Plan-Based Paragraph Comprehension

Scott D. Anderson
Penelope Sibun
David R. Forster
Beverly Woolf

COINS Technical Report 89-121

Abstract

We investigated the use of plans in comprehending natural language paragraphs. We built a system which reads a paragraph about a tutoring situation and demonstrates its understanding of the paragraph by answering questions about the situation. For parsing, inference, and knowledge representation, we used SNePS, to which we made modifications and extensions. We did not use a planner to accomplish plan recognition, but instead depended heavily on context, domain knowledge, and Condition-Action pairs representing tutoring strategies to understand the text. We include our theoretical results in understanding situations, including Model Anaphora, an extension to theories in anaphora resolution.

This work was supported in part by the Air Force Systems Command, Rome Air Development Center, Griffiss AFB, New York, 13441 under contract No. F30602-85-C-0008. This contract supports the Northeast Artificial Intelligence Consortium (NAIC).

We thank James Pustejovsky of Brandeis University, who provided invaluable assistance during the research presented here.

Contents

1 Introduction

We investigated the use of plans in comprehending natural language paragraphs. We built a system that reads a paragraph about a situation in some domain and then demonstrates its understanding of the paragraph by answering questions about the situation. We chose tutoring as a domain because its complexity would drive our research: we had to stretch our understanding of planning and goal-driven behavior, and we were confronted with texts that required new techniques of Natural Language Processing (NLP). This report reviews our research into Tutoring as Planning (section 2), and our NLP techniques for Understanding Situations (section 3). Next, it gives an overview of our implemented program, which we call TOM, for Tutoring Observation Module. Section 4 also covers our implementation of the grammar, the rules in the knowledge base, and the focussing algorithm. We then discuss our experiences with using SNePS at a new site, our laboratory at the University of Massachusetts (section 5). Finally, we review our research conclusions and point to directions for future research.

2 Tutoring as Planning

One of the original intents of this project was to use plans to understand tutoring paragraphs. It was felt that Dr. Woolf's tutoring expertise, the extensive work at SUNY Buffalo on SNePS, and the availability at UMass of a planning system, GRAPPLE (Huff & Lesser 1987), would provide many of the pieces necessary for putting together a system to read and understand paragraphs describing tutoring sessions.

Our first several months on this project were spent in an effort to reconcile SNePS and GRAPPLE. At that time, there was no working SNePS system, so we concentrated on designing a way to use GRAPPLE's plans to understand tutoring paragraphs.[1] This effort resulted in our learning several things, discussed in detail in the rest of this section.

- Reading a story about people using plans is not the same as watching people use plans (or being told how to use plans). GRAPPLE was designed for the latter, but we are working on the former.

- Natural language input contains information (for example, tense, clausal connectives, discourse focussing clues) that requires a recognition mechanism designed to capture and represent it.

- Tutoring is a complex human activity, not amenable to representation by traditional planning methodologies. We felt, however, that of all the planning paradigms available, one closer to that called "reactive planning" best captures the way in which a tutor works, since after each action, the tutor typically monitors the response of the student and may consult other aspects of the current context.

- Recognizing a tutor's goals, through either direct observation or reading paragraphs, is not furthered by recognizing, for example, that she asked a question. Instead, understanding

[1]It should be noted that the version of GRAPPLE we examined in 1987 is perhaps quite different from the current system.

depends heavily on rich knowledge of both tutoring strategies and the subject matter being taught.

2.1 How GRAPPLE Might be Integrated into TOM

Plan-based Natural Language Understanding usually considers a story to be "understood" if it has matched a story with a known plan or script so that it can infer actions and subgoals that were not explicitly mentioned in the text. We had wanted to use GRAPPLE to match the events in the input paragraph with a known plan. Clearly, GRAPPLE would have been most useful had it been able to narrow its set of hypotheses to a single plan, and allow us to recover an "instantiated plan," one that shows the order and hierarchy of accomplished subgoals and what subgoals were matched with what actions. Unfortunately, GRAPPLE suffered two deficiencies in this regard: abandoned plans and erroneous hypotheses that were never purged; and redundant plans, one for each successfully matched step in the plan. Additionally, GRAPPLE did not keep track of the *order* in which subgoals were achieved; for its purposes, it was merely sufficient to know that they all were.

For tutoring, we would need abstracted plans that represent either *choice points* or *loops* or both. As an example of the former, we wanted an abstract plan to cover a topic which includes questioning the student, and *depending on the answer*, teaching the topic or moving on to something different. As an example of the latter, we wanted an abstract plan for presenting (as in a lecture) a series of examples, without hardwiring the number of examples into the plan.

We came to the conclusion that we would not be able to rely solely on GRAPPLE output for our understanding task, but would also, in some sense, need to save the parse. "Saving the parse" means that we would be holding on to more information from our processing of the input paragraph than the GRAPPLE plan(s) and the obvious instantiations of plan variables such as "Tom" for "student." In general, we would need to save information that GRAPPLE had no interest in and would throw away. More specifically, we came up with two independent reasons for saving the parse.

The first problem arises when GRAPPLE cannot provide us with a single plan, and thus, presumably, returns us a set of plans. We would still want to be able to answer questions about what went on: we would want to be able to give a full description of the actions mentioned, even if we could not characterize the strategies (plans) behind them. It would be very difficult and tedious to try to recover this description from a bunch of partially instantiated GRAPPLE plans.

Second, there are times when the paragraph will describe all the actions that are to be matched by GRAPPLE, but in a different order from their occurrence. For example, in our tutoring paragraph, the first few sentences could be restated thus:

> Nancy asked Tom if an inanimate object exerts a force. Nancy pointed to a book on the table. She asked if the table exerts an upward force on the book. Tom said no to both questions.

The order of occurrence of events is essential to plan understanding, and GRAPPLE itself has no way to reorder steps, because it is designed for "real time" plan recognition. Therefore, we would need to reorder as necessary in our own implementation. Assuming we would be feeding GRAPPLE information after processing each sentence, we would be faced with the possibility of having to backtrack in order to correctly order the steps. Alternatively, we could hold *onto all* the results of the parse, do a reordering pass, and hand the results to GRAPPLE in the correct order.

The first option would allow us to work incrementally, but would become wasteful when backtracking occurred. The second option avoids wasted work, but disallows the possibility of using GRAPPLE to form any predictions about upcoming input, which could conceivably help the parsing process.

We found neither of these options attractive, and this situation, in addition to GRAPPLE's own problems, persuaded us that GRAPPLE would be of limited benefit to our project. We therefore decided not to use it and turned our energies toward finding a better-suited planning formalism.

2.2 Condition-Action Pairs

Our study of the tutoring paragraph convinced us that to understand the paragraph our system need rely only minimally on plans. A tutor may incorporate overarching strategies into her tutoring, such as "examples are good" and "present easier material before harder," but when she decides what action to take next in teaching a student, she typically examines the current context rather than consults some tutoring schema. This context includes what she has already taught, what she believes the student understands, what the student's most recent response was, what material is yet available to teach on the topic—*domain knowledge*—and what *tutoring strategies* may apply for teaching it.

Domain knowledge of physics, for example, includes information that relates the concept of "inanimate object exerting a force" to the concrete example "table exerting an upward force on a book." The concept is also related to the *anchor example* (one that a student is expected to understand because it involves his direct experience) of "hand exerting an upward force on a book." The availability of such examples helps guide the tutoring process.

Tutoring strategies can best be conceived of as *Condition-Action pairs*: for any tutoring situation or set of conditions, any number of actions may be appropriate; conversely, an action may be appropriate to any number of situations. A tutor will choose an action based on the conditions. A system trying to understand a tutor's activity will match the tutor's action and attempt to match a corresponding set of conditions. It is in this weak sense that plans are used in our system. Figure 1 gives some sample conditions and actions.

We believe that the planning formalisms being developed by Dr. Shapiro and his group at SUNY Buffalo are adequate to represent these Condition-Action pairs. Using their paradigm, the Condition-Action knowledge is implicitly incorporated in our inference rules.

3 Understanding Situations

In this section, we discuss the definition and processing of words such as "situation," "example," and "case." We have concluded that a *Situation*, the referent of this word, is an aggregate of model objects picked out by a *Situation Index*. Situations are states of affairs characterized by concreteness and tension. We show further that context must inform the comprehension of this difficult lexical item.

Words such as "situation" are important because they occur frequently in natural discourse, and they are interesting because they are like anaphoric terms ("he," "it") and deictic terms ("this," "that"), but they have a much richer semantics. Consider the following usage:

> Penni and Scott are writing an IJCAI paper. Their first draft was shot down by their fellow researchers. In this situation, they squabbled more than usual.

Condition	Action
student doesn't understand topic	give anchor example (found in KB)
tutor wants to bridge	repeat question
	go back to an earlier concept
	give anchor example (found in KB)
tutor wants to introduce topic	ask question about topic
tutor wants to test student's knowledge	give concrete example of topic
tutor wants to review a previous topic	tell student to explain topic

Figure 1: sample Condition-Action pairs

Most people would agree that this paragraph contains a description of a situation, and more importantly, that the lexical item "situation" successfully refers to it. However, understanding the exact reference of the word "situation" is complicated by three factors:

1. "Situation" places significant, but ill-defined, constraints on its possible referents. This example shows elements of concreteness (particular people writing a particular paper), states (a bad first draft), and change (such as increased squabbling). These are much more complex constraints than the simple number, person, and gender features of pronouns.

2. "Situation" is nevertheless a weak descriptor, because situations come in so many different kinds—different sets of entities can be referred to as a situation, depending on the context.

3. Situations might not exist *a priori*, but instead be created *a posteriori*. That is, the set of objects that make up the situation may not previously have been aggregated. The objects aggregated by "situation" reside in the Model,[2] which is constructed in the process of understanding the text.

This section attempts to define the semantics and processing of situations.

3.1 Types of Reference

Model Anaphora, which we define to be the determination of the reference of words such as "situation," naturally extends other work on anaphora. Many researchers have studied pronoun anaphora (coreference between a noun phrase and a personal pronoun): Hobbs (1978), Grosz and Sidner (1986), Hankamer and Sag (1976), and Webber (1983). Work on temporal anaphora (coreference between temporal expressions) has been done by Partee (1984) and Hinrichs (1986), among others. Webber has also worked on the reference of demonstratives such as "this" and "that"; she has termed this variously as "event reference" (1987) and "discourse anaphora" (1988). The 1987 paper also introduced the notion of *individuating reference*, which plays a key role in our Model Anaphora.

Text comprehension involves *reference*, which we define as the link from an expression in the text (for example, a noun phrase or a proposition) to a Model Object,[3] called the *referent*. The arrows in figure 2 are examples of such reference links.

Sometimes, the referent of an expression is a *set* of things in the Model. Such a set acts as an *individual*, in that it can be referred to, can have qualities attributed to it, and can participate in relations. Webber (1987) calls a reference that creates a new individual an *individuating reference*, depicted in figure 3. One can also view this as creating a new first-class object, since only first-class objects can be referred to and predicated.

[2]This Model is informed by, but is distinct from, our domain models, which contain world knowledge of various possible domains of discourse. The Model may contain explicit structural or lexical information; this needs to be saved for some understanding tasks (for example, Webber's discourse anaphora (1987)), but it is not necessary for processing situations.

[3]A Model Object is any first-class object in the Model (essentially, a first-class object is anything that can be pointed to). Therefore, Model Objects include relations, events, states, and so forth, as well as physical objects. Recall that the Model records our understanding of the text, and is distinct from domain models.

Figure 2: the reference links from the text to the Model

Figure 3: An expression referring to a set of objects as though it were an individual. The reference that creates this set is called an *individuating reference.*

Not all aggregate model objects are the result of individuating reference: some aggregates already exist as part of a domain model. For instance, the story of Romeo and Juliet is clearly an aggregate, being composed of many people and events, but it is easily available as a referent in discourse without being created anew.

We believe that a situation is represented as an aggregate model object and that the word "situation" is often used as an individuating reference. There are two issues to address: defining just what can be termed a situation, and processing a reference to a situation (which may involve an individuating reference). The next section deals with the problem of a proper definition of this term, and section 3.3 deals with processing.

3.2 The Definition of Situations

We have identified several factors which influence whether an expression describes a situation. Consider examples (1) through (4) below. When asked to judge which examples can be characterized as situations, speakers generally agree that (4) can be a situation, but (1) cannot. However, there is less agreement on whether the intermediate expressions may refer to situations.

(1) running
(2) running the Boston Marathon
(3a) James running the Boston Marathon
(3b) James having run the Boston Marathon
(4) James having run the Boston Marathon last April

It is possible to imagine contexts in which any of these phrases describes a situation, but in some cases, the contexts are more natural and more available than in others. We believe that these expressions differ by their *concreteness.*

Concreteness is characterized in the psychological literature by imageability (for example, Paivio *et al.*, 1968). In our examples, the increasing specificity supplies the details that aid in building up a mental image. Specificity, however, is not the right measure of a situation. Consider example (5):

(5a) playing a violin
(5b) playing a Stradivarius

Most people judge these as equally good (or bad) as situations, yet (5b) is clearly more specific, though it is no more concrete (imageable). This is similar to Rosch's (1976) notion of *basic level categories*; one formulation of a basic level is that it is the highest level for which the *prototype* is imageable (hence, concrete). Thus, concepts at or below the basic level are equally concrete, and therefore their concreteness equally affects judgements of whether something is a situation.

Intuitively, a situation is a state of affairs—a description of the world (or some small part of it) at some point in time. Note that the questions "What is the situation with X?" and "What is the state of X?" will elicit similar, if not identical, descriptions. Certainly, the description may combine present and past actions, especially those that are related (causally or otherwise) to the current state of affairs:

> The situation is that I dialed this long-distance number and it's been over thirty seconds and it still hasn't rung. Strange.

Clearly, this situation includes the past action of dialing the phone. On the other hand, a situation need not mention past actions. For example, the situation at the beginning of *Romeo and Juliet* can adequately be described as the feud between the Montagues and the Capulets, without mentioning any particular actions.

A situation usually contains *tension*, which we take as *potential for change*. In Example (6) below, (6a) is a better situation than (6b) because there is a greater potential for change. Most people consider (6b) too uninteresting to be a situation at all. Of course, in certain contexts, (6b) could be considered a situation: a textbook on erosion might discuss the changes in that situation.

(6a) A boulder teetering on the edge of a cliff.
(6b) A boulder sitting in the middle of a plain.

Example (6) also demonstrates that tension need not result from agents and their goals, as it does in the Romeo and Juliet situation.

We conclude that situations are states of affairs characterized by concreteness and tension. Concreteness reflects how easily people can construct a mental image of the situation. Tension reflects people's judgements of the potential for change.[4] Obviously, judgements of these characteristics will vary from person to person; nevertheless, they put significant constraints on what can be a situation.

3.3 Processing

Situations are commonly referenced with phrases such as "the situation with X" or "the X situation." Sometimes, depending on X, more specific phrases are used, as with "the situation <u>in</u> the Middle East," which is preferable to "the situation <u>with</u> the Middle East." These modifying phrases include the *situation index* for each of these Situations. A *situation index* is a key that picks the situation out of the Model, distinguishing foreground from background, so to speak.

The situation index is usually some common or unifying aspect of the situation. Thus, if we speak of "the IJCAI paper situation," (see our first example), the situation index unites the situation. This situation is also successfully referred to by "the situation with Penni." Indeed, the situation index need not be explicitly mentioned in the text. For example:

> I was driving to work with the carpool, as usual, and the engine threw a rod. It's shot, so now I've got to get a new one. That's the situation with my car.

This reference is successful because the mention of "driving" and "engine" bring "car" into *focus* (see Sidner 1979). Furthermore, replacing the referring phrase with "the situation with the carpool" does not as easily refer to the car situation. Therefore, we conclude that, to a first approximation, it is elements in focus, regardless of explicit mention in the text, that are available as situation indexes. We have found that the *topic* of the paragraph is usually what is in focus by the end of the second sentence, and the topic is the most likely source of the situation index.[5]

We take references that make explicit mention of the situation index to be the canonical case; we process other references by first determining the situation index, thereby reducing them to the canonical case. Thus, processing Model Anaphora has two steps:

[4]Some situations, unfortunately, seem to be ones in which people nevertheless judge there to be little potential for change. For example, consider the situation of being trapped in the center of a row at a boring lecture.

[5]See section 5.1 for an alternative method of finding the paragraph's topic.

1. **Determine the situation index.** Where this is explicitly mentioned, the determination is trivial. Otherwise, the situation index must be computed from the context.

2. **Using the situation index, search the Model and look for matches.** Since we take as a departure point the actual result of parsing the text, the search will be through a subspace of the Model. The matching Model Objects will yield a set of candidates, some of which will be discarded because they are not states of affairs or because they lack concreteness or tension. If this does not reduce the set of candidates to a single situation, we choose the most recent candidate.

Our current research concerns the comprehension of the following paragraph, so we will use it to illustrate Model Anaphora:

> Nancy asked Tom if an inanimate object exerts a force. Tom said no. Nancy pointed to a book on the table. She asked if the table exerts an upward force on the book. Tom said no. Nancy placed the book on Tom's hand. Tom exerted a great force to keep the book steady. Nancy asked Tom to compare the two situations. Tom said the table exerted a force on the book.

Nancy sets the topic of the paragraph with her first question to Tom. This marks "inanimate object exerts a force" as the topic; the Model Objects **inanimate object**, **exerts**, and **a force** are available as situation indexes. Because of our domain knowledge of physics, we know that her second question, which is more specific, is subsumed by her first, and does not change the topic. When Nancy refers to "the two situations" without supplying a situation index, either "inanimate object" or "force" is probably the index she has in mind.

Supposing we take "force" as the situation index, we search in the Model for Model Objects that match it. We find the first situation because "an upward force " is a kind of force and Nancy has asked Tom whether the books exert an upward force on the table. Hence, **the books on the table** is a situation. The second situation is found again because force is explicitly mentioned, hence **the books on Tom's hand** is also a situation.[6] Although force is mentioned in the first sentence, "if an inanimate object exerts a force," this is discarded as a situation because it lacks concreteness.

3.4 Summary

We have outlined the two phases of processing Model Anaphora and have shown how we can find the referents for "situation" in a sample text. The first step in the process is finding a situation index, which we have defined as the key that picks the situation out of the Model. The second step involves searching the Model for occurrences of the situation index and determining whether any of these is part of a situation.

In conclusion, we claim that situations are states of affairs that exhibit concreteness and tension, and that the process of referring to them involves a situation index. We believe that Model Anaphora is a phenomenon common to a large class of nouns, including "case," "disaster," and "example." These nouns differ somewhat in their semantics, but they all make individuating references to sets of Model Objects.

[6]Both situations are larger than we have described them here. For instance, Nancy would include in her representations the forces involved in the two situations. Tom will, too, once he understands the concept.

4 Implementation of TOM

Our program, TOM (Tutoring Observation Module), reads a paragraph, using the SNePS Augmented Transition Network (ATN) to parse the sentences and represent their meaning with SNePS nodes, which are automatically integrated into the SNePS network. Next, the Sidner Focussing Algorithm is executed. This algorithm tracks the focus movement through the paragraph, that is, as the focus changes from some element of the previous sentence to some element of the current sentence. The focussing algorithm was originally designed to work sentence by sentence and, although we considered implementing more dynamic focussing, we decided to stay with sentence-based processing. This required us to make the focussing algorithm a separate stage after the parser, since the ATN might backtrack at any time. Because paragraphs in the tutoring domain will contain many indirect questions, we decided not to trigger question-answering within the ATN. Instead, we look at the topmost node that represents the meaning of the current sentence and, if it's a question, we invoke the inference engine to deduce the answer. When this is completed, the answer is printed. Since the research is not concerned with Natural Language Generation, the answer is printed simply by looking up a sentence corresponding to the answer that was deduced. This technique is obviously deficient in the long run, but it enabled us to concentrate on the primary task of text comprehension. We believe that the SNePS generator will eventually be able to handle our generation demands. A flowchart of TOM's organization appears in figure 4.

4.1 Implementing the Grammar

The paragraph that we have studied most extensively and that TOM can now read contains very natural English, and that presented some problems. Some phenomena are simply outside the scope of this research and are among the open problems in AI and NLP today. For example, the original paragraph made great use of modals: "Tom had to exert..." rather than "Tom exerted...." To properly represent these meanings would demand representing possible worlds and intentions of agents, which would have been extremely complex. Therefore, we simply rewrote these sentences in simpler language.

The first sentence originally read:

1. Nancy asked Tom if an inanimate object, such as a table, can exert a force.

The parenthetical expression "such as a table" presented unexpected difficulties. It could be parsed, but there was much controversy over the meaning of the expression, the representation of its meaning, and the ultimate effect of that meaning. Eventually, we decided that the effect of "such as" phrases is to focus the attention of the human listener or, by analogy, to advise the computer's inference engine so that it can focus its inferences in a particular way. In a way, it's as if the speaker expects that the hearer will attempt to answer the question by using a case-based approach: examining cases of inanimate objects and deciding if any of them can exert a force. The "such as" phrase prompts particular kinds of cases to be examined. Consider asking a naive student whether a hammer or an anvil can exert a force. The student is likely to say that the hammer can and the anvil cannot, simply because the hammer is active and the anvil passive. Thus, the "such as" expression focusses the hearer's attention on passive inanimate objects, without articulating that concept. Once we agreed that this is the correct view of the "such as" phrase, we also knew that it could have no effect on the SNePS inference engine, since the SNePS inference engine is not

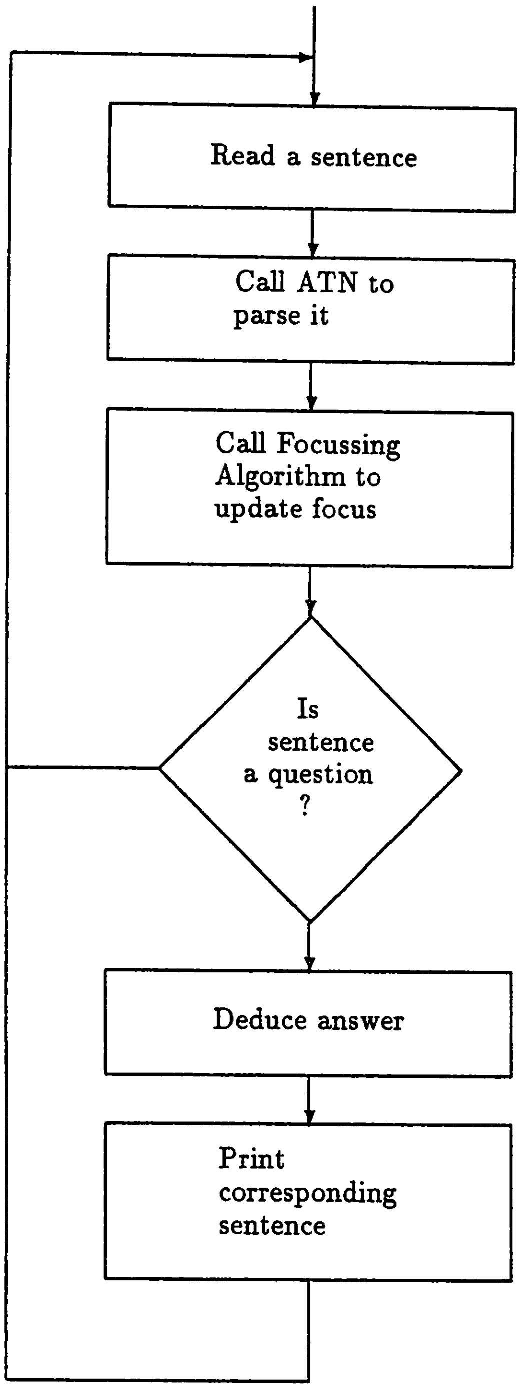

Figure 4: a flowchart of TOM's organization

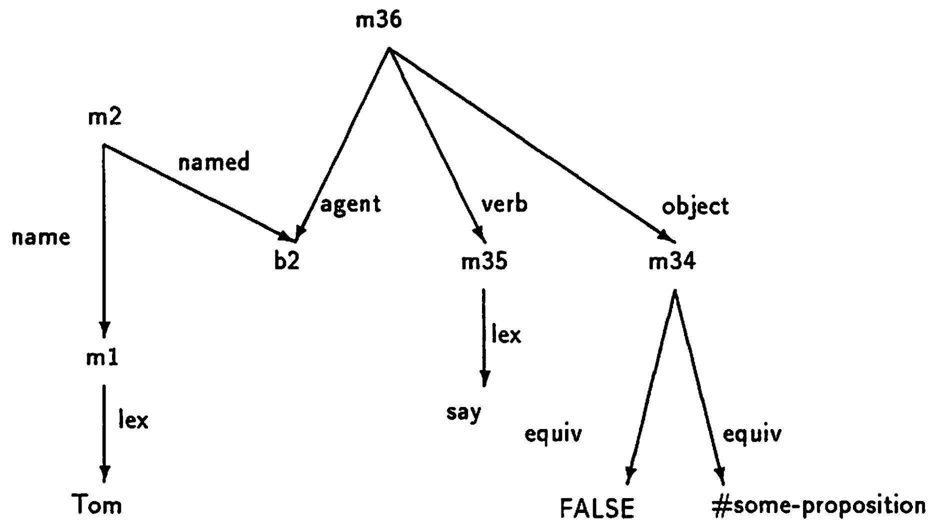

Figure 5: the parser's representation of the meaning of "Tom said no"

case-based and has no attentional mechanisms. Therefore, we dropped the phrase; we would be parsing it to no purpose.

The second sentence is both very simple and very interesting.

2a. Tom said no.

Actually, the original sentence was

2b. Tom said he didn't think so.

We changed it because we considered the latter simply a needlessly complicated idiomatic expression with essentially the same semantics as the former. The most important difference in semantics is that the latter expression conveys Tom's doubt is his belief—a difference we could not have accounted for, since the SNePS inference engine is a symbolic logical system, not a probabilistic system with degrees of belief. Still, "Tom said no" is more than interesting enough, because the key problem is determining what he said no *about.* He is negating some proposition, but what proposition? We call this problem *Propositional Anaphora.*[7] Properly understanding this sentence requires awareness of the continuity of sentences at the discourse level, higher than the sentence level. In section 4.2 we discuss how we use an extension to Sidner's focussing algorithm to handle Propositional Anaphora.

Because the focussing algorithm is executed after the ATN has finished parsing the sentence, the representation of the meaning of "Tom said no" is as depicted in figure 5. The "#some-proposition" is just a convenient label for what is an atomic base node, meaningful only by what points to it; for our purposes, it's just a placeholder. So, the network reads "Tom said that <some proposition> is equivalent to False." "False," of course, is a constant in the network. At this point, we don't know what he said was false. Later, after the focussing algorithm runs, we get the network fragment in figure 6. You can see the need for the placeholder node, as it is the connection between 'False' and the node M12, which is the focussed element.

[7]The same issue arises, of course, with affirmative sentences such as "John said yes."

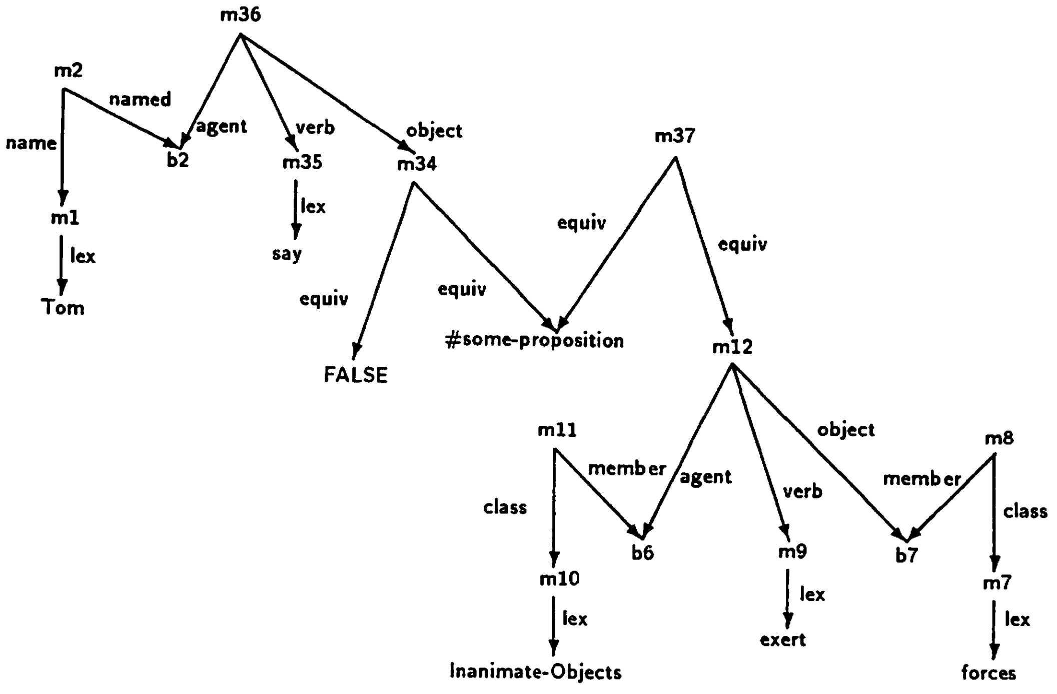

Figure 6: the representation of the meaning of "Tom said no" after the focussing algorithm runs

The next interesting sentence is the following

7. Tom exerted a great force to keep the book steady.

In current linguistics literature, the phrase "to keep the book steady" is known as a *rationale clause* (Huettner 1989). Rationale clauses are deemed to attach directly to the **S** node, rather than to **VP**, as a relative clause would, yet they are like relative clauses in that there is usually a *trace*—a item which is deleted because it is identical to something in the main clause. In this rationale clause, the trace is the *agent*, Tom, who is the one keeping the book steady. This sentence also exhibits a *small clause*, namely the phrase "the book steady." We parse this into the proposition `steady(book)`, which is then the object of "keep." It makes sense for "keep" to take a state (the steadiness of the book) as its object. We do not claim that our ATN can handle all small clauses and rationale clauses, but we have gotten it to handle, in a robust and general way, sentences like sentence 7.

The most difficult sentence we faced is the next one:

8. Nancy asked Tom to compare the two situations.

This has already been discussed at length in section 3. The sentence also posed a slight parsing problem, because the phrase "to compare the two situations" can be read as a rationale clause. We were forced to be much more rigorous in annotating verbs with features to say what arguments they can or must have. In this case, we solved the problem by having "ask" require a **VP** as its object. By adding features to items in the lexicon to direct the ATN, we improved its parsing ability, but this continues to be a weakness of the system. Ideally, the system would have a fairly complete set of subcategorization frames and thematic grids, with a mapping from the former to the latter, and

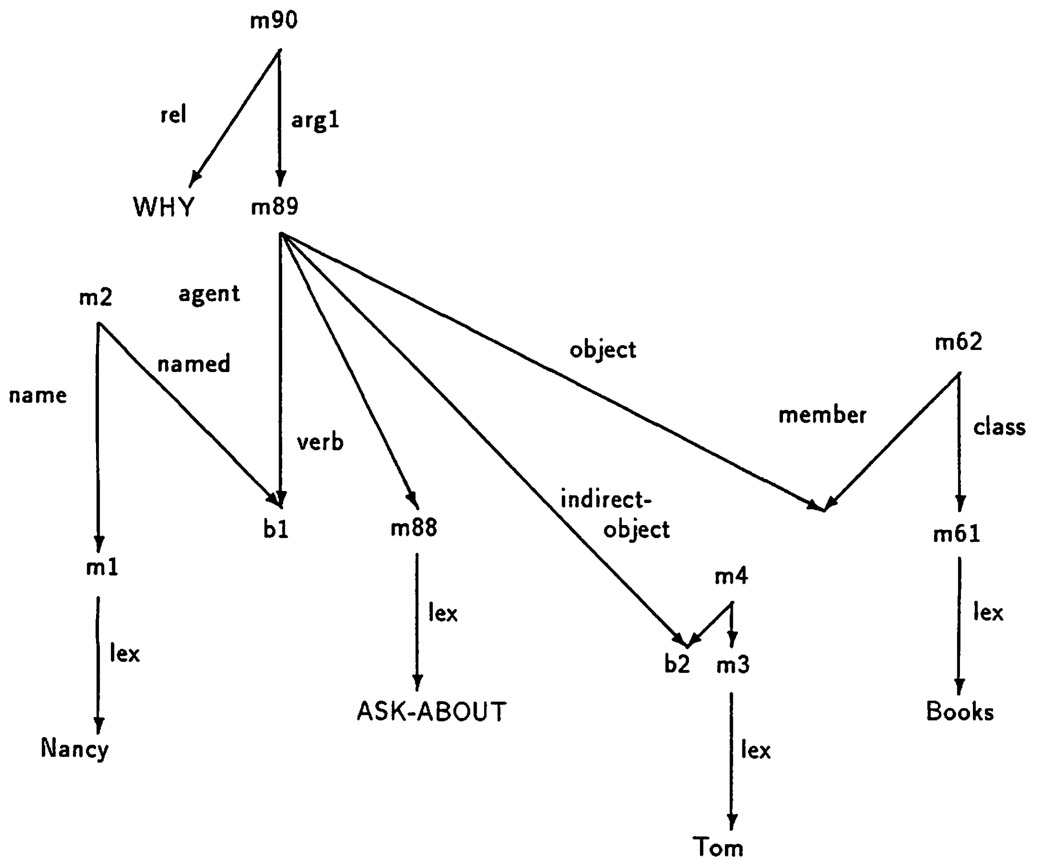

Figure 7: the representation of the meaning of "Why did Nancy ask Tom about the book on the table?"

have the subcategorization frames take a much stronger role in the parsing, rather than letting the structure of the ATN control the parsing.

The last grammatical problem we tackled was parsing the questions that follow the paragraph:

Q1. Why did Nancy ask Tom about the book on the table?
Q2. Why did Nancy place the book on Tom's hand?
Q3. Why did she ask him if inanimate objects exert a force?

Previous SNePS implementations had concentrated on "wh-questions," such as "what did Lucy pet," which would, as a side-effect of parsing, cause SNePS to search the network and reply, for example, "Sweet Lucy petted a yellow dog named Rover." Because we believe indirect questions—statements such as "Nancy asked Tom what the book weighed"—are common in our domain, we did not want the system to answer *every* question it read. Therefore, we chose a simple representation for questions, and then, *after the parsing*, if the topmost node of the representation of a sentence is a question node (which means the sentence is a direct question, not an indirect question), we answer the question. Figure 7 shows the representation for the first question.

Note that the verb in figure 7 is "ask-about." We felt that "ask...about" could be treated as a verb-particle combination, rather than a simple verb with a prepositional phrase. Technically, it should be the latter, since "about" does not undergo particle movement (one of the key diagnostics of a true particle construction). The change was motivated primarily on semantic grounds, because we wanted "the book on the table" to be the object of the sentence, so that it would fulfill the requirement that the verb "ask" take an object. (See the discussion of sentence 4.1, above.) Moreover, several other functions also look for the object of the sentence, and this change makes those functions simpler. The problem arises because the system conflates syntactic and semantic roles in the label **object**. While our solution works, it is not sufficiently general, and a uniform distinction between syntactic positions (of subcategorization frames) and thematic roles (of thematic grids) would solve the fundamental problem. Then, at the syntactic level, "the book on the table" can be the object of the preposition "about," and yet be mapped to **theme** at the semantic level, and all semantic functions can uniformly refer to **theme** rather than **object**.

Parsing the questions also required us to substantially rewrite the grammar so that it could handle the inversion, auxiliary-insertion and other syntactic phenomena of questions in English. The resulting grammar has been tested on a large suite of sentences and is quite robust. The whole ATN grammar is included in appendix A.

4.2 Implementing Sidner's Focussing Algorithm

As described in section 3, a major concern of our research has been developing a theory of Model Anaphora. Since we considered this effort an extension of other theories of anaphora resolution, and since, in particular, our theory depended on the notion of focus, we decided to incorporate Sidner's focussing algorithm (1979) into our system. Her work remains one of the cornerstones of focussing theory, and since her algorithm is so well-known, we thought it would be well-understood and easy to customize for our own purposes.

Unfortunately, several difficulties arose from our assumptions. While implementations are reputed to exist, we could not find someone actually able to give us one; Sidner's thesis itself contains only an English version of the algorithm, which is obscure in some places. Further, it became apparent that someone else's implementation would probably not help us too much, since the implementation naturally depends greatly on the structure of both the knowledge representation and the parser. So we eventually decided to implement the algorithm from scratch, using the thesis as a guide. This exercise proved instructive of the pitfalls of such an endeavor.

Sidner's focussing algorithm was motivated by a need to find ways to constrain the search for cospecifications of anaphora in multi-sentential text. An *anaphor* is a lexical item that derives its "meaning," or specification in a knowledge base, through association with some other lexical item. The most common anaphora are definite noun phrases—pronouns and noun phrases (NPs) with definite articles (such as "the"). Such NPs *cospecify* with other NPs in the text.[8] Thus, for example, in the following text "she" cospecifies with "Nancy" and "the book" cospecifies with "a big red book."

> Nancy was leafing through a big red book. She remembered fondly the time she used the book to tutor Tom about physics.

The idea behind focussing is that only one item in the text can be in focus at a time. What is in

[8]Most researchers refer to this phenomenon as coreference rather than cospecification; Sidner prefers the latter because in a program no referring to the real world is happening, only specification of particular items in a knowledge base.

focus constrains the resolution of anaphora; conversely, use of anaphora signals either continuation of the focus or the movement of the focus to some other item in the text.[9]

The algorithm works by processing one sentence at a time. For each sentence, an ordered list is constructed of all items that may come into focus at some point. For the first two sentences, this is the Default Expected Focus list (DEF); for all subsequent sentences, this is the Potential Focus list (PFL), which differs from the DEF by not including the agent of the sentence. Each sentence's Alternative Focus List (ALFL) is the DEF or PFL from the previous sentence. The processing of each sentence results in either *confirming* the focus or *moving* it; the ALFL is one set of possible new focuses. If the focus is moved, the old focus is pushed onto the focus stack. The focus stack is another source of new focuses. If the focus is moved to one of these (that is, back to an item that has already been in focus), the items above it in the stack are discarded. There is no focus for the first sentence, since focus is a property of multi-sentential text. Processing for the second sentence, then, is a matter of *establishing* the focus, rather than confirming or moving it.

The focussing algorithm has a companion focussing algorithm for actors that handles the resolution of personal pronouns. We did not implement this because this issue is tangential to our main concerns and a simplistic solution was already available by use of the SNePS Uniqueness Principle. We did not implement the entire focussing algorithm, either, since some of it was not needed for the paragraph we were understanding. Our implementation also includes several extensions for cases not covered in the original, as described below.

The interface between SNePS and the focussing algorithm is not very clean. The algorithm is called after each sentence is parsed, and handed a list of SNePS nodes corresponding to phrases. Unfortunately, the algorithm depends on the syntactic roles of each of the phrases in a sentence as well as the actual linear order in which the phrases appear. The SNePS parser, on the other hand, is only concerned with the semantic roles played by elements of the sentence, and therefore in principle does not save any syntactic information. Hence, the first task of the focussing algorithm is to recover as much of this information as possible. We also had to add to the parser a mechanism for explicitly recording the definiteness of each NP, something to which SNePS is indifferent but which is clearly crucial to the algorithm.

The rest of this section briefly sketches some of the interesting aspects of the algorithm, as it functions in our tutoring paragraph, reproduced here.

> (1) Nancy asked Tom if an inanimate object exerts a force. (2) Tom said no. (3) Nancy pointed to a book on the table. (4) She asked if the table exerts an upward force on the book. (5) Tom said no. (6) Nancy placed the book on Tom's hand. (7) Tom exerted a great force to keep the book steady. (8) Nancy asked Tom to compare the two situations. (9) Tom said the table exerted a force on the book.

Sentence (2) immediately presents us with a type of anaphora not covered in the original algorithm. When Tom says "no," he is negating some proposition, and to understand the sentence we need to find out which proposition this is; thus, we call this Propositional Anaphora. Following the principles of the focussing algorithm, we believe that the proposition being negated will be the one in focus, and indeed, "if an inanimate object exerts a force" is the predicted focus at the second sentence.

[9]For a much fuller discussion of these ideas, see Sidner 1979.

This example demonstrates another extension we made to the algorithm: Sidner's version did not account for sentences with subordinate clauses. We decided that subordinate clauses, including the rationale clause in sentence (7), fill roles similar to those filled by NPs, and so we could treat them equivalently. However, we also felt that the constituents of a subordinate clause should be available to the focussing algorithm as well, so these are also included in the DEF/PFL.

Our most interesting use of the algorithm occurs in sentence (8). Previously, the topic of the paragraph was determined, at the second sentence, to be the element in focus, that is, "an inanimate object exerts a force." The word "situations" triggers a part of the focussing mechanism responsible for resolving such Model Anaphora (see section 3). Any of the elements of the topic, namely **exert**, **inanimate object**, and **force**, may be used as the situation index; the current implementation uses the first of these. "The two situations" is thus found to specify the aggregate of **table exerts an upward force on the book** and **Tom exerts a great force to keep the book steady**. Each of these situations is itself an aggregate of the entire situation and all of its elements.

At this point in the paragraph, during the processing of sentence (8), the focus is the **book** cospecified by "the book" in sentence (7). The focussing algorithm includes a step by which if a definite noun phrase in the current sentence (here, "the two situations") *implicitly cospecifies* the focus, the focus is confirmed. Sidner defines implicit cospecification as a case in which the definite NP specifies a Model Object that is closely related to the specification of the focus. In this example, the **book** is a situation element of one of the two situations; this relationship is close enough to warrant confirmation of the **book** as the focus.

Implementing Sidner's focussing algorithm and making the necessary extensions to it proved to be more work than we had anticipated and in retrospect was probably a poor design decision. The apparent appeal of an "off the shelf" algorithm was offset by the difficulty of engineering the interface with SNePS, as well as the limited coverage of the original algorithm. In fact, we learned the hard way that an off the shelf algorithm is a far cry from off the shelf software, and we would have been better off in this case building a mechanism of our own rather than trying to fit an algorithm to our needs.

4.3 Implementing the Knowledge Base

TOM's knowledge of tutoring and physics is represented as inference rules, stored in the SNePS network and executed by the SNePS inference engine. They are triggered by the need to answer questions asked of the system. Calling the inference engine results in new nodes being asserted and returned, whereupon these nodes are mapped to sentences that answer the user's question. This section will review the deductions involved in answering the first question.

The representation of the first question, "Why did Nancy ask Tom about the book on the table?" was shown in figure 7 (page 16). "Why" questions are taken to ask what goal some action or plan is subserving. Therefore, we turn that question into the following deduction:

```
(deduce plan (build agent           b1
                    verb            ask-about
                    indirect-object b2
                    object          b9)
      goal v1)
```

The node "V1" is a new variable node, which will be bound to the node which is deduced to be the answer to the question, which is essentially "what is the goal of this plan." The call to DEDUCE initiates the backward-chaining ability of the SNePS inference engine. It creates a pattern node which matches the consequent of rule 1, shown in figure 8.

This rule is essentially knowledge about tutoring, for it concerns asking someone about specific examples of a topic in order to probe his knowledge of the topic. Note that the agent, "x," which in this case is Nancy, is mentioned several times in the rule: Nancy did the asking, so it's Nancy who wants (Nancy) to know Tom's knowledge of the topic. The use of the agent is a philosophical difference between UMass Amherst and SUNY Buffalo, since we believe it is important to mention the agent, so that the agent can constrain the inferences and be represented in the answer. The SUNY Buffalo research group, on the other hand, believes it is important to represent the commonality of different agents' plans and not mentioning the agent in a rule produces more general inferences. We believe this rule demonstrates the correctness of our view, since the agent, "x," is so crucial in the representation of the goal, but the Buffalo researchers may have an alternative solution to this problem.

Once the inference engine has matched the consequent of rule 1, it wants to prove the antecedents of that rule. The first antecedent is just the act itself and so is trivially true.[10] The second antecedent is that "B9" is a situation index for some (as yet unknown) situation "e." B9 *is* a situation index (we determined so when processing the Model Anaphora in "Nancy asked Tom to compare the two situations"), and so "e" is bound to the corresponding situation, the node B19, shown in figure 9.

Antecedent 3 is that "e" is a situation. This follows straightforwardly from its having situation elements such as B9, so we won't go into this. The fourth antecedent, however, is somewhat complicated, as it determines whether "e" is an example of statics. That matches the consequent of rule 4, shown in figure 10, with "z" unified with "statics."

This rule captures some of the program's knowledge of physics, since it defines conditions sufficient for an example of statics. The first antecedent is that "s" (which is our "e" from rule 1) is a situation; we already know this is true. Second, we check that "s" has situation elements "x" and "y." This binds "x" to B9 and "y" to B10. Next, we check that both "x" and "y" are physical objects. The variable "f," in the next antecedent, is open and will be bound to something deduced from yet another rule (one which deduces that if "x" supports "y," then there exists a force, "f," between them). We check that "f" is a force exerted by "x" on "y" and by "y" on "x." This completes the backward chaining through rule 4, and we can now return to rule 1.

The next two antecedents of rule 1 involve a variable "k" which can be thought of as the pupil's ("y's") knowledge of statics. The variable "k" gets bound to a node built by another rule, part of the program's general world knowledge, that says there always exists a node to represent someone's knowledge of something. The backward chaining through rule 1 is now complete. The consequent is asserted, which lets the system state that Nancy wanted to know Tom's knowledge of statics.

This section has concentrated on part of one inference, the one which answers the first question, in order to demonstrate the general technique, and to show the representation of tutoring knowledge and physics knowledge in the SNePS formalism. We include here a summary of our work on the other questions.

[10]Actually, it's not quite so trivial. The act is true because the parser asserted it to be true; that is, the parser takes the statements it reads to be true. One could imagine a more suspicious system that might well answer this question: "I'm not certain that Nancy did ask about the books on the table." This is a minor quibble, as most AI systems believe what they're told, but it's worth mentioning why this proposition is true.

```
(assert
  forall ((new-variable-node x) ... y, z, e, si, k)
  &ant  ((build agent            (node-named x)
                verb             ask-about
                indirect-object  (node-named y)
                object           (node-named si))
         (build arg1 (node-named si)
                rel  situation-index
                arg2 (node-named e))
         (build arg1 (node-named e)
                rel  situation)
         (build arg1 (node-named e)
                rel  specific-example-of
                arg2 (node-named z))
         (build arg1 (node-named y)
                rel  (build lex know)
                arg2 (node-named k))
         (build arg1 (node-named z)
                rel  in-domain
                arg2 (node-named k)))
  cq     (build plan (build agent            (node-named x)
                            verb             ask-about
                            indirect-object  (node-named y)
                            object           (node-named si))
                goal (build arg1 (node-named x)
                            rel  (lex want)
                            arg2 (build arg1 (node-named x)
                                        rel  (build lex know)
                                        arg2 (node-named k)))))
```

Figure 8: Rule 1 from the Knowledge Base
If **x** asks **y** about **si**, which is a situation-index to the situation **e**, a specific example of **z** (and there's a node **k**, which is **y**'s knowledge of **z**), then the asking-about was a plan for that goal.

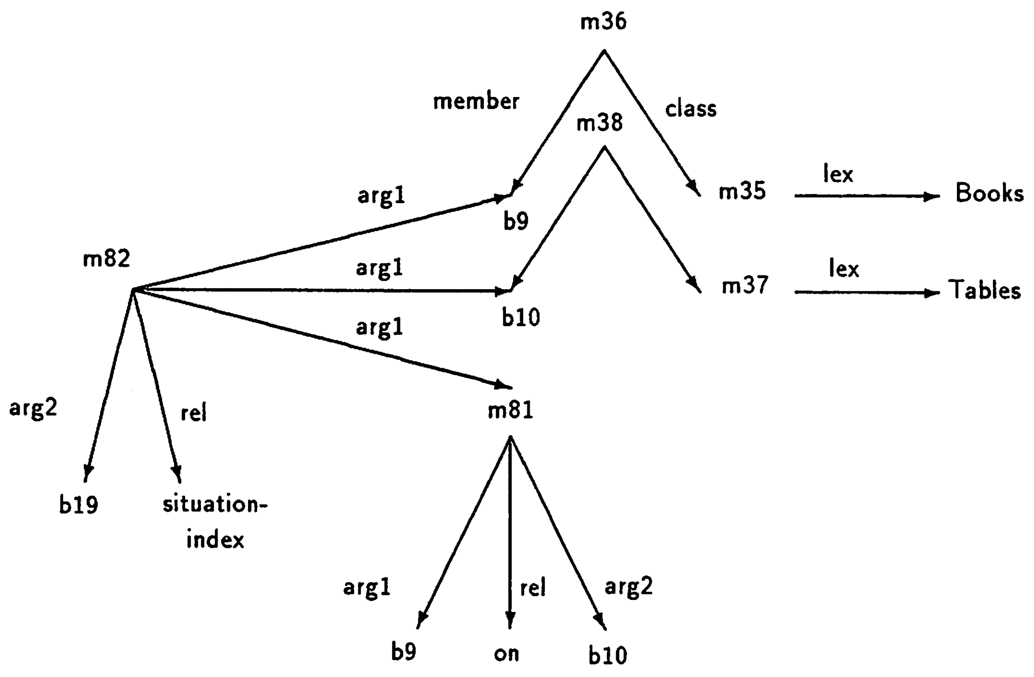

Figure 9: The representation of "e," the example referred to in the first question with the phrase "the book on the table?"

The representation of the second question, "Why did Nancy place the book on Tom's hand?" is shown in figure 11 (page 24). The question is interpreted as requesting the deduction:

```
(deduce plan m145 goal v1)
```

"M145" is the node representing the declarative form of the question. The variable node "V1" will be bound to the node which is deduced to be the answer to the question.

The pattern node created matches the consequent of rule 7, shown in figure 12. Like the first rule, this rule is essentially knowledge about tutoring. It states that an anchor example of a topic can be provided to a student by performing a suitable action which results in the student having direct physical evidence of the laws of the topic domain.

The first and third antecedents of this rule concern the action, namely that it be an action, and that the action have a particular topic. Here the action is "place," which has "statics" as its topic.

The second and fourth antecedents concern the student: The second antecedent requires that the object which experienced the action directly ("w") be a part of a sentient entity ("u"). The fourth antecedent requires that this sentient entity have direct physical evidence of the topic, i.e., of "statics."

The rules underlying the second and fourth antecedents are shown in figures 13 and 14. For the question posed, the object is "B21" (Tom's hand), for which the sentient entity is "B2" (Tom).

The rule underlying the fourth antecedent requires that the sentient entity (Tom) sense an example of the domain (statics). Rules concerning such sensing are domain-specific. We will not go into further detail than to say that Tom senses an example of statics because he senses an example of forces, because a force is being exerted on a part of him, and because that part is capable of sensing forces.

```
(assert forall ((new-variable-node s)
                (new-variable-node x)
                (new-variable-node y)
                (new-variable-node f))
        &ant   ((build arg1 (node-named s)
                       rel  situation)
                (build arg1 (node-named s)
                       rel  situation-element
                       arg2 ((node-named x)
                             (node-named y)))
                (build member (node-named x)
                       isa    Physical-Objects)
                (build member (node-named y)
                       isa    Physical-Objects)
                (build member (node-named f)
                       isa    forces)
                (build rel  exerts
                       arg1 (node-named x)
                       arg2 (node-named y)
                       arg3 (node-named f))
                (build rel  exerts
                       arg1 (node-named y)
                       arg2 (node-named x)
                       arg3 (node-named f)))
        cq     (build arg1 (node-named s)
                      rel  specific-example-of
                      arg2 statics))
```

Figure 10: Rule 4 from the Knowledge Base
If s is a situation, and x & y are elements of s and are **Physical-Objects**, and f is a force, and **exerts(x,y,f)** and **exerts(y,x,f)** then s is a specific example of **statics**.

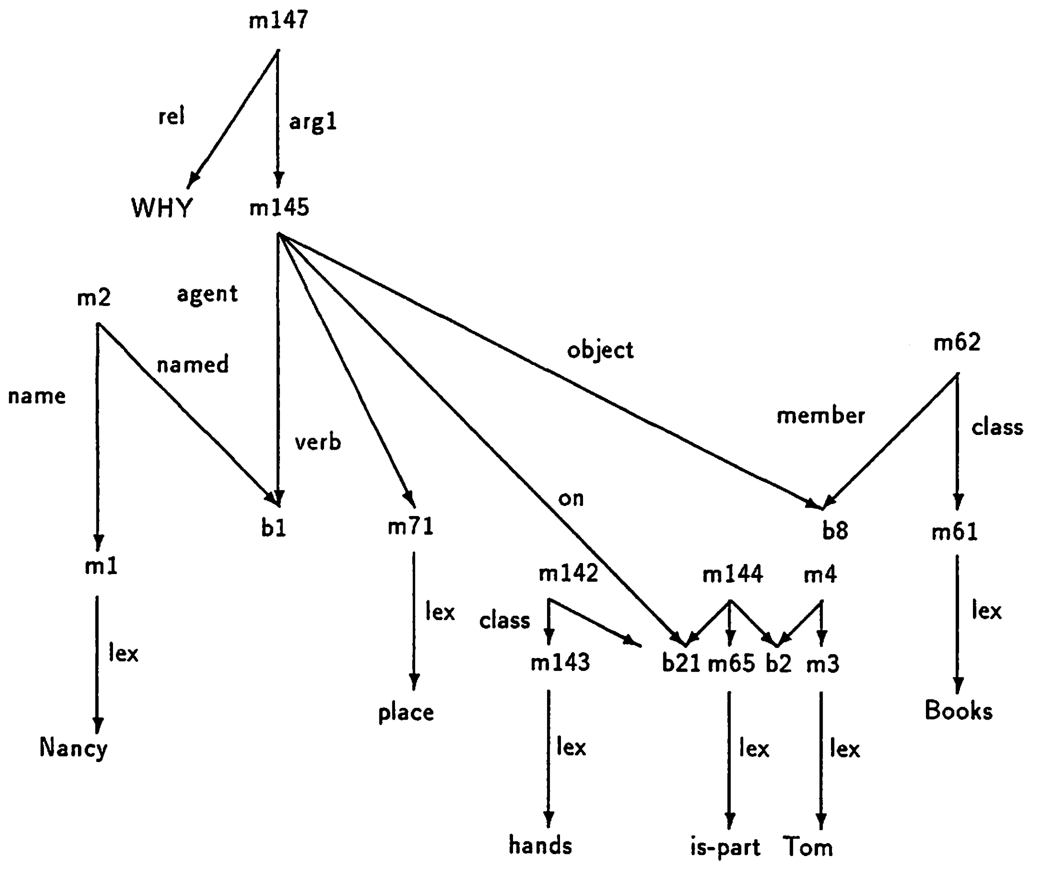

Figure 11: the representation of the meaning of "Why did Nancy place the book on Tom's hand?"

```
(assert
   forall ((new-variable-node u) ... v, w, x, y, z)
   &ant   ((build arg1 (node-named v)
                  rel  has-property
                  arg2 action)
           (build arg1 (node-named u)
                  rel  sentient-head
                  arg2 (node-named w))
           (build arg1 (node-named v)
                  rel  has-topic
                  arg2 (node-named z))
           (build arg1 (node-named u)
                  rel  has-direct-evidence-of
                  arg2 (node-named z)))
   cq     (build plan (build agent  (node-named x)
                             verb   (node-named v)
                             object (node-named y)
                             phys-relation (node-named w))
                 goal (build agent  (node-named x)
                             verb   (build lex provide)
                             indirect-object (node-named u)
                             object (node-named z))))
```

Figure 12: Rule 7 from the Knowledge Base
If **v** has an action property and topic **z**, and if **w** is a part of sentient being **u** which has evidence of **z**, then **x**'s carrying out **v** with **y** such that a physical relation is established with **w** could be a plan to provide the sentient being **u** with knowledge of **z**.

```
(assert
   forall ((new-variable-node x))
   ant     (build  member (node-named x)
                   isa    sentient-thing)
   cq      (build  arg1 (node-named x)
                   rel  sentient-head
                   arg2 (node-named x)))
(assert
   forall ((new-variable-node x)
            (new-variable-node y))
   &ant    ((build arg1 (node-named y)
                   rel  (build lex is-part)
                   arg2 (node-named x))
            (build member (node-named x)
                   isa    sentient-thing))
   cq      (build  arg1 (node-named x)
                   rel  sentient-head
                   arg2 (node-named y)))
```

Figure 13: Rules 8 and 9 from the Knowledge Base
If x is a **sentient-thing**, then it is its own **sentient-head**. If **y** is part of a sentient-thing **x**, then **x** is **y**'s sentient-head.

```
(assert
   forall ((new-variable-node x)
            (new-variable-node y)
            (new-variable-node z)
            (new-variable-node k))
   &ant    ((build arg1   (node-named z)
                   rel    in-domain
                   arg2   (node-named k))
            (build arg1   (node-named y)
                   rel    example
                   arg2   (node-named z))
            (build agent  (node-named x)
                   verb   (build lex sense)
                   object (node-named y)))
   cq      (build arg1 (node-named x)
                  rel  has-direct-evidence-of
                  arg2 (node-named k)))
```

Figure 14: Rule 10 from the Knowledge Base
If **x** senses **y**, an example of **z**, which is in domain **k**, then **x** has direct evidence of **k**.

All the antecedents of rule 7 having been satisfied, TOM can now assert that Nancy wanted to give Tom an example of statics.

Question 3 appeared to be solved until very recently, when we realized that answering the question was much more complicated than we had thought. While we no longer have time to devise and program a complete revised solution, we can sketch out our thoughts on the problem.

Essentially, to answer the question

> Q3. Why did Nancy ask Tom if inanimate objects exert a force?

the system must understand that Tom's knowing the domain of statics is sufficient for him to answer Nancy's question, and therefore the question is a good one for probing his knowledge of the domain. We would like to be able to say that is it *necessary* and sufficient that one know a domain in order to answer a question about it. However, this is clearly not always the case; for example, Tom could cheat.

For the system simply to know that the question is about statics is difficult, because it is not enough to know that each term in the question is in the domain, since the terms may be related in such a way that the composite meaning falls outside the domain. Now, to some extent we can control which relations appear in the composite meaning representation, via our control of the ATN, but this would be an unsatisfactory *ad hoc* solution.

Because a realistic depth of understanding in the knowledge representation is so difficult to achieve, we have simply chosen to leave this question for possible subsequent studies.

5 Using SNePS at UMass

Any time two research groups from different institutions attempt to cooperate there will be difficulties as well as advantages. The benefits of working with the SUNY Buffalo group were primarily intellectual: we were able to discuss our approach to problems with interested, knowledgeable, and objective researchers. It was particularly refreshing to work with a group that comes from a different background and therefore challenged many of our assumptions about possible solutions. In retrospect, we feel the two groups should have met more often than twice a year so as to derive greater benefit from this interaction. While we were pleased with the intellectual interaction, we were disappointed with the interaction involving implementation issues. This section details some of the experiences we had with SNePS.

We expected to benefit greatly from getting SNePS, a working system that could handle our needs for representation, parsing, inference, and generation. However, SNePS was too immature a system to meet our expectations: releases of SNePS tended not to work, and SNePS was difficult to debug when it did fail.

We encountered major implementation bugs, for example in the ATN's restoration of registers when backtracking and in the use of TI Common Lisp's[11] "displacing" macros. We were also tripped up by philosophical differences in coding style. The Buffalo group's adoption of conventions that usurp the Common Lisp language, including important characters such as # and *, and functions such as DESCRIBE, PUSH, POP, ASSERT, +, *, and = made our own coding very difficult. We believe that such idiosyncratic practices should be avoided in any code that is meant to be shared between two or more research groups.

[11]the implementation of Common Lisp on the Texas Instruments Explorer

We had particularly frustrating difficulty with the Inference Engine under development by the Buffalo researchers. Some problem causes the conjunction of several antecedents, each of which is itself true, to be false. The exact source of this bug is as yet unknown and still being investigated. The very fact that this bug is so difficult to track down is indicative of how opaque the Inference Engine is, particularly to users at sites other than Buffalo.

We believe that the lesson to be learned from this experience is that for one research group to fruitfully use another research group's program, either the software should be relatively mature, stable, and likely to work, or the two groups should work very closely together.

5.1 Future Directions: Understanding More Complex Paragraphs

We would like to conclude this report by describing a promising extension to this research. In general, the goal of our work in paragraph understanding is to comprehend ever more complex text. Here, we show preliminary work on a paragraph which involves processing Model Anaphora and appreciating complex human interactions and motivations. We believe that a system based on our current framework can handle such text.

We have shown that processing Model Anaphora requires more complex representation and more subtle inference than other sorts of anaphora. Anaphora that involve human interaction, plans, and goals are more complex and subtle yet. The paragraph below is typical of such text.

> (1) Last week, 32 shabby peasants appeared at the gates of the U.S. embassy in Moscow. (2) They put a startling request to American diplomats: help us get out of Russia. (3) Local authorities in Siberia had threatened to imprison the adult members of the peasants' religious group. (4) With the vague notion that a foreign embassy might help them, the Siberians had come by train to Moscow. (5) The Americans listened sympathetically, but Ambassador Foy Kohler had to stick to regulations. (6) He called the Soviet Foreign Ministry, explained the situation and asked that the peasants be removed. (7) Embarrassed by the whole thing, the U.S. officials prevented foreign correspondents from photographing or speaking with the visitors.

Two distinct Model Anaphora are used in this paragraph: "the situation" in sentence (6) and "the whole thing" in sentence (7). Notice that "the whole thing" is more inclusive than "the situation." Since our algorithm resolves each Model Anaphor in the course of processing the sentence in which it appears, the additional structure that is part of "the whole thing" will be built after "the situation" has been resolved, thus capturing this difference.

As described in section 3.3, we find the situations by first finding the situation index, which is generally the topic of the paragraph. In the tutoring paragraph, the topic was determined to be what was in focus by the second sentence; this happened to be the indirect question of the first sentence, namely, "if an inanimate object exerts a force." However, the topic of a paragraph need not always be given in the first sentence, so we need a more sensitive mechanism for recognizing it than checking the focus at a particular moment.

Often, the topic is strongly marked by some action. This may be, for example, a demonstative action, such as pointing, or, as in the tutoring paragraph, a speech act (Nancy asking a *question*). In the current paragraph, the topic is also marked by a speech act:

> (1) Last week, 32 shabby peasants appeared at the gates of the U.S. embassy in Moscow.
> (2) They *put a* startling *request* to American diplomats: *help us get out of Russia.*

(1)	None.		
(2)	Peasants	*Goal*	be out of Russia
		Plan	ask American authorities for help
(3)	Authorities	*Goal*	preventing peasants' religious practices
		Plan	threaten peasants
	Peasants	*Goal*	keeping religion
		Plan	??
(4)	Peasants	*Subgoal*	ask a foreign embassy for help
		Plan	go to embassy in Moscow
	(This is an elaboration of the plan in (2).)		
(5, 6)	Americans	*Goal*	stick to regulations
		Plan	ask Russians to remove peasants
		Goal	be nice to peasants
		Plan	act sympathetic
(7)	Americans	*Goal*	cover up embarrassment
		Plan	keep reporters away

Figure 15: Plans and Goals in the "Shabby Peasants" paragraph

To put a request is to perform a speech act; the object of the request is a likely candidate for the topic of the paragraph and is selected as the topic because it is the earliest candidate in the text.

This analysis shows that the topic is determined functionally, rather than just structurally. To get a better grasp on how "topic" is defined and how the topic of a paragraph is determined, as well as related issues in resolving Model Anaphora, we suggest that future research include testing modifications to the structure of the paragraph to discover how different rhetorical or syntactic structures result in different specifications for the Model Anaphora.

In figure 15 we present our preliminary research into the goals and plans that may be present in this paragraph. Future research should include refinement and implementation of this work.

6 Bibliography

Akmajian, A. and Frank W. Heny. *An Introduction to the Principles of Transformational Syntax.* The MIT Press, 1975.

Grosz, B. and C. Sidner, "Attention, Intentions, and the Structure of Discourse." *Computational Linguistics*, Vol. 12, No 3, pp. 175-204, 1986.

Hankamer, J. and I. Sag, "Deep and Surface Anaphora." *Linguistic Inquiry*, **7**(3), pp. 391-426, 1976.

Hinrichs, E., "Temporal Anaphora in Discourses of English." *Linguistics and Philosophy*, **9**(1), pp. 63-82, 1986.

Hobbs, J., "Resolving Pronoun References." *Lingua* **44**, pp. 311-338, 1978. Also in B. Grosz, K. Sparck-Jones, and B. Webber, eds., *Readings in Natural Language Processing*, pp. 339-352, Morgan Kaufman Publishers, Inc., 1986.

Huettner, Alison K., "Adjunct Infinitives in English." unpublished Ph.D. thesis, Linguistics Department, University of Massachusetts at Amherst, 1989. A complete citation was unavailable at time of publication.

Huff, K. and V. Lesser, *The GRAPPLE Plan Formalism.* University of Massachusetts, COINS Technical Report 87-08, 1987.

Paivio, A., J. Yuille, and S. Madigan, "Concreteness, Imagery, and Meaningfulness Values for 925 Nouns." *Journal of Experimental Psychology Monograph Supplement*, **76**, pp. 1-25, 1968.

Partee, B., "Nominal and Temporal Anaphora." *Linguistics and Philosophy*, **7**(3), pp. 243-286, 1984.

Rosch, E., C. Mervis, W. Gray, D. Johnson, and P. Boyes-Braem, "Basic Objects in Natural Categories." *Cognitive Psychology*, **8**, pp. 382-439, 1976.

Sidner, C., *Towards a Computational Theory of Definite Anaphora Comprehension In English Discourse.* Massachusetts Institute of Technology, Technical Report TR-537, 1979.

Webber, B., "So What Do We Talk About Now?" In M. Brady and R. Berwick, Eds., *Computational Models of Discourse*, The MIT Press, pp. 331-371, 1983.

Webber, B., "Event Reference." In *Position Papers for TINLAP-3: Theoretical Issues in Natural Language Processing-3*, Las Cruces, NM, pp. 137-142, 1987.

Webber, B., "Discourse Anaphora." In *Proceedings of the 26th Annual Meeting of the Association for Computational Linguistics*, Buffalo, pp. 113-122, 1988.

Woods, William, "Transition Network Grammars for Natural Language Analysis." In *Communications of the ACM*, **13**(10), 1970.

A Grammar

A.1 Verbal Elements in the SNePS ATN

This section explains the way the SNePS ATN processes words that part of the verb, meaning the main verb and any auxiliaries, negatives and such. We call such words "verbal elements." Since the SNePS ATN as delivered did not handle verbal elements correctly, and since we could not understand the code that seemed like it ought to handle them, we threw out the old code and completely reimplemented it.

Our first attempt at handling verbal elements was a failure. We were trying hard to go forward in a principled way, from CFG + TG rules to either a pure CFG or an FSM.[12] The problems mostly arose because as we multiply out the CFG + TG rules, we first get a huge increase in the number of rules, which then must be reduced by looking for identities or similarities between rules or states. Similarities between rules or states don't really result in any reduction in the number of rules or states, but there is a reduction in the size of the NOTATION, because rules or states can be parameterized, yielding an identity can then be collapsed. But this is still really hard, and properly collapsing the notation was very confusing.

This grammar presents a new approach, by trying to go directly from the CFG + TG rules to an ATN, which is in consonance with the original intention of Woods' ATN formalism (1970). In fact, we will end up parameterizing the rules even more than they are already! These rules are primarily drawn from Akmajian and Heny (1975).

We start with:

> S → NP VS VP-REST
> VS → (MODAL) (have) (be) V
> plus the TG rules of Aux-Hopping, Negation, and Question Formation. And don't forget DO-support.

First, we note that for each of the auxiliaries, much the same thing is happening on the resulting ATN arc, to wit: if the auxiliary is of the right category (MODAL, 'have,' or 'be'), and the auxiliary is in the right form (does it have the suffix which has hopped in from the left?), traverse the arc. So let's simplify the CFG rule and add a constraint.

> VS → AUX* V
> AUX → MODAL | have| be
> Constraint: No auxiliary appears more than once. Auxiliaries go in the following order: MODAL, 'have,' and 'be.'

The rule of Negation adds that if the auxiliary is the first one, it can be followed by a negative morpheme. That simply goes into the constraint:

> VS → AUX | not * V
> AUX → MODAL | have | be
> Constraint: No auxiliary appears more than once. Auxiliaries go in the following order: MODAL, 'have,' and 'be.' Optionally, 'not' immediately follows the first auxiliary.

[12]CFG = Context-Free Grammar; TG = Transformational Grammar; FSM = Finite State Machine.

Let's throw in DO-support:

> VS → AUX | not * V
> AUX → do | MODAL | have | be
> Constraint: No auxiliary appears more than once. Auxiliaries go in the following order: MODAL, 'have,' and 'be.' If 'do' is an auxiliary, then it is the first and only auxiliary (if you don't believe in DO-support for positive declaratives, then 'do' must be followed by 'not'). Optionally, 'not' immediately follows the first auxiliary.

The second step is going to look weird, but bear with me. We want to handle Question Formation. So, let's eliminate the ordering of subject and auxiliary from the CFG rules and put it into the constraint:

> S → VS VP-REST
> VS → NP | AUX | not * V
> AUX → do | MODAL | have | be
> Constraint: No auxiliary appears more than once. Auxiliaries go in the following order: MODAL, 'have,' and 'be.' If 'do' is an auxiliary, then it is the first and only auxiliary (if you don't believe in DO-support for positive declaratives, then 'do' must be followed by 'not'). Optionally, 'not' immediately follows the first auxiliary. NP appears exactly once and precedes or follows the first auxiliary (together with 'not').

The third step is to convert this stuff into an ATN. We only need two states to deal with the VS rule (the S rule is obvious), with the second of those states being trivial (it reads the V). The arcs of the first state are: PUSH NP, CAT V, CAT NEG, and JUMP. Why the JUMP? That's because many of these items are optional, and the JUMP is like an epsilon transition in an NFA.

The constraints are implemented via registers in the ATN, plus lisp code in the tests and actions on the arcs. The information about modals and auxiliaries is available when the node is finally built (in VP-END), but is discarded at that point, since we don't know how to represent tense, aspect and modal information. If someone has an idea, the information is all there.

A.2 The ATN

This section contains all of the actual code of the ATN.

```
(S
  ;; The words 'no' and 'yes' can be sentences on their own
  (wrd (yes no) t
       (setr propositional-anaphor |*|)
       (to YES-NO))
  (cat wh-adjunct t
       (initialize-vs-machine)
       (setr mood :why-question)
       (to VS))
  (jump VS t
        (initialize-vs-machine)))


(YES-NO (pop (progn
               (lisp::push (cons 'contains-propositional-anaphora? 't)
                           sneps::*sentence-facts*)
               (setr net
                     (build equiv1 (lisp-escape (new-base-node some-proposition))
                            equiv2 (lisp-escape
                                     (if (string-equal
                                           "yes"
                                           (getr propositional-anaphor))
                                         (build lex true)
                                         (build lex false)))))
               (getr net))
            t))
```

```
(VS (PUSH NP (and (null (getr subj))
                  (member (length (getr auxes)) '(0 1)))
          (sendr v v)
          (setr prev (getr |*|))
          (setr subj (getr |*|))
          (new-case-frame :agent '|*|)
          (if (and (null (getr auxes))
                   (null (getr v)))
              (setr mood :declarative)
              (if (null (getr mood))
                  (setr mood :yes-no-question)))
          (TO VS))
    (CAT MODAL (acceptable-verbal-element? :modal)
         (process-auxiliary :modal)
         (TO VS))
    (CAT V (acceptable-verbal-element? :do)
         (process-auxiliary :do)
         (TO VS))
    (CAT V (acceptable-verbal-element? :have)
         (process-auxiliary :have)
         (TO VS))
    (CAT V (acceptable-verbal-element? :be)
         (process-auxiliary :be)
         (TO VS))
    (CAT NEG (and (eql (length (getr auxes)) 1)
                  (member (getf ctgy (getr prev)) '(v modal))
                  ;; the following means the ATN doesn't allow double negatives,
                  ;; such as 'John can't not go to the party.' or
                  ;; 'John can't be not going to the party.'
                  (null (getr neg)))
         (setr prev *current-word*)
         (setr neg t)
         (TO VS))
    (CAT V (and (acceptable-verbal-element? :v)
                ;; The only main verbs that can appear before the subject are
                ;; 'have' and 'be.'  See the test suite for examples.
                (or (not (null (getr subj)))
                    (member (getr |*|) '("have" "be") :test #'string-equal)))
         (talk 2 "main verb")
         (setr v (get-lexical-feature 'root (getr features)))
         (when (null (getr auxes))
           (internal-setr neg (get-lexical-feature :negative (getr features))))
         (if (and (null (getr subj))
                  (null (getr mood)))
             (setr mood :yes-no-question))
          (TO VS))
     (JUMP VP-ARGS (and (not (null (getr subj)))
                        (not (null (getr v)))
                        (typep (getr mood) 'legal-mood))))
```

```
;;; This node collects arguments of the verb.

(vp-args
  (push np (or (getf trans v) (getf obj+pplocative v))
        (sendr v v)
        (setr obj |*|)
        (new-case-frame :object '|*|)
        (to vp-args))
  (wrd if (getf condv v)
       (to vp-if))
  (cat adj (equal (getf root (getr v)) "be")
       (setr predicate |*|)
       (setr type :predicate-adjective)
       (to vp-adjuncts))
  (push scomp (getf infinitive-scomp v)
        (sendr case-frames case-frames)
        (sendr subj obj)
        (setr iobj obj)
        (setr obj |*|)
        (new-case-frame :object '|*|)
        (new-case-frame :indirect-object 'iobj)
        (to vp-adjuncts))
  (push s (getf scomp v)
        (sendr case-frames case-frames)
        (setr obj |*|)
        (new-case-frame :object '|*|)
        (to vp-adjuncts))
  (cat prep (member (getr |*|) (getf :particles (getr v))
                    :test #'string-equal)
       (setr particle |*|)
       (to vp-particle))
  (push pp  t
        (sendr v (getr v))
        (addr vmods |*|)
        (let ((node  (cadr (getr |*|)))
              (role  (intern (the (and symbol (not null)) (car (getr |*|)))
                             *keyword-package*)))
          (new-case-frame role node))
        (to vp-args))
  (jump vp-adjuncts t))
```

```
;;; This node ought to collect adjuncts to the verb, such as time/location
;;; NPs and PPs or rationale clauses.  It only does the last, since the PPs
;;; are treated uniformly as vmods, and therefore are collected in the
;;; VP-ARGS node.
(vp-adjuncts
  (push rationale-clause t
        (sendr subj (getr subj))
        (sendr case-frames case-frames)
        (setr rationale |*|)
        (to vp-end))
  (jump vp-end t))


;;; This node assumes that if you have two NPs, the one preceding the
;;; particle is the IOBJ and the one following is the OBJ.
(vp-particle
  (push np t
        (sendr v v)
        (setr iobj obj)
        (setr obj |*|)
        (new-case-frame :object '|*|)
        (setf (car (member :object (getr case-frames))) :indirect-object)
        (to vp-args))
  ;; For sentences like ``call John up''
  (jump vp-args t))
```

```
(vp-end
  (pop (progn
         (setr net
               (lisp-escape
                 (if (eq (getr type) :predicate-adjective)
                     (eval
                       '(build
                          ,@(if *increment-time?*
                                '(stime   ,(node-named sneps::stm)
                                          etime   ,(node-named sneps::etm)))
                          rel     ,(build lex (lisp-escape (getr predicate)))
                          arg1    ,(lisp-escape (getr subj))))
                     (eval
                       '(build
                          ,@(if *increment-time?*
                                '(stime   ,(node-named sneps::stm)
                                          etime   ,(node-named sneps::etm)))
                          agent           (lisp-escape (getr subj))
                          verb            ,(if (getr particle)
                                               (sneps-symbol
                                                 (string-append
                                                   (getr v) "-" (getr particle)))
                                               '(build lex (lisp-escape
                                                             (getr v))))
                          ;; if an arc would point to NIL, it's not built
                          indirect-object (lisp-escape (getr iobj))
                          object          (lisp-escape (getr obj))
                          rationale       (lisp-escape (getr rationale))
                          . ,(lisp-escape (getr vmods)))))))
         (if (not (null (getr neg)))
             (setr net (lisp-escape
                         (eval
                           '(build rel  (build lex not)
                                   arg1 ,(getr net))))))
         (setr net
               (lisp-escape
                 (ecase (getr mood)
                   (:declarative     (getr net))
                   (:yes-no-question (eval '(build rel  (build lex truth-value)
                                                   arg1 ,(getr net))))
                   (:why-question    (eval '(build rel  (build lex why)
                                                   arg1 ,(getr net))))
                   (:how-question    (eval '(build rel  (build lex how)
                                                   arg1 ,(getr net)))))))
         (new-case-frame :case-frame 'net)
  (liftr case-frames case-frames)
  ;; This works because VP-END will be the last node executed, so the
  ;; case-frames register will be right.
  (setq sneps::*case-frames* (getr case-frames))
  (when *increment-time?*
    (build after  (new-base-node sneps::nstm)
           before (node-named    sneps::stm))
    (build after  (new-base-node sneps::netm)
           before (node-named    sneps::etm))
    (remember-node (node-named sneps::netm) sneps::etm)
    (remember-node (node-named sneps::nstm) sneps::stm))
  (getr net))
;; an incredible amount of consistence-checking here:
(or (and (getf intrans v)
         (nullr obj)
         (nullr iobj))
    (and (getf trans v)
         (not (nullr obj))
         (nullr iobj))
    (and (getf scomp v)
         (not (nullr obj))
         (nullr iobj))
    (and (getf infinitive-scomp v)
         (not (nullr obj))
         (not (nullr iobj)))
    (and (getf obj+pplocative v)
         (not (nullr obj))
         (nullr iobj)
         (not (nullr vmods))
         (contains-locative-pp? (getr vmods)))
    (and (getf small-clause v)
         (not (nullr obj))
         (nullr iobj))
    (and (getf condv v)
         'what??)
    (and (getf prop-att v)
         'what??))))
```

```
(vp-if
  (push s t
        (sendr case-frames case-frames)
        (setr type 'tv)
        (setr iobj obj)
        (setr obj  (lisp-escape (build arg1 (lisp-escape (getr |*|))
                                       rel  (build lex truth-value))))
        (new-case-frame :if-obj 'obj)
        (sneps::= (lisp-escape (find arg2- (lisp-escape (getr obj))))
                  some-proposition)
        (to vp-args)))


(scomp
  (jump scomp-subj t))


(scomp-subj
  (wrd to t
       ;; This is much like (initialize-vs-machine), but omits some things
       ;; because we're in a more constrained situation.
       (setr neg  nil)
       (setr mood nil)
       (setr v nil)
       (to scomp-to)))


(scomp-to
  ;; the following means the ATN doesn't allow double negatives in infinitive
  ;; complements, such as 'Mary requested John to not not go to the party.'
  (CAT NEG (null (getr neg))
       (setr neg t)
       (TO SCOMP-TO))
  (CAT V ([u]member :infinitive
                    (get-lexical-feature :untensed-forms (getr features))
                    englex:*legal-untensed-forms*)
       (setr v |*|)
       (setr mood :declarative)
       (to scomp-v)))
```

```
(scomp-v
  (push vp-args t
        (sendr subj subj)
        (sendr v    v)
        (sendr neg  neg)
        (sendr mood mood)
        (sendr case-frames case-frames)
        (setr  predicate |*|)
        (to scomp-end)))


(scomp-end
  (pop (progn (liftr case-frames)
              (getr predicate))
       t))


(comp
  (cat v ([u]member :infinitive
                    (get-lexical-feature :untensed-forms (getr features))
                    englex:*legal-untensed-forms*)
       (setr v |*|)
       (to vp-args)))
```

```
;;; The following deals with rationale clauses

(RATIONALE-CLAUSE
  (wrd to t
       (to rationale-clause-v)))


(rationale-clause-v
  (cat v ([u]member :infinitive
                    (get-lexical-feature :untensed-forms (getr features))
                    englex:*legal-untensed-forms*)
       (setr v |*|)
       (to rationale-clause-args)))


;; doesn't deal with double-object verbs:
(rationale-clause-args
  (push np (and (getf trans v)
                (not (getr obj)))
        (sendr v v)
        (setr obj |*|)
        (new-case-frame :rat-obj 'obj)
        (to rationale-clause-adjective))
  (jump rationale-clause-adjective t))


(rationale-clause-adjective
  (cat adj (getf small-clause v)
       (progn "to deal with elided small-clause subjects, as in
              'john lifted weights to keep <John> strong'"
              (if (null (getr obj))
                  (setr obj subj)))
       (setr obj
             (build rel  (build lex (lisp-escape (getr |*|)))
                    arg1 (lisp-escape (getr obj))))
       (to rationale-clause-adjuncts))
  (jump rationale-clause-adjuncts t))


(rationale-clause-adjuncts
  (push pp t
        (sendr v v)
        (addr vmods |*|)
        (to rationale-clause-adjuncts))
  (jump rationale-clause-end t))
```

```
(rationale-clause-end
  (pop (progn (setr rc
                    (lisp-escape
                       (eval
                          '(build agent  (lisp-escape (getr subj))
                                  verb   (build lex (lisp-escape (getr v)))
                                  object (lisp-escape (getr obj))
                                  . ,(lisp-escape (getr vmods))))))
              (new-case-frame :rationale-clause 'rc)
              (liftr case-frames)
              (getr rc))
       t))
```

```
;;; The following deals with Noun Phrases.  The naming is confusing: np-X
;;; usually means that the X was read by the preceding state--you have
;;; read the code to figure out what the current node does.

(NP
  (cat det t
       (setr art |*|)
       (to np-art))
  (jump np-art t))


(np-art
  (cat quant t
       (setr quant |*|)
       (to np-quant))
  (jump np-quant t))


(np-quant
  (cat adj t
       (addl adjs |*| (if (get-lexical-feature 'pos (getr features)) :pos :adj))
       (to np-quant))
  (jump np-adj t))
```

```
;;; Dealing with the head noun

(np-adj
  (cat npr t    ; real noun
       (setr nh
             (find named-
                   (lisp-escape
                    (or (find (sneps::compose sneps::name sneps::lex)
                              (lisp-escape (getr |*|)))
                        (build named (lisp-escape (new-base-node dummy))
                               name  (build lex (lisp-escape (getr |*|))))))))
       (lisp:assert (atom (getr nh)) (getr nh)
                    "NH should be a single node, not a list: ~s" (getr nh))
       (case (get-lexical-feature 'gen (getr features))
         (m  (sneps::=-fun (getr nh) 'most-recent-male))
         (f  (sneps::=-fun (getr nh) 'most-recent-female))
         (T  nil))
       (setr n |*|)
       (setr n-type :proper-noun)
       (setr nu (get-lexical-feature 'num (getr features)))
       (to np-n))
  (cat pron t
       (setr nh (case (get-lexical-feature 'gen (getr features))
                      (m  (node-named most-recent-male))
                      (f  (node-named most-recent-female))
                      (n  (error "We don't deal with 'it'"))
                      (t  (error "Unknown gender feature"))))
       (lisp:assert (not (null (getr nh))) ()
                    "Couldn't find reference for pronoun.~%~
                     Have we encountered a proper noun of the right gender yet?")
       (setr n |*|)
       (setr n-type :pronoun)
       (to np-n))
(cat n t      ; real noun
     (setr n |*|)
     (setr n-type :common-noun)
     (setr noun-class (lisp-escape (class-name (getr |*|))))
     (if (equal (getr art) "the")
         (setr nh (lisp-escape
                    (eval '(let ((node  (find (sneps::member- sneps::class)
                                              ,(getr noun-class))))
                             (lisp:assert
                               (null (cdr node)) (node)
                               "'The X' should yield only one node: ~s" node)
                             (if node
                                 (progn (eval '(assert rel     definite
                                                       defarg  ,node))
                                        node)
                                 (progn (assert class  ,(getr noun-class)
                                                member (new-base-node dummy))
                                        (assert rel    definite
                                                defarg   (node-named dummy))
                                        (node-named dummy)))))))
         (setr nh (lisp-escape
                    (eval '(progn (assert class  ,(getr noun-class)
                                          member (new-base-node dummy))
                                  (assert rel    indefinite
                                          defarg   (node-named dummy))
                                  (node-named dummy))))))
     (case (get-lexical-feature 'gen (getr features))
       (m  (sneps::=-fun (getr nh) most-recent-male))
       (f  (sneps::=-fun (getr nh) most-recent-female))
       (T  nil))
     (setr nu (get-lexical-feature 'num (getr features)))
     (to np-n))
(cat n nil    ; for complex noun?
     (setr n |*|)
     (addl adjs |*| :noun)
     (to np-adj)))
```

```
;; In certain circumstances, proper nouns and pronouns can take PP modifiers
;; (e.g. ``Tom in the next room'' or ``You with the nose,'' but allowing this
;; usually means that the first parse is an unwanted parse, so we are
;; disallowing PP modification of nouns other than common nouns.  SDA 5/10/89
(np-n
  (push pp ;; hack alert!
        (and
          (eq (getr n-type) :common-noun)
          ;; either both can apply to a physical object, or neither (hack!)
          (eq (get-lexical-feature 'non-physical (getr features))
              (get-lexical-feature 'non-physical (getr n))))
        (sendr v v)
        (build arg1 (lisp-escape (getr nh))
               rel  (lisp-escape (car (getr |*|)))
               arg2 (lisp-escape (cadr (getr |*|))))
        (to np-n))
  (pop  (progn
          (lisp:assert (not (null (getr nh))) () "register NH is null")
          (do* ((adjs (getr adjs) (cddr adjs))
                (word (car adjs) (car adjs))
                (prop (cadr adjs) (cadr adjs)))
               ((null adjs))
            ; because of odd behavior by ADDL
            (eval '(assert arg1 ,(getr nh)
                           rel  (build lex ,(case prop
                                               ((:adj :noun) 'has-property)
                                               (:pos 'is-part)))
                           arg2 (build lex ,word))))
          (getr nh))
            ;; adjs - adjectives -- associate via hasproperty-link
            ;; the predicate DETAGREE tests for agreement between
            ;; the ART and N registers to screen out ``a books'' and ``an table''
        (detagree)))
```

```
;;; The following deals with prepositional phrases.

(pp
  (cat prep (not (member (getr |*|) (getf :particles (getr v))
                         :test #'string-equal))
       ;; This code converts the prep to a relation
       (setr tmp |*|)
       (if (not (stringp (getr tmp)))
           (error "preposition is not a string:  ~s" (getr tmp)))
       (setr tmp (string-upcase (getr tmp)))
       (setr tmp (intern (getr tmp) (find-package 'sneps)))
       (if (not (relation? (getr tmp)))
           (error "preposition is not a relation:  ~s" (getr tmp)))
       (setr prep tmp)
       (to pp-prep)))


(pp-prep
  (push np (member (getf ctgy) '(n npr det adj))
        ; predicate fails if the next word cannot begin an NP
        (sendr v v)
        (setr np |*|)
        (to pp-np)))


(pp-np
  (pop (list (getr prep) (getr np))
       t))
```