26

Inference and Knowledge in Language Comprehension

Eugene Charniak

Institute for Semantic and Cognitive Studies
Castagnola, Switzerland

INTRODUCTION

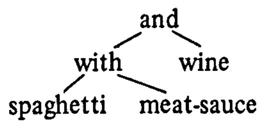
Why make inferences?

To use language one must be able to make inferences about the information which language conveys. This is apparent in many ways. For one thing, many of the processes which we typically consider "linguistic" require inference making. For example, structural disambiguation:

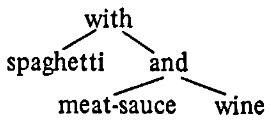
(1) Waiter, I would like spaghetti with meat sauce and wine.

You would not expect to be served a bowl of spaghetti floating in meat sauce and wine. That is, you would expect the meal represented by structure (2) rather than that represented by (3).

(2)



(3)



Note however that if you had asked for spaghetti with butter and garlic you would have wanted structure (3). So to get your order right the waiter must have made an inference based on his knowledge of food.

The same is also true for word sense disambiguation and reference determination, as seen in examples (4) and (5) respectively.

(4) The box was in the *pen*. [not the writing implement]
(5) At the supermarket Fred found the shelf where the milk was. He payed for *it* at the checkout counter and left. [Not the shelf but the milk. And to be precise, not the "milk" mentioned in the story, which is in some sense all the milk on the shelf, but a single carton of milk.]

Furthermore, any task which requires that a person understand language invariably requires the making of inferences. Some obvious examples are: question answering (What did Fred buy?); translation (Translating (5) into German requires finding the referent for "it" since "shelf" and "milk" have different genders); summarizing (It's hard to give a short example of why inference is needed here, but the general idea behind summarizing a story is to find the "main thread". However, the connecting links of the thread are generally not stated explicitly, but rather have to be inferred).

The need for knowledge

In each of the above examples, in order to make the necessary inference we needed information beyond that provided by the example. So in (5) we need to know that shelves are usually not purchased at supermarkets, and that one would not generally buy all the milk which the supermarket had on its shelves. In example (1) we must know about wine and its use as a beverage, and that while it is sometimes used as an ingredient for food, one typically does not specify the ingredients of a dish to a waiter, but rather chooses something from the menu. This paper, then,is about how *knowledge* is used to make *inferences* in the comprehension of language.

Five questions

A complete solution to this problem would require answering the following five questions. I will only state them briefly here. Their precise meaning will be best understood by the repeated reference to them throughout the paper.

1. SEMANTIC REPRESENTATION. What concepts, and in what combinations are needed to record our impressions of the world? This subject will only be referred to peripherally here.
2. INFERENCE TRIGGERING. Under what circumstances, and for what reasons do we make inferences? For example, when do we make an inference, when new information comes in, or when a question is asked? We also make inferences to help determine reference. Does this affect our answers to any of the other questions?
3. ORGANIZATION. Given we want to make an inference, how do we locate the needed information? The all purpose response here is that we have "pointers" to the information. The question then becomes how do we organize these pointers? So we might, for example, imagine that certain pieces of information, like that about supermarkets, are "grouped" together. What "grouping" might mean is that there is a pointer from some topic to each piece of information.
4. INFERENCE MECHANISM. Once we have located a fact, how do we know how to use it? For example, if facts were represented as programs, the interpreter for the program would tell us how to use the fact.
5. CONTENT. What is the knowledge which we have of the world that

enables us to understand language? Note that while the answers to the other questions would hopefully be culture independent (with the possible exception of (1)), the knowledge we have is clearly quite culture dependent, and even idiosyncratic.

FIRST ORDER PREDICATE CALCULUS

One set of answers, or at least partial answers, is given by assuming that the first order predicate calculus (FOPC) can be used as a theory of inference and knowledge. I will assume the reader is familiar with FOPC (and more specifically resolution theorem proving) and go on to consider what answers it gives to these questions.

1. SEMANTIC REPRESENTATION. Does FOPC tell us what predicates are needed to represent our impressions of the world? Well, for the most part the answer must be "no". We have seen that FOPC leaves us free to use whatever predicates we choose, with only a few restrictions. So it does not say if we should have a predicate TRADE(w,x,y,z) which means that w gave x to y for z, or if we should have a more basic, or "primitive" predicate. For example, Schank (Schank, 1973) argues that there is a more primitive verb, which he calls "ATRANS" which underlines not only "trade", but also "buy", "sell", "give", "take", etc. In particular, he would roughly represent TRADE(w,x,y,z) as

(6)	ATRANS (w,w,x,y)	[(w was the actor in transferring x from w to y)]
(7)	ATRANS (y,y,z,w)	[(y gave z to w)]
(8)	CAUSE ((6),(7))	[(6) and (7) mutually caused each other)]
(9)	CAUSE ((7),(6))	

   (Technically statements (8) and (9) as written are not legal FOPC statements, but there are well known ways to change things so that they are legal.) So the only answer FOPC gives to the question is that certain items like AND, OR and NOT are useful to use, but otherwise you can do much as you choose.
2. INFERENCE TRIGGERING. What does FOPC say about when and why we make inferences? Well, it is hard to be too firm about this, since one can always change things slightly, but the general idea behind FOPC is that one only makes inferences when one is asked a question. At the moment, this may not seem like much of a limitation but we will see that it is.
3. ORGANIZATION. How do we locate a needed fact in FOPC? FOPC in general, and resolution theorem proving in particular, say very little on the subject. The result has been the inability of systems

based on FOPC to handle large data bases, or complex inferences because the system gets lost in the combinatorial explosion.

4. INFERENCE MECHANISM. Given that we have located the facts we intend to use, does FOPC tell us what to do with them? Here the answer is an unequivocal "yes", especially when we are talking about resolution theorem proving. In resolution theorem proving, to produce a new fact from old there is only one thing you can do with the old facts–resolve them together. When you come right down to it, FOPC is primarily a theory of inference mechanism.
5. CONTENT. FOPC says nothing about what facts one needs to know about the world. It turns out that there are some facts which are difficult, if not impossible, to express in FOPC, but aside from that FOPC makes no suggestions.

To summarize, FOPC says a lot about inference mechanism, and a little about semantic representation. As we see then, one problem with FOPC is that it does not provide answers to most of our questions.

WHEN DO WE MAKE INFERENCES

And there are still other problem with FOPC. To understand them however it will be necessary to take a close look at the circumstances under which we will want our inference systems to make inferences. In particular we want to answer the simple question, when do we make an inference?

Let us assume we are talking about a question-answering system which accepts information expressed in a natural language and will answer questions based on it. Then there are two obvious times when we might make an inference. The first is when a question is asked which requires the inference be made. The second is that point in the input when the system has been given enough information to make the inference. We will call these two possibilities "question time" and "read time", respectively.

There are obvious advantages to making inferences only at question time. Given a particular set of facts there are an incredible number of inferences which might be drawn (remember the combinatorial explosion). We clearly cannot make all possible inferences, so by waiting until a question is asked we guarantee that we will only be making those inferences which we must make in order to answer the system user's question. However, there are good reasons why this is a bad thing to do.

As a psychological question

If we wanted to build a model which simulated people reading a text, we could ask the purely empirical question, do people make inferences about the text while they read. In a typical psychological experiment on the question, a subject will be given a piece of text to recall, only to have him "remember" facts which were not present in the text, but which were inferred from the text. To me this strongly suggests that the inferences were made at read time, although the psychologists note that the inferences could have been made at recall time.

Is it even possible to postpone all inference?

But even if we were not interested in a psychological model, we have already seen reasons why our model could not postpone all inference making until question time. It is not possible to do word sense or structural disambiguation, or noun phrase reference without making inferences. We saw several examples of this at the start of the paper, let me give one more here.

(10) Today was Jack's birthday. Janet and Penny went to the store. They had to get presents. "I will get a top," said Janet. "Don't do that," said Penny. "Jack has a top. He will make you take it back."

The problem here is that the "it" in the last sentence does not refer to the last mentioned inanimate object (Jack's top) but rather the second to last (the top Janet was thinking of getting). To get this reference correct one must make use of one's knowledge about exchanging things at stores.

Problem occasioned inference

Considerations like this have produced a consensus that some inference must be done at read time. The question now becomes how much and of what sort. A useful distinction in this regard is that of Wilks (Wilks, 1975) between "problem-occasioned" and "non-problem-occasioned" inference. A problem-occasioned inference is one which we perform in order to accomplish the translation into internal representation. It is called "problem-occasioned" because we perform such inference only when we run into a problem like an ambiguous word, or a noun phrase whose referent is in doubt. A typical example of a non-problem-occasioned inference is exhibited by the story "Janet shook her piggy-bank. There was no sound". The inference that there is nothing in the piggy-bank is non-problem-occasioned since no ambiguity in the story required this inference for its resolution.

It is my opinion that non-problem-occasioned inferences are needed even to perform the translation into internal representation. To see why I hold this view, consider the following text about chimpanzees taken from the book *In the Shadow of Man* by Jan van Lawick-Goodall.

(11) When Flint was very small his two elder brothers, although they sometimes stared at him, paid him little attention. Occasionally while *he* was grooming with his mother, Faben very gently patted the infant.

The problem here is to figure out who the underlined "he" refers to. In fact if you have not read the book it is doubtful that you could do so, but those who come across passage (11) in the course of reading the book have no trouble because they know fact (12):

(12) Baby chimpanzees do not groom.

This fact is of course covered earlier in the book, and using it we conclude that "he" cannot refer to Flint, who is a baby, but rather to Faben.

But suppose we were only doing problem-occasioned inferences. While I do not remember how fact (12) was introduced, it is quite likely that there was no statement in the book which directly stated (12). This could happen, for example, if the text said "Young chimpanzees do not groom with non-relatives. We later learned that the young chimpanzees' behavior is the same with relatives". So while on one hand the system would have to infer (12) by making inferences on earlier parts of the text, the problem is even worse because nothing in (11) tells us that grooming is the clue. It is true that we are told that "he" was grooming with his mother, but we are also told that the grooming was with "his mother", and that Flint has two elder brothers, that Faben patted Flint, etc. The combination of both of these problems makes it hard for me to imagine how a system which only performed problem-occasioned inference could handle (11). Furthermore, I think such examples are more common than we might think. In particular, I am prepared to argue that (10) (the "take it back" story) exhibits exactly the same problem although the argument showing that this is the case is quite complex.

Question answering and problem-occasioned inference

Finally, I suspect that it will also be extraordinarily difficult to do question answering on complex stories unless the system performs non-problem-occasioned inference. Consider the following story.

(13) Janet wanted to trade her coloured pencils for Jack's paints. Jack was painting a picture of an aeroplane. Janet said to him, "Those paints make your aeroplane look bad."

Notice that a system which only makes problem-occasioned inference would not note at read time that Janet's comment is less than unbiased truth. But suppose we asked such a system whether Janet believed what she said in this instance. How would the system decide to say no? Presumably the system would have a rule to the effect that we assume a person believes what he says unless we have reason to believe otherwise. But how would we show that Janet had reason to believe otherwise? The number of possible reasons why a person might lie seems too large to try to look for each one in the story. It would seem much more to the point to start with the facts in the story and infer that Janet has reason to lie (rather than vice-versa). But while the crucial lines in (13) occurred just prior to Janet's statement, in a real story the separation could be much larger, so it is again hard to imagine how the system could know where to look for the needed information if it had not been performing non-problem-occasioned inferences.

Implications

Assuming then that our program will be making inferences as it reads, what does this say about the system we use to make inferences?

1. CANNOT HAVE ONLY QUESTION DRIVEN INFERENCE. As we noted earlier, FOPC is a question driven system for making inferences. But we saw that not all inferences will be due to user questions, and even if we broaden our concept of question to include what we have called problem-occasioned inference, it is likely that many of our inferences will still not be question driven. On the other hand, we could not make all possible inferences at read time, there are simply too many. Hence our system must be able to distinguish between those inferences which are "important" and hence should be made immediately, and those which are not, and hence could be postponed until (and if) the need arises. For example, in story (10) it would seem reasonable to postpone the inference that Janet has lungs and a heart.

   This is not to say, however, that these problems could not be overcome in a FOPC system. Perhaps it will be possible to come up with some set of all purpose questions like "Why was I told that?" and "Why did that happen" such that a FOPC system could handle "data driven inference" by stopping after every line and asking each of the questions. But one is entitled to be suspicious of a system when it is necessary to circumvent its natural properties. Besides, in stopping to ask questions like this one has already moved some distance from a pure FOPC system.

2. TOLERANCE FOR CONTRADICTION. It is not hard to see that any system which is making inferences as the story comes in is bound to make a mistake now and then. For example, instead of story (10) where we assumed that Janet and Penny were going to the store in order to buy presents we could have had:

   (14) Today was Jack's birthday. Janet and Penny went to the store. They had to get presents. They had decided to make the presents and needed glue from the store.

   That is to say, while it is reasonable to assume that they are going to buy presents at the store, it is not an absolute certainty. So in (14) the new information must be seen as contradicting our inference, so that the inference must be withdrawn. So our system must be able to cope with seeming contradictions. However, this is another weakness of FOPC. In fact, it is a well known property of FOPC that anything can be proved from a contradiction. To see this in the resolution theorem prover we outlined, all we have to do is note that if both A and NOT A are clauses, all we have to do is resolve them and we get the null clause, hence proving whatever we set out to prove. Again, one might come up with various ways to get around this problem, but it is once again the case that the inherent properties of FOPC do not seem well suited to the task at hand.

PLANNER

What then are the alternatives to FOPC? One possibility is just to use the natural properties of LISP or some other programming language to make inferences. This is, in fact, what was done in some of the early question answering programs, for example Raphael's SIR (Semantic Information Retrieval) (Raphael, 1968).

SIR

To explain what Raphael did, it is necessary to back up a minute and explain LISP property lists. In LISP every atom has something called a property list. So, if we wanted to say that a nose is part of a person, one could represent this in LISP in the following way.

$$\begin{array}{ll} \text{PERSON} & \xrightarrow{\text{property list}} \text{(SUBPART(NOSE))} \\ \text{NOSE} & \xrightarrow{\hspace{5em}} \text{(SUPERPART(PERSON))} \end{array}$$

That is, the property list is just a list of pairs, where the first item is the property, and the second is the value of the property. If we wanted to add that a heart was also part of a person, and that girls were a subclass of people we would add on:

$$\begin{array}{ll} \text{HEART} & \xrightarrow{\hspace{5em}} \text{(SUPERPART(PERSON))} \\ \text{GIRL} & \xrightarrow{\hspace{5em}} \text{(SUPERSET(PERSON))} \end{array}$$

and we would change the property list of PERSON.

$$\text{PERSON} \xrightarrow{\hspace{5em}} \text{(SUBSET(GIRL) SUBPART(HEART NOSE))}$$

Raphael did exactly this. He had programs which translated a few kinds of sentences into such property list structures, and then he wrote programs which searched these structures in order to answer questions. So if we asked "Does a girl have a heart," the program for SUBPART would first look to see if there was a SUBPART property on GIRL, and failing this would note that GIRL was a subset of PERSON so it would look to see if PERSON had a SUBPART property. This of course would succeed in the above case and the program would respond YES.

But LISP has problems as an inference system. Raphael at the end of (Raphael, 1968) notes that SIR was becoming unmanageable, because when he wanted to add some new facility, it often required rewriting some of the old property list search routines. He then suggests that FOPC would be the solution to this problem. Indeed it would, but as we have already seen, at the cost of giving us many new problems.

An alternative is to make programming languages more suited to the needs of inference making. This is exactly what Hewitt did when he designed the programming language PLANNER (Hewitt, 1969). How PLANNER has much in common with unicorns: we know quite a bit about it, but it never existed. That is, no language called PLANNER has yet been implemented. Nevertheless, the idea of PLANNER has been quite influential, probably due to the fact that Winograd used a hastily implemented subset of PLANNER called MICRO-PLANNER (Sussman, *et al.*, 1971) in his now famous SHRDLU program. As with FOPC, I will assume the reader is familiar with at least the rudiments of PLANNER.

PLANNER vs. LISP

What exactly does PLANNER offer over LISP as a language for making inferences?

1. DATA BASE MANAGEMENT. As we saw in SIR, it is possible to construct a data base in LISP, but the functions ASSERT, GOAL, ERASE make it much easier to do it.
2. PATTERN MATCHING MICRO-PLANNER offers a primitive pattern matching facility, so we can pull things out of the data base by knowing part of the pattern. We can bind variables at the same time. PLANNER, if it is ever written, will have a much more sophisticated pattern matching facility.
3. PATTERN DIRECTED INVOCATION. Theorems could be called on the bases of their pattern. In effect this allows us to write a function which calls a second function without ever knowing the name of the second function, but rather what it is supposed to do (insofar as we can represent this in the pattern of the theorem).
4. BACK TRACKING. This is one of the controversial features of PLANNER, but there seems little doubt that used in moderation backtracking is a useful feature. For example, one use made of it in SHRDLU occurs when the reference procedure wishes to locate a "big blue block". The general idea is to set up three goals, the first looking for a block, the second checking that it is blue, and the third checking for size. It makes sense in the case like this to use simple back tracking because there is no way to "guess" in advance which block B1, B2 or B3 say, is the one which will have all three properties. The criticism of back tracking is that it tends to encourage the construction of programs which depend too heavily on blind search. This criticism is well taken, and it was clearly a design mistake in MICRO-PLANNER that it is almost impossible to turn back tracking off. But there are cases where the search must be blind, and backtracking is a good thing to have in such cases.

PLANNER vs. FOPC

PLANNER does not suffer from the problems of FOPC. In particular:

1. COMBINATORIAL EXPLOSION. It is not hard to write programs in

PLANNER which suffer from combinatorial explosion, but on the other hand the language offers at least the possibility to write inference systems which do not. The prime idea here is that PLANNER (but not FOPC) allows the user to specify how a goal (i.e., theorem) is to be established. For example, we can tell it only to look to see if it is already in the data base, or we can specify that it should try to infer the fact from other facts. Even this already gives us some control over how much computation is done in search of an answer, but there are other options also, like "use the theorems named. . . .first, only use the theorems named. . . ., use any theorem with the following property first. . . ., use only theorems with the following property. . . . This in effect allows the user to include information which should help the system make the needed inferences without dying a horrible death by combinatorial explosion.

2. COPING WITH CONTRADICTION. This is no problem in PLANNER. Since one writes one's own theorems, one would have explicitly to write a theorem which derived anything from a contradiction before that would happen. Furthermore, PLANNER is very well suited to one particular kind of semi-contradiction which comes up all the time. We would like to be able to say that all people have two legs, without worrying about the few rare cases where this is not true. On the other hand if Bill only has one leg we should be able to note the fact. In PLANNER this is done by making "Bill has one leg" an assertion, and "All people have two legs" a theorem. Since the data base is checked first when trying to establish a goal, the system will find that Bill has one leg before it attempts to use the general theorem to show that he has two legs.
3. DATA VS. QUESTION DRIVEN INFERENCE. While it was difficult to accommodate data driven inference in FOPC, there is no problem in PLANNER. Indeed, antecedent theorems exactly cover this situation.

Answers to the five questions

Finally what does PLANNER say in response to the five questions we asked initially?

1. SEMANTIC REPRESENTATION. Essentially nothing. There is nothing in PLANNER which in any way restricts what we can put in an assertion. If we assume that some of our impressions of the world get translated into theorems, then PLANNER does say a little about the form of the theorem, namely that it is written in PLANNER, but that is hardly much of a restriction.
2. INFERENCE TRIGGERING. Again very little. As we have already noted, one of the benefits of PLANNER is that it does not make the

restrictions that FOPC makes in this area.

3. ORGANIZATION. Here PLANNER does tell us something, but in a very qualified way. PLANNER offers several built-in organizational features. The primary one is pattern directed invocation. The secondary one is the means given to choose which theorems will be used to satisfy a goal, or react to a new piece of information. However, it should be remembered that PLANNER is a programming language, and it is certainly possible to program in other organizations. Nevertheless, it would seem fair to criticise PLANNER if the built-in organizational features it offered were not the ones we needed. As we will see later, there is some reason to believe that this is the case.
4. INFERENCE MECHANISM. Like FOPC, PLANNER is primarily a theory of inference mechanism. Given a PLANNER theorem, there is no doubt about how to use it, hand it to the PLANNER interpreter and have it executed.
5. CONTENT. Again nothing.

A DEMON BASED SYSTEM

But what are the problems with PLANNER? Surely it must have some or else it would have been announced some time ago that all the problems of memory and knowledge have been solved. Well, we have seen one problem with PLANNER already in passing. Since it is a programming language, as well as a theory of fact use, it is possible to use PLANNER in so many ways that it is hard to pin down exactly what PLANNER commits one to. Now a programmer would call this flexibility, but a theorist must call it vagueness. So the first problem with PLANNER as a theory of knowledge and inference is that it is a vague theory.

Naturally, this first problem makes it quite difficult to find other problems. To do so we will have to stop looking at PLANNER itself, and instead look at some of the ways it has been used. In particular we will look at my work on children's stories. I will then suggest that some of the problems we see in my model are really problems with PLANNER itself.

The problem my model (or system) is concerned with is that which I have tacitly assumed throughout this paper; namely, given a piece of simple narration (or children's story) like (15) the system should be able to answer reasonable questions like (16)-(18).

(15) Janet needed some money. She got her piggybank (PB) and started to shake it. Finally some money came out.

Some typical questions would be:

(16) Why did Janet get the PB?
(17) Did Janet get the money?
(18) Why was the PB shaken?

The reader should be sufficiently attuned by this point to recognize that to answer these questions real world knowledge is needed. The model is solely concerned with the problem of using the necessary real world knowledge to make inferences and it does not explicitly consider problems of natural language *per se*. In particular it does not deal with syntax, and while it does deal to some degree with those problems at the boundary between syntax and inference (like determination of noun phrase reference) I will not consider these topics here.

We will assume that as the story comes into the program it is immediately translated into an internal representation which is convenient for doing inference. The internal representation of a sentence will be a group of MICRO-PLANNER assertions. The model will try to "fill in the blanks" of the story on a line by line basis. That is, as it goes along, it will try to make connections between events in the story (usually causal connections) and fill in missing facts which seem important.

Demons and base routines

Consider a fact like:

> (19) If "it is (or will be) raining" and if "person P is outside" then "P will get wet."

We have an intuitive belief that (19) is a fact about "rain", rather than, say, a fact about "outside". Many things happen outside and getting wet is only one of them. On the other hand only a limited number of things happen when it rains.

We will embody this belief in our system by associating (19) with "rain" so that only when "rain" comes up in the story will we even consider using rule (19). We will say that rain is the "topic concept" of (19). To put this another way, when a concept is brought up in a story, the facts associated with it are *"made available"* for use in making inferences. (We will also say that the fact are *"put in"* or *"asserted".*) So, if "circus", say, has never come up, the program will not be able to make inferences using those facts associated only with "circus".

Note however that we are not saying that "rain" has to be mentioned explicitly in the story before we can use (19). It is only necessary that there be a "rain" assertion put into the data base. Other parts of the story may provide facts which cause the program to assert that it is raining. For example:

> (20) One afternoon Jack was outside playing ball with Bill. Bill looked up and noticed that the sky was getting dark. "I think we should stop" said Bill. "We will get wet if we keep playing".

Here, the sky's getting dark in the afternoon suggests that it is going to rain. If this is put into the data base it will be sufficient to bring in facts associated with "rain".

Also note that a topic concept need not be a single "key word". A fact may not become available to the system until a complex set of relations appears in the data base. A fact may be arbitrarily complex, and in particular may activate other facts depending on the presence or absence of certain relations in the story.

Looking forward, looking back. When a fact is made available we might not have all the information needed to make use of the fact. Since we are making inferences as we go, if the necessary information comes in after the rule has been asserted we want to make the inference when the information comes in. So we might have:

(21) Jack was outside. It was raining.
(22) It was raining. Jack was outside.

In (21) there is no problem. When we introduce "rain" we have sufficient information to use (19) and infer that Jack is going to get wet. But in (22) we only learn that Jack is outside after we mentioned rain. If we want to use (19) we will need some way to have our fact "look forward" in the story. To do this we will represent facts by (MICRO-PLANNER) antecedent theorems, so a fact will have two parts, a pattern and body (an arbitrary program). We will execute the body of the fact only when an assertion is in the data base which matches the pattern. In (19) the pattern would be "someone outside". Then in (22) when we introduce (19) no assertion matches the pattern. But the next line creates a matching assertion, so the fact will be executed. We will say that a fact is *"looking foward"* when its topic concept appears before the assertion which matches the pattern. When the assertion which matches the pattern comes first we will say that the fact is *"looking backward"* (as in (21)). (This is a slight extension over MICRO-PLANNER antecedent theorems which can only look forward.)

We can see how important looking forward is with a few examples.

(23) Janet was thinking of getting Jack a ball for his birthday. When she told Penny, Penny said, "Don't do that. Jack has a ball." Here we interpreted the line "Jack has a ball" as meaning that he did not want another. The common sense knowledge is the fact that in many cases having an X means that one will not want another X. This piece of information would probably be filed under "things to consider when about to get something for somebody else". Naturally it was an earlier line which mentioned that Janet was thinking of getting Jack a ball.

(24) Bill offered to trade his pocket knife for Jack's dog Tip. Jack said "I will ask Janet. Tip is her dog too." The last line is interpreted as the reason Jack will ask Janet because of information about the relation between trading and ownership.

(25) "Janet wanted to get some money. She found her piggybank and started to shake it. She didn't hear anything." The last line means that there was nothing in the piggy-bank on the basis of facts about piggy-banks.

In each of these cases it is an earlier line which contains the information which is used to assign the interpretation. So in (23) there is nothing inherent in the line "Jack has a ball" which means "don't get him another". If there were, something in the line would also have to trigger a check for the following situations:

(26) Bill and Dick wanted to play baseball. When Jack came by Bill said "There is Jack. He has a ball."

(27) Tom asked his father if he would buy him a ball. "Jack has a ball," said Tom.

(28) Bill's ball of string was stuck in the tree. He asked Jane how he could get it out. Jane said "You should hit it with something. Here comes Jack. He has a ball."

I formulated facts as antecedent theorems because I was so impressed with the need to "look forward". However, rather than call the facts antecedent theorems, I call them *"demons"* since it is a shorter and more mnemonic name.

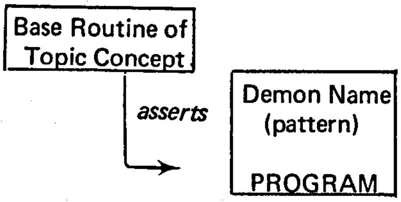
Specification and removal of demons. It should be emphasized that the model does not "learn" the information contained in the demons. This information is put in by the model maker. On the other hand, the demons are not specific to the story in the sense that they mention Jack, or "the red ball". Rather, they talk about "a person X" who at one point in the story could be Jack, at another, Bill. We will assume a mechanism for binding some of the variables of the demon ("specifying" the demon) at the time the demon is asserted.

We want demons to be active only while they are relevant to the story. A story may start by talking about getting a present for Jack, but ultimately revolve around the games played at his party. We will need some way to remove the "present getting" demons when they have outlived their usefulness. (An irrelevant but active demon not only wastes time and space, but can cause us to misinterpret a new line.) As a first approximation we will assume that a demon is declared irrelevant after a given number of lines have gone by.

Base routines. So far we have said that demons are asserted when the proper concept has been mentioned. But this implies that there is something attached to the concept telling us what demons should be put in.

If we look at a particular example, say (24), it is Bill's offer to trade which sets up the context for the rest of the fragment. I will assume that the information to do so is in the form of a program. Such routines, which are available to set up demons, will be called *"base routines"*.

The relation between base routines and demons can be represented as:



These base routines will be responsible for more than setting up demons. Suppose we are told that Jack had a ball, and Bill a top. Then Jack traded his ball to Bill for the top. One question we might ask is "Who now has the top?" Naturally since questions of "who has what" are important in understanding stories we will want to keep tabs on such information. In this particular case, it must again be the "trade" statement which tells us to switch possession of the objects. Every time a trade occurs we will want to exchange objects, so whenever we see "trade" we execute the "trade" base routine. Of course, the program can't be too simple-minded, since it must also handle "I will trade..." and perhaps even "Will you trade...?"

A good test as to whether a given fact should be part of a base routine or a demon is whether we need several lines to set it up or whether we can illustrate the fact by presenting a single line. (Naturally several lines could be made into one by putting "and's" between them, but this is dodging the point. I am only suggesting an intuitive test.) So we saw that "Jack has a ball" was not enough by itself to tell us that Jack does not want another ball. Hence this relation is embodied by a demon, not a base routine.

Bookkeeping and fact finders

Up to this point we have introduced two parts of the model, demons and base routines. In this section we will introduce the remaining two parts.

Updating and bookkeeping. Again let us consider the situation when Jack had a ball, Bill a top, and they traded. When we say that Bill now has the ball, it implies that Jack no longer does. That is to say, we must somehow remove the fact that Jack has the ball from the data base. Actually we don't want to remove it, since we may be asked "Who had the ball before Bill did?" Instead, we want to mark the assertion in some way to indicate that it has been updated. We will assume that there is a separate section, pretty much independent of the rest of the model, which is responsible for doing such updating. We will call this section "bookkeeping".

Fact finders. But even deciding that one statement updates another requires special knowledge. Suppose we have:

(29) Jack was in the house. Sometime later he was at the store.

If we ask "Is Jack in the house?" we want to answer "No, he is at the store." But how is bookkeeping going to figure this out? There is a simple rule which says that (<state> A B) updates (<state> A C) where C is not the same as B. So (AT JACK FARM) would update (AT JACK NEW-YORK). But in (29) we can't simply look for JACK AT <someplace which is not the store>, since he is IN the house. To make things even worse, we could have:

(30) Jack was in the house. Sometime later he was in the kitchen.

To solve this problem we will need:

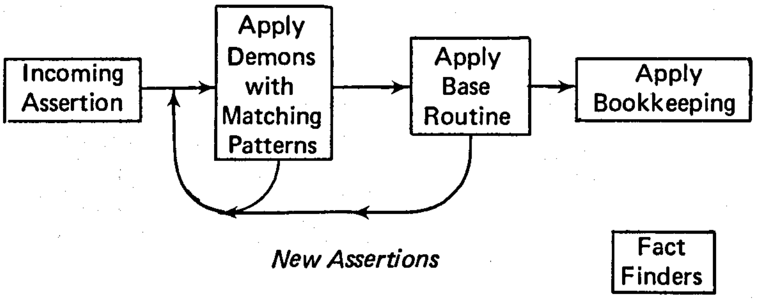
(31) To establish that PERSON is not at location LOC

```
Find out where PERSON is, call it X
If X = LOC, then theorem if false so return "NO".
If X is part of LOC then return "No".
If LOC is part of X, then try to find a different X
Else return "Yes".
```

In (29) the bookkeeping would try to prove that Jack is not at the store, and it would succeed by using (31) and the statement that Jack is in the house. Bookkeeper would then mark the earlier statement as updated. Theorems like (31) are called *"fact finders".* Fact finders are represented as MICRO-PLANNER consequent theorems.

The basic idea behind fact finders is that they are used to establish facts which are comparatively unimportant, so that we do not want to assert them and hence have them in the data base. So in (29) we do not want to assert "Jack is not in the house" as well as "Jack is at the store". In the same way we will have a fact finder which is able to derive "<person> knows <fact>" by asking such questions as "was the <person> there when <fact> was mentioned or took place?" Again, this information is easily derivable, and not all that important, so there would seem to be no reason to include it explicitly in the data base.

Then, we can represent our model as:



The five questions

To summarize this, let us return to our five questions and see what answers this model gives. In answering them however we will have to be careful, since in a peculiar way the model just presented (and the ones we will consider henceforth) answers all of them. What I mean is that since this model was made into a computer program, and since one has to specify everything in a program, the model must have some answer to each of the questions. Yet, very little was said in the preceding pages about, say, semantic representation, and I would deny that I really said anything about semantic representation at all.

The way out of this peculiar contradiction is to make a distinction between a computer program and the theory a computer program embodies. There has been much written on the relation between programs and theories, and the last thing I want to do is add to this literature, but some distinction of this sort is necessary. The distinction I will make is quite simple. If person P writes an AI program which performs task T, I will say that P's theory of T is those parts of the program which P considers significant. This definition has as a consequence that a single program could embody two different theories. This could happen, for example, if two people wrote a single program, but held different opinions about which portions of the program were significant.

So with this distinction in mind, what answers does my theory provide?

1. SEMANTIC REPRESENTATION. I have very little to say about semantic representation. Indeed the only statement which the preceding commits me to is that a semantic representation with *only* a small number of primitives is wrong. For example, in the preceding I used more specific concepts like "rain", "piggy-bank" and "present giving". However, to the best of my knowledge no one I know actually holds the contrary position so it is not clear that it is worth arguing the point. (Some people mistakenly believe that Schank holds this position, and Schank's writings often encourage this misapprehension, but from personal conversation I know that he does not.)
2. INFERENCE TRIGGERING. I am firmly committed to making non-problem occasioned inferences at read time, and for the reasons we went into above. This was the entire point of the demon apparatus.
3. ORGANIZATION. By and large the theory just presented is a theory of organization. In particular it states that given a particular assertion, the way we find those facts which we should use to make inferences from the assertion is to look in two places, first the base routine for assertions of that form, and second for any demons which happen to have been activated which are looking for assertions of that form. To put this slightly differently, the system presented states that the PLANNER mechanism of pattern directed invocation is the way facts are located.
4. INFERENCE MECHANISM. The theory assumes that the inference mechanism is MICRO-PLANNER demons (or the equivalent in some other language). A subsequent section will argue against this assumption.
5. CONTENT. As presented, the model said nothing about content in that it makes no claims about exactly what it is we know about piggy-banks, or anything else. One might use this model to make claims about what it is we know of these topics, but nothing in the presentation here does so.

Problems with the model

There are many places where one could find fault with the model just proposed, but I will pick one issue which seems to me important because it touches on the more general issue of the role of high level programming languages as vehicles for theories of knowledge and inference.

Consider a fact like:

(32) Umbrellas are used to keep rain off one's head.

To fit such a fact into the model just presented, we would most naturally treat it as follows:

(33) Base routine which activates demon: Possibility of rain.

Pattern:	Person gets umbrella.
Program:	If person might be caught in rain, he got the umbrella to prevent getting wet.

This will work quite well for stories like:

(34) It looked like rain. Jack got his umbrella.

It would even be possible, using mechanisms in the model which were not explained in the previous section to handle a story like:

(35) As Jack was leaving the house, he heard on the radio that it might rain. He went to the closet.

If asked why he did this we would respond that he was probably getting an umbrella. (It would be also possible to answer that he was getting his raincoat, but this is not important since "raincoat" would also have information connecting it to rain.) The extra mechanism which is needed here is the ability to put together the "expectation" of getting an umbrella, with our knowledge of where umbrellas are normally kept to conclude that he is going to get his umbrella in spite of the fact that the word "umbrella" was never mentioned in (35).

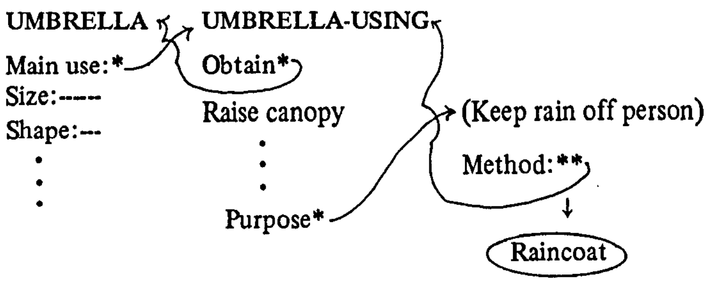
The trouble with this solution is that it would not account for the following story:

(36) Jack began to worry when he realized that everyone on the street was carrying an umbrella.

Question:	What was Jack worrying about?
Answer:	That it might rain, and he was without an umbrella.

While it is intuitively clear that a fact like (32) comes into play here, the formulation in (33) as vague as it is, is incapable of accounting for (36). The problem is that since (36) never mentioned rain, the demon expressed in (33) would never have been activated. To put this in terms of pointers, the fact in (33) only allows a pointer from "rain" to "umbrella", it does not allow a pointer from "umbrella" to "rain" and hence cannot be used to help us conclude in (36) that the problem is rain. Now it is not hard to come up with solutions to this problem. I can think of several alternatives, of which my favorite is something like:

(37)



What (37) says is that there is a heading under "umbrella" which indicates the use of umbrellas by pointing to a set of facts about umbrella usage. Umbrella usage naturally points right back at umbrella because to use an umbrella you must have one. It also notes that the purpose of using an umbrella is to keep rain off one, and keeping rain off one notes that one standard way to accomplish this end is to use an umbrella. Now with the possible exception of the facts listed under "umbrella-using" none of this looks very much like a program, or a demon. It is rather a complex set of pointers, which is just another way to say "data structure". Furthermore, I have implicitly argued elsewhere (Charniak, 1975) that even umbrella-using cannot be considered to be "program" but rather looks like "data".

Now all this argument is highly speculative. First, I am arguing that the proper representation for fact (32) looks more like data than program, and from that conclusion I wish to argue against PLANNER, and the "proceduralist" view in general. Both phases of the argument can be attacked. For one thing, it is conceivable that we could change demon (33) so that it could also be activated by "umbrella". This would not be easy since it would require that the program take into account under which circumstances it was activated, but it conceivably could be done. The response, of course, is the same one we gave concerning FOPC. When we have to subvert the natural inclinations of a system there is probably something wrong. (It is also true that there is no hard line between what might be considered program, and what data. One of the factors which helps determine whether we consider a particular structure as one or the other is to what degree the structure determines how it is to be used. By allowing programs to become usable in more than one way we are making them slightly less "programlike". One must watch for the possibility that in gradually modifying one's program to account for cases like (36) it becomes "data" without one's realizing it.)

Also note that by giving up the use of demons to represent individual facts like (32) we are implicitly giving up the mechanism of pattern directed invocation as our means of locating useful facts. That is, since we no longer represent (32) as a demon there is no longer the pattern-program distinction which makes the concept of pattern directed invocation meaningful.

But saying that (32) is better expressed in a data format like (37) than a demon format like (33) is not necessarily to say that knowledge should not be expressed in PLANNER. As we noted earlier, PLANNER is a programming language and hence has a certain amount of flexibility. To consider only one possibility, we could agree that (32) should not be expressed as a distinct demon, while arguing instead that it is better expressed as part of a program which, say, includes all of the information expressed in (37), but unlike (37) is written in PLANNER. My own opinion is simply to repeat what by now has become an old refrain, I am suspicious when one has to manoeuvre around the problems inherent in a formalism.

FRAMES

There are other problems with the demon based scheme (see "A better looking supermarket frame," below). My reaction to them has been a shift to a different formalism. In particular my most recent work has been based on an idea suggested by Minsky (Minsky, 1974), the "frame". It is perhaps indicative of the convergence of ideas reflected in (or perhaps inspired by) Minsky's paper that the overall organization proposed here (although not the details) is quite similar to the independently developed "scripts" of (Schank and Abelson, 1975).

I take a frame to be a static data structure about one stereotyped topic, such as shopping at the supermarket, taking a bath, or piggy banks. Each frame is primarily made up of many statements about the frame topic, called *"frame statements"* (henceforth abbreviated to FS). These statements are expressed in a suitable semantic representation, although I will simply express them in ordinary English in this paper.

The primary mechanism of understanding a line of a story is to see it as instantiating one or more FS's. So, for example, a particular FS in the shopping at the supermarket frame would be:

(38) SHOPPER obtain use of BASKET

(SHOPPER, BASKET, and, in general, any part of an FS written in all capitals is a variable. These variables must be restricted so that SHOPPER is probably human, and certainly animate, while BASKET should only be bound to baskets, as opposed to, say, pockets.) This FS would be instantiated by the second line of story (39).

(39) Jack was going to get some things at the supermarket.
The basket he took was the last one left.

Here we assume that part of the second line will be represented by the story statement (SS):

(40) Jack1 obtain use of basket1

(Of course, really both (38) and (40) would be represented in some more abstract internal representation.) Naturally, (40) would be an instantiation of (38), and this fact would be recorded with a special pointer from (40) to (38).

The supermarket frame will contain other FS's which refer to (38), such as:

(41) (38) usually occurs before (42)
(42) SHOPPER obtains PURCHASE-ITEMS

Any modification (like (41)) of a particular FS (like (38)) will be assumed true of all SS's which instantiate that FS (like (40)), unless there is evidence to the contrary. Hence, using (41) we could conclude that Jack has not yet finished his shopping in (39). Other modifications of (38) would tell us that Jack was probably already in the supermarket when he obtained the basket, and that he got the basket to use during shopping.

The variable SHOPPER in (38) also appears in (42), and in general a single variable will appear in many FS's. Hence the scope of these variables must be at least that of the frame in which they appear. When an SS instantiates an FS the variables in the FS will be bound. Naturally it is necessary to keep track of such bindings. For example, failure to do so would cause the system to fail to detect the oddness in (43) and (44).

(43) Jack went to the supermarket. He got a cart and started up and down the aisles. Bill took the goods to the checkout counter and left.
(44) Jack went to the supermarket to get a bag of potatoes. After paying for the milk he left.

It is probably a bad idea to actually change the frame to keep track of such bindings. Instead I assume that the frame remains pure, and that the variable bindings are recorded in a separate data structure called a *"frame image"* (abbreviated FI). For frames which describe some action, like our shopping at supermarket frame, we will create a separate FI for each instance of someone performing the action. So two different people shopping at the same time, or the same person shopping on two different occasions, would require two FI's to record those particulars which distinguish one instance of supermarket shopping from all others.

Much of this information will be stored in the variable bindings (shopper, purchase items, store, shopping cart used, etc.) However, the variable bindings do not exhaust the information we wish to store in the FI, for example it will probably prove necessary to have pointers from the FI to some, if not all, of the SS's which instantiate FS's of the frame in question. Of slightly more interest is that the FI of a frame describing an action will keep track of how far the activity has progressed. So, for example, we would find the following story odd:

(45) Jack drove to the supermarket. He got what he needed, and took it to his car. He then got a shopping cart.

We have already said that FS's are modified by time ordering statements, so to note the oddity of (45) it is only necessary to have one ore more *progress pointers* in the FI to the most time-wise advanced FS yet mentioned in the story. Then when new statements are found in the story which instantiate FS's in the frame, the program will automatically check to see if these FS's are consistent with the current progress pointer(s). If so, the FI progress pointer(s) may be advanced to indicate the new state of progress. If not, as in (45), the oddity should be noted, and, if possible, the story teller questioned about the oddness of the time sequence.

I have not commented so far about how, given a new SS, we locate an FS which it instantiates. In general this is a difficult problem, and I will have little to say about it. Roughly speaking the problem falls into two parts. First, the system must recognize that a given frame is relevant to a particular story. I am assuming that the presence of a key concept in the story will trigger a given frame. (It should be clear however that this is much too simple minded. For example, the scene setting description of a city block as containing a supermarket, bank, tailors, shoe repair shop, etc., should probably not activate the frames for the activities normally done in each.) Secondly, given that one or more frames have been selected as relevant to the story, how does the program find the particular FS which is instantiated by a particular SS? Here I will assume that a list of current frames is kept and frames which have not been used recently are thrown away. To find the particular FS which is instantiated by the SS I will simply assume that all FS's of the recently used frames are checked for a match. A more sophisticated procedure would be to first check FS's which follow the progress pointer. Another improvement would be to have an index for each frame so that it would not be necessary to check all FS's of the frame. This would, in effect, make each frame into a local data base. If a frame has a sub-frame it too will be checked, although it will probably be necessary to put some limit on how deep one should look into sub-frames.

One final note before moving on to more detailed issues. This paper is concerned primarily with the use of frames in the comprehension of simple narration. However, it seems only reasonable to me to assume that whatever knowledge we have built up into frames was done, in large part, in order to get around in the world, rather than to read stories. I will assume then that the same knowledge structures should be usable for either task, and upon occasion I will make arguments that structuring a frame in a particular way will make it easier to perform actions based on the frame structured knowledge. It seems to me one of the great advantages of frames is that they seem capable of being used in multiple ways, something which is not obviously true, for example, of demons.

FRAMES AND SUB-FRAMES

In this and the next two sections we will take a closer look at the internal structure of a frame. In particular I will try to show that Minsky's notion that different frames will have "terminals" in common is applicable to the kind of frames I have in mind. Minsky's idea was that frames applied to problems of vision would store information about what one was likely to see in a certain situation (e.g., a room) and from a certain vantage point (e.g., just having walked in the door). Upon changing vantage points one moved to different frames, but many of the "terminals" of the frame (e.g., right wall, center wall, lamp, etc.) would appear in both views, and hence in both frames. While Minsky used the term "terminal" when discussing scenario frames (his "terminals" very roughly correspond to my "frame statement") he never applied the idea of common terminals between frames to scenario frames. Nor is it clear that frames such as the ones discussed in this paper have anything directly corresponding to the sharing of a wall terminal between two room frames. However, I will try to show that Minsky's notion is useful in a somewhat different way.

Let us start by giving a naive outline of some FS's about supermarkets.

```
(46) a)  Goal (SHOPPER own ITEMS)
     b)  SHOPPER be at SUPERMARKET
     c)  SHOPPER have use of BASKET
     d)  do for all ITEM ϵ ITEMS
     e)        SHOPPER at ITEM
     f)        BASKET at ITEM
     g)        ITEM in BASKET
     h)  end
     i)  SHOPPER at CHECKOUT COUNTER
     j)  BASKET at CHECKOUT COUNTER
     k)  SHOPPER pay for ITEMS
     l)  SHOPPER leave SUPERMARKET
```

I am assuming in (46) an implicit time ordering from top to bottom. In the actual frame this time ordering would be made explicitly. The reader might also notice that most of the FS's are states which must be achieved at some point in the course of the action. For reasons why I use states rather than actions to express what happens in an action sequence, see (Charniak, 1975).

There are countless things missing or wrong with (46), but I wish to concentrate on only one of them, the relation between shopping and using a shopping cart. It should be obvious that one can do shopping without using a cart, although (46) would make it seem that cart usage is an indispensable part of shopping. I will suggest that a good way to gain this flexibility in the supermarket frame is to have a separate cart frame which shares information with the supermarket frame, by having, in effect, some FS's common between the two.

One way to account for the ability to shop with or without a cart (and to understand stories about same) would be to have a second frame for shopping without a cart. However, not only is the idea unsatisfying, since it would require the duplication of the many facts the two activities have in common, but it would also lead to problems in the comprehension of certain types of stories. For example, one could easily imagine a story which starts out with Jack using a cart, the wheel of the cart sticking, and rather than going to get a second cart Jack finishes his shopping without a cart. It seems *a priori* that such combinations would be extraordinarily hard to account for with two completely separate frames for the two forms of supermarket shopping. (Alternatively, it is common practice in crowded situations to park one's cart in a general vicinity of several items and pick up the individual items without the cart, only to bring them all to the cart at a later point and resume shopping with the cart.)

One possibility for a single frame which handles both kinds of supermarket shopping is:

```
(47) a)   Goal: SHOPPER owns PURCHASE-ITEMS
     b)   SHOPPER decide if to use a basket. If so, set CART to T
     c)   If CART then SHOPPER obtain BASKET
     d)   SHOPPER obtain PURCHASE-ITEMS
          |    method
     e)   └──→ Do for all ITEM ε PURCHASE-ITEMS
     f)             SHOPPER decide on next ITEM
     g)             SHOPPER at ITEM
     h)             If CART then BASKET also at ITEM
     i)             SHOPPER hold ITEM
     j)             If CART then ITEM in BASKET
     k)        End
     l)   SHOPPER at CHECK-OUT COUNTER
     m)   if CART then BASKET at CHECK-OUT COUNTER
     n)   SHOPPER pay for PURCHASE-ITEMS
     o)   SHOPPER leave SUPERMARKET
```

Now one problem with (47) is that to my eye the constant repetitions of "If CART then . . ." give it a rather *ad hoc* appearance. And while it was this ugliness which gave me the initial impetus to find a better representation, there are more concrete problems with (47). One major one is the lack of any information about why these various actions are to be performed and why in this particular order. This lack comes out most strongly when we consider stories where something goes slightly wrong in the course of the shopping. For example:

(48) Jack was shopping at the supermarket. After getting a few things he returned to his cart only to find that some prankster had taken everything out of it and put the things on the floor. After putting the things in the cart, Jack finished his shopping, but was not able to find out who had done it.

Question: Why did Jack put the groceries which were on the floor into his cart?

The question here is so simple that one might be at a loss to know what sort of answer is desired, but one certainly knows that the items were put back in the cart so Jack could continue his shopping. The point here is that frame (47) does not give any explanation for Jack's action in (48). Naturally we would not expect any supermarket frame to explicitly take into account strange situations like (48), but this is not necessary to be able to answer the question in (48). All that is needed is an understanding that the purpose of a cart is to transport goods from place to place in the supermarket and that to do this the goods must be in the cart. Hence, in this situation, if, as is most likely, Jack still wants to transport the goods elsewhere he should once again put the items in the cart. But (47) does not give this information. It tells us to put the items in the cart, but not why or for how long.

I should point out that if (48) seems like an extraordinarily odd story on which to base any conclusions, there are much more normal ones which make the same point. For example:

(49) Jack was shopping at the supermarket. After getting a few items the wheel of his cart stuck. He got a second cart and finished his shopping.

Question: Why did Jack transfer his groceries to a second cart?

The story does not say that Jack transferred his groceries, and to infer that he did require essentially the same reasoning process required to understand why Jack put the groceries back in the cart in (48).

To handle stories like (48) and (49) we must therefore put two pieces of information into (47) which are not there at present. First, that in those cases like lines (47 h) and (47 m) where we have the basket going along with the person, the reason is to keep the previously collected items with one. Second, that to carry something with a cart requires that it be in the cart. We will indicate the first of these by:

```
(50) SHOPPER at ITEM
       side condition - DONE at ITEM also
       | method - suggested
       └→ cart-carry (SHOPPER, BASKET, DONE, ITEM)
```

Here DONE is a list of those items already collected. Cart-carry is a frame (and hence a sub-frame of the supermarket frame) describing the use of a cart for carrying things. (I am introducing a bit of terminology (method-suggested) from (Charniak, 1975); I will assume that it is reasonably self-explanatory. Consult (Charniak, 1975) for some explanation and justification.) The FS's in (50) replace lines (47 g) and (47 h) and differ from them in two respects. First (50) formulates the goal in a manner neutral with respect to using a basket or not, with only a suggestion that a basket be used. This is obviously necessary if we are to handle stories where the person does not use a basket. Secondly, it assumes the existence of a separate cart-carry frame in which we store information about using carts to carry things. We will see other advantages of this move later, but at the moment we can at least note that if one were to ask "Why does

Jack use a basket?" two answers (at least) would be possible–"to do shopping", or "to carry his groceries". (50) allows for both of these answers, wheras (47 g) and (47 h) only allow for the former, since there is no separate "carry" level.

The second piece of information we needed to handle stories (48) and (49) was that to carry something with a basket it is necessary that the thing be in the basket. The most natural place to put such information would be in our newly created cart-carry frame where it would be some sort of a pre-requisite (or more precisely a "strict" (as opposed, for example, to "suggested") substrate to use the terminology of (Charniak, 1975). By creating the cart-carry frame and locating information about the action within it, we also circumvent the need to duplicate this information elsewhere. For example, an expanded supermarket frame would include the fact that in some circumstances it is permitted (and suggested) that one uses one's basket to take the groceries to one's car. By stating this as suggesting cart-carry (SHOPPER, BASKET, PURCHASE-ITEMS, CAR) we no longer need an instruction to put the groceries into the basket again before setting out.

SHARING FRAME-STATEMENTS BETWEEN FRAMES

So far, then, I have argued that frames must be able to reference sub-frames, and in particular, the supermarket frame needs some sub-frame like cart-carry. There is nothing exceedingly strange in this, but the next step will perhaps be a bit more interesting. Here I will suggest that some of the frame statements in our supermarket frame be shared with the cart-carry frame. To see the reasons for this, let us start by noting that the failure to allow for common FS's will lead to some curious redundancies in our frame. One of these occurs in the DO loop of (47) which handles the collection of the PURCHASE-ITEMS. With out latest changes, this portion of (47) (lines (e) through (k)) looks like:

```
(51) a) Do for all ITEM ∊ PURCHASE-ITEMS
     b)    SHOPPER choose next ITEM ∊ PURCHASE-ITEMS - DONE
     c)    SHOPPER at ITEM
     d)       side-condition DONE at ITEM also
     e)       | method - suggested
     f)       └→cart-carry (SHOPPER, BASKET, DONE, ITEM)
     g)    SHOPPER hold ITEM
     h)    If CART then ITEM in BASKET
     i)    DONE ← DONE + ITEM
     j) End
```

The redundancy is this: we have already stated that cart-carry has a strict sub-state that the things to be carried must be in the basket. But in (51) we further specify in line (h) that ITEM is to be in BASKET, which is simply a special case.

Once again my initial reason for being concerned with this is that I find such redundancy unappealing, but again there seem to be more solid reasons for doing away with it. In particular consider the following story:

(52) Jack was doing some shopping at the supermarket using a shopping cart. The last thing he got was a package of gum, which he picked up right at the check-out counter. Jack put the gum and everything else down in the counter.

Question: Why didn't Jack put the gum in his basket?

Again the question seems silly, but again we all know that there would be no reason to put the gum into the cart simply because Jack already has everything at the check-out counter. But (51) as stated does not allow us to make this inference. It simply says to put each ITEM into the BASKET and since no reasons are given there is no way to see that in the particular case of the gum in (52) there is no reason to put the gum into the basket.

My solution to this problem is to see line (51 h) as shared between the supermarket frame and the cart-carry frame. Looked at in this light, the reason for obeying (51 h) is then the same as the reason for having items in the basket as stated in the cart-carry frame—if you wish to use the cart to carry an item it must be in the cart. Since Jack has no reason for carrying the gum in the cart he has no reason to put it in the cart. (To be a bit more precise, Jack might have a reason to put the gum in the cart, namely using the cart to carry the groceries to his car afterwards, but this only occurs after putting the groceries on the check-out counter, hence does not count as a reason for doing it at the particular point in time we are discussing.)

Now when I suggest that the two frames share an FS, I do not mean that the one FS physically appears in both frames, although this would be perfectly possible in a list processing language like LISP. There are, however, several good reasons for not implementing FS sharing by physical identity. For one thing it would mean that different frames would have to have the same variables, which at the very least would create a major debugging problem. From a theoretical point of view it seems likely that such an attempt will run into trouble because two or more statements in one frame will share the same FS in a second frame. If the two FS's in frame one have different variables there would be no way for the FS in frame two to be identical to both, and hence could not be physically the same. (However, I do not have a clear-cut example of this happening.)

So I will not assume that FS (51 h) is physically identical to the corresponding FS in cart-carry, but rather that there is a pointer from (51 h) to the FS in cart-carry which says that (51 h) should be considered to be the same FS. That is, we would have an arrangement somewhat like:

(53) Frame for supermarket   Frame for cart-carry

$$\begin{tikzcd}
\vdots & \\
\text{ITEM in BASKET *} \arrow[r, "\text{identity}", "\text{pointer}"'] & \text{CARRIED in CARRIER} \\
\vdots & \vdots
\end{tikzcd}$$

Here I have created two new variables for the cart-carry frame, CARRIED which specifies what is carried, and CARRIER which is the cart used to carry. Naturally, to actually use the cart-carry information for understanding the ITEM and BASKET line of the supermarket frame it will be necessary to see that ITEM corresponds to CARRIED, etc. (It may also be necessary to see that SHOPPER in supermarket corresponds to the variable for the actor in cart-carry, and this would require more formalism, but I am not sure it is necessary.) Finally, note that there seems to be no reason to have a corresponding pointer from the FS in cart-carry to ITEM in BASKET, since there is no need to know about supermarkets in order to use a basket.

Now if all of this seems eminently reasonable to you, feel free to skip the next section. But for those of you to whom this seems a strange sort of data structure, what follows is an attempt to justify it by considering one alternative and showing how the shared FS proposal is superior.

Beyond the "ugliness" of the redundancy, the only argument we gave for replacing (51 h) with (53) was the story (52) where Jack did not put the gum into the shopping cart. One way to solve both the redundancy, and the problem spotlighted by (52), would be to simply remove (51 h) from the supermarket frame. This clearly solves the redundancy problem, and it also solves (52) since now the only FS to the effect that the goods should be in the basket appears in cart-carry, and since Jack has no reason for carrying the gum in the cart he has no reason to put the gum into the cart.

The argument against this possibility must start from the recognition that it has some counter-intuitive properties. Intuitively one sees putting ITEM into BASKET as the last state of collecting ITEM. With this new solution this is no longer the case. Instead, putting ITEM into BASKET is a result of wanting to move to the next ITEM. (To distinguish we will call the item one has just obtained ITEM_n and the item one is going to obtain next ITEM_{n+1}.) Hence, putting ITEM_n into the basket is not the last thing of the N'th cycle, but one of the first of the N+1'st. This is against my intuition and makes me immediately suspect it.

Furthermore, this counter intuitiveness seems to have more substantive implications. For example:

(54) Jack was shopping at the supermarket. After getting a basket he went to the milk counter and picked up a carton of milk. He then thought about what to get next.

Question: Did Jack put the milk in the basket?
Answer: I would assume so.

The shared FS model would allow for this answer since deciding on ITEM_{n+1} occurs after putting ITEM_n into the basket. But by deleting (51 h) we remove this information so there would be no way to answer the question other than "I don't know". (Of course, "I don't know" is also an acceptable answer, but our model must allow for the various alternative answers people can give.)

A second argument against deleting (51 h) comes from using our supermarket frame in actually doing shopping. By deleting (51 h) we would be saying in effect that one puts $ITEM_n$ into the basket when checking to see that all of DONE is in the basket when going to get $ITEM_{n+1}$. Computationally it seems horribly inefficient to bother to check on all of DONE each time (and in fact I am sure that people do not do it).

Finally by deleting (51 h) we make it difficult to account for "mistakes" that people make. For example, suppose we had the variation of (52) where Jack does put the gum into his cart only to immediately take it out again. We then ask the question:

(55) Question: Why did Jack put the gum into the cart, since he only had to immediately take it out again?

Answer: Well, I suppose one normally puts things into the cart immediately after picking them up.

Such considerations lead me to reject the alternative of deleting (51 h).

A BETTