible action (possibly including splitting the state) until it reaches one which requires reading a new input token. At this point the state will be 'frozen' and pushed back on to the agenda for processing when the next word is read from the input.

Actually, the discussion above is a bit of an oversimplification. Rather than putting choice points into the belief network when encountering alternatives, the parser actually puts these choice points in only after freezing its state. This allows it to collect only the choice points that make a difference to the rest of processing. No choice points need be added for parser alternatives that have no semantic or pragmatic consequences.

The parser generates statements of two types: word-instances and syntactic-relations. These are generated as side-effects of executing parser actions.

6.2 Network Construction

The principles of dynamic network construction have been outlined in Chapter 5. In this section we will discuss network construction as practiced in Wimp3. As mentioned above, the parser generates statements of two types: syntactic-relations, and word-instances, which may be related to each other as alternatives, in the case of syntactic ambiguity. These statements are placed on a queue (which we will, for historical reasons, refer to as a forward-chaining queue). When the parser has completed its computations for one word of the input, the statements on the queue will trigger network construction rules, which may add more statements. After the queue is completely processed, but before processing any new queue

```
(→(inst ?i ?f) :label ?A
   (→←(role-inst ?f ?slot ?sfrm)
        (Exists (x :name ?sfrm)
        (and (inst ?x ?sfrm) :label ?B
             (== '(?slot ?x) ?i) :label ?C)))
   :prob ((?B →?A) ((t t 1))))
```

Figure 6.5: The rule Wimp3 uses for plan-recognition.

entries that have been created, the resulting network will be passed on to the network evaluation component.

There are two features of the network construction process which we will discuss in detail.

- The use of the marker-passer in building the parts of the networks which concern plan-recognition.
- Controlling the network evaluation process.

6.2.1 Wimp3's marker-passer

We have outlined the advantages of marker-passing in Chapter 2. In brief, marker-passing allows us to combine information from many levels of the plan-hierarchy when searching for explanations, effectively combining a limited amount of top-down inference with our generally bottom-up regime. Wimp3 incorporates a marker-passer in its network construction component. In this section we will discuss how this marker-passer is used. We will not discuss the internals of the marker-passer. The marker-passer is the work of Carroll and Charniak, and its functioning is discussed in more detail in [9], and in forthcoming articles.

We give a simplified version of one of our plan-recognition rules in Figure 6.5.[5] What this rule says is the following:

> If we see a new action (inst ?i ?f), and this object could fill ?slot in a frame of type ?sfrm [(role-inst ?f ?slot ?sfrm)], then hypothesize that this action is explained by filling ?slot of an entity of type ?sfrm (this entity is the ?sfrm-explanation of ?i).

Some consideration will reveal why a rule like this could get us in trouble. In fact, with a hierarchy like ours, which contains cycles, this rule could fire forever! *e.g.,*, in our axiomatization, an **airplane-trip** isa **go-**. It also contains a **go-step** slot, which is filled by instances of the frame **go-**: this is the step of going to the airport.

We will use a marker-passer to help us control the search. In essence, what the marker-passer does is to limit us to considering hypotheses which will explain at least two pieces of the input. Intuitively, this may seem too restrictive. It is not as restrictive as it seems, however. We must keep in mind to what extent the problem of text understanding is decomposed before the network construction module sees it: even verbs and their direct objects are different pieces of its input. So, for example, in the sentence "Jack went to the supermarket." **supermarket-shopping** is found as the point of intersection between marks passed from the supermarket and the going action.

[5]These rules will be discussed thoroughly in Chapter 7.

The marker-passer is integrated into the network construction component. When processing the forward-chaining queue, the marker-passer will be invoked on any new instances created. The paths which the marker-passer finds connecting the new instance with earlier ones will be collected. After the paths are collected, they will be passed through a number of path-filters. These filters remove paths which do not correspond to useful inferences. Then the marker-passer will use the paths to fire the plan-recognition inference rules.

6.2.2 Controlling network evaluation

The network evaluation component is the module of Wimp3 which consumes the vast majority (by at least an order of magnitude) of its run-time. Therefore, it behooves us to do whatever we can to speed up its computations. We do by interleaving network evaluation with network expansion, and by committing to beliefs.

We alternate network evaluation and network expansion in order to limit the size of the networks Wimp3 must evaluate. To do this, we interleave processing the forward-chaining queue with evaluating the network. When processing the queue, we chain from all the statements on the queue, putting the resulting statements on a new queue. When Wimp3 has exhausted the initial queue, but before processing the second queue, it invokes the network evaluation component.

Before passing the network on to the evaluation program, we remove any portions which have not changed in the latest round of network-construction. To do so, we use the criterion of d-separation. We keep track of the nodes which have been added to the network, or which have been altered. We remove from the network any nodes which are d-separated from (conditionally independent of) the changed nodes.

The reason we interleave network evaluation with marker-passing is that the marker-passer uses the probabilities of statements to filter the paths it retrieves. These filters are the work of Glenn Carroll and Eugene Charniak [9]. We will not explain them in detail here, just try to give the intuition behind them.

Good paths are, broadly speaking, of two types: there are peak paths, where marks from two pieces of input meet at an explanatory hypothesis, and there are straight-line paths, where a piece of input hooks up with an explanatory hypothesis, which has already been instantiated. For example, when reading the sentence "Jack went to the liquor-store." Wimp3 finds the peak path between **liquor-store** and **go**, through **liquor-shop**. If, after that, it reads "He bought some bourbon." it will find straight-line paths from the already existing liquor-shop hypothesis and **buy-** and **bourbon**.

The intuitions behind the probability filters are as follows. First of all, if an explanatory hypothesis (either the peak of a peak-path, or one end of a straight-line path) is present and not wildly improbable, accept the path. Otherwise, consider its explanatory quality. If its probability is not substantial with respect to the actions to be explained, then reject it. This helps us eliminate, *e.g.*, paths between very common actions. Without a filter like this, any story which contained forms of the words 'go' and 'get' would call into consideration virtually all of the plan hierarchy, because the actions named by those words could play roles in virtually any plan.

The actual test which is used to implement this filter for peak-paths is as follows:

$$pathquality = P(end1 \mid evidence)P(end2 \mid evidence)\frac{P(exp)}{P(end1)P(end2)}$$

If *path-quality* is greater than a cut-off value, the path passes this filter. The second factor in the formula, $\frac{P(exp)}{P(end1)P(end2)}$, represents the explanatory quality of the hypothesis. The first part allows us to bring in even hypotheses which are not terribly likely, if we have enough reasons to believe in their explanands. The formula for the straight-line path filter is similar.

Committing to beliefs limits the cost of network evaluation by allowing us to prune nodes from the networks. By committing to a belief, we mean accepting a node's value categorically. We do this by placing evidence at a node. Because of the properties of the networks, evidence entered at a node may be propagated to that node's children. Then the links from the evidence node to its children may be cut, and if the evidence node has no parents, it may be removed from the diagram.

For the moment, we have adopted a pragmatic, ad hoc approach to committing to beliefs. For each statement in the network, we wait for a set amount of time (for the moment we have adopted the time necessary to process 4 words). After that time, we examine the posterior distribution of that statement. If the probability of some state of the random variable exceeds a threshold (we have adopted .8 for the time being), we will accept that state categorically as being the state of this random variable.

6.3 Network Evaluation

We have said a little bit about network evaluation in Chapter 5. To recap briefly, we use the IDEAL system [106, 107] to provide the network evaluation capability. IDEAL is a toolkit, or test-bench program. It defines a set of data structures for representing and operating on belief networks and influence diagrams. We have a gateway program which maps our belief networks, which are FRAIL data structures, into IDEAL's data structures. Then we have a choice of network evaluation strategies to apply.

IDEAL offers belief network evaluation algorithms of three types: conditioning, simulation and clustering.[6] We refer the reader to our review of the three approaches in Chapter 3.

For use in Wimp3, we have chosen the clustering algorithm of Jensen, et. al. [60, 59], as tailored for our particular problem. We have had to reject simulation approaches, because of features of our distributions. Simulation approaches converge poorly, if at all, when operating on networks which are not strictly positive (see [23, 25, 22, 27]). We have experimented with conditioning algorithms, but with little success. There are two reasons for this. First of all, the performance of conditioning algorithms depends crucially on the size of the cutset used. We have not been able to find a heuristic that gives results comparable to those we have obtained with the clustering algorithms. Secondly, implementation of conditioning algorithms is difficult, especially where the computation of the 'mixing weights' is concerned.

Jensen's algorithm gives us the best results we have found to date. This algorithm is a further development of the algorithm of Lauritzen and Spiegelhalter [72]. The use of 'sepsets' as communication channels between the nodes in the junction tree (the tree of macro-nodes or 'belief universes') substantially speeds processing. Furthermore, there exists a good method for triangulating the moral graph, the step of the algorithm which determines the structure of the junction tree and thus the efficiency of network updating [68].

We have further improved Jensen's algorithm by building the evidence into the junction-tree. In the algorithm as described by Jensen, et al, one first creates a junction tree representation of the input belief network. Then, if one wishes to evaluate the network, one adds evidence to the tree by removing states from the state space, and renormalizes. This

[6]It also offers influence diagram algorithms, but we are not concerned with them here.

approach is well-suited for a diagram which will be used many times.

Our belief networks, however, will be used only once. Because of their throwaway nature, we may build a particular configuration of evidence into the junction tree from the start. This has the beneficial effects of limiting the state space by allowing us to construct a junction-tree which is suited to computing with exactly the evidence we have at hand.

6.4 Summary

In this chapter, we have given a broad outline of the structure of Wimp3, our story understanding program. We have discussed the major components: the parser, the network constructor (with its marker-passer), and the network evaluator. In the next chapter we will expand on the network construction component, discussing the rules it uses. Following that, we will conclude our coverage of the program by giving a detailed example.

Chapter 7

Network Construction in Wimp3

In this chapter we go deeper into the details of Wimp3's belief network construction. We do so by examining the network construction rules it uses. These rules are of four types, broadly speaking, to each of which we will devote a section of the chapter:

1. The 'semantics' rules – rules triggered directly by the parser;
2. Coreference rules – the rules triggered as side effects of the semantics rules. These rules do the 'local pragmatics' [54] processing;
3. Plan-recognition rules, used under control of the marker-passer and
4. Auxiliary rules, which add the effects of assuming various kinds of equality.

In this chapter we will occasionally take the liberty of simplifying rules in order to make the discussion clearer. We will also add some 'syntactic sugar' to the rules, as we have done in Chapter 5.

We do not make any claims about the adequacy of the rules presented here. Our point is merely that it is *possible* to write such rules for language understanding, and that such rules make possible an integrated, probabilistic approach to this problem. Obviously, a small-scale undertaking like our own cannot attempt to produce a program which incorporates the state-of-the-art in the hotly-contested fields of syntax, semantics and pragmatics.

7.1 Semantics rules

The semantics rules, as we classify them here, are the rules which are triggered by the parser assertions – descriptions of words and syntactic relations. We except rules which are designed to draw the portions of the network pertaining to reference relations. Those will be discussed in the following section.

7.1.1 Word senses

The rules for postulating word-senses of open-class words are given in Figure 7.1.[1] There are two such rules. The first rule handles the simple case where the denotation of a word can be described as an instance of a frame. *e.g.,* the word 'supermarket' can simply be translated into an assertion like **(inst foo supermarket)**. The latter case is for words which cannot so simply be mapped into our hierarchy of frames. *e.g.,* 'busdriver' needs to be translated into "an instance of the **person** frame, who operates a bus."

[1]Rules for other referring expressions, like pronouns and proper nouns will be found in the next section.

```
(→ (word-inst ?i ?set ?pos ?word) :label ?A
        (→← (and (not (= ?pos pronoun))
                     (not (= ?pos reflexive))
                     (not (= ?pos propernoun))
                     (not (= ?pos prep))
                     (set-mem (?frame ?PROB) ?set))
              (inst ?i ?frame) :label ?C)
        :prob ((?C →?A) ((t | t ?PROB))))
```

(a) The rule for simple word instances.

```
(→ (word-inst ?i ?set noun ?word) :label ?A
   (→← (set-mem (?frame ?slot ?superframe ?PROB) ?set)
         (and (inst ?i ?frame) :label ?E
              (cond ((inst-frame-compat ?sfrmi ?superframe ?i)
                     (and (inst ?sfrmi ?superframe) :label ?C
                          (== '(?slot ?sfrmi) ?i) :label ?D))
                    (t (Exists (x :name ?superframe)
                               (and (existential ?x ?i)
                                    (inst ?x ?superframe) :label ?C
                                    (== '(?slot ?x) ?i) :label ?D))))))
   :prob ((?C ?D ?E →?A) ((t | t t t ?PROB)) ))
```

(b) The rule for complex word-senses.

Figure 7.1: Network-construction rules for lexical item recognition.

The simple rule, in Figure 7.1(a), may be translated as follows: When you see a word-instance of a particular part-of-speech (**pos**), first check to see that it is not a part-of-speech which Wimp3 treats as a special case (a pronoun, reflexive, proper noun or preposition). If it is not a special case, then check the possible word-senses. The parser will have bound the variable **?set** to the possible word-senses. Every simple entry in the lexicon will have a frame-name, and a probability. For each hypothesis, we will create a statement, and add it to the belief network as a parent of the triggering statement.

For example, when Wimp3 reads "supermarket," the morphological analyzer will first identify the string as the word 'supermarket.' The parser will then look up that word in the lexicon, and finally, assert the statement

(word-inst supermarket11635 ((super-m- 1)) noun supermarket)

supermarket11635 is just a name which the parser has generated for this particular word-token. Because there is no danger of confusion, Wimp3 uses the same constant to refer to the denotation of this word-token.

'Supermarket' is neither a pronoun, a reflexive, a proper noun nor a preposition, so the simple word-instance rule will fire. The invocation of the predicate **set-mem**, serves to bind the variables **?frame** and **?PROB** to **super-m-** and 1, respectively.[2] **set-mem** is an auxiliary predicate, which serves only to translate the list of bindings, **?set** into a set of binding-lists. So, for example, if the word in question were 'went,' **?set** would be bound to **((go- 0.9)(die- 0.1))**, which would translate to the two binding lists **((frame go-) (prob 0.9))** and **((frame die-) (prob 0.1))**.

Continuing with the example, the bindings for **?frame** and **?PROB** are applied to the consequent of the rule, to yield the assertion **(inst supermarket11635 super-m-)**. In accordance with the **:prob** section of this rule, an arc (a 'pform') is added, connecting this statement with the word-inst statement that triggered the rule. This section of the rule also specifies that P(word-inst-statement | inst-statement) = 1. The probability P(word-inst-statement | not(inst-statement)) is dictated by the default associated with the word-inst predicate.

If there were more than one word-sense, the probability of the word-inst node, conditioned on more than one of its parents being true would be 0, as dictated by the combination rule associated with **word-inst**, **oneof-gate**. As before, the probability of the word-inst statement given that none of the parents are true would be dictated by the default.

The second rule, as mentioned above, was created to handle words whose meanings could not be described simply in terms of frames in our isa-hierarchy. For example, we wish the meaning of busdriver to be

> There exists a person (the denotation of this instance of the word 'busdriver') such that there exists a bus that person drives.

More formally:

$$\begin{aligned}&\text{person}(\text{denotation}(\text{word})) \wedge \\ &\qquad \exists(x)(\text{bus}(x) \wedge \text{operator}(x) = \text{denotation}(\text{word}))\end{aligned}$$

This rule handles such cases. As in the simpler rule, the predicate **set-mem** relates the word to its meanings. However, in this case the arguments are somewhat more complex.

[2]**super-m-** is a name we have chosen to describe the concept of supermarket, as opposed to the word 'supermarket.' The trailing hyphen is a convention we use to distinguish concepts/frames from words which refer to them.

In addition to containing a frame and a probability, the lexicon also dictates another frame in which the denotation of the word will fill a slot. For example, when the parser sees an instance of the word 'busdriver,' it will assert the proposition :

(word-inst busdriver12310 ((person- operator bus- 1)) noun)

After finding a type for the denotation of the word – in the case of our example, **person-** – Wimp3 goes on to instantiate the relation dictated by the lexicon. This is done by the portion of the rule beginning with **cond**. **cond** is syntactic sugar for exclusive disjunction. Here the two alternatives are to find a previously existing instance of the superframe, or to create a new one.

The first alternative would arise when processing the story "Bill went to the supermarket. He pointed a gun at the owner." Here the meaning of the word 'owner' is "person that is the **owner-of** a **store-**." Wimp3 would consider that the owner was the owner of the supermarket Bill went to. The second alternative would arise when reading "Jack gave the busdriver a token." Wimp3 would postulate a new bus that the busdriver operates.

Finding a previously-existing instance (the supermarket in the previous example) is done in a straightforward way. Retrieving **(inst-frame-compat ?sfrmi ?superframe ?i)** will find all instances of the **?superframe**, which are eligible. This can be done efficiently because of the way FRAIL indexes instances of frames. For each such instance, Wimp3 will hypothesize that the triggering instance fills the appropriate slot.

The second alternative is somewhat more complicated, since it involves use of a special side-effecting predicate, **Exists**, and the predicate **existential**. FRAIL provides the capability of defining special 'asserter' functions for distinguished predicates. We have taken advantage of this facility for **Exists**. When told to assert a statement with the exists predicate, FRAIL will create a new instance token to bind the existential variable (x in this case), and will use the optional **:name** argument as the prefix of the name. So if this rule was triggered by the assertion **(word-inst busdriver22 ?set noun busdriver)**, and there was no appropriate bus, FRAIL would be told to assert

```
(Exists (x :name bus-)
        (and (existential ?x busdriver22)
             (inst ?x bus-)
             (== '(operator ?x) busdriver22)))
```

This would cause FRAIL to generate a new constant, say bus-29, and the following statements:

(existential bus-29 busdriver22)
(inst bus-29 bus-)
(== '(operator bus-29) busdriver22)

A predication of the form **(existential x y)** is used to represent the fact that x is predicated existentially on y: roughly speaking, that x is an existential variable scoped under y. In response to such predications, we will alter the conditional probability matrices, to ensure that these existential variables are not 'orphaned' – so that they are bound to the variable they are scoped under. This is desirable so that, *e.g.,* we do not continue to believe in a **bus-** created for a possible busdriver if we reject the busdriver. **Existential** statements are used here, and in the plan-recognition rules, where they ensure that the constants we create as names for explanations of objects do not 'outlive' the hypotheses to be explained.

```
(→ (syn-rel ?prep ?verb ?val) :label ?A
   (→← (and (or (= ?prep subject)
                (= ?prep object)
                (= ?prep indir-obj))
            (case ?frame ?prep ?case ?PROB)
            (inst* ?verb ?frame))
       (== '(?case ?verb) ?val) :label ?C)
   :prob ((?C →?A) ((t | t ?PROB) (t | f 0.0))))
```

Figure 7.2: The rule for subjects and objects.

7.1.2 Syntactic relations

Wimp3 has three rules which it uses to translate syntactic relations into relations in the world. The first rule handles the arguments of verbs – the subject, direct and indirect objects. The other two rules are used to translate prepositional phrases. Prepositional phrases which modify verbs are held to specify case-slot fillers as with verb arguments. Prepositional phrases modifying nouns specify relationships between objects.[3]

These rules use information about the cases of lexical items to infer relations in the world. Following Charniak [12], we identify verb cases with slots of the frames represented by the verbs. So, for example, our database records the fact that one may represent the recipient of a giving action either by using the indirect-object relation, or a prepositional phrase headed by 'to':

(case give- indir-obj recipient .5)
(case give- to recipient .5)

e.g., we can say "Jack gave the busdriver a token." or "Jack gave a token to the busdriver." Note that the probability argument indicates that we are assuming the two constructions are equally likely.

Verb arguments

As we've said, the first rule for pp's, which we give as (Figure 7.2), handles assertions about the arguments of verbs. For example, when Wimp3 reads "Jack gave the busdriver a token." the following assertions will be generated:

(syn-rel subject gave12640 jack12449)
(syn-rel object gave12640 token12609)
(syn-rel indir-obj gave12640 busdriver12481)

When we retrieve the additional antecedents of this rule – the first argument to the →← operator – we generate the following consequents:

(== '(agent gave12640) jack12449)
(== '(patient gave12640) token12609)
(== '(recipient gave12640) busdriver12481)

[3]There is certainly room for a better theory of pp syntax in Wimp3. For the moment, we simply assume that a pp modifying a phrase may be represented as a modification of the head component of that phrase.

```
(→ (syn-rel pp ?arg ?prepi) :label ?A
   (→← (and (not (syn-rel det ?arg ?det))
            (word-inst ?prepi ?set prep ?prep)
            (case ?frame ?prep ?case ?PROB)
            (inst* ?arg ?frame))
       (→ (syn-rel prep ?prepi ?val) :label ?B
          (== '(?case ?arg) ?val) :label ?C
       :prob ((?C →?A) ((t | t ?PROB))
              (?C →?B) ((t | t ?PROB))))))
```

Figure 7.3: Semantic rule for verb-modifying pp's.

In doing so, we have used the predicate **inst***, which is the inst relation, closed transitively over **isa**. **inst*** holds when the first argument is an instance of the second (*i.e.,* when Wimp3 knows a proposition of the form **(inst x y)**), or when the first argument is an instance of a frame which **isa***[4] **y**. **inst*** is used to make frames inherit case relations. For example, in order to understand our example, "Jack gave ..." it is necessary for Wimp3 to know that the frame **give-** inherits from **action-** the case relations **(case action- subject agent)** and **(case action- object patient)**.

PP's which modify verbs

The other two syntactic relation rules, given as Figures 7.3 and 7.4 handle prepositional phrases. The former handles prepositional phrases which modify verbs, and the latter ones which modify nouns.

The rule in Figure 7.3 works almost exactly like the rule for verb arguments. The differences arise because prepositional phrases are represented by two syntactic-relation statements, rather than a single one. We have a statement which indicates what word-inst is modified and one which indicates what the modifier is. If we were to change our previous example to read "Jack gave a token to the busdriver." the prepositional phrase would be represented as follows:

(syn-rel pp gave12699 to12806)
(syn-rel prep to12806 busdriver12900)

Because the parser generates the former statement first, and then the latter, we have written this rule so that it creates a new forward-chaining rule. This rule 'waits' for the addition of a statement specifying the modifier, then adds corresponding case-relation hypotheses. In our example, after reading the first statement, Wimp3 would generate a rule which waited for the noun following 'to,' and then hypothesize that this noun names the recipient of the giving event.

PP's which modify nouns

The final prepositional phrase rule (Figure 7.4), for pp's modifying verb-phrases, is somewhat more complex. It is, in a way, a combination of a word-sense rule, and a case rule. What it does is find a relation which may be denoted by the preposition, and then make the modifier and modified np's the arguments of this relation.

[4]The transitive closure of the isa relation.

```
(→ (syn-rel pp ?arg ?prepi) :label ?A
   (→← (and (syn-rel det ?arg ?det)
            (word-inst ?prepi ?set prep ?prep)
            (set-mem (?frame ?N) ?set))
       (→(syn-rel prep ?prepi ?val) :label ?B
          (and (inst ?prepi ?frame) :label ?D
               (== '(a1 ?prepi) ?arg) :label ?E
               (== '(a2 ?prepi) ?val) :label ?F
               (has-referent ?prepi) :label ?G)
       :prob ((?E ?D ?G →?A)
              ((t | t t t .5) (t | t t f .01)
               (t | t f t 0.0) (t | f t t 0.0) (t | f f t 0.0))
              (?F →?B) ((t | t .5))))))
```

Figure 7.4: Semantic rule for noun-modifying pp's.

For example, when Wimp3 reads the sentence "Jack found some milk on the shelf." it will trigger this second rule, because 'milk' has the determiner 'some.' Note that the presence of this determiner keeps it from firing the verb-modifier rule for the relation **(syn-rel pp milk13035 on13089)**.

When the parser encountered the preposition **on13089**, it consulted the lexicon, and found the possible word-sense **position-rel-**, yielding the following statement: **(word-inst on13089 ((position-rel- 1)) prep on)**. In response, it generates a rule which waits for the assertion of the modifier – in this case **(syn-rel prep on13089 shelf13013)** – and then creates an instance of the appropriate relation frame:[5]

(inst on13089 position-rel-)
(== '(a1 on13089) milk13035)
(== '(a2 on13089) shelf13013)

7.2 Reference rules

We need reference rules for a number of different kinds of referring expressions. First of all, there are the conventional referring expressions: pronouns and definite np's. However, we also need to be able to handle other kinds of reference. We need to be able to identify the named characters – *e.g.,* to recognize that the second use of 'Fred' in "Fred robbed the liquor-store. Fred pointed a gun at the owner." refers to the same person as the first. We want Wimp3 to infer that 'the owner' cannot refer to the same person as 'Fred' in this context. We also need to be able to handle verb reference. The word 'went' in "Jack got on the bus. He went to the supermarket." must be understood as referring to the **bus-trip-** we have inferred from Jack's getting on the bus. In this section we will discuss the rules for these situations, and some supporting parts of the belief network.[6]

Pronoun reference

[5]The **has-referent** statement is discussed in the following section.

[6]There are also rules for handling reflexive reference, but since they aren't used in any of our test examples, we do not discuss them here.

```
(→ (word-inst ?i ?set pronoun ?word) :label ?A
   (→← (and (set-mem (?frame ?n) ?set)
            (definite-pronoun ?word))
       (and (inst ?i ?frame) :label ?B
            (has-referent ?i) :label ?C))
   :prob ((?C ?B → ?A) ((t | t t 1) (t | f t :p)
            (t | t f 0.0) (t | f f 0.0))) )
```

Figure 7.5: The rule for pronoun reference.

```
(→ (has-referent ?i) :label ?A
   (cond ((and (inst ?i ?f) :label ?B
               (inst-inst-compat ?i2 ?f ?i)
               (not (reflexive-coref ?i ?i2)))
          (and (inst ?i2 ?f) :label ?C
               (same-type ?i ?i2 ?f) :label ?SAME-TYPE
               (== '?i ?i2) :label ?D))
         (t #L(add-evidence ?A :false)))
   :prob ((?B ?C → ?SAME-TYPE)
              ((n | t t 1) (im | t f 1) (im | f t 1) (ir | f f 1))
          (?SAME-TYPE → ?D) ((t | n =inst-prior) (t | im 0.0))
          (?D → ?A) ((t | t 1))))
```

Figure 7.6: The has-referent rule. An auxiliary rule used in reference relations.

We start our coverage of the reference rules with the simplest rule: pronoun reference. This is given as Figure 7.5. This rule does two things: as with the word-inst rule, discussed in Section 7.1.1, it creates a type predication about the denotation of the pronoun. *e.g.,*, if an instance of the word 'he' was read – let's call it he234 – this rule would generate **(inst he234 boy-)**.

Unlike the word-inst rule, this rule also asserts a **has-referent** statement, which indicates that Wimp3 should consider the possibility that **he234** is the same as some other boy already encountered. The probabilities given encode Wimp3's preference for finding a referent for a pronoun. Literally, they say that the probability of using a pronoun to describe a previously-introduced person or object, given that there is such a person to be introduced, is many orders of magnitude higher than the probability of using a pronoun to introduce an entity into the universe of discourse, given that there is such an entity to be introduced.

Has-referent

As one might imagine, the conditions for a has-referent predication being true are straightforward. If this entity is equal to – *i.e.,* refers to – a previously-known entity, the predication is true. Otherwise it is false. The has-referent rule, given as Figure 7.6, directs Wimp3 to search for possible referents for **?i**, and enforces constraints on the reference relations.

The **has-referent** rule is a cond, with two branches. The second, and simpler branch,

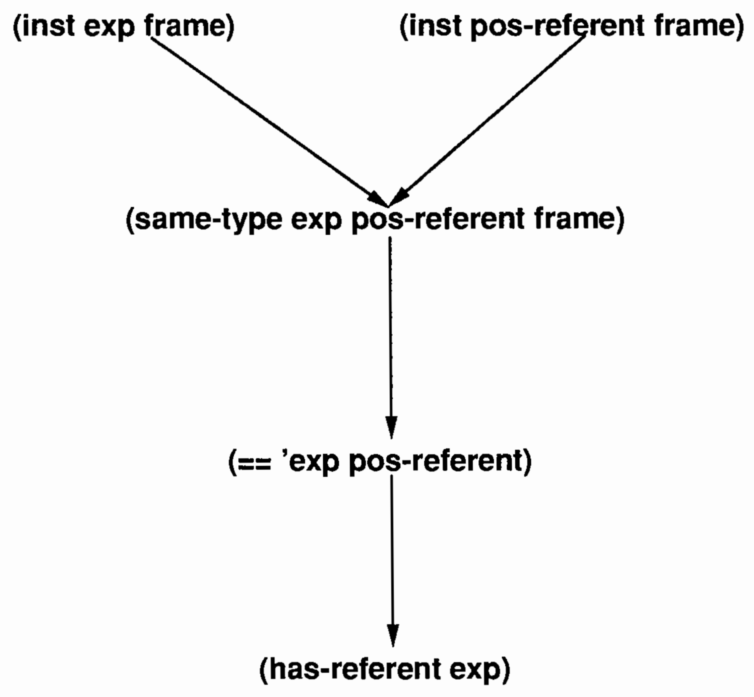


Figure 7.7: The portion of the network constructed to represent a coreference hypothesis.

```
(→ (word-inst ?i ?set propernoun ?word) :label ?A
   (→← (set-mem (?frame ?n) ?set)
        (and (inst ?i ?frame) :label ?B
             (has-referent ?i) :label ?C))
   :prob ((?C ?B → ?A) ((t | t t .2) (t | f t .02)
                         (t | t f 0) (t | f f 0))))
```

Figure 7.8: The rule for proper noun reference.

states that if no eligible referent can be found, the has-referent statement is false. The first branch collects possible referents, and constructs a section of the network representing the reference hypothesis. Because the rule is somewhat difficult to decipher, we give a pictorial representation in Figure 7.7.

The **same-type** nodes in this network are intermediate variables which we have added in order to make the nets easier to evaluate. They are not representationally necessary, and could be deleted. They are three-valued nodes, whose sample space is {necessary (n), impossible (im), irrelevant (ir)}. The two parents of a **same-type** node are inst predications of the form **(inst** x **type)** and **(inst** y **type)**. The **same-type** node is necessary if both parents are true, impossible if one is true and the other false, and otherwise irrelevant.

Proper noun reference

The proper noun reference rule (Figure 7.8), like the pronoun reference rule, relies on the **has-referent** predicate. In fact, the only real difference is in the probabilities. These reflect the fact that proper nouns are more commonly used to introduce entities into a universe of discourse than are pronouns.

```
(→ (inst ?i ?f) :label ?A
   (→← (and (current ?i)
              (syn-rel det ?i the) :label ?B)
        (cond ((and (inst-inst-compat ?i2 ?f ?i)
                    (not (reflexive-coref ?i ?i2))
                    (not (== '?i2 ?i3)))
               (and (inst ?i2 ?f) :label ?C
                    (same-type ?i ?i2 ?f) :label ?SAME-TYPE
                    (check-coref ?i ?i2)
                    (== '?i ?i2) :label ?D))
              ((inst! ?i ?f)
               (→ (== '(?slot ?w) ?i) :label ?Q
                  (→← (syn-rel det ?w ?any)
               :prob ((?Q → ?B)
                      ((t | t .1) (t | f .001))))))))
   :prob ((?A ?C → ?SAME-TYPE) ((n | t t 1) (im | t f 1)
                                (im | f t 1) (ir | f f 1))
          (?SAME-TYPE → ?D) ((t | n =inst-prior) (t | im 0.0))
          (?D → ?B) ((t | t .4) (t | f .001))))
```

Figure 7.9: The rule for definite noun phrase reference.

Definite noun phrases

The rule for definite noun phrases, given as Figure 7.9, allows for reference to two kinds of discourse entities. These two kinds of reference are handled by the two branches of the cond clause in the rule. The first is the conventional kind of reference, where a np is used to refer to an entity which has been explicitly introduced into the universe of discourse. An example of this kind of reference is seen in the use of 'the boy with the sweater' in the story "Jack wore a suit. Bill wore a sweater. Janet killed the boy in the sweater." The second kind of reference is reference to a slot of another entity in the universe of discourse: *e.g.*, "Jack went into the kitchen. The stove was lit." 'The stove' refers to a part of the kitchen which has been mentioned.

Verb reference

Verb reference works in a way very similar to the simple case of definite np reference. The complexity of the rule (Figure 7.10) comes from preconditions added to prevent Wimp3 from considering impossible reference relations. For example, the precondition **(current ?i)** is to ensure that Wimp3 doesn't allow reference forward in time: to make sure that when Wimp3 reads "Jack got on the bus. He went to the supermarket." Wimp3 makes 'went' refer to the bus-trip, rather than *vice versa*.

7.3 Plan-recognition rules

We have discussed the way the marker-passer directs plan-recognition in Chapter 6. Here we will discuss the rules used for plan-recognition (see Figures 7.11 and 7.12), and elaborate a little on our earlier account.

```
(→ (inst ?i ?f) :label ?A
   (→← (and (current ?i) ; no forward reference
            (not (syn-rel det ?i ?any)) ;make sure this is a verb
            (word-inst ?i ?set ?pos ?le)
            (inst-inst-compat ?i2 ?f ?i))
       (and (inst ?i2 ?f) :label ?C
            (same-type ?i ?i2 ?f) :label ?SAME-TYPE
            (== '?i ?i2) :label ?D))
   :prob ((?A ?C → ?SAME-TYPE) ((n | t t 1) (im | t f 1) (im | f t 1)
                                (ir | f f 1))
          (?SAME-TYPE → ?D) ((t | n =inst-prior) (t | im 0.0))))
```

Figure 7.10: The rule for verb reference.

```
(→ (inst ?i ?f) :label ?A
   (→← (and (informative-role-inst* ?f ?slot ?sfrm) ; 1
            (not (inst-frame-compat ?j ?sfrm ?i)) ; 2
            (or (not (isa* ?f physob)) ; 3
                (not (isa* ?sfrm event-))))
       (Exists (x :name ?sfrm)
               (and (existential ?x ?i)
                    (inst ?x ?sfrm) :label ?B
                    (== '(?slot ?x) ?i) :label ?C)))
   :prob ((?B → ?A) ((t| t 1))))
```

Figure 7.11: The plan-recognition rule which introduces new hypotheses.

```
(→ (inst ?i ?f)
   (→← (and (informative-role-inst* ?f ?slot ?sfrm)
            (inst-frame-compat ?sfrmi ?sfrm ?i)
            (not-redundant ?sfrmi ?slot))
       (and (inst ?sfrmi ?sfrm)
            (== '(?slot ?sfrmi) ?i))))
```

Figure 7.12: The plan-recognition rule which recognizes new parts of previously-recognized plans.

After every round of forward-chaining, and before network evaluation, there is a round of marker-passing. The marker-passer propagates marks through the plan network. It collects the paths found, and then runs them through a set of filters. Any paths which successfully pass the filters are then used to fire the network construction rules in this section. The statements in the path are used to bind the variables **?i**, **?f**, **?slot** and **?sfrm** (see below), *i.e.,* to specify the instance to be explained and how that instance is to be explained (as filling **?slot** of **?sfrm**).

7.3.1 Introducing new plan hypotheses

There are three preconditions to this rule (Figure 7.11), added by the $\rightarrow\leftarrow$ operator. These are (we have marked the corresponding lines in the rule):

1. An instance of the appropriate type must be able to fill this slot;
2. There does not already exist an instance of **?sfrm** that is eligible to explain **?i** and
3. The explanand and explanation are of the appropriate types.

This last precondition is to enforce our restriction, discussed earlier,[7] that plans only be *postulated* to explain actions. Note that this doesn't mean that plans do not *explain* objects – just that objects don't introduce plans into Wimp3's consideration.

Let us consider how this rule operates. When reading the sentence "Jack went to the supermarket." Wimp3 finds a path from **supermarket-** to **go-** that passes through the plan **supermarket-shop-**. This plan passes the path-filters, and then is used to fire this rule.

Wimp3 first attempts to fire this rule from the start of the path:**(inst supermarket27 supermarket-)**. The bindings used are:

$$\begin{gathered}\textbf{?i} := \textbf{supermarket27}\\ \textbf{?f} := \textbf{supermarket-}\\ \textbf{?slot} := \textbf{store-of}\\ \textbf{?sfrm} := \textbf{supermarket-shop-}\end{gathered}$$

These bindings will satisfy the first two preconditions of the rule, but not the third. A **supermarket-** can fill the **store-of** slot of **supermarket-shop-**, and there is no previously-existing **supermarket-shop-** that could explain this **supermarket-**. However, were Wimp3 to hypothesize a new **supermarket-shop-** to explain **supermarket27**, it would be explaining an object by a plan, violating precondition 3. So the rule fails to fire.

The marker-passer then reverses the path and tries the rule again, this time starting from 'went.' The following bindings are used:

$$\begin{gathered}\textbf{?i} := \textbf{went22}\\ \textbf{?f} := \textbf{go-}\\ \textbf{?slot} := \textbf{go-step}\\ \textbf{?sfrm} := \textbf{supermarket-shop-}\end{gathered}$$

This time the preconditions *are* satisfied. Wimp3 postulates a new **supermarket-shop-**, **superming29**. The statement introducing this plan, **(inst superming29 supermarket-shop-)** is given an existential statement, linking it to **went22**, to record the fact that **superming29 = supermarket-shop-explanation(went22)**.

[7]See p. 47.

7.3.2 Linking up with previously-existing explanations

The second plan-recognition rule (Figure 7.12) makes it possible for Wimp3 to recognize new pieces of input as playing roles in previously-hypothesized plans. To continue our example, if Wimp3 read "He found some milk on the shelf." after "Jack went to the supermarket." we would like it to recognize 'found' as filling the **loc-step** slot of the already-postulated supermarket-shop (**superming29**). Let us consider how this is done.

When Wimp reads 'found,' it will generate a word-sense hypothesis like (**inst found87 locate-**). When the marker-passer passes marks from this statement, it will find a 'straight-line' path[8] from **locate-** to **store-shop-** to **supermarket-shop-**. A **locate-** action can fill the **loc-step** slot of a **store-shop** (the step of finding the object to be purchased), and a **supermarket-shop-** isa **store-shop**.

The marker-passer will try to fire the second plan-recognition rule with the bindings:

?i := **found87**
?f := **locate-**
?slot := **loc-step**
?sfrm := **supermarket-shop-**

Looking for an instance of **store-shop** whose **loc-step** could be filled by **found87**, Wimp3 will find **superming29**, the hypothesis it created to explain the previous sentence.

7.4 Auxiliary rules

The reader will note that the marker-passer rules do not fill out the networks. They just suggest hypotheses. It is left to the auxiliary rules, which we introduce in this section, to flesh out these hypotheses. Because of the way our knowledge representation scheme works – slot-filler relations represented by equations – these rules will primarily be concerned with equality.

There are three types of these rules. The first are those which are concerned with fleshing-out the explanatory hypotheses. The second are those which add constraints to the network based on the structure of the frames/plans. The final sort of rules are those which close over equality relationships.

7.4.1 Fleshing-out explanations

The two rules given as Figures 7.13 and 7.14 build the portions of the network around the statements introduced by the marker-passer rules discussed in the preceding section. The rule given in Figure 7.13 corresponds to the rule in Figure 7.11: it builds the portion of the network around a new hypothesis. The rule given in Figure 7.14 corresponds to Figure 7.12: it builds the portion of the network necessary to incorporate new input in an old hypothesis.

The exact details of the functioning of these rules are of little interest. What is important is to understand the portion of the networks built by these rules. We give an illustration as Figure 7.15. What this shows is that the reason we can believe that **?filler** fills **?slot** of **?frame** is that (**slot frame**) and **filler** are the same type of object. We have introduced (**slot-inst slot filler frame**) as syntactic sugar for (**inst** (**slot filler**) **frame**).

The difference between the two rules lies in the probability distributions they specify. The network supporting a new hypothesis (see Figure 7.13) essentially specifies that if the

[8]See page 81.

```
(→ (== '(?slot ?frm) ?filler) :label ?A
   (→← (and (existential ?frm ?filler)
            (role-inst ?f ?slot ?sf)
            (inst* ?frm ?sf) :label ?B)
       (and (slot-inst ?slot ?frm ?f) :label ?C
            (same-type '(?slot ?frm) ?filler ?f) :label ?SAME-TYPE
            (inst ?filler ?f) :label ?D))
   :prob ((?B → ?C) ((t | t 1))
          (?C ?D → ?SAME-TYPE) ((n | t t 1) (im | t f 1)
                                (im | f t 1)(ir | f f 1))
          (?SAME-TYPE → ?A) ((t | n 1) (t | im 0.0))))
```

Figure 7.13: The rule which builds the part of the network supporting a new hypothesis.

```
(→ (== '(?slot ?frm) ?filler) :label ?A
   (→←(and (not (existential ?frm ?filler))
           (role-inst ?f ?slot ?sf))
      (→ (inst ?frm ?o)
         (→← (inst* ?frm ?sf) :label ?B
             (and (slot-inst ?slot ?frm ?f) :label ?C
                  (same-type '(?slot ?frm) ?filler ?f) :label ?SAME-TYPE
                  (inst ?filler ?f) :label ?D))
         :prob ((?B → ?C) ((t | t 1))
                (?C?D → ?SAME-TYPE)
                   ((n | t t 1) (im | t f 1) (im | f t 1) (ir | f f 1))
                (?SAME-TYPE → ?A)
                   ((t | n =inst-prior) (t | im 0.0))))))
```

Figure 7.14: The rule which builds the part of the network supporting the addition of new input to an old explanation.

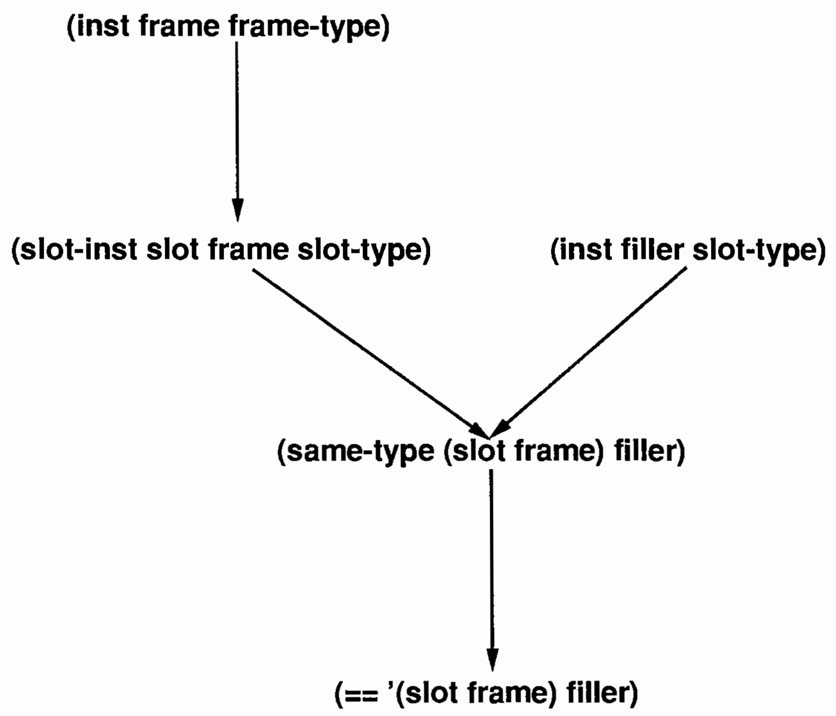

(inst frame frame-type)
(slot-inst slot frame slot-type)
(inst filler slot-type)
(same-type (slot frame) filler)
(== '(slot frame) filler)

Figure 7.15: The network fragment which supports an explanatory hypothesis.

explanation inst node – drawn as **(inst frame frame-type)** in Figure 7.15 – is true, then the rest of the hypothesis will be true. This is because this instance is an explanation of **filler**. It makes no sense for this node to be true, and the explanation to be false.

This is not true in the case where we are linking new input into an old action. Let us return to the example given in the previous section, but change it somewhat, to read "Jack went to the supermarket. He found a revolver." Now it is clear that the finding mentioned in the second sentence is *not* the **loc-step** of Jack's supermarket-shopping.

7.4.2 Rules reflecting frame structure

The rules discussed in this section, which we call 'diamond-rules,' allow Wimp3 to use its frame knowledge to make predictions from its plan-recognition hypotheses. If such predictions are confirmed, they will raise the probability of an hypothesis. Conversely, if

```
(→ (== (?slot1 ?i1) ?i2) :label ?A
        (→ (== (?slot2 ?sf) ?i1) :label ?B
           (→← (and (diamond= ?fr ?slot2 ?slot3 ?slot1) :label ?D
                     (inst* ?sf ?fr) :label ?C)
                (and (== (?slot3 ?sf) ?i2) :label ?E
                     (diamond-center ?i1 ?i2 ?sf ?slot1) :Label ?F))
           :prob ((?B ?E → ?F) ((n | t t 1) (im | t f 1) (ir | f f 1)
                                (ir | f t 1))
                  (?F → ?A) ((t | n 1) (t | im 0.0)))))
```

Figure 7.16: A rule Wimp3 uses to predict slot-filler relations from its frame knowledge.

```
(defframe gshop- ?gs
   isa achieve
   slots (store-of (gstore-))
         (bought (inanimate-))
   acts (go-stp (go- (agent (agent ?gs)) (destination (store-of ?gs)))) *
        (pay-stp (pay- (agent (agent ?gs)) (paid-for (bought ?gs)))))

(defframe store-shop- ?ss
   isa gshop-
   slots (store-of (store-))
   acts (loc-stp (locate- (agent (agent ?ss)) (patient (bought ?ss)))))

(defframe supermarket-shop- ?sm
   isa store-shop-
   slots (bought (food-))
         (store-of (super-m)))
```

Figure 7.17: Some sample frames from Wimp3's knowledge-base.

they are discredited, it will tend to lower the probability. We call these rules diamond-rules because of the topology of the subnetworks they create (see Figure 7.18).

Wimp3 is able to predict further slot-filler relations based on the ones it has already observed. Consider the fragment of the isa hierarchy given in Figure 7.17. Among other things, this records the fact that the agent of the **go-step** of a **gshop-** frame[9] is the same as the **agent** of the **gshop-** plan (see the starred line). This allows Wimp3 to infer, reading the story "Jack went to the supermarket." that Jack is the agent of the **supermarket-shop-** plan. Later, reading "He found some milk on the shelf." when it hypothesizes that 'found' is the **loc-step** of this plan, it will predict that Jack is the agent of the **loc-step**. This prediction is borne out when Wimp3's network evaluation shows that 'he' refers to 'Jack' lending further credence to the **supermarket-shop-** interpretation.

The diamond rule, given as Figure 7.16 is somewhat difficult to read because we have made free use of additional predicates. **diamond=** is a predicate which makes frame knowledge available in a convenient way. **(diamond= ?frame ?slot1 ?slot2 ?slot3)** is true if

$$\forall(x)\{(\text{inst x frame}) \supset \\ (\text{slot2 x}) == (\text{slot3 (slot1 x)}) \}$$

An example may make this more clear:

(diamond= store-shop- loc-step purchased patient)

means that the **patient** of the **loc-step** of a **store-shop** is the same as the purchased of the **store-shop**. See the frames in Figure 7.17. So after reading "Jack went to the supermarket. He found some milk on the shelf." Wimp3 would predict that the milk found in sentence 2 would be the object purchased.

Let us consider how this rule works in the supermarket-shopping example. Initially, from information supplied by the parser, Wimp3 learns that the agent of **went22** is Jack:

[9]**gshop** means 'generalized-shopping:' a **store-shop** isa **gshop**, but so is a **restaurant-meal** plan.

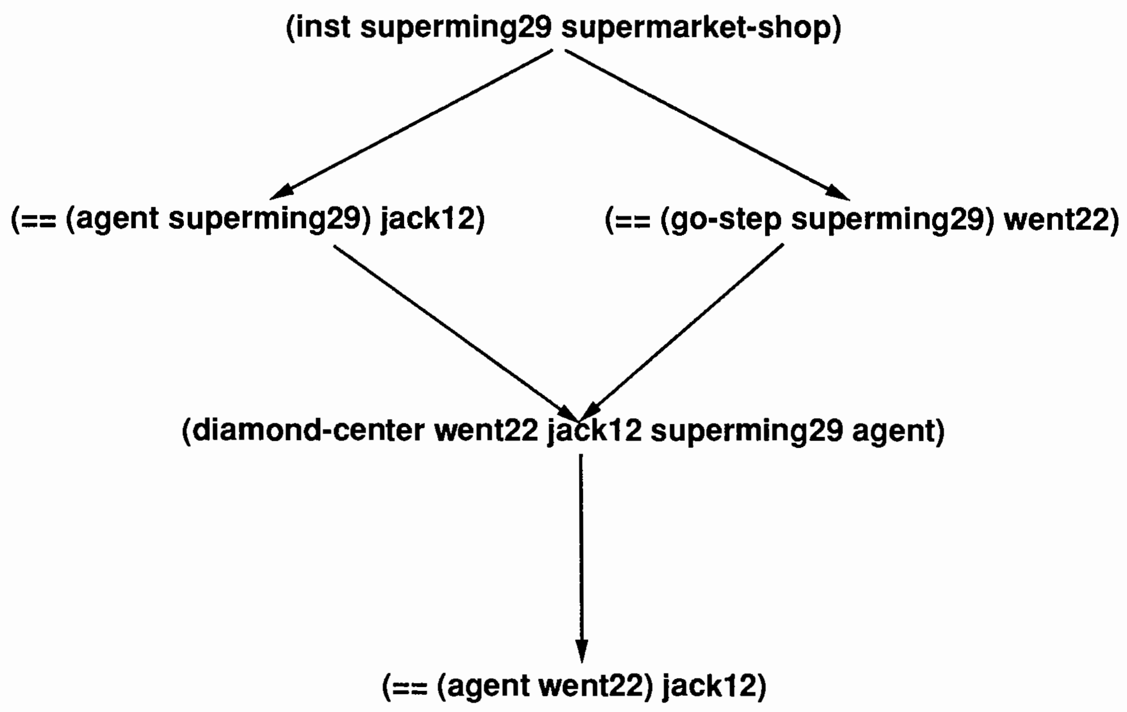

(inst superming29 supermarket-shop)
(== (agent superming29) jack12)
(== (go-step superming29) went22)
(diamond-center went22 jack12 superming29 agent)
(== (agent went22) jack12)

Figure 7.18: An example of the use of the diamond rule.

(== (agent went22) jack12). This causes the rule in Figure 7.16 to fire, creating a forward-chaining rule which waits for assertions about **went22** – a rule whose precondition is (== (?slot2 ?sf) went22).

Later on, Wimp3 considers explaining **went22** as part of a **supermarket-shop-** plan. It asserts (== (go-step superming29) went22). This triggers the rule added earlier, because Wimp3 knows the following facts:

(diamond= gshop go-step agent agent)	*The agent of the go-step of a gshop is also the agent of the gshop.*
(inst* superming29 gshop)	*superming29 is a supermarket-shop which isa* gshop.*

This causes Wimp3 to add the statements

(== (agent superming29) jack12)
(diamond-center went22 jack12 superming29 agent)

The resulting subnetwork is illustrated in Figure 7.18.

Diamond-center predications, like **same-type**, have been introduced to make the updating of the belief networks more efficient, and to make the belief networks easier to understand. Equality statements often have many parents. We wanted to clearly separate out the different reasons for believing an equality statement. Introducing **same-type** and **diamond-center** predications makes it clear which pairs of antecedents combine into a particular reason for believing a statement. Furthermore, they limit the connectivity of the network. *e.g.,* if an equality statement has two **diamond-center** pforms and a **same-type** pform, it will have only 3 parents in the formulation with these intervening nodes, rather than 6 in the naive formulation. This translates into savings in the network evaluation algorithms, particularly conditioning algorithms, which are more sensitive to the connectivity of the input belief network than to its size.

```
(→ (== ?np ?onp) :label ?A
   (→← (atomic ?np)
         (→ (== (?slof ?frm) ?np) :label ?B
            (→← (and (ok-sub-order ?np ?frm)
                     (acceptable-slot-filler ?frm ?onp))
                 (== (?slof ?frm) ?onp) :label ?C)
            :prob ((?A ?C → ?B) ((t | t t 1) (t | t f 0.0))))))

(→ (== ?np ?onp) :label ?A
   (→← (atomic ?np)
         (→ (== (?slof ?frm) ?np) :label ?B
            (→← (and (== ?frm ?ofrm)
                     (acceptable-slot-filler ?frm ?onp))
                 (== (?slof ?frm) ?onp) :label ?C)
            :prob ((?A ?C → ?B) ((t | t t 1) (t | t f 0.0))))))


(→ (== ?new ?old) :label ?A
   (→← (atomic ?new)
         (→ (== (?slof ?new) ?val) :label ?B
            (→← (and (ok-sub-order ?new ?val)
                     (acceptable-slot-filler ?old ?val))
                 (== (?slof ?old) ?val) :label ?C)
            :prob ((?A ?C → ?B)
                   ((t | t t 1) (t | t f 0.0))))))
```

Figure 7.19: Rules for managing equality statements.

Note that we do not allow Wimp3 to make all available predictions. Rather, the rules are triggered when Wimp3 has some reason to believe an equation. At this point, Wimp3 looks to find which hypotheses would support this equality.

We give here only the simplest of two rules that Wimp3 uses for the purpose of applying plan knowledge.

7.4.3 Managing equalities

Wimp3 is unable to cope with a full equality relation, for efficiency reasons. Fortunately, the nature of our language understanding problem is such that a better-name relation [79] is a reasonable approximation. Our better-name relation is structured so that less-deeply-nested expressions are better-names than more deeply-nested ones, and at the atomic level, atoms introduced earlier are better names than atoms introduced later. This accords well with our intuitions about how reference works.

The three rules we will discuss in this section (see Figure 7.19) manage this better-name relation by drawing out some of its implications. These rules are important because they collect evidence for and against various equality hypotheses.

The first two rules in this figure ensure copying of slot-assignments. They take care of the case where we know a better-name for a symbol which fills a slot of a frame. To continue with our running example, after Wimp3 has read "Jack went to the supermarket. He found..." it will consider the two hypotheses:

(== (agent found87) he234)
(== he234 jack12)

These rules cause Wimp3 to conclude

(== (agent found87) jack12)

The third rule causes Wimp3 to copy the slot-filler statements it knows about a worse-name to its better-name. For example, if we knew that **he234**'s **blood-type** was O-positive, we generate a corresponding statement about **jack12**: **(== (blood-type he234) O-positive)** would yield **(== (blood-type jack12) O-positive)**

7.5 Summary

In this Chapter we have given a summary of the network-construction rules used by Wimp3. We have divided these rules into semantics rules, reference rules, plan-recognition rules and auxiliary rules, and given simple examples of their functioning in isolation. In the following chapter we will give a detailed example of Wimp3's processing of a story, and show how these rules work together.

Chapter 8

A Detailed example

In this chapter we illustrate the operation of Wimp3 by giving a detailed account of its processing of the story "Jack gave the busdriver a token. He got off at the supermarket." The desired conclusion, which Wimp3 reaches, is to assume that Jack has a supermarket-shopping plan, and that he is taking the bus to get to the supermarket to go shopping. In the following pages we will show how Wimp3 discovers this.

8.1 Jack

The story begins with the introduction of the character, Jack. Wimp3's parser reads the string "Jack", begins processing a s-maj, and asserts the following statement:

(word-inst jack717 ((jack- 1)) propernoun jack)

The network constructor component takes this statement and adds it to its database. This causes the propernoun rule, which we discussed in the last chapter (see page 93), to fire. This adds two statements to the database:

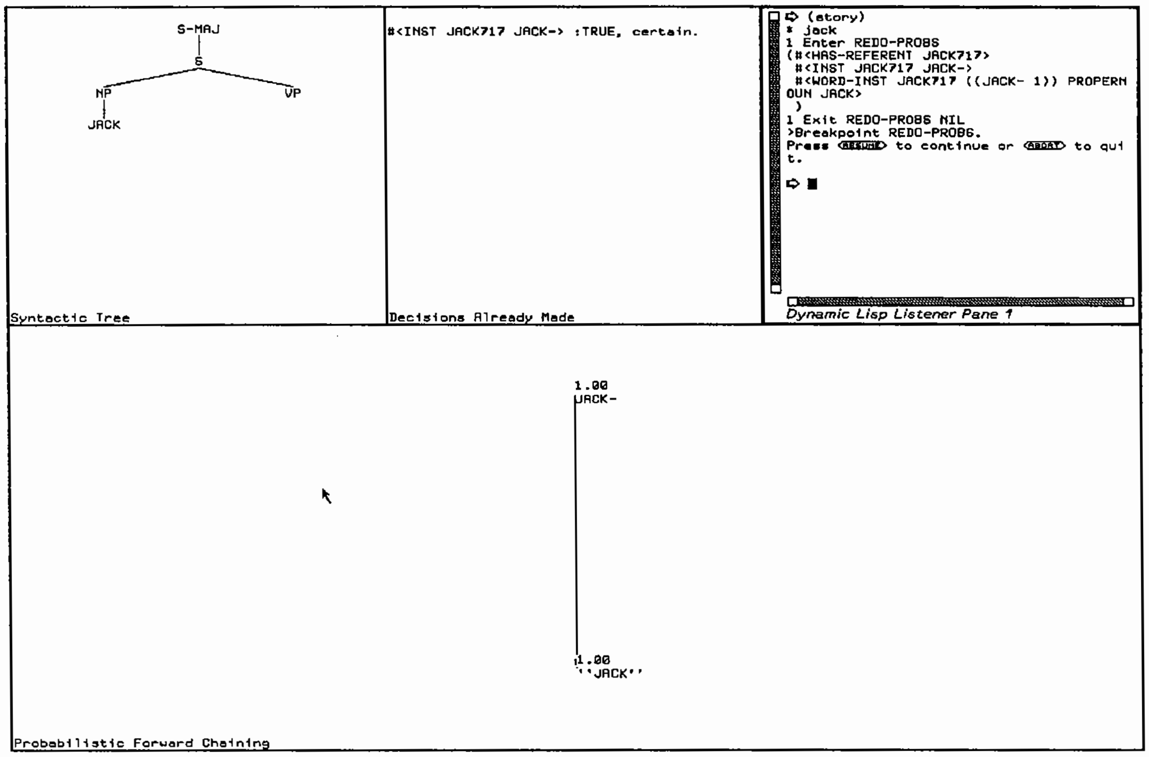
(inst jack717 jack-)
(has-referent jack717)

Note that we have introduced frames for every proper name. This is simply done to make processing of proper nouns easier. We could have defined "Jack" as "A person whose name is Jack." but it would have been a bit more cumbersome. Instead we have a class of persons, **jack**-s, who are male and whose name is "Jack."

The **has-referent** statement triggers further chaining, attempting to retrieve a previously-known boy that "Jack" might be referring to. Wimp3 does not know of one, so this fails.

The marker-passer does not generate any statements either. First of all, we do not pass marks from instances of the **person-** frame. We do not do this because marks from **person-** do not help us choose a plan explanation: people can be the agents of any plan that Wimp3 knows about. Not only that, but even if we did pass marks from **jack717**, there are no statements whose marks these could meet.

The set of affected statements is now collected and passed to the network evaluation component. Before evaluating the network, the gateway component (see page 82) translates it into a set of IDEAL data structures. The gateway program recognizes that the **has-referent** statement is an evidence statement with no parents. The effect of the evidence at this node can be propagated into the network, and then it can be removed.



```
#<INST JACK717 JACK-> :TRUE, certain.
```

```
⇨ (story)
* jack
1 Enter REDO-PROBS
(#<HAS-REFERENT JACK717>
 #<INST JACK717 JACK->
 #<WORD-INST JACK717 ((JACK- 1)) PROPERN
OUN JACK>
 )
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

⇨ ■
```

Figure 8.1: The situation after Wimp3 reads "Jack".

Figure 8.1 displays the situation after network evaluation. This screen dump, and the others which follow, have been taken from the symbolics version of Wimp3. There are four windows onto Wimp3. These are:

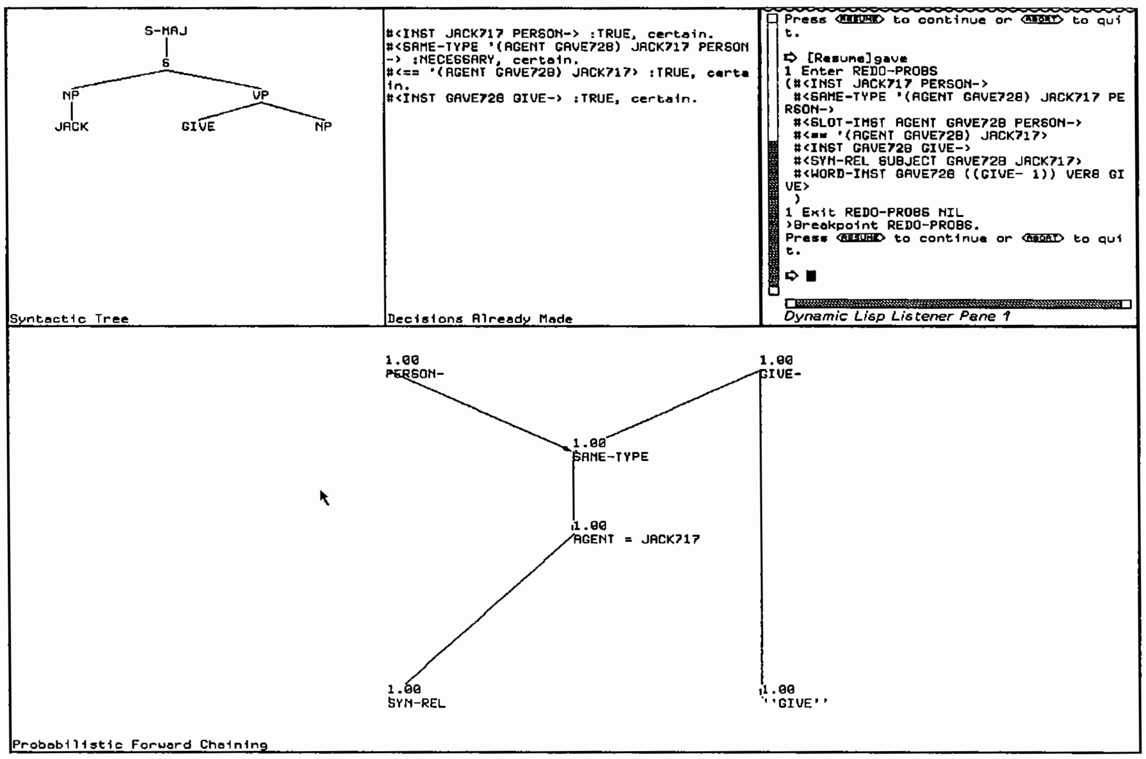
1. Syntax window – this displays as much of the parse as is unambiguous. Wimp3 believes it has read the head np of the first s of an s-maj, and is waiting for a vp.

2. Decisions window – Wimp3 writes to this window whenever it commits to belief in a statement. Right now, it has committed to **(inst jack717jack-)** (see below).

3. Lisp interaction window. The contents of this window will not be of interest to the reader. It simply serves to accept the input, and in these illustrations, it is contaminated by breakpoint output generated as a side effect of interrupting the program to collect screen-dumps.

4. Display of the belief network. The arcs in these graphs are always directed from the top to bottom. The numbers associated with the nodes are the posterior probabilities, to two significant digits.

As mentioned above, after evaluating the network, Wimp3 accepts the statement **(inst jack717jack-)** as true. Wimp3 makes decisions like this for two reasons (see page 82). First of all, it commits to statements which have probability 1.0 of being some value (recall that the sample space of most, but not all, of our random variables/statements is {true,false}). This is what happens with the statement **(inst jack717jack-)**. Wimp3 believes this statement to be true with probability 1. This belief is founded on the fact that there is no competing word-sense hypothesis, and the assumption that every proper noun has a denotation. The other case where Wimp3 will commit to belief is when a statement has been



```
#<INST JACK717 PERSON-> :TRUE, certain.
#<SAME-TYPE '(AGENT GAVE728) JACK717 PERSON
-> :NECESSARY, certain.
#<== '(AGENT GAVE728) JACK717> :TRUE, certa
in.
#<INST GAVE728 GIVE-> :TRUE, certain.
```

```
Press RESUME to continue or ABORT to qui
t.

⇨ [Resume]gave
1 Enter REDO-PROBS
(#<INST JACK717 PERSON->
 #<SAME-TYPE '(AGENT GAVE728) JACK717 PE
RSON->
 #<SLOT-INST AGENT GAVE728 PERSON->
 #<== '(AGENT GAVE728) JACK717>
 #<INST GAVE728 GIVE->
 #<SYN-REL SUBJECT GAVE728 JACK717>
 #<WORD-INST GAVE728 ((GIVE- 1)) VERB GI
VE>
 )
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

⇨ ■
```

Figure 8.2: The situation after Wimp3 reads "Jack gave".

in its database for a fixed period of time, and has a high probability of being a particular value. We will see an example of this kind of commitment later in this example.

8.2 gave

Figure 8.2 displays the situation after Wimp3 reads the word 'gave.' This time the parser adds two statements to the database:

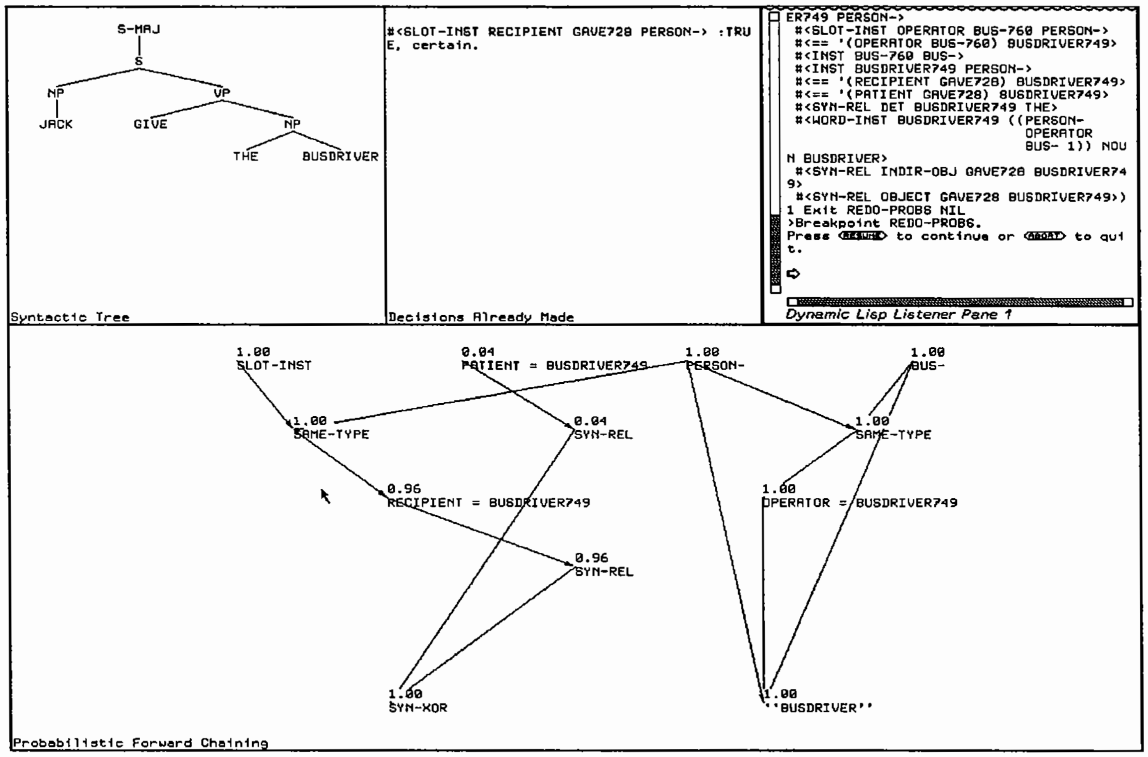
(word-inst gave728 ((give- 1)) verb give)
(syn-rel subject gave728 jack717)

The first statement causes the word-sense rule (see p. 85) to add the hypothesis **(inst gave728 give-)**. This statement, combined with the fact that 'Jack' is the subject of 'gave' leads Wimp3 to hypothesize that Jack is the agent, (== **(agent gave728) jack717)**.

When this equality is postulated, Wimp3 will look for other support. It finds that it knows that the agent of a giving is a person, and that **jack717** is a person. We see these facts reflected in the portions of the network above the `AGENT` = **jack717** node, in Figure 8.2.

When the statement **(inst gave728 give-)** is added to the database, the marker-passer passes marks from **give-**. As yet, the marker-passer yields no statements, since there are no marks for those from **gave728** to meet.

The network is evaluated, and since there are no competing word-sense hypotheses, Wimp3 commits to believing in the **give-** interpretation of **gave728**. Similarly, it decides that **jack717** is the agent of **gave728**.



```
#<SLOT-INST RECIPIENT GAVE728 PERSON-> :TRU
E, certain.
```

```
ER749 PERSON->
 #<SLOT-INST OPERATOR BUS-760 PERSON->
 #<== '(OPERATOR BUS-760) BUSDRIVER749>
 #<INST BUS-760 BUS->
 #<INST BUSDRIVER749 PERSON->
 #<== '(RECIPIENT GAVE728) BUSDRIVER749>
 #<== '(PATIENT GAVE728) BUSDRIVER749>
 #<SYN-REL DET BUSDRIVER749 THE>
 #<WORD-INST BUSDRIVER749 ((PERSON-
                            OPERATOR
                            BUS- 1)) NOU
N BUSDRIVER>
 #<SYN-REL INDIR-OBJ GAVE728 BUSDRIVER74
9>
 #<SYN-REL OBJECT GAVE728 BUSDRIVER749>)
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press <RESUME> to continue or <ABORT> to qui
t.

⇨
```

Figure 8.3: The screen after Wimp3 reads "Jack gave the busdriver"

8.3 the busdriver

Reading "the busdriver" introduces two complications.[1] First of all, busdriver has a complex word-sense (see p. 87). Secondly, we now encounter syntactic ambiguity: Wimp3 does not yet know whether 'the busdriver' is the direct or indirect object of 'gave.'

The part of the network pertaining to the denotation of 'busdriver' is seen on the right of the screendump given as Figure 8.3 (in the belief network window). In the upper right, we see the node representing the instance of **bus-** introduced for 'busdriver.' To its left is the node for the statement **(inst busdriver749 person-)**. These two statements, and the statement **(== (operator bus-760) busdriver749)**, are the parents of the new **word-inst** assertion. The **same-type** assertion is added by the rules which look for support for equality statements.

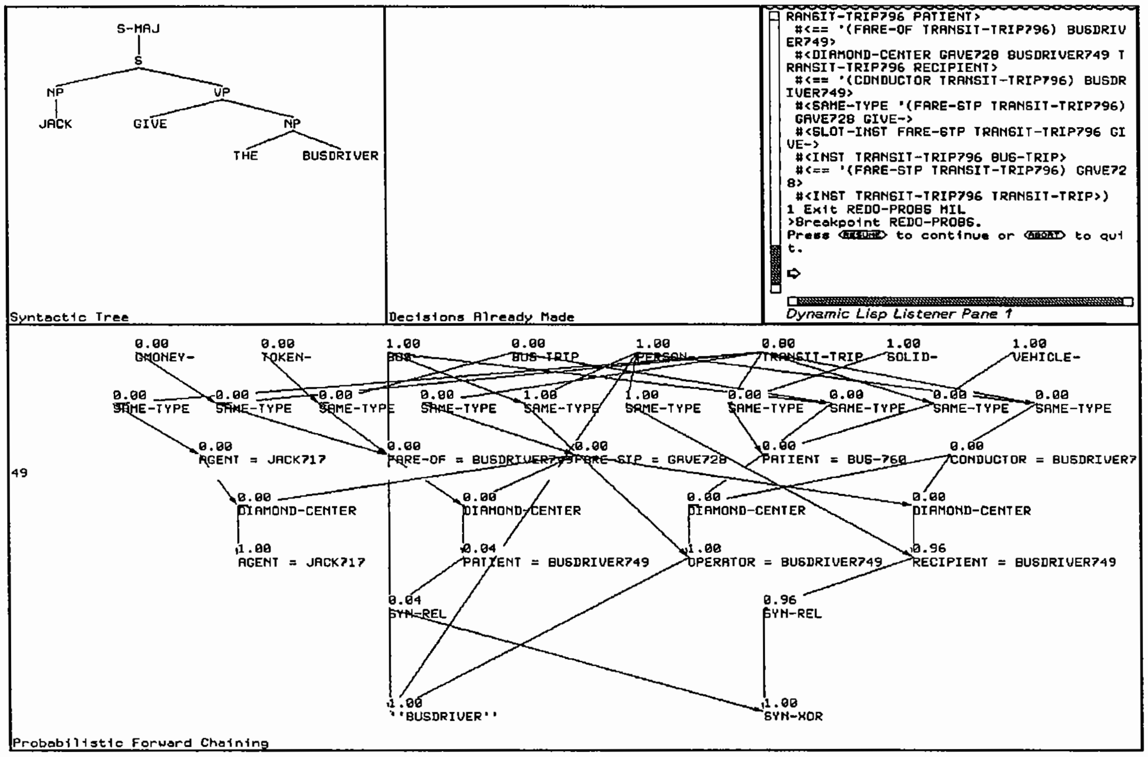
Note that, had there been an eligible referent for 'the busdriver,' a **has-referent** sub-network would have been added above the **word-inst** node. Since there is no such, Wimp3 can dispense with this piece of the belief network.

The left hand side of the screen represents the portion of the network concerned with the syntactic ambiguity. The syntactic-xor node at the bottom (displayed as SYN-XOR) indicates that the choices are exclusive.[2] This node has two parents: the two SYN-REL nodes on the screen. These indicate the two choices

(syn-rel object gave728 busdriver749)
and
(syn-rel indir-obj gave728 busdriver749)

[1]No processing of interest is done after reading just "the".

[2]'xor' is a misleading name for such nodes: we really mean one and only one of the alternatives.



```
RANSIT-TRIP796 PATIENT>
 #<== '(FARE-OF TRANSIT-TRIP796) BUSDRIV
ER749>
 #<DIAMOND-CENTER GAVE728 BUSDRIVER749 T
RANSIT-TRIP796 RECIPIENT>
 #<== '(CONDUCTOR TRANSIT-TRIP796) BUSDR
IVER749>
 #<SAME-TYPE '(FARE-STP TRANSIT-TRIP796)
 GAVE728 GIVE->
 #<SLOT-INST FARE-STP TRANSIT-TRIP796 GI
VE->
 #<INST TRANSIT-TRIP796 BUS-TRIP>
 #<== '(FARE-STP TRANSIT-TRIP796) GAVE72
8>
 #<INST TRANSIT-TRIP796 TRANSIT-TRIP>)
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.
```

Figure 8.4: The screen after Wimp3 passes marks from **bus-760**.

The one which Wimp3 has judged most probable is that 'the busdriver' is the indirect object. It prefers this alternative because it is explained by **busdriver749** being the **recipient** of **gave728**. In turn, **busdriver749** being the **recipient** of **gave728** is supported by the fact that the busdriver is a person: according to Wimp's axiomatization, only people can be recipients of givings, but any object can be the patient of a giving.

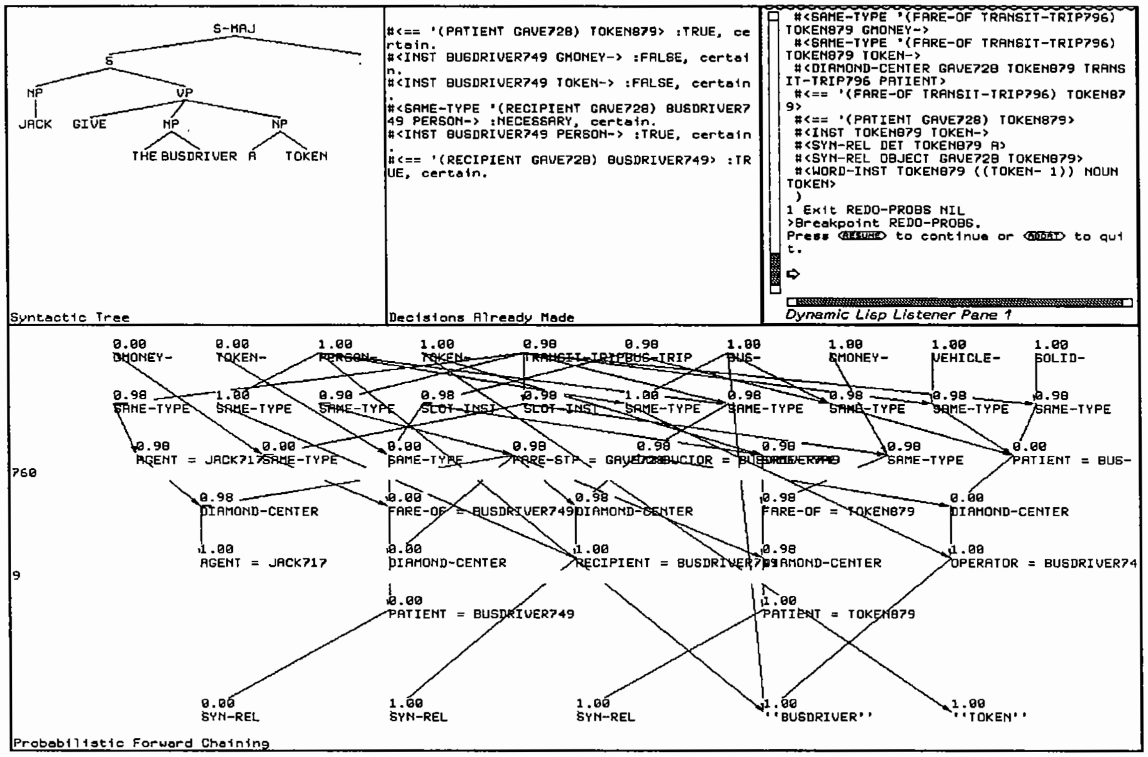
After this first network is evaluated, Wimp3 invokes its marker-passer. No marks are passed from **busdriver749**, because it is an instance of **person-**, but marks are passed from the **bus-** created as part of the meaning of 'busdriver.' This results in the situation shown in Figure8.4. Wimp3 has assembled a bus-trip hypothesis, but does not yet find it a very probable explanation; it judges the probability to be about .2 % .

Wimp3's uncertainty is also reflected in the confused nature of the belief network. Because the busdriver is a candidate for the patient for the giving event – in fact right now it is the *only* such candidate – Wimp3 considers the possibility that **busdriver749** is the fare of the **bus-trip**. See the node labelled `FARE-OF = BUSDRIVER749` in Figure 8.4. At the moment, this is enough to confuse Wimp3, because the **bus-trip-** hypothesis does not explain enough events. The probability of this interpretation will rise when Wimp3 sees 'token,' because tokens are fairly uncommon entities, and are predicted only by the **bus-trip-** plan.

8.4 a token.

After reading "a token"[3] Wimp3's interpretation of the story begins to take shape. The resulting belief network is shown in Figure 8.5. It is, unfortunately, very difficult to read, but some salient features can be picked out.

[3]Again, nothing of interest takes place after reading the article



```
#<== '(PATIENT GAVE728) TOKEN879> :TRUE, ce
rtain.
#<INST BUSDRIVER749 GMONEY-> :FALSE, certai
n.
#<INST BUSDRIVER749 TOKEN-> :FALSE, certain
.
#<SAME-TYPE '(RECIPIENT GAVE728) BUSDRIVER7
49 PERSON-> :NECESSARY, certain.
#<INST BUSDRIVER749 PERSON-> :TRUE, certain
.
#<== '(RECIPIENT GAVE728) BUSDRIVER749> :TR
UE, certain.

Decisions Already Made
```

```
 #<SAME-TYPE '(FARE-OF TRANSIT-TRIP796)
TOKEN879 GMONEY->
 #<SAME-TYPE '(FARE-OF TRANSIT-TRIP796)
TOKEN879 TOKEN->
 #<DIAMOND-CENTER GAVE728 TOKEN879 TRANS
IT-TRIP796 PATIENT>
 #<== '(FARE-OF TRANSIT-TRIP796) TOKEN87
9>
 #<== '(PATIENT GAVE728) TOKEN879>
 #<INST TOKEN879 TOKEN->
 #<SYN-REL DET TOKEN879 A>
 #<SYN-REL OBJECT GAVE728 TOKEN879>
 #<WORD-INST TOKEN879 ((TOKEN- 1)) NOUN
TOKEN>
 )
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

Dynamic Lisp Listener Pane 1
```

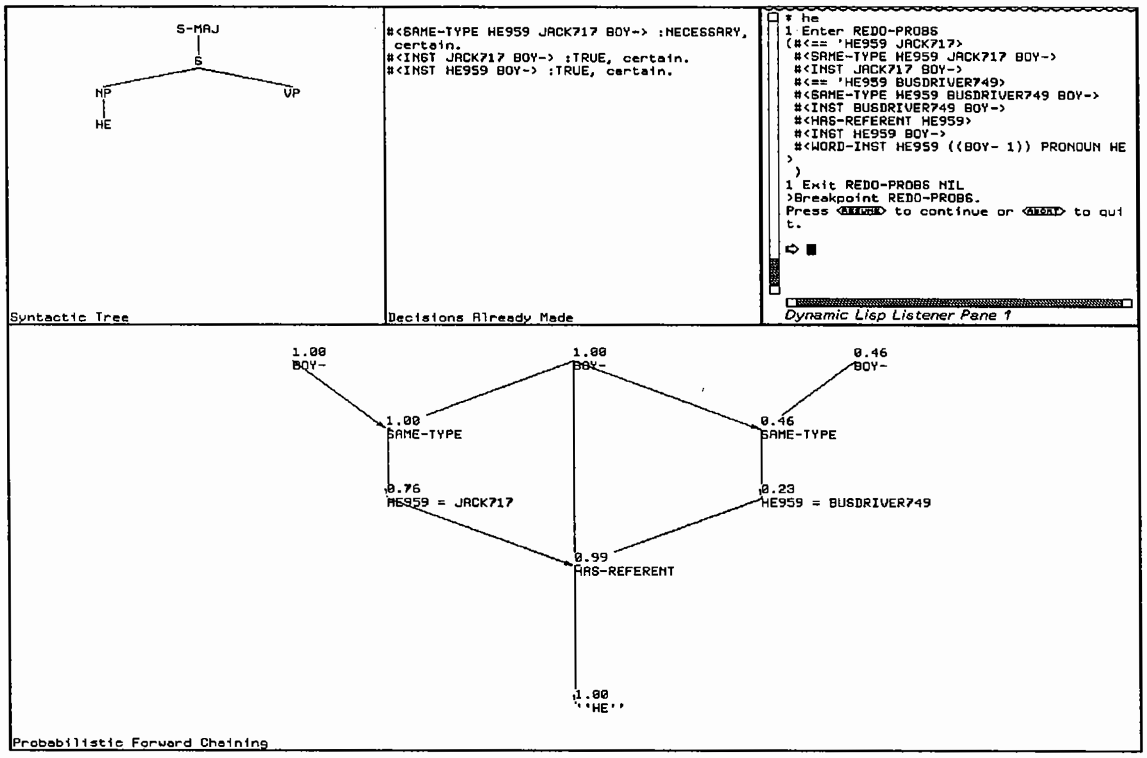
Figure 8.5: The screen after Wimp3 reads the entire first sentence.

First of all, the syntactic ambiguity has been resolved. There is no longer a **syntactic-xor** on the screen, just three **syn-rels**, two believed, and one rejected. The rejected one, on the left, is that the busdriver is the direct object of **gave728**. This is now known to be false. The two believed **syn-rel** nodes represent the fact that **token879** is the direct object, and **busdriver749** is the indirect object of the verb 'gave.'

This syntactic evidence supports Wimp3's **bus-trip-** interpretation of the story. If Wimp3 assumes that there is a bus-trip, that the token is the fare of the bus-trip, that **jack717** is the agent and the bus is the vehicle, it predicts the case-slot fillers that are, in fact observed. This is 'cheaper' for Wimp3 to assume (*i.e.*, more probable) than to assume that all these objects lined themselves up in this way by chance.

Note that we have not had to call on the marker-passer to elaborate this hypothesis any further. Once we have created the **bus-trip** hypothesis to explain the giving action (as the **fare-step**), the diamond rules (see p. 99) take care of the rest: **jack717** is inferred to be the agent of the plan because he is the agent of the **fare-step**; **busdriver749** the conductor, since he or she is the **recipient** of the **fare-step**; and so forth.

The absurd version of the **bus-trip-** has been rejected, as can be seen by examining the decisions window. This was the version which was suggested by the incorrect syntactic parse: that the busdriver was the patient of the giving. If that were true, and the giving was the **fare-step** of the **bus-trip-**, then **busdriver749** would have to be the fare. This possibility has been rejected, for two reasons. First, based on the fact that the busdriver is not a **token-**, so he cannot be the patient of the **fare-step** of a **bus-trip-**. Second, the parser has rejected the possibility that 'the busdriver' is the direct object of 'gave,' since the verb phrase is of the form **v np np**.



```
#<SAME-TYPE HE959 JACK717 BOY-> :NECESSARY,
 certain.
#<INST JACK717 BOY-> :TRUE, certain.
#<INST HE959 BOY-> :TRUE, certain.
```

```
* he
1 Enter REDO-PROBS
(#<== 'HE959 JACK717>
 #<SAME-TYPE HE959 JACK717 BOY->
 #<INST JACK717 BOY->
 #<== 'HE959 BUSDRIVER749>
 #<SAME-TYPE HE959 BUSDRIVER749 BOY->
 #<INST BUSDRIVER749 BOY->
 #<HAS-REFERENT HE959>
 #<INST HE959 BOY->
 #<WORD-INST HE959 ((BOY- 1)) PRONOUN HE
>
 )
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press <RESUME> to continue or <ABORT> to qui
t.
```

Figure 8.6: The screen after Wimp3 reads "He"

8.5 He

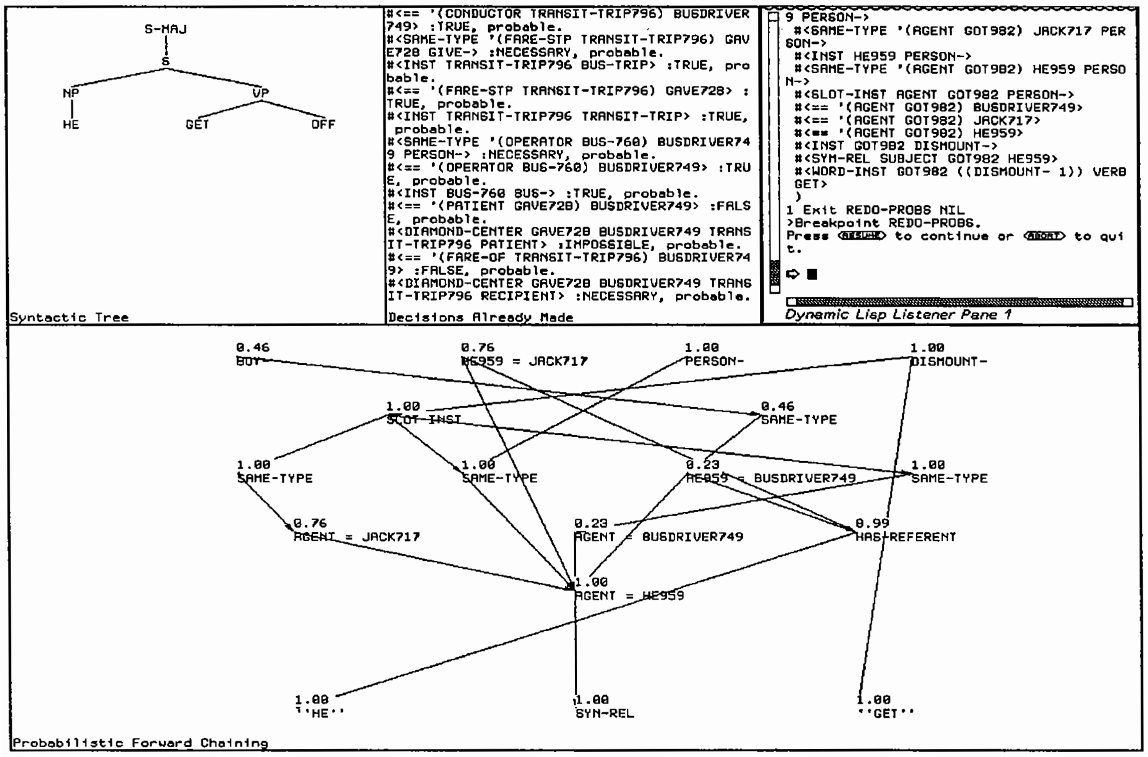
The first word of the second sentence is the pronoun 'he.' Reading this word causes Wimp3's pronoun rule (see p. 91) to fire. This adds the assertions

(inst he959 boy-)
(has-referent he959)

The addition of the latter statement begins a search for possible referents for **he959**. The two candidates are **jack717** and **busdriver749**(see Figure 8.6). Note that of the two, Wimp3 prefers the hypothesis that **he959** refers to **jack717**. At this point, it does so because it knows that **jack717** is a **boy-**, but only knows that **busdriver749** is a **person-**. Obviously, this is not a sufficient theory about resolving pronoun reference! However, for the moment we will make do with it (but see our discussion of future work in Chapter 10).

Note that only the statements just added are in the network evaluated at this iteration. At the moment this newly-added sub-network is independent of the rest of the network. Its only points of connection are the statements **(inst jack717boy-)** and **(inst busdriver749person-)**, and as they are both root nodes which are categorically believed, they block the flow of evidence between parts of the global network. Also, note that the one line down the middle of the belief network window actually represents two edges, which are superimposed. The first is from the **has-referent** parent of **he959**, and the other from the type predication of **he959**: **(inst he959 boy-)**, which is in the top-center of this window.

8.6 got off



```
#<== '(CONDUCTOR TRANSIT-TRIP796) BUSDRIVER
749> :TRUE, probable.
#<SAME-TYPE '(FARE-STP TRANSIT-TRIP796) GAV
E728 GIVE-> :NECESSARY, probable.
#<INST TRANSIT-TRIP796 BUS-TRIP> :TRUE, pro
bable.
#<== '(FARE-STP TRANSIT-TRIP796) GAVE728> :
TRUE, probable.
#<INST TRANSIT-TRIP796 TRANSIT-TRIP> :TRUE,
 probable.
#<SAME-TYPE '(OPERATOR BUS-760) BUSDRIVER74
9 PERSON-> :NECESSARY, probable.
#<== '(OPERATOR BUS-760) BUSDRIVER749> :TRU
E, probable.
#<INST BUS-760 BUS-> :TRUE, probable.
#<== '(PATIENT GAVE728) BUSDRIVER749> :FALS
E, probable.
#<DIAMOND-CENTER GAVE728 BUSDRIVER749 TRANS
IT-TRIP796 PATIENT> :IMPOSSIBLE, probable.
#<== '(FARE-OF TRANSIT-TRIP796) BUSDRIVER74
9> :FALSE, probable.
#<DIAMOND-CENTER GAVE728 BUSDRIVER749 TRANS
IT-TRIP796 RECIPIENT> :NECESSARY, probable.

Decisions Already Made
```

```
9 PERSON->
 #<SAME-TYPE '(AGENT GOT982) JACK717 PER
SON->
 #<INST HE959 PERSON->
 #<SAME-TYPE '(AGENT GOT982) HE959 PERSO
N->
 #<SLOT-INST AGENT GOT982 PERSON->
 #<== '(AGENT GOT982) BUSDRIVER749>
 #<== '(AGENT GOT982) JACK717>
 #<== '(AGENT GOT982) HE959>
 #<INST GOT982 DISMOUNT->
 #<SYM-REL SUBJECT GOT982 HE959>
 #<WORD-INST GOT982 ((DISMOUNT- 1)) VERB
 GET>
 )
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

⇨ ■

Dynamic Lisp Listener Pane 1
```

Figure 8.7: The first update after reading "He got off"

The next chaining of interest is done only after Wimp3 has read both 'got' and 'off.' This is because after reading 'got,' the parser is waiting to see whether it is reading a simple verb, or a verb + particle construction. Because it treats such constructions, like 'got off' as a single verb, it makes no assertions in the midpoint.

After 'off' is read, this momentary ambiguity is resolved. Wimp3 consults its lexicon, and finds the sense **dismount-** for the construction 'get off.' The results are shown in Figure8.7. Wimp has not yet passed marks from **dismount-**. As mentioned earlier, it first chains and evaluates the resulting network before doing so. This allows it to eliminate any hypotheses found to be impossible.

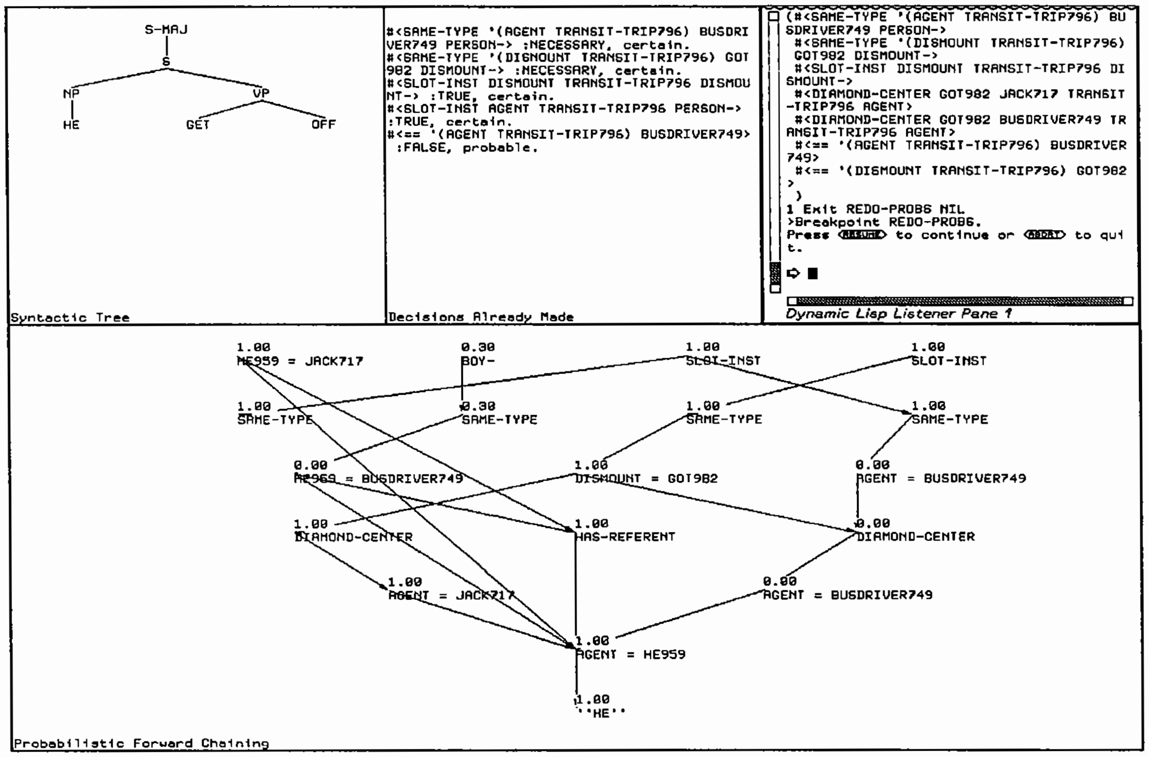
Note that the window displaying Wimp3's commitments is full (in fact it has overflowed and wrapped around). What has happened is that the time period for committing to belief after the introduction of the **bus-trip-** hypothesis has just expired. Since the statements about the **bus-trip-** have very high probabilities, Wimp3 has committed to these beliefs.

The benefits of this commitment can be seen in the belief network after marker-passing, shown here as Figure8.8. Accepting those statements as true has the effect of making the statements about the **dismount-step** of the **bus-trip-** independent of most other statements about the **bus-trip-**.

Note that tying this new action into the plan has the side-effect of resolving the reference ambiguity about 'He.' The diamond-rules dictate that the agent of the **dismount-step** of the **bus-trip-** must be the same as the agent of the **bus-trip-**, ergo 'he' must be **jack717**.

8.7 at the supermarket.

After reading 'at' and 'the,' no network construction or evaluation is done. However, Wimp3 continues the process of committing to strongly-held beliefs about the **bus-trip-**



```
#<SAME-TYPE '(AGENT TRANSIT-TRIP796) BUSDRI
VER749 PERSON-> :NECESSARY, certain.
#<SAME-TYPE '(DISMOUNT TRANSIT-TRIP796) GOT
982 DISMOUNT-> :NECESSARY, certain.
#<SLOT-INST DISMOUNT TRANSIT-TRIP796 DISMOU
NT-> :TRUE, certain.
#<SLOT-INST AGENT TRANSIT-TRIP796 PERSON->
:TRUE, certain.
#<== '(AGENT TRANSIT-TRIP796) BUSDRIVER749>
 :FALSE, probable.
```

```
(#<SAME-TYPE '(AGENT TRANSIT-TRIP796) BU
SDRIVER749 PERSON->
 #<SAME-TYPE '(DISMOUNT TRANSIT-TRIP796)
 GOT982 DISMOUNT->
 #<SLOT-INST DISMOUNT TRANSIT-TRIP796 DI
SMOUNT->
 #<DIAMOND-CENTER GOT982 JACK717 TRANSIT
-TRIP796 AGENT>
 #<DIAMOND-CENTER GOT982 BUSDRIVER749 TR
ANSIT-TRIP796 AGENT>
 #<== '(AGENT TRANSIT-TRIP796) BUSDRIVER
749>
 #<== '(DISMOUNT TRANSIT-TRIP796) GOT982
>
 )
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.
```

Figure 8.8: After passing marks from **dismount-**. Wimp3 ties the second sentence into the bus-trip hypothesis.

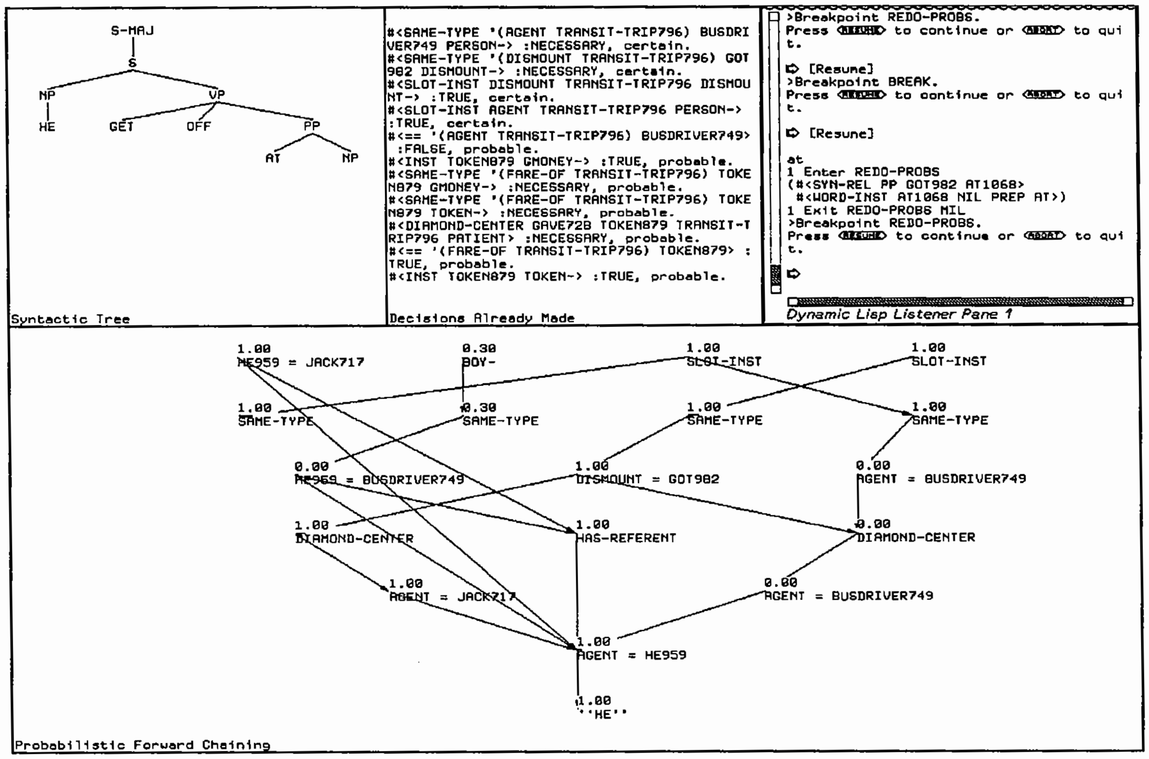
interpretation. In particular, it commits to beliefs about the role that **token879** plays in the plan. It commits to these statements later than the others because they were added later (only after the word 'token' was read, rather than after 'busdriver'). In Figure 8.9 we give the screen after reading each of these two words; only the decisions window is of interest.

The screen immediately after reading 'supermarket' is as shown in Figure 8.10. The belief network represented here shows that Wimp3 is fitting **supermarket1075** into the **bus-trip-** plan. In the lower center of the diagram is the deep case relation (**== (destination got982) supermarket1075**) statement, represented as `DESTINATION = SUPERMARKET1075`. Evidence is provided this statement by the fact that the supermarket is in a pp of the form "at x" which is attached to "get off." In turn, this statement provides support for the hypothesis that the getting off is the **dismount-step** of the **bus-trip-** and that the supermarket is its **destination**. This can be seen by examining the parents of the `DIAMOND-CENTER` above `DESTINATION = SUPERMARKET1075` in the window: `DESTINATION = SUPERMARKET1075`[4] and `DISMOUNT = GOT982`.

Further support for this interpretation is provided by the fact that 'he' is the agent of the getting-off. This combines with the fact that Jack is the **agent** of the **bus-trip-** and our earlier assumption that 'he' refers to Jack. This can be seen in the part of the belief network displayed in the left-center of the window in Figure 8.10.

After updating the network and making commitments, Wimp3 passes marks from the newly-introduced **supermarket-**. It finds an acceptable path between **bus-trip-** and **supermarket-** that passes through **supermarket-shop-**, as follows:

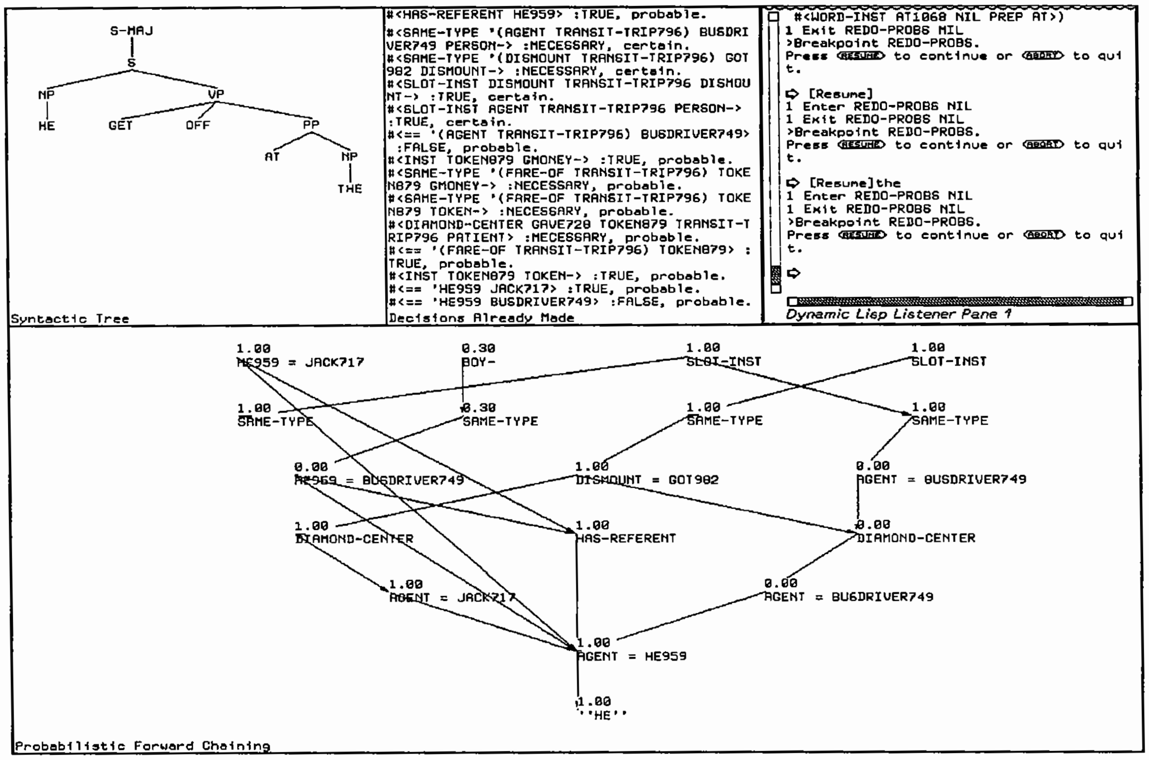
[4]We apologize for the unfortunate way we've abbreviated the statements.



```
#<SAME-TYPE '(AGENT TRANSIT-TRIP796) BUSDRI
VER749 PERSON-> :NECESSARY, certain.
#<SAME-TYPE '(DISMOUNT TRANSIT-TRIP796) GOT
982 DISMOUNT-> :NECESSARY, certain.
#<SLOT-INST DISMOUNT TRANSIT-TRIP796 DISMOU
NT-> :TRUE, certain.
#<SLOT-INST AGENT TRANSIT-TRIP796 PERSON->
:TRUE, certain.
#<== '(AGENT TRANSIT-TRIP796) BUSDRIVER749>
 :FALSE, probable.
#<INST TOKEN879 GMONEY-> :TRUE, probable.
#<SAME-TYPE '(FARE-OF TRANSIT-TRIP796) TOKE
N879 GMONEY-> :NECESSARY, probable.
#<SAME-TYPE '(FARE-OF TRANSIT-TRIP796) TOKE
N879 TOKEN-> :NECESSARY, probable.
#<DIAMOND-CENTER GAVE728 TOKEN879 TRANSIT-T
RIP796 PATIENT> :NECESSARY, probable.
#<== '(FARE-OF TRANSIT-TRIP796) TOKEN879> :
TRUE, probable.
#<INST TOKEN879 TOKEN-> :TRUE, probable.

Decisions Already Made
```

```
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

⇨ [Resume]
>Breakpoint BREAK.
Press RESUME to continue or ABORT to qui
t.

⇨ [Resume]

at
1 Enter REDO-PROBS
(#<SYN-REL PP GOT982 AT1068>
 #<WORD-INST AT1068 NIL PREP AT>)
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

⇨

Dynamic Lisp Listener Pane 1
```

(a)

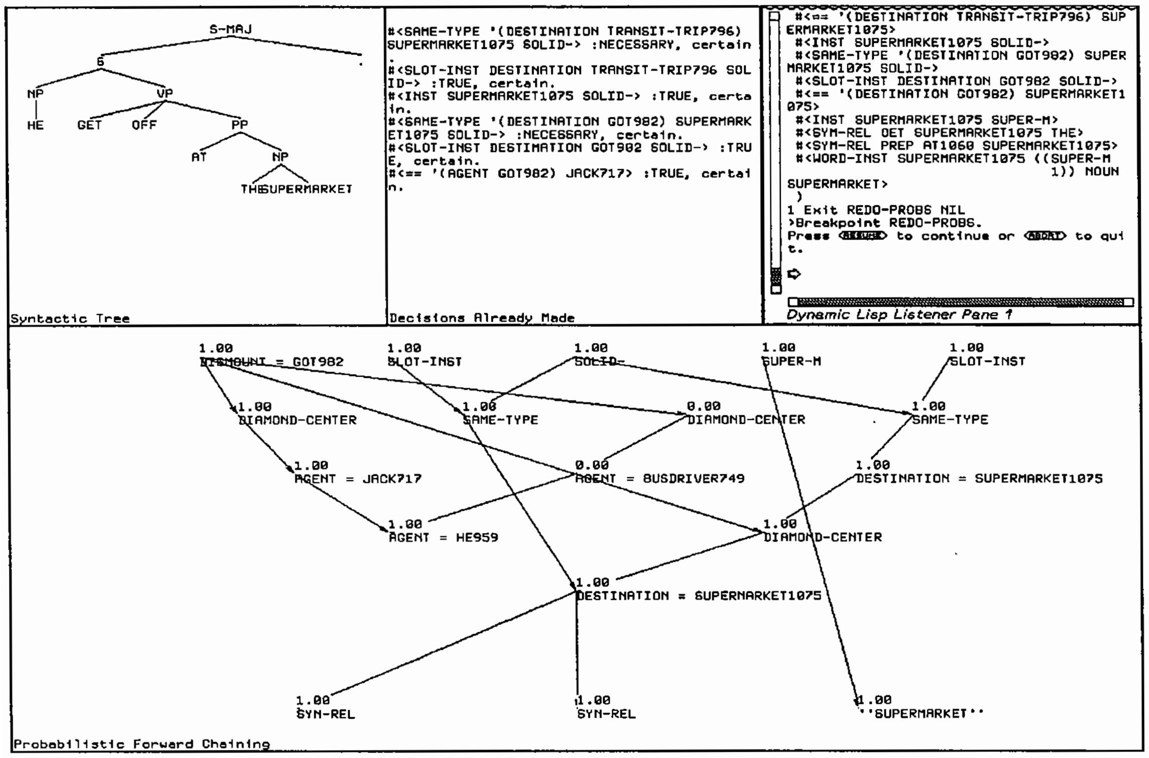


```
#<HAS-REFERENT HE959> :TRUE, probable.
#<SAME-TYPE '(AGENT TRANSIT-TRIP796) BUSDRI
VER749 PERSON-> :NECESSARY, certain.
#<SAME-TYPE '(DISMOUNT TRANSIT-TRIP796) GOT
982 DISMOUNT-> :NECESSARY, certain.
#<SLOT-INST DISMOUNT TRANSIT-TRIP796 DISMOU
NT-> :TRUE, certain.
#<SLOT-INST AGENT TRANSIT-TRIP796 PERSON->
:TRUE, certain.
#<== '(AGENT TRANSIT-TRIP796) BUSDRIVER749>
 :FALSE, probable.
#<INST TOKEN879 GMONEY-> :TRUE, probable.
#<SAME-TYPE '(FARE-OF TRANSIT-TRIP796) TOKE
N879 GMONEY-> :NECESSARY, probable.
#<SAME-TYPE '(FARE-OF TRANSIT-TRIP796) TOKE
N879 TOKEN-> :NECESSARY, probable.
#<DIAMOND-CENTER GAVE728 TOKEN879 TRANSIT-T
RIP796 PATIENT> :NECESSARY, probable.
#<== '(FARE-OF TRANSIT-TRIP796) TOKEN879> :
TRUE, probable.
#<INST TOKEN879 TOKEN-> :TRUE, probable.
#<== 'HE959 JACK717> :TRUE, probable.
#<== 'HE959 BUSDRIVER749> :FALSE, probable.
Decisions Already Made
```

```
 #<WORD-INST AT1068 NIL PREP AT>)
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

⇨ [Resume]
1 Enter REDO-PROBS NIL
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

⇨ [Resume]the
1 Enter REDO-PROBS NIL
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.

⇨

Dynamic Lisp Listener Pane 1
```

(b)

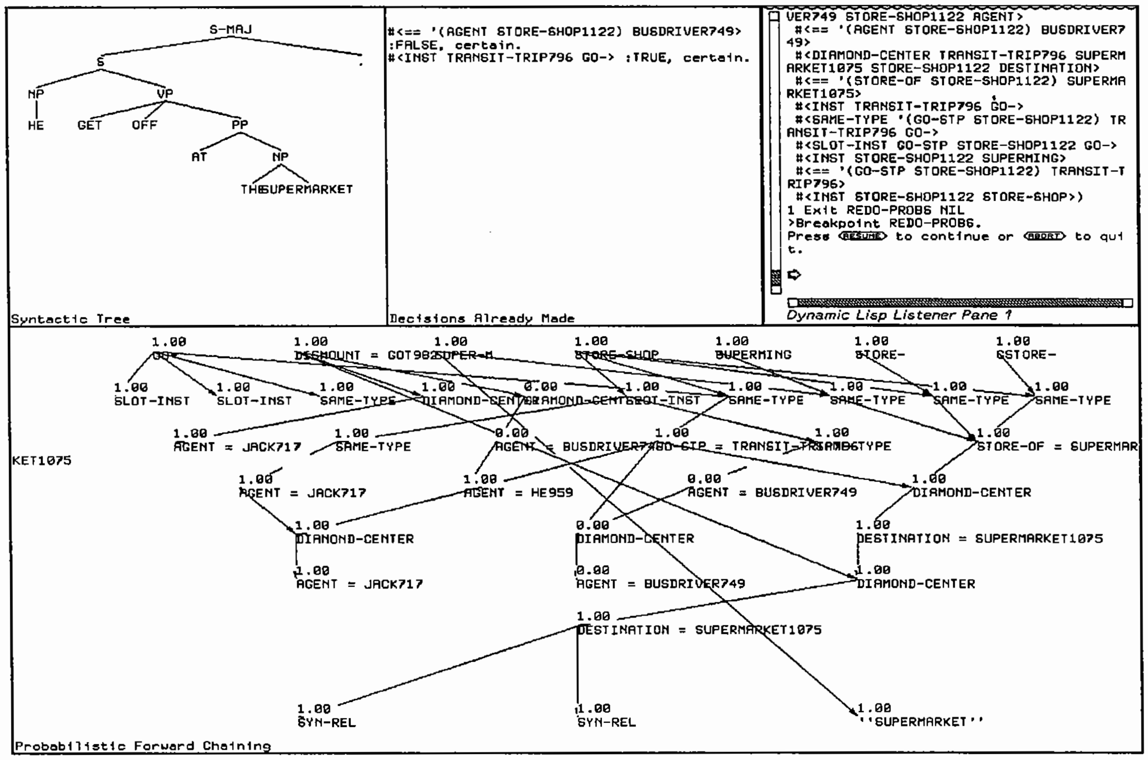
Figure 8.9: The beliefs Wimp3 commits to when reading "...at the" Nothing but the decisions windows is interesting.



```
#<SAME-TYPE '(DESTINATION TRANSIT-TRIP796)
SUPERMARKET1075 SOLID-> :NECESSARY, certain
.
#<SLOT-INST DESTINATION TRANSIT-TRIP796 SOL
ID-> :TRUE, certain.
#<INST SUPERMARKET1075 SOLID-> :TRUE, certa
in.
#<SAME-TYPE '(DESTINATION GOT982) SUPERMARK
ET1075 SOLID-> :NECESSARY, certain.
#<SLOT-INST DESTINATION GOT982 SOLID-> :TRU
E, certain.
#<== '(AGENT GOT982) JACK717> :TRUE, certai
n.
```

```
 #<== '(DESTINATION TRANSIT-TRIP796) SUP
ERMARKET1075>
 #<INST SUPERMARKET1075 SOLID->
 #<SAME-TYPE '(DESTINATION GOT982) SUPER
MARKET1075 SOLID->
 #<SLOT-INST DESTINATION GOT982 SOLID->
 #<== '(DESTINATION GOT982) SUPERMARKET1
075>
 #<INST SUPERMARKET1075 SUPER-M>
 #<SYM-REL DET SUPERMARKET1075 THE>
 #<SYM-REL PREP AT1060 SUPERMARKET1075>
 #<WORD-INST SUPERMARKET1075 ((SUPER-M
                               1)) NOUN
SUPERMARKET>
 )
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.
```

Figure 8.10: The screen immediately after Wimp3 reads "...supermarket.



```
#<== '(AGENT STORE-SHOP1122) BUSDRIVER749>
:FALSE, certain.
#<INST TRANSIT-TRIP796 GO-> :TRUE, certain.
```

```
VER749 STORE-SHOP1122 AGENT>
 #<== '(AGENT STORE-SHOP1122) BUSDRIVER7
49>
 #<DIAMOND-CENTER TRANSIT-TRIP796 SUPERM
ARKET1075 STORE-SHOP1122 DESTINATION>
 #<== '(STORE-OF STORE-SHOP1122) SUPERMA
RKET1075>
 #<INST TRANSIT-TRIP796 GO->
 #<SAME-TYPE '(GO-STP STORE-SHOP1122) TR
ANSIT-TRIP796 GO->
 #<SLOT-INST GO-STP STORE-SHOP1122 GO->
 #<INST STORE-SHOP1122 SUPERMING>
 #<== '(GO-STP STORE-SHOP1122) TRANSIT-T
RIP796>
 #<INST STORE-SHOP1122 STORE-SHOP>)
1 Exit REDO-PROBS NIL
>Breakpoint REDO-PROBS.
Press RESUME to continue or ABORT to qui
t.
```

Figure 8.11: The screen after Wimp3 has passed marks from **supermarket-**, introducing the **supermarket-shopping** hypothesis.

node	link
supermarket-	store-of
supermarket-shop-	isa
store-shop-	isa
gshop-	go-step
go-	isa
transit-trip-	isa
bus-trip-	

The resulting belief network is shown in Figure 8.11. The interesting additions to the network are in the upper right of the belief network. Above the previously-existing node representing (== **(destination transit-trip796) supermarket1075)** there is a new **diamond-center** node. The parents of this node are statements about the newly-introduced **supermarket-shop-**[5] plan, drawn as `STORE-OF = SUPERMARKET1075` and `GO-STP = TRANSIT-TRIP796` in the screendump.

After network evaluation, this hypothesis is found to be very likely. Not as likely as the screendump indicates, but approximately 77%. This is because the **supermarket-shop-** hypothesis is a very good explanation for supermarkets, even in isolation, and in this case it also explains why the destination of the bus-trip is the supermarket.

8.8 Summary

In this chapter we have tried to show how the various components of Wimp3 work together to find an interpretation of a text. We have done so by reviewing Wimp3's processing of a simple, two-sentence story. In this story, we have seen the way Wimp3's plan-recognition, semantic and syntactic processing work together, mediated by the belief network, to constrain the interpretation process. We have also seen how the rules various rules combine to construct a coherent belief network.

[5]**supermarket-shop-** is abbreviated to `SUPERMING` in the screen display

Chapter 9

Testing the implementation

We have developed a test corpus of 25 stories for Wimp3. These stories were chosen to demonstrate particular features of Wimp3's architecture: stories requiring integrated inference for interpretations, stories for which there is more than one consistent interpretation so that different interpretations must be compared, etc. Our implementation of Wimp3 successfully processes all of the stories in this training suite. In order to avoid some of the problems of 'wishful programming,' we have devised a simple 'single blind' test for Wimp3 involving 25 stories in a hidden test set. With a few additions to its lexicon and case rules, Wimp3 processes 24 of the 25 stories in the test set.

In this chapter we describe these evaluations in detail. We begin by discussing the construction of the training suite, and comment on the features of Wimp3 the various stories were intended to illustrate. We then discuss the organization of the single-blind test, and review the results.

9.1 The Training set

The twenty-five stories of the training set are given as Figure 9.1. In this section we will briefly comment on the features of Wimp3 that are demonstrated by the training set. We will also discuss the database queries which were used to demonstrate Wimp3's 'understanding' of the training set.

Many of the stories in Wimp3's training set were designed to illustrate that the integrated, probabilistic approach can handle classes of texts which presented difficulties to earlier plan-recognition and story understanding systems (see Chapter 2). Several of the stories require a mixture of top-down and bottom-up inference. Others were chosen to demonstrate the way probabilities allow Wimp3 to evaluate different interpretations relative to each other. They show that Wimp can reach tentative conclusions about the meaning of a story, and revise these conclusions in the face of further evidence. Some of the examples were chosen to show that Wimp3 can handle plan-recognition problems which were not permitted in Kautz' idealization of plan-recognition. Finally, some exemplify the integrated processing which is another one of Wimp3's distinguishing features.

The purpose of some of the stories is to demonstrate that Wimp3 performs both top-down and bottom-up plan-recognition inference. Some stories introduce a plan first, and then mention components of that plan (8). Other stories mention sub-acts and then mention a plan (15). A clearer example of this is the pair of stories "Jack wanted to kill himself. He got a rope." and "Jack got a rope. He wanted to kill himself."[1] Finally, and most

[1]Wimp3 makes the correct inferences in these two examples (Jack is getting a rope in order to hang

1. "Jack went to the supermarket. He found some milk on the shelf. He paid for it."
2. "Bill went to the supermarket. He paid for some milk."
3. "Jack gave the busdriver a token. He got off at the supermarket."
4. "Jack got off the bus at the liquor-store. He pointed a gun at the owner."
5. "Jack went to the liquor-store. He found some bourbon on the shelf."
6. "Bill went to the liquor-store. He pointed a gun at the owner."
7. "Bill gave the busdriver a token."
8. "Fred robbed the liquor-store. Fred pointed a gun at the owner."
9. "Bill got a gun. He went to the supermarket."
10. "Fred went to the supermarket. He pointed a gun at the owner. He packed his bag. He went to the airport."
11. "Jack took the bus to the airport. He bought a ticket."
12. "Bill packed a suitcase. He went to the airport."
13. "Jack got on a bus. He got off at the park. Jack went to the supermarket."
14. "Jack gave the busdriver a token. He got off at the park. He went to the airport. He got on a plane."
15. "Fred sat down on the bus. He went to the supermarket."
16. "Jack went to a restaurant. He got a milkshake."
17. "Bill drank a milkshake with a straw."
18. "Fred got off the bus at a restaurant. He got a milkshake."
19. "Janet put a straw in a milkshake."
20. "Bill got on a bus. He got off at a restaurant. He drank a milkshake with a straw."
21. "Bill took a bus to a restaurant. He drank a milkshake. He pointed a gun at the owner. He got some money from him."
22. "Fred gave the busdriver a token. He got off the bus at the park. He went to a restaurant. He got some money from the owner."
23. "Jack took a taxi to the park."
24. "Bill took a taxi."
25. "Fred took a taxi to the bus-station. He got on a bus."

Figure 9.1: Wimp3's training set.

commonly, there are stories which never explicitly mention the character's plan (4).

In the training set, there are several pairs of stories which indicate the way Wimp3 updates probabilities in the course of plan-recognition. These are stories which begin with some mention of a trip to a store or restaurant. These stories then diverge to describe either a shopping plan[2] or a robbery plan. Some such are:

supermarket shopping:	1,2,3,15	robbery:	9
liquor-store shopping:	5	robbery:	4,6,8
eating in restaurants	16,18,20	robbery:	21,22

These examples demonstrate that Wimp3 can initially favor an explanation whose a priori probability is higher – the humdrum stories – but can shift its attention to a more unusual explanation – robbery – when there is a preponderance of evidence favoring it. Note that these stories also demonstrate the importance of a priori probabilities. Wimp3 will correctly assume a shopping plan when it knows just that some character is going to the supermarket, rather than reporting the disjunction (shopping or robbery or ...).

For some stories, a correct understanding requires that Wimp3 recognize that a character is executing more than one top-level plan (rather than executing a number of plans all in pursuit of one overall plan). Stories 10,13,14,21 and 22 are of this type. For example, in order to understand story 10, Wimp3 must understand that Fred is executing both the rob and air-travel plans, and that neither one is in service of the other.[3]

Finally, there are stories which demonstrate that Wimp3 can handle classes of plan-recognition inference which were ruled out in Kautz' framework [63]. Kautz' framework requires that plan hierarchies be acyclic – no plan of type t can have a substep of type t' such that t' *isa*$*$ t. This makes it impossible to have plans like our air-trip plan, which isa go plan, and which has, as one of its steps, that the agent goes to the airport which is the source of the air-trip. See stories 11,12 and 14. Another example is the bus-trip plan; see story 25.

Another simplification made in Kautz' framework was that a set of top-level, or self-explanatory, plans must be specified as input to the program. This makes it difficult to process stories like our story 24: "Bill took a taxi." For this story, Wimp3 simply reports that there is a taxi-trip, and that Bill is the agent. In Kautz' framework, this intuitively appealing interpretation would be only one element of a large disjunction, which would include every top-level plan which could explain going somewhere.

Several of the stories were chosen to demonstrate Wimp3's integrated approach to text understanding. These are stories which show the way syntactic, semantic and pragmatic information work together to constrain the interpretation. Some examples are the way plan-recognition inference constrains the process of reference resolution: e.g., in story 3, 'he' (the person who gets off the bus) is correctly recognized as Jack, rather than the busdriver. Story 19, "Janet put a straw in a milkshake," shows how syntax, semantics and world knowledge can be combined. Knowledge about drinking is used to resolve the problems of prepositional phrase attachment (drinking-with vs. Janet-with) and word-sense ambiguity (drinking-straw vs. the kind of straw animals eat).

These, then, are the kind of points we are trying to demonstrate in having Wimp3 interpret these stories. However, so far we have simply relied on a fuzzy, intuitive notion of understanding. What do we mean by 'understanding'?

We have developed a set of queries which we claim represents an intuitive understanding

himself), but they were not part of our test arrangement.

[2]Where eating in a restaurant is considered a generalized form of shopping

[3]Wimp3's knowledge base does not contain the notion of fleeing from the law. One would not want to trust it to plan a heist.

of the stories in the training set. These are queries which Wimp3's database will answer affirmatively after reading the corresponding story. They are given in Appendix A, some with English glosses to make the meaning more clear.

Processing time for the stories in the test set ranges from c. 15 seconds to c. 10 minutes (user time, not CPU time) on a Sun SPARCstation 1 with 135Mb swap, 16Mb memory, running Sun Common Lisp 4.0. The code is compiled, but not optimized.

9.2 The Single-blind Test

The stories above were the training set for Wimp3. They were developed as part of a single-blind test of Wimp3's effectiveness. Eugene Charniak developed twenty-five pairs of synonymous stories. These stories were developed for the demonstrative purposes outlined in the previous section. They were designed to require only knowledge of everyday human activities and some unusual alternatives like hijacking. One of the stories from each pair was chosen (randomly) to be in the training set. The rest were set aside to be a test set, and were kept hidden from the author until after the tests were completed.

The evaluation task for Wimp3 was to correctly match each story in the test set with its mate in the training set. In order to allow for variations in wording, we decided in advance on several simple criteria for matching. Each story was categorized by giving the number of top-level plans described in it, and by a set of sketchy frames, one for each top-level plan. A plan, p is a top-level plan if there does *not* exist another plan, p' such that for some slot, s, $(s\ p') == p$. I.e., a top-level plan does not fill a slot in any other plan.

The sketchy frame representations are records which are made up of the most-specific type of the top-level plan, and bindings for the agent, and generalized patient of the plan. The author defined for each plan type a slot to be the generalized patient of that plan. E.g., the generalized patient of a shopping plan was the `bought` slot of the shopping plan. The value of the agent and generalized patient was the most specific known type of the filler of the corresponding slot. Because, as a notational convenience, named persons were represented by types (e.g., a `Jack` isa `person` whose name is "Jack"), this representation was sufficient to determine the agent of the plans.

For example, the categorization of story 1:

Jack went to the supermarket. He found some milk on the shelf. He paid for it.

was:

```
(1 (SUPERMING JACK- MILK-))
```

There is one top-level plan, it is a supermarket-shopping plan, its agent is Jack, and its patient is some milk. For story 13,

Jack got on a bus. He got off at the park. Jack went to the supermarket.

the categorization was:

```
(2 (SUPERMING JACK- FOOD-) (PARK-TRIP JACK- NIL))
```

There are two plans: 1) Jack goes shopping at the supermarket, and 2) Jack goes on a park-trip. Note that for the first plan, the generalized patient was determined by Wimp3's knowledge about supermarket-shopping, because the story did not give more specific information about what Jack purchased. For the second plan, the generalized patient of park-trips was defined to be `NIL`, because Wimp3 doesn't know anything about specific parks.[4]

The matching criterion was held to be frame compatibility, since some of the synonymous stories were more or less specific than their mates. So, for example,

```
(a)  (1 (SUPERMING JACK MILK-))
                and
(b)  (1 (SUPERMING JACK FOOD-))
```

are synonymous. This weakens the test a little bit, since now two of the stories in the training set are synonymous: stories 1 and 3 are represented as (a) and (b), above.

The stories in the test set are given as Figure 9.2. After Wimp3 successfully processed the training set, it was tested with this set of stories. At that point, 24 of the 25 test stories could be processed without causing an error. 19 of the 25 were correctly classified.

Examining the stories which were not successfully classified, we found that the dictionary was missing some entries, and that Wimp3 did not have sufficient knowledge about verb cases. E.g., there was nothing in Wimp3's database that stated that 'getting onto a vehicle' is the same as 'getting on.' When these deficiencies were remedied – without changing any code or any of Wimp3's inference rules – Wimp3 was able to successfully match 24 of the 25 stories: all of the test-set stories which could be read without error.

9.3 Conclusions

We have tried to present examples illustrative of the strengths of our approach. We hope that the single blind test will help to convince the reader that we have not simply written a program that handles exactly the examples discussed in the thesis. We grant that this is only the most simple of tests. However, to the best of our knowledge, even such a rudimentary test of a story understanding system is completely novel (cf. [30, 86, 110]).[5]

[4]Perhaps it would have been more elegant to make the generalized patient always be `PARK-`, but it makes no concrete difference.

[5]For a discussion of the difficulties of evaluating natural language processing programs, see [88]. Further, note that the discussants at this workshop were all government contractors: how much more difficult are the problems for those working with the more limited resources of academic institutions.

1. "Bill robbed the supermarket."
2. "Bill went to a restaurant. He put a straw in the milkshake."
3. "Fred got off the bus at the park. He took a bus to a restaurant. He robbed the restaurant."
4. "Bill went to the airport. He got on the airplane."
5. "Bill got on a bus. He gave the busdriver a token."
6. "Fred robbed the supermarket. He went to the airport. He got on a plane."
7. "Jack went to the supermarket."
8. "Fred robbed the liquor-store."
9. "Bill bought some milk at the supermarket."
10. "Jack took a bus to a restaurant. He drank a milkshake."
11. "Jack got on a bus. He went to the supermarket."
12. "Jack got a gun. He took the bus to the liquor-store."
13. "Bill got into a taxi. He paid the driver."
14. "Fred went to a restaurant. He paid for a milkshake."
15. "Bill got a milkshake. He picked up a straw."
16. "Jack bought a ticket. He got on the airplane."
17. "Bill got off the bus at the liquor-store. He robbed it."
18. "Janet drank a milkshake with a straw."
19. "Jack got into a taxi. He got out at the park."
20. "Bill went to a restaurant. He put the straw into the milkshake. Bill robbed the restaurant."
21. "Jack got on a bus. He got off at a liquor-store. He paid for some bourbon."
22. "Jack gave the busdriver a token. He went to the park. He bought some cheese at the supermarket."
23. "Jack sat down in the bus. He went to the park. He packed his suitcase. He went to the airport."
24. "Fred got into a taxi. He got out at the bus station. He gave the bus driver a token. "
25. "Fred went to the supermarket."

Figure 9.2: The stories in Wimp3's test set.

Chapter 10

Conclusion

In this, the concluding chapter of the thesis, we discuss open problems, and directions suggested by the Wimp3 project, and review the progress made. We start by discussing the parser, which is not as integrated into the architecture as thoroughly as we would like. We discuss some practical problems we have had using belief networks in knowledge representation. In particular, we have found it difficult to write rules for databases like our belief networks or an ATMS [37]; also, it is by no means straightforward to mix logical and probabilistic inference in belief networks. We discuss some shortcomings that Wimp3 shares with most other story understanding programs, notably ignoring the issue of time and not doing enough top-down inference. The issue of time is closely related to Wimp3's weak representation of the structure of discourse. Finally, we sum up the contributions of this work.

10.1 The Parser

Wimp3's parser is not smoothly integrated into the project of probabilistic text understanding. As we've explained, Wimp3's parser is a conventional ATN parser, modified slightly for incremental processing, and to produce all possible parses of the input. Unlike the rest of our architecture, where different knowledge sources communicate through the belief network, all communication is from the parser to the rest of Wimp3. The parser is not able to take advantage of any probabilistic information. In particular, it cannot use Wimp3's ranking of different parses based on semantics and plan-recognition.

The one-way communication between the parser and the rest of Wimp3, although inelegant, may not be altogether a bad thing. After all, it means that the parser acts as a filter on possible interpretations. What *is* undeniably a bad thing is that the conventional design of Wimp3's parser makes it impossible for Wimp3 to handle noisy or ill-formed input. The ability to handle noise is something one expects from a probabilistic approach. There have been some attempts to handle ill-formed input within a conventional parser architecture (see, for example [81]). However, these rely on patching the results of a failed parsing process: otherwise they would be searching too large a space. Such an approach is not compatible with Wimp3's architecture.

10.2 Writing rules for all-solutions databases

One practical problem we have found in constructing both Wimp2 and Wimp3 is the construction of rules for 'all-solutions' databases. Both Wimp2's ATMS (see Chapter 2), and

Wimp3's belief network represent simultaneously an entire space of possible interpretations, in a way that conventional TMS's do not. Furthermore, they do so in a way which shares structure, unlike conventional search trees.

Database architectures like this require a different sort of rules from those in conventional architectures. One of the most important differences is that it is difficult to write rules which have as preconditions the absence of statements.

An example may make this more clear: Our diamond rules take entities which play parts in plans, and fit related entities into the same plans. For example, if we know that a buying event is fills the pay-step slot of a shopping event, then we know that the object which is paid for is the object for which the agent is shopping. If we know that Jack is the person who is paying, then we know he is the person who has gone shopping.

These rules add snarls to our networks – subgraphs which are almost completely connected – and thus slow network evaluation. So if we can avoid firing these rules too often, without losing any information, it will be beneficial.

One way of controlling the diamond rules would be to take into account coreference. For example, if we know that 'he' refers to 'Jack,' and we know that 'he' is the agent of the pay-step of a shopping, we would prefer to just add a diamond about the fact that Jack is the agent of the shopping, and omit the part concerned with 'he.' So we might add as a precondition for firing the diamond rule, that the entity concerned not have a better name.

Unfortunately, a precondition like this can cause Wimp3 to miss many important inferences. This is because it might be considering a coreference relation hypothesis which is incorrect.

One possible fix to this problem would be to set a probability cutoff, and make statements whose probability is below that cutoff invisible. Unfortunately, this is made very difficult by the nature of our problem. Because the space of interpretations is so large, the prior probability of any hypothesis is going to be astronomically low. So we cannot ignore improbable statements; the correct solution will, at some time or other, appear improbable, and such a cutoff would cause Wimp3 to fail to expand the correct interpretation. If we are to impose a cutoff like this, we will have to find one which will be sensitive to the state of Wimp's computation.

10.2.1 Mixing logical and probabilistic inference

One of the primary selling points of belief networks has been that they are suited to representing functional dependencies – conditional probabilities of one and zero – as well as causal relations. This is one of the reasons why they have been touted as superior to Markov networks (see [89, Section 3.1.3] or [72, pp. 160, 164, 180]). We have taken advantage of the ability to represent both functional and probabilistic dependencies in our networks, but with mixed results.

Our networks mix explanatory and taxonomic inference. We need taxonomic inference because we may need to refine our hypotheses about the types of objects and actions in stories. For example, when reading the story "Jack wanted to kill himself. He got a rope." Wimp3 needs to consider the possibility that Jack's **kill-** plan is, more specifically, a **hanging-** plan.

In FRAIL, information about the type of an object is represented by **inst** statements, *e.g.,* **(inst kill24 kill-)**. We need taxonomic inference for cases where we postulate more than one **inst** statement about an object.

Since belief networks are capable of representing functional dependencies, it should not be difficult to arrange a set of **inst** statements into a network. There are two obvious

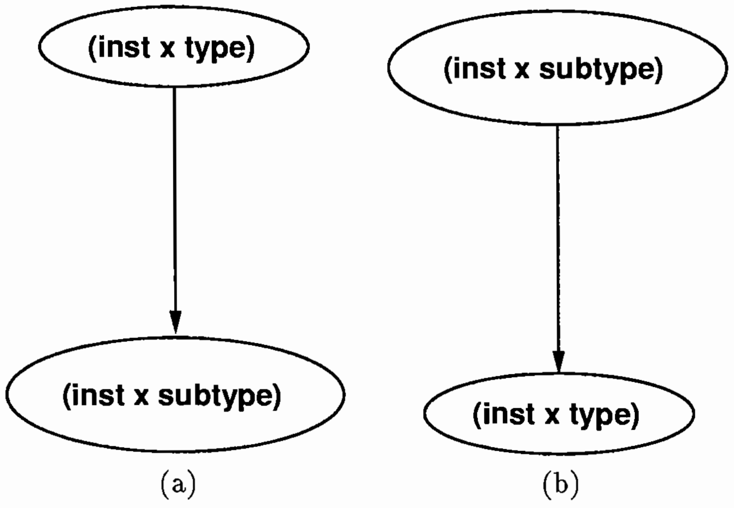


Figure 10.1: Two possible arrangements of **inst** nodes.

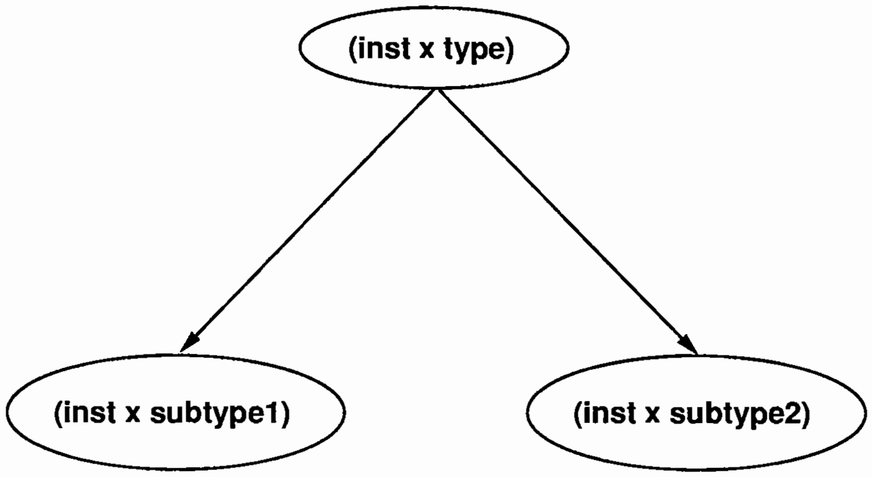


Figure 10.2: A type with two subtypes, drawn according to the scheme of Figure 10.1(a).

choices: either statements about the subtypes of an object of a given type should go above or below statements about the type (see Figure 10.1).

In Figure10.1(a), the statement about the type of an object is shown above statements about the subtype of an object. Now, since we have defined a distribution over our isa-hierarchy, it is simple to quantify this subnetwork: The prior for the type comes from this distribution. The probability of the subtype statement being true given the type statement is $\frac{P(subtype)}{P(type)}$, and $P(subtype \mid not(type)) = 0$.

The difficulty with this network topology comes when combining statements about more than one subtype of a given type. See Figure10.2. Here if we extend the same approach, we are assuming that whether x is of *subtype*1 is independent of whether it is of *subtype*2, given that it is of *type*. But this is clearly untrue, since *subtype*1 and *subtype*2 are mutually exclusive hypotheses. So this topology is not suitable.

The network may equally well be formulated as in Figure 10.1(b), with the subtype over the supertype. Here $P(subtype)$ comes from the isa-hierarchy definition. $P(type \mid subtype) = 1$ and $P(type \mid not(subtype)) = P(type) - P(subtype)$. With this scheme, we do not run into any troubles in taxonomic inference, since we can exclude the possibility of x

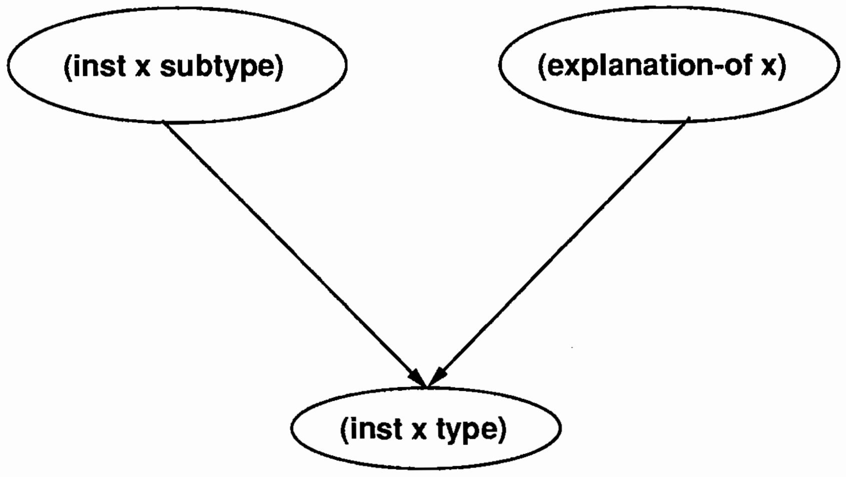


Figure 10.3: An example where explanatory links are combined with taxonomic ones.

being both fish and fowl by setting $P(type \mid subtype1, subtype2) = 0$.

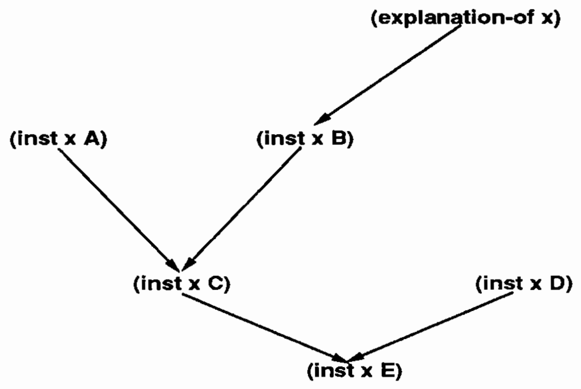
The difficulty comes when combining explanatory inference with taxonomic inference. Consider the diagram given as Figure 10.3. Here we have reason to believe that x is an instance of *subtype*. We also have an explanation which predicts that x is an instance of *type*. This kind of situation occurs frequently in Wimp3. For example, when Wimp3 reads the sentence "Jack took the bus to the supermarket." it postulates a **supermarket-shop-** explanation for the story. This explanation predicts that there will be a **go-**, a supertype of the actually-occurring **bus-trip-**.

How are we to assign the conditional probability matrix at **(inst x type)**? In particular, what is the correct value for $P(type \mid subtype, not(explanation))$ and $P(type \mid subtype, explanation)$? Clearly, if the object is of *subtype*, it must be of *type*. Ergo, the correct conditional probability is one. However, this has the effect of making the explanation independent of the type assertion, given the subtype assertion.

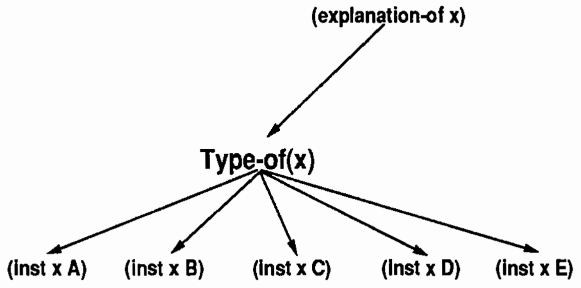
In the event, we have had to take a different approach, which essentially removes the taxonomic inference from our belief networks. What we have done is to introduce new random variables for the types of objects. The details of this transformation have been suppressed in the earlier discussion of Wimp3. Because this is a recent change to the probabilistic models, it has not yet been incorporated in the network-construction rules. Instead, the belief networks generated by the rules are rearranged in the gateway component, which translates FRAIL belief networks into IDEAL data structures (see p. 82 for more discussion of this part of Wimp3).

The rules generate networks which are arranged as per Figure 10.1(b): inverted trees of **inst** statements with subtypes represented above supertypes. The gateway component introduces a new random variable, $type-of(x)$. This state space of this node is, abstractly speaking, the set of leaves of the isa-hierarchy. In fact, of course, we do not represent this whole space explicitly; it is far too large. What we do is divide it up into relevant and irrelevant portions. Then the **inst** statements in the FRAIL network are translated into IDEAL nodes. Each of these nodes has the type-of node as its sole parent. The explanation nodes are made parents of the type-of node. An example of this translation is given in Figure 10.4.

We have not incorporated these new nodes into the rules of Wimp3 is that FRAIL does not, at the moment, have expressive enough rules. We are currently working on an extension



(a)



(b)

	other	$E \backslash D \cup C$	D	$C \backslash B \cup A$	B	A
E		x	x	x	x	x
D			x			
C				x	x	x
B					x	
A						x

Figure 10.4: The translation of inst nodes: The original graph (generated by Wimp3's rules) is given in (a). The translation into IDEAL nodes is shown as (b). The state-space of the node type-of(x) is given in (c). In the table the elements of the state space are shown across the top. Below are the 5 types, and the values of type-of(x) which make them true.

of the network-construction language which will allow us to specify large state-spaces, only small parts of which are of interest. FRAIL will detect, based on the parents and children of such nodes, how to divide the state space into interesting regions. We are trying to decide what other uses we might have for this facility and design the extension to FRAIL.

10.3 Discourse structure

At the moment, Wimp3 essentially treats a paragraph as an unordered bag of sentences. This makes it very difficult for Wimp3 to understand even a very simple story like: "Jack went to the liquor-store. Jack went to the supermarket."[1] What happens here is that Wimp3's marker-passer has two objects and two actions, and hooks them up in all possible ways. The equality hypotheses spawned as a result yield a network of about 130 nodes, the largest we have encountered, and one which is so bushy that it takes about two hours to evaluate.

This account is actually oversimplified to our discredit. Carroll has since improved the marker-passer to eliminate much of the unnecessary search in this example. Nevertheless, the general problem remains – Wimp3 just simply does not have enough information about the structure of discourse.

We have mentioned discourse structure briefly before this, when discussing the probabilistic model (see p. 57). There has been a fair amount of work on discourse theory in AI (see the review in [2, Chapter 14] or [45, 46, 61, 62, 77, 94]). Most of these approaches have suggested some constraints on the search for referents. We hope that it will be possible to give a crisper definition to these constraints in terms of conditioning, and that they will help us limit our processing.

Incorporating a good theory of discourse structure and discourse planning into Wimp3's model would answer an objection Norvig and Wilensky have to our approach [87]. They phrase what we believe to be the same argument in two different ways. First of all, they argue that basing a model of descriptions of actions on the frequency with which actions occur in the real world is incorrect. We agree with this assessment – we have simply used this approach as a simplification. We are also interested in a model that takes into account the intentions of the language-user.

10.4 Time

A common disclaimer by an AI author is that he has neglected temporal considerations to avoid complication. The implication is nearly made that adding a temporal dimension to the research (on engineering, medical diagnosis, etc.) would be a familiar but tedious exercise that would obscure the new material presented by the author. Actually, of course, no one has ever dealt with time correctly in an AI program, and there is reason to believe that doing it would change everything. — Drew McDermott [80].

We are as guilty of the offense of neglecting time as any other AI authors. This is an important shortcoming of Wimp3, and one which is related to our ignoring issues of discourse structure. The only mitigating circumstance that we can plead is that ignoring time is by now conventional in NLP and particularly in story understanding. Early work in

[1]We are indebted to Glenn Carroll for this example.

story understanding, *e.g.,* [30, 31, 110], simply assumed that narrative time moved forward with the text – that events were named one after another as they occurred.

There has been some recent work on reasoning about time in natural language texts, and an obvious next step in this, or any other story understanding project, would be to try to adapt that work. There has been some work on inferring relations between narrative time, and the time of utterance, based on textual cues [47, 109]. Schubert has recently suggested a knowledge representation scheme designed to address the problem of representing the flow of events in narratives [102]. We have hopes that in exchange for the increased complexity of the representation, we would be able to limit our search to remove some implausible interpretations of texts like the one referred to above (p. 128).

10.4.1 Bottom-up search

One problem which limits Wimp3's ability to understand stories is that our approach is still primarily a bottom-up one of explaining the input. This makes it impossible for Wimp3 to understand a story like:

> The detective was tailing the suspect. He went into a hardware store. He saw that the suspect was looking at him. He bought some rope. – Larry Birnbaum[2]

In this example the detective's buying the rope cannot properly be understood without simulating the detective's planning. Furthermore, this planning is based on the detective's inferences about the suspect's beliefs. We have little to say about this issue, except to note that it poses difficulties for *any* story understanding program based on a schema-recognition approach; it is orthogonal to the issues which concern us in this thesis. For more discussion of issues of mixing planning and understanding, see Birnbaum's thesis [4].

10.5 Computing the probabilities

In this section we discuss two aspects of the network evaluation component of Wimp3. We discuss methods for improving the speed of this component, which takes most of its run-time. We also discuss the pro's and con's of a different approach to network evaluation, finding MAP instantiations of the belief networks. We argue that the ideal architecture would be one which is able to both find MAP instantiations *and* posterior distributions. Finally, we discuss the prospect of alternatives to full-blown probability theory.

10.5.1 Performance

The network evaluation component consumes the vast majority, by an order of magnitude at least, of the run-time of Wimp3. We have made great strides in speeding up this part of Wimp3, building on the fact that there has been a lot of work on improving belief network evaluation algorithms in the past several years. We have also been able to speed up network evaluation by changing the structure of the networks we create. The process of committing to categorical beliefs has helped us greatly. Perhaps this could be improved. Finally, it is possible that we could improve Wimp3's performance by postponing belief net evaluation until it will make a difference.

For improvements in the algorithm itself, we look with interest at recent work by Cooper on applying recursive decomposition to belief networks [26]. This technique can be combined with other network evaluation algorithms to improve their performance.

[2]Reconstructed to the best of our memory.

As far as improving the models is concerned, so far we have had success introducing intermediate variables to make the networks less fully-connected. The **slot-inst**, **same-type** and **diamond-center** nodes are examples of this (see Chapter 7). It is possible that adding further variables particularly as parents of equality nodes, will be helpful.

Other speed-ups may be had from improving the process by which Wimp3 commits to beliefs. What happens now is that we examine nodes some fixed time after their creation. If they are believed to be some value with high probability, we accept that as categorically true. This was discussed in more detail in Chapter 6.

One possible improvement would be to make the commitment process sensitive to the types of the nodes involved. Wimp3 would commit more quickly to "lower-level" statements like word-sense hypotheses, and less quickly to plan-recognition hypotheses. If we had a way of detecting incorrect commitments, we could add the ability to backtrack over commitments, and could be less conservative about making them.

A final area of improvement would be to be more intelligent about when to evaluate the networks. For example, we evaluate the network right after reading a pronoun at the start of a sentence (see pp. 111ff. for an example). This processing has no effect on Wimp3's interpretation of the sentence. This is because any assumptions made here can later be overridden, and because the part of the network concerned with pronoun reference does not yet link up with the rest of the belief network. This evaluation could be skipped. We could also postpone reasoning about verbs until we have parsed all of their arguments. Consider the first of the sentences discussed in Chapter 8: "Jack gave the busdriver a token." If we postponed network evaluation until the end of the sentence, we could avoid considering the possibility that 'the busdriver' is the direct object of 'gave,' since that possibility is ruled out on syntactic grounds.

10.5.2 Posteriors vs. MAPs

One possible alternative to solving the belief networks for posterior distributions would be to find maximum a posteriori (MAP) instantiations for the networks. The MAP of a network is the instantiation of the network – the single element of the global state space – which has the highest probability, conditioned on the evidence.

There are two reasons why we haven't used MAP estimation in Wimp3. First of all, when we began this project, there was no fast MAP algorithm for multiply-connected belief networks, and little research being done on the issue, while there has been an explosion of work on calculating posterior distributions.

Secondly, MAP estimates are not good for the kind of incremental results we want. This is because knowing that a random variable has a particular value in the MAP instantiation of the network tells us very little about the posterior probability of that random variable having that value. It may be that all values of that random variable are approximately equally likely. An MAP will then represent an essentially arbitrary choice of value for that node. Now, if the arbitrarily-assigned node represents an hypothesis about the sense of the word we've just read, we don't want to make any immediate decisions based on that choice, because the immediately-following input may change our decision.

A related counter-argument is offered by Pearl, who shows that MAP interpretations (which he refers to as MPE's – most probable explanations) are too sensitive to the structure of the networks:

> Suppose, for example, that I am concerned about having a certain fatal disease, and by a stroke of good luck, a medical test reveals that there is an 80% chance that I am totally healthy. According to the MPE acceptance rule, I should

> commit all my belief to a world model in which I am health. So, I start imagining all kinds of possible scenarios....I imagine 10 mutually exclusive scenarios, $S_1, S_2, \ldots, S_{10}$..., each having a probability 0.1 of being realized. Now I repeat the MPE exercise, but this time on a larger scale, embracing the earlier facts about the disease and the newly imagined scenarios. Lo and behold, any world model in which I am healthy now receives only an 8% chance of being realized, so the most probable "explanation" of the evidence is that I *do* suffer from that horrible disease–and all for being a hasty daydreamer! [89, p. 285]

The only argument for MAPs that we can see is the speculation that they could provide a faster evaluation process. It is unlikely this is *in general* true, since finding an MAP instantiation is also an NP-hard problem [25, 27]. However, Charniak and Shimony are currently working on developing a variant of MAP estimation for Wimp3, based on best-first search [21, 104]. This work attempts to overcome some of the limitations of MAPs by allowing some nodes in the networks to remain 'uncommitted.'

10.6 Alternatives to probability theory

We are often asked why we don't use some approximation to probability theory, especially since we argue that the absolute value of the probabilities in our models are not relevant. The reason is that it follows from the proof of the rationality of probability theory [29], that any alternative will not have all of the desirable properties of probability theory.

So any alternative will, at best, be a trade-off between correctness of results and ease of computation. Now, if we could characterize the kinds of networks we will be constructing and evaluating in the process of story understanding, such a trade-off might be desirable. It would then be at least theoretically possible to find an alternative which would get good enough answers on our problems.

It seems to us to be premature to make this kind of trade-off. Even if our approach proved to be entirely correct, which seems unlikely, we would still want to make sure the details of the model were correct before investing too much time in honing its performance. These details would involve altering the distributions, and altering the terms of any accuracy–speed tradeoff. Until such time as we have a much firmer foundation, the important consideration is to keep Wimp3's calculations accurate, and just fast enough that it be a convenient experimental framework.

10.7 Summary

The problems discussed in the last sections are, for the most part, ones which follow from the unique properties of the work described here. As such they do not begin to exhaust the difficulties confronting the natural language understanding researcher. The problems of knowledge representation, search, and the idiosyncracies of language in general, mock us all. The work described here has not lessened them, or not by much.

The contribution of this work is, rather, in how to choose the best interpretation of a text. We have recast this problem as a problem in probability theory, and have, as a result, gathered various benefits which probability theory is heir to.

First, any natural language text is, inherently, underconstrained in its interpretation, at least from a formal point of view. Texts typically have enormous numbers of interpretations. Typically people view all but one or two as incredibly unlikely, but how to make this judgment has proved difficult. Our approach sheds some light on this.

Second, typically the interpretations of different portions of the text interact. The plan-recognition problem will typically interact with the word-sense ambiguity problem, the case determination problem and the pronoun resolution problem, and these latter three will interact among themselves. In many approaches such interactions lead either to circularity in the rules (e.g., we will have rules which say to use the noun to determine the verb case, and use the verb case to determine the noun word-sense), or to difficult search problems (which rule to use first) or both. Furthermore, the sources of information we have in each of these areas will be completely different. As we said earlier, probability theory gives us a common currency in which the different sources of information can be combined into a single interpretation.

Third, even approaches which try to solve these problems by recasting them as one in constraint satisfaction still fall to pieces when the constraints are not absolute, but soft, as they so often are in NLU work. This is not a problem for our approach.

Fourth, because probability theory in general, and belief networks in particular, are well understood, commonly used, normative procedures, we have been able to hook our wagon to the efforts of a large number of researchers who are interested in extending the approach and making it more efficient. This makes it possible to share ideas, and even code. A large portion of the code used in this thesis was written by other people at other institutions. How often can this be said?

Lastly, and this is still speculative, probability theory gives one a new body of mathematics which has yet to be applied to the problems of natural language understanding. This is most obvious in the case of search, which is, as we mentioned earlier, a common problem which confronts all NLU researchers. Having recast NLU problems as ones in probability theory suggests that one could now, say, evaluate a search technique by how likely it is to change the probabilities involved. Or one might add on expected costs for the search and apply utility theory. To us, at least, probability theory opens a fresh window onto some not-so-fresh problems.

Appendix A

Test queries for the training set.

1. "Jack went to the supermarket. He found some milk on the shelf. He paid for it."

   ```
   (and (inst ?jack jack-)
        (inst ?sming superming)
        (inst ?went go-)
        (inst ?find locate-)
        (inst ?pay pay-)
        (inst ?milk milk-)
        (== '(agent ?sming) ?jack)
        (== '(go-stp ?sming) ?went)
        (== '(pay-stp ?sming) ?pay)
        (== '(bought ?sming) ?milk)
        (== '(loc-stp ?sming) ?find))
   ```

   *There is a supermarket-shopping plan (*superming*) with Jack is its agent, and the actions mentioned in the story (going to the supermarket, finding milk on the shelf, and paying for 'it') all play roles in this plan. Furthermore, the 'it' paid for in the story is the milk, which is the thing for which Jack went shopping.*

2. "Bill went to the supermarket. He paid for some milk."

   ```
   (and (inst ?bill bill-)
        (inst ?sming superming)
        (inst ?went go-)
        (inst ?pay pay-)
        (inst ?milk milk-)
        (== '(agent ?sming) ?bill)
        (== '(go-stp ?sming) ?went)
        (== '(pay-stp ?sming) ?pay)
        (== '(bought ?sming) ?milk))
   ```

3. "Jack gave the busdriver a token. He got off at the supermarket."

   ```
   (and (inst ?jack jack-)
        (inst ?sming superming)
        (inst ?went bus-trip)
        (== '(agent ?sming) ?jack)
        (== '(go-stp ?sming) ?went))
   ```

In the absence of other information, Wimp3 is to assume that going to the supermarket is probably done in order to go shopping.

4. "Jack got off the bus at the liquor-store. He pointed a gun at the owner."

```
(and (inst ?jack jack-)
     (inst ?rob rob-)
     (inst ?ls liquor-store-)
     (inst ?went go-)
     (inst ?gt gun-threaten-)
     (inst ?point point-)
     (not (== '(owner ?ls) ?jack))
     (== '(agent ?rob) ?jack)
     (== '(go-stp ?rob) ?went)
     (== '(patient ?rob) ?ls))
```

Although Wimp3 assumes that people usually go to the liquor-store in order to buy alcoholic beverages, given additional evidence, it can revise its interpretation.

It must Recognize that the bus-trip was part of Jack's plan to rob the liquor store.

The negated clause is a check to make sure that Wimp3 is correctly rejecting the hypothesis that 'the owner' refers to Jack.

5. "Jack went to the liquor-store. He found some bourbon on the shelf."

```
(and (inst ?jack jack-)
     (inst ?lsing liquor-shop)
     (inst ?went go-)
     (inst ?find locate-)
     (inst ?bourbon bourbon-)
     (== '(agent ?lsing) ?jack)
     (== '(go-stp ?lsing) ?went)
     (== '(bought ?lsing) ?bourbon)
     (== '(loc-stp ?lsing) ?find))
```

This story and the following further illustrate Wimp3's probabilistic updating of its beliefs in the face of accumulating evidence.

6. "Bill went to the liquor-store. He pointed a gun at the owner."

```
(and (inst ?bill bill-)
     (inst ?rob rob-)
     (inst ?ls liquor-store-)
     (inst ?went go-)
     (inst ?gt gun-threaten-)
     (inst ?point point-)
     (not (== '(owner ?ls) ?bill))
     (== '(agent ?rob) ?bill)
     (== '(go-stp ?rob) ?went)
     (== '(patient ?rob) ?ls))
```

We check to make sure that Wimp3 doesn't confuse Bill and the owner of the liquor store. This requires combining syntactic and plan-based information. 'He' must be Bill, because Bill is the agent of the robbery, and so the owner cannot be Bill, because of the restrictions on well-formed reflexive sentences (cf. the sentence "Bill pointed a gun at himself.")

7. "Bill gave the busdriver a token."

```
(and (inst ?bill bill-)
     (inst ?b-t bus-trip)
     (inst ?gave give-)
     (== '(agent ?b-t) ?bill)
     (== '(fare-stp ?b-t) ?gave))
```

8. "Fred robbed the liquor-store. Fred pointed a gun at the owner."

```
(and (inst ?fred fred-)
     (inst ?rob rob-)
     (inst ?ls liquor-store-)
     (inst ?gt gun-threaten-)
     (inst ?point point-)
     (not (== '(owner ?ls) ?fred))
     (== '(agent ?rob) ?fred)
     (== '(threaten-stp ?rob) ?gt)
     (== '(point-stp ?gt) ?point)
     (== '(patient ?rob) ?ls))
```

9. "Bill got a gun. He went to the supermarket."

```
(and (inst ?bill bill-)
     (inst ?rob rob-)
     (inst ?sm super-m)
     (inst ?went go-)
     (inst ?get obtain-)
     (inst ?gun gun-)
     (inst ?gt gun-threaten-)
     (not (== '(owner ?sm) ?bill))
     (== '(agent ?rob) ?bill)
     (== '(go-stp ?rob) ?went)
     (== '(patient ?rob) ?sm)
     (== '(threaten-stp ?rob) ?gt)
     (== '(instr ?gt) ?gun)
     (== '(arming ?gt) ?get))
```

Wimp3 should infer from this story that Bill is getting a gun in order to be able to threaten the owner of the supermarket, as part of a robbery plan. Wimp3 has robbery on its mind a bit – it leaps to this conclusion because it does not know many reasons for getting a gun (e.g., self-protection) that are consistent with the trip to the supermarket.

10. "Fred went to the supermarket. He pointed a gun at the owner. He packed his bag. He went to the airport."

```
(and (inst ?fred fred-)
     (inst ?rob rob-)
     (inst ?sm super-m)
     (inst ?went go-)
     (inst ?pack pack-)
     (inst ?went2 go-)
     (inst ?airport airport-)
     (inst ?gun gun-)
     (inst ?gt gun-threaten-)
     (inst ?air-trip air-trip)
     (not (== '(owner ?sm) ?fred))
     (== '(agent ?rob) ?fred)
     (== '(go-stp ?rob) ?went)
     (== '(patient ?rob) ?sm)
     (== '(threaten-stp ?rob) ?gt)
     (== '(instr ?gt) ?gun)
     (== '(agent ?air-trip) ?fred)
     (== '(go-stp ?air-trip) ?went2)
     (== '(destination ?went2) ?airport)
     (== '(pack-stp ?air-trip) ?pack))
```

11. "Jack took the bus to the airport. He bought a ticket."

```
(and (inst ?jack jack-)
     (inst ?took bus-trip)
     (inst ?bus bus-)
     (inst ?airport airport-)
     (inst ?air-trip air-trip)
     (inst ?bought buy-)
     (inst ?ticket ticket-)
     (== '(agent ?air-trip) ?jack)
     (== '(go-stp ?air-trip) ?took)
     (== '(patient ?took) ?bus)
     (== '(source ?air-trip) ?airport)
     (== '(buy-stp ?air-trip) ?bought)
     (== '(fare-of ?air-trip) ?ticket))
```

12. "Bill packed a suitcase. He went to the airport."

```
(and (inst ?bill bill-)
     (inst ?went go-)
     (inst ?packed pack-)
     (inst ?suitcase luggage-)
     (inst ?airport airport-)
     (inst ?air-trip air-trip)
     (== '(agent ?air-trip) ?bill)
     (== '(go-stp ?air-trip) ?went)
     (== '(pack-stp ?air-trip) ?packed)
     (== '(source ?air-trip) ?airport)
     (== '(patient ?packed) ?suitcase))
```

13. "Jack got on a bus. He got off at the park. Jack went to the supermarket."

```
(and (inst ?jack jack-)
     (inst ?bus bus-)
     (inst ?got mount-)
     (inst ?bt bus-trip)
     (inst ?pt park-trip)
     (inst ?park nature-park-)
     (inst ?went go-)
     (inst ?sm superming)
     (inst ?smarket super-m)
     (== '(go-stp ?pt) ?bt)
     (== '(park-of ?pt) ?park)
     (== '(go-stp ?sm) ?went)
     (== '(store-of ?sm) ?smarket)
     (== '(agent ?pt) ?jack))
```

The last conjunct is to check to see that Wimp3 has made the assumption that the two uses of the name 'Jack' refer to the same person.

14. "Jack gave the busdriver a token. He got off at the park. He went to the airport. He got on a plane."

```
(and (inst ?jack jack-)
     (inst ?give give-)
     (inst ?bt bus-trip)
     (inst ?pt park-trip)
     (inst ?park nature-park-)
     (inst ?went go-)
     (inst ?at air-trip)
     (inst ?airport airport-)
     (inst ?plane airplane-)
     (== '(fare-stp ?bt) ?give)
     (== '(go-stp ?pt) ?bt)
     (== '(park-of ?pt) ?park)
     (== '(go-stp ?at) ?went)
     (== '(source ?at) ?airport)
     (== '(agent ?at) ?jack)
     (== '(agent ?pt) ?jack)
     (== '(patient ?at) ?plane))
```

15. "Fred sat down on the bus. He went to the supermarket."

```
(and (inst ?fred fred-)
     (inst ?sming superming)
     (inst ?went bus-trip)
     (== '(agent ?sming) ?fred)
     (== '(go-stp ?sming) ?went))
```

16. "Jack went to a restaurant. He got a milkshake."

```
(and (inst ?jack jack-)
     (inst ?eat-out eat-out)
     (inst ?milkshake milk-shake)
     (inst ?got obtain-)
     (inst ?went go-)
     (word-inst ?he ?x pronoun he)
     (== '?he ?jack)
     (== '(go-stp ?eat-out) ?went)
     (== '(bought ?eat-out) ?milkshake)
     (== '(get-stp ?eat-out) ?got))
```

We check to make sure that Wimp3 recognizes that 'he' refers to Jack.

17. "Bill drank a milkshake with a straw."

```
(and (inst ?bill bill-)
     (inst ?milkshake milk-shake)
     (inst ?drank straw-drink)
     (inst ?straw drink-straw)
     (== '(agent ?drank) ?bill)
     (== '(patient ?drank) ?milkshake)
     (== '(instr ?drank) ?straw))
```

We make sure that Wimp3 knows that the straw concerned is not `animal-straw` *(the kind you find on a farm).*

18. "Fred got off the bus at a restaurant. He got a milkshake."

```
(and (inst ?fred fred-)
     (inst ?eat-out eat-out)
     (inst ?milkshake milk-shake)
     (inst ?got obtain-)
     (inst ?went bus-trip)
     (word-inst ?he ?x pronoun he)
     (== '?he ?fred)
     (== '(go-stp ?eat-out) ?went)
     (== '(bought ?eat-out) ?milkshake)
     (== '(get-stp ?eat-out) ?got))
```

19. "Janet put a straw in a milkshake."

```
(and (inst ?janet janet-)
     (inst ?milkshake milk-shake)
     (inst ?drank straw-drink)
     (inst ?straw drink-straw)
     (inst ?put put-)
     (== '(agent ?drank) ?janet)
     (== '(patient ?drank) ?milkshake)
     (== '(instr ?drank) ?straw)
     (== '(put-stp ?drank) ?put))
```

In order for Wimp3 to understand this story, it must know that the straw is the instrument of the drinking action (rather than the co-agent).

20. "Bill got on a bus. He got off at a restaurant. He drank a milkshake with a straw."

```
(and (inst ?bill bill-)
     (inst ?eat-out eat-out)
     (inst ?milkshake milk-shake)
     (inst ?drink straw-drink)
     (inst ?went bus-trip)
     (== '(agent ?eat-out) ?bill)
     (== '(go-stp ?eat-out) ?went)
     (== '(bought ?eat-out) ?milkshake)
     (== '(eat-stp ?eat-out) ?drink))
```

21. "Bill took a bus to a restaurant. He drank a milkshake. He pointed a gun at the owner. He got some money from him."

```
(and (inst ?bill bill-)
     (inst ?eat-out eat-out)
     (inst ?milkshake milk-shake)
     (inst ?rest restaurant-)
     (inst ?got obtain-)
     (inst ?went bus-trip)
     (inst ?rob rob-)
     (inst ?money money-)
     (== '(agent ?eat-out) ?bill)
     (== '(go-stp ?eat-out) ?went)
     (== '(bought ?eat-out) ?milkshake)
     (== '(loot-stp ?rob) ?got)
     (== '(agent ?rob) ?bill)
     (== '(patient ?rob) ?rest)
     (== '(swag-of ?rob) ?money))
```

There is a problem with Wimp3's understanding of this story. It ought to have in its database that `(== '(go-stp ?rob) ?went)`, *but does not. This is because it already believes that this going is explained by Bill's plan to eat at the restaurant, and does not see that this action serves two purposes. So Wimp understands this story as 'Bill went out to eat and then robbed the restaurant.' rather than 'Bill planned to go to the restaurant to eat some food and commit an armed robbery.'*

22. "Fred gave the busdriver a token. He got off the bus at the park. He went to a restaurant. He got some money from the owner."

```
(and (inst ?fred fred-)
     (inst ?pt park-trip)
     (inst ?rest restaurant-)
     (inst ?got obtain-)
     (inst ?bt bus-trip)
     (inst ?rob rob-)
```

```
          (inst ?went go-)
          (inst ?money money-)
          (== '(agent ?pt) ?fred)
          (== '(go-stp ?pt) ?bt)
          (== '(agent ?rob) ?fred)
          (== '(patient ?rob) ?rest)
          (not (== '?went ?bt))
          (== '(go-stp ?rob) ?went)
          (== '(swag-of ?rob) ?money))
```

23. "Jack took a taxi to the park."

```
     (and (inst ?jack jack-)
          (inst ?taxi taxi-)
          (inst ?tt taxi-trip)
          (inst ?pt park-trip)
          (== '(agent ?pt) ?jack)
          (== '(patient ?tt) ?taxi)
          (== '(go-stp ?pt) ?tt))
```

24. "Bill took a taxi."

```
     (and (inst ?bill bill-)
          (inst ?taxi taxi-)
          (inst ?tt taxi-trip)
          (== '(agent ?tt) ?bill)
          (== '(patient ?tt) ?taxi))
```

25. "Fred took a taxi to the bus-station. He got on a bus."

```
     (and (inst ?fred fred-)
          (inst ?taxi taxi-)
          (inst ?tt taxi-trip)
          (inst ?bt bus-trip)
          (inst ?bs bus-station-)
          (== '(agent ?bt) ?fred)
          (== '(patient ?tt) ?taxi)
          (== '(go-stp ?bt) ?tt)
          (== '(source ?bt) ?bs)
          (word-inst ?he ?x pronoun he)
          (== '?he ?fred))
```

Bibliography

[1] Agogino, Alice M. and Rge, Ashutosh, IDES: Influence Diagram Based Expert System, *Mathematical Modelling*, 227–233, (Pergamon Press, 1987), 227–233.

[2] Allen, James, *Natural Language Understanding*, (Benjamin / Cummings Publishing Company, Menlo Park, California, 1987).

[3] Bacchus, Fahiem, Statistically Founded Degrees of Belief, *Proceedings of the Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence*, Morgan Kaufmann Publishers, Inc., 1988, 59–66.

[4] Birnbaum, Lawrence, *Integrated Processing in Planning and Understanding*, Technical Report 489, Yale University Department of Computer Science, December 1986.

[5] Brachman, Ronald J., On the epistemological status of semantic networks, Findler, N.V., (Ed.), *Associative Networks: Representation and Use of Knowledge by Computers*, 3–50, (Academic Press, New York, 1979), 3–50. Republished in [6].

[6] Brachman, Ronald J. and Levesque, Hector J., (Eds.), *Readings in Knowledge Representation*, (Morgan Kaufmann Publishers, Inc., 1985).

[7] Breese, John S., Construction of belief and decision networks, forthcoming, 1989.

[8] Bundy, Alan, Incidence Calculus: A Mechanism for Probabilistic Reasoning, In Lemmer and Kanal [76], 177–184.

[9] Carroll, Glenn and Charniak, Eugene, Finding Plans with a Marker-Passer, *Proceedings of the Plan-recognition workshop*, 1989.

[10] Chang, Kuo-Chu and Fung, Robert, Node Aggregation for Distributed Inference in Bayesian Networks, *Proceedings of the 11th International Joint Conference on Artificial Intelligence*, 1989, 265–270.

[11] Charniak, Eugene, On the use of framed knowledge in language comprehension, *Artificial Intelligence*, **11** (1978) 225–265.

[12] Charniak, Eugene, A common representation for problem solving and language comprehension information, *Artificial Intelligence*, **16** (1981) 225–255.

[13] Charniak, Eugene, Context Recognition, In Lehnert and Ringle [75].

[14] Charniak, Eugene, The Bayesian basis of common-sense medical diagnosis, *Proceedings of the Second National Conference on Artificial Intelligence*, 1983, 70–73.

[15] Charniak, Eugene, A neat theory of marker passing, *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986, 584–589.

[16] Charniak, Eugene, Gavin, Michael, and Hendler, James, *Frail 2.1, Some Documentation*, Technical report, Brown University Department of Computer Science, 1982.

[17] Charniak, Eugene and Goldman, Robert P., A logic for semantic interpretation, *Proceedings of the Annual Meeting of the Association for Computational Linguistics*, 1988, 87–94.

[18] Charniak, Eugene and Goldman, Robert P., Plan Recognition in Stories and in Life, *Proceedings of the Workshop on Uncertainty and Probability in AI*, Morgan Kaufmann Publishers, Inc., 1989.

[19] Charniak, Eugene and McDermott, Drew, *Introduction to Artificial Intelligence*, (Addison Wesley, 1985).

[20] Charniak, Eugene, Riesbeck, Christopher K., and McDermott, Drew V., *Artificial Intelligence Programming*, (Lawrence Erlbaum, Hillsdale, N.J., 1980).

[21] Charniak, Eugene and Shimony, Solomon, *Probabilistic semantics for Cost-based Abduction*, Technical Report CS-90-02, Brown University Department of Computer Science, 1990.

[22] Chavez, R. Martin and Cooper, Gregory F., An Empirical Evaluation of a Randomized Algorithm for Probabilistic Inference, In Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence [91], 60–70.

[23] Chin, Homer L. and Cooper, Gregory F., Stochastic simulation of bayesian belief networks, *Proceedings of the Workshop on Uncertainty and Probability in Artificial Intelligence*, 1987, 106–113.

[24] Clocksin, W. and Mellish, C., *Programming in Prolog*, (Springer-Verlag, New York, 1981).

[25] Cooper, Gregory F., *Probabilistic inference using belief networks is NP-hard*, Technical Report KSL-87-27, Medical Computer Science Group, Stanford University, 1987.

[26] Cooper, Gregory F., Bayesian Belief-Network Inference Using Recursive Decomposition, Forthcoming., 1990.

[27] Cooper, Gregory F., The Computational Complexity of Probabilistic Inference Using Bayesian Belief Networks, *Artificial Intelligence*, **42**(2–3) (1990) 393–405.

[28] Cooper, Gregory Floyd, *NESTOR: A Computer-Based Medical Diagnosis Aid that Integrates Causal and Probabilistic Knowledge*, PhD thesis, Stanford University, 1984.

[29] Cox, R.T., Probability, frequency and reasonable expectation, *American Journal of Physics*, 14 (1946) 1–13, Republished in [103].

[30] Cullingford, Richard E., *Script Application: Computer Understanding of Newspaper Stories*, Technical report, Yale University Department of Computer Science, 1978.

[31] Cullingford, Richard E., SAM, Schank, Roger C. and Reisbeck, Christopher, (Eds.), *Inside Computer Understanding*, 75–89, (Lawrence Erlbaum Associates, Hillsdale, 1981), 75–89.

[32] D'Ambrosio, Bruce, A Hybrid approach to reasoning under uncertainty, *International Journal of Approximate Reasoning*, **2** (1988) 29–45.

[33] D'Ambrosio, Bruce, Process, structure and modularity in reasoning with uncertainty, *The Fourth Workshop on Uncertainty in Artificial Intelligence*, 1988, 64–72.

[34] Dasigi, Venugopala R., *Word Sense Disambiguation in Descriptive text interpretation*, PhD thesis, Department of Computer Science, University of Maryland at College Park, 1988.

[35] Dasigi, Venugopala R. and Reggia, James A., Parsimonious covering as a method for natural language interfaces to expert systems, *Artificial Intelligence in Medicine*, **1** (1989) 49–60.

[36] DeJong, Gerald, An Overview of the FRUMP system, In Lehnert and Ringle [75], 149–176.

[37] deKleer, Johan, An assumption-based TMS, *Artificial Intelligence*, **28** (1986) 127–162.

[38] deKleer, Johan, Extending the ATMS, *Artificial Intelligence*, **28** (1986) 163–196.

[39] deKleer, Johan, Problem-solving with the ATMS, *Artificial Intelligence*, **28** (1986) 197–224.

[40] Doyle, Jon, *Truth Maintenance Systems for Problem Solving*, Technical report, MIT Artificial Intelligence Laboratory, 1978.

[41] Doyle, Jon, A truth maintenance system, *Artificial Intelligence*, **12**(3) (1979) 231–272.

[42] Geman, Stuart and Geman, Donald, Stochastic relaxation, Gibbs distributions and the Bayesian restoration of images, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, **6** (1984) 721–741, Republished in [103].

[43] Goldman, Robert P. and Charniak, Eugene, A probabilistic ATMS for plan recognition, *Proceedings of the Plan-recognition workshop*, 1988.

[44] Grenander, Ulf, *Tutorial in Pattern Theory*, Technical report, Division of Applied Mathematics, Brown University, Providence, RI 02912, December 1983.

[45] Grosz, Barbara J., Discourse knowledge, Walker, D., (Ed.), *Understanding Spoken language*, 229–347, (North-Holland, New York, 1978), 229–347.

[46] Grosz, Barbara J. and Sidner, Candace, Attention, Intention and the Structure of Discourse, *Computational Linguistics*, **12** (1986).

[47] Harper, Mary P. and Charniak, Eugene, Time and Tense in English, *Proceedings of the 24th Annual Meeting of theAssociation for Computational Linguistics*, 1986, 3–9.

[48] Hayes, Patrick, *The Naive Physics Manifesto*, Technical report, University of Essex Computer Science Department, 1978.

[49] Hayes, Patrick, The second naive physics manifesto, Hobbs, J., (Ed.), *Formal Theories of the Commonsense World*, (Ablex, Hillsdale, N.J., 1984).

[50] Hayes, Patrick J., *Naive Physics I: Ontology for Liquids*, Technical report, Institute for Semantic and Cognitive Studies, 1978.

[51] Hendler, James A., *Integrated Marker-Passing and Problem-Solving: A Spreading-Activation Approach to Improved Choice in Planning*, (Lawrence Erlbaum Associates, Hillsdale, NJ, 1988).

[52] Henrion, Max, Propagating Uncertainty by Logic Sampling in Bayes' Networks, In Lemmer and Kanal [76], 149–163.

[53] Hinton, Geoffrey and Sejnowski, Terence J., Learning and Relearning in Boltzmann Machines, Rumelhart, David E., McClelland, James L., and the PDP Research Group, (Eds.), *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, volume 2, (MIT Press, 1986).

[54] Hobbs, Jerry R. and Martin, Paul, Local Pragmatics, *Proceedings of the 10th International Joint Conference on Artificial Intelligence*, 1987, 520–523.

[55] Hobbs, Jerry R., Stickel, Mark, Edwards, Douglas, and Martin, Paul, *Interpretation as Abduction*, Technical Report 499, Artificial Intelligence Center, SRI International, Menlo Park, California, 1990.

[56] Hobbs, Jerry R., Stickel, Mark, Martin, Paul, and Edwards, Douglas, Interpretation as Abduction, *Proceedings of the 26th Annual Meeting of the ACL*, 1988, 95–103.

[57] Horvitz, Eric J., Suermondt, H. Jacques, and Cooper, Gregory F., *Bounded Conditioning: Flexible Inference for Decisions Under Scarce Resources*, Technical Report KSL-87-27, Medical Computer Science Group, Stanford University, April 1989.

[58] Howard, Ronald A. and Matheson, James E., Influence Diagrams, Howard, Ronald A. and Matheson, James E., (Eds.), *The Principles and Applications of Decision Analysis*, volume 2, chapter 37, 719–762, (Strategic Decisions Group, 3000 Sand Hill Road, Menlo Park, CA 94025, 1984), 719–762.

[59] Jensen, Finn V., *Bayesian Updating in Recursive Graphical Models by Local Computations*, Technical Report R 89-15, Institute for Electronic Systems, Department of Mathematics and Computer Science, University of Aalborg, 1989.

[60] Jensen, Finn V., Oleson, Kristian G., and Anderson, Stig Kjær, *An Algebra of Bayesian Belief Universes for Knowledge-based Systems*, Technical Report R 88-25, Institute for Electronic Systems, Department of Mathematics and Computer Science, University of Aalborg, 1988.

[61] Johnson, Mark and Klein, Ewan, *Discourse, Anaphora and Parsing*, Technical Report CSLI-86-63, CSLI, Stanford University, Stanford, California, 1986.

[62] Kamp, Hans, A Theory of Truth and Semantic Representation, Groenendijk, J.A.G., Janssen, T.M.V., and Stokhof, M.B.J., (Eds.), *Formal Methods in the Study of Language*, 277–322, (Mathematical Centre Tracts, Amsterdam, 1981), 277–322.

[63] Kautz, Henry, *A Formal Theory of Plan Recognition*, Technical report, University of Rochester, Rochester, N.Y., May 1987.

[64] Kautz, Henry and Allen, James, Generalized plan recognition, *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986, 32–38.

[65] Kim, Jin H. and Pearl, Judea, A computation model for causal and diagnostic reasoning in inference systems, *Proceedings of the 6th International Joint Conference on Artificial Intelligence*, 95 First Street, Los Altos, CA 94022, 1983, 190–193, Morgan Kaufmann Publishers, Inc.

[66] Kindermann, Ross and Snell, J. Laurie, *Markov Random Fields and their Applications*, volume 1 of *Contemporary Mathematics*, (American Mathematical Society, Providence, RI, 1980).

[67] Kirkpatrick, S., C.D. Gelatt, Jr., and Vecchi, M.P., Optimization by Simulated Annealing, *Science*, **220** (1983) 671–680.

[68] Kjærulff, Uffe, *Triangulations of graphs – algorithms giving small total clique size*, Technical Report R 90-09, Institute for Electronic Systems, Department of Mathematics and Computer Science, University of Aalborg, 1990.

[69] Kowalski, Robert A., *Logic for Problem Solving*, (Elsevier North-Holland, Amsterdam, 1979).

[70] Laskey, Kathryn B. and Lehner, Paul E., Belief Maintenance: An integrated approach to uncertainty management, *Proceedings of the Seventh National Conference on Artificial Intelligence*, 1988, 210–214.

[71] Laskey, Kathryn B. and Lehner, Paul E., Assumptions, Beliefs and Probabilities, *Artificial Intelligence*, 41 (1989) 65–77.

[72] Lauritzen, S.L. and Spiegelhalter, David J., Local Computations with probabilities on Graphical Structures and their Application to Expert Systems, *Journal of the Royal Statistical Society*, **50** (1988) 157–224, Republished in [103].

[73] Lauritzen, Steffen L., *Lectures on Contingency Tables, 3rd. edn.*, Technical Report R 89-24, Institute for Electronic Systems, Department of Mathematics and Computer Science, University of Aalborg, 1989.

[74] Lauritzen, Steffen L., Dawid, A.P., Larsen, B.N., and Leimer, H.-G., *Independence properties of directed markov fields*, Technical Report R 88-32, Institute for Electronic Systems, Department of Mathematics and Computer Science, University of Aalborg, 1988.

[75] Lehnert, W.G. and Ringle, M.H., (Eds.), *Strategies for Natural Language Processing*, (Lawrence Erlbaum, Hillsdale, N.J., 1982).

[76] Lemmer, John F. and Kanal, Laveen F., (Eds.), *Uncertainty in Artificial Intelligence*, volume 2, (North-Holland, 1988).

[77] Litman, Diane, Understanding plan ellipsis, *Proceedings of the Fifth National Conference on Artificial Intelligence*, 1986, 619–626.

[78] McAllester, David, *Reasoning Utility Package User's Manual, Version One*, Technical report, MIT Artificial Intelligence Laboratory, 1982.

[79] McAllester, David Allen, *The Use of Equality in Deduction and Knowledge Representation*, Technical Report AI-TR-550, Massachusetts Institute of Technology, Artificial Intelligence Laboratory, Cambridge, Mass., January 1980.

[80] McDermott, Drew V., A temporal logic for reasoning about processes and plans, *Cognitive Science*, **6** (1982) 101–155.

[81] Mellish, Chris, Some Chart-based Techniques for Parsing Ill-formed Input, *Proceedings of the 27th Annual Meeting of theAssociation for Computational Linguistics*, 1989, 102–109.

[82] Miller, A.C., Merkhofer, M.M., and Howard, R.A., *Development of Automated Aids for Decision Analysis*, (Stanford Research Institute, Menlo Park, CA, 1976).

[83] Minsky, Marvin, A framework for representing knowledge, Winston, P.H., (Ed.), *The Psychology of Computer Vision*, (McGraw-Hill, New York, 1975). Republished in [6].

[84] Nilsson, Nils, Probabilistic Logic, *Artificial Intelligence*, **28** (1986) 71–88, Republished in [103].

[85] Norvig, Peter, Inference in Text Understanding, *Proceedings of the Sixth National Conference on Artificial Intelligence*, 1987, 561–565.

[86] Norvig, Peter, *Unified Theory of Inference for Text Understanding*, PhD thesis, Computer Science Division (EECS), Berkeley, 1987.

[87] Norvig, Peter and Wilensky, Robert, A Critical Evaluation of Commensurable Abductive Models for Semantic Interpretation, *COLING-90*, 1990.

[88] Palmer, Martha and Finin, Tim, Workshop on the Evaluation of Natural Language Processing Systems, *Computational Linguistics*, **16**(3) (1990) 175–181.

[89] Pearl, Judea, *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference*, (Morgan Kaufmann Publishers, Inc., 95 First Street, Los Altos, CA 94022, 1988).

[90] Peng, Yun and Reggia, James A., Plausibility of Diagnostic Hypotheses: the nature of simplicity, *Proceedings of the Fifth National Conference on Artificial Intelligence*, Morgan Kaufmann Publishers, Inc., 1986, 140–145.

[91] *Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence*. Morgan Kaufmann Publishers, Inc., 1989.

[92] *Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence*. General Electric Corporation, 1990.

[93] Quillian, M. Ross, The teachable language comprehender: a simulation program and theory of language, *Communications of the ACM*, **12**(8) (1969) 459–476.

[94] Reichman, Rachel, Conversational coherency, *Cognitive Science*, **2**(4) (1978) 283–327.

[95] Richard H. Granger, Jr, *Adaptive Understanding: Correcting Erroneous Inferences*, Technical Report Research Report 171, Yale University Department of Computer Science, 1980.

[96] Riesbeck, Christopher K. and Martin, Charles E., *Direct memory access parsing*, Technical report, Department of Computer Science Yale University, 1985.

[97] Schachter, Ross D., Evaluating Influence Diagrams, *Operations Research*, **34**(6) (1986) 871–882, Republished in [103].

[98] Schachter, Ross D., Intelligent Probabilistic Inference, Lemmer, John F. and Kanal, Laveen F., (Eds.), *Uncertainty in Artificial Intelligence*, North-Holland, 1986, 371–382.

[99] Schachter, Ross D., Andersen, Stig K., and Poh, Kim L., Directed Reduction Algorithms and Decomposable Graphs, forthcoming, 1990.

[100] Schachter, Ross D. and Peot, Mark A., Simulation Approaches to General Probabilistic Inference on Belief Networks, In Proceedings of the Fifth Workshop on Uncertainty in Artificial Intelligence [91], 311–318.

[101] Schank, Roger C. and Abelson, Robert P., *Scripts, Plans, Goals, and Understanding*, (Lawrence Erlbaum, Hillsdale, N.J., 1977).

[102] Schubert, Lenhart K. and Hwang, Chung Hee, An Episodic Knowledge Representation for Narrative Texts, *First International Conference on Principles of Knowledge Representation and Reasoning*, Morgan Kaufmann Publishers, Inc., 1989, 444–458.

[103] Shafer, Glenn and Pearl, Judea, (Eds.), *Readings in Uncertain Reasoning*, (Morgan Kaufmann Publishers, Inc., 95 First Street, Los Altos, CA 94022, 1990).

[104] Shimony, Solomon and Charniak, Eugene, A New Algorithm for Finding MAP Assignments to Belief Networks, In Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence [92], 98–103.

[105] Spiegelhalter, David J., *Fast algorithms for probabilistic reasoning in influence diagrams with application in genetics and expert systems.*, chapter 16, 361–384, Wiley Series in Probability and Mathematical Statistics. (John Wiley & Sons, 1990), Proceedings of the Conference entitled 'Influence Diagrams for Decision Analysis, Inference and Prediction,' held at the Engineering Systems Research Center, University of California, Berkeley, May 9-11, 1988.

[106] Srinivas, Sampath and Breese, Jack, *IDEAL: Influence Diagram Evaluation and Analysis in Lisp.*Rockwell International Science Center, May 1989.

[107] Srinivas, Sampath and Breese, Jack, IDEAL: A software package for analysis of Influence Diagrams, In Proceedings of the Sixth Conference on Uncertainty in Artificial Intelligence [92], 212–219.

[108] Stickel, Mark E., *A Prolog-like Inference System for Computing Minimum-cost Abductive Explanations in Natural-language Interpretation*, Technical report, Artificial Intelligence Center,SRI International, 1988.

[109] Webber, Bonnie Lynn, The interpretation of tense in discourse, *Proceedings of the 25th Annual Meeting of the ACL*, 1987.

[110] Wilensky, Robert, *Understanding Goal-based Stories*, Technical Report 140, Yale University Department of Computer Science, 1978.

[111] Wilensky, Robert, Why John married Mary: Understanding stories involving recurring goals, *Cognitive Science*, **2**(3) (1978) 235–266.

[112] Wilensky, Robert, *Planning and Understanding*, (Addison-Wesley, Reading, Mass., 1983).

[113] Wong, Douglas, Language comprehension in a problem solver, *Proceedings of the 4th International Joint Conference on Artificial Intelligence*, volume 7, 1981, 7–12.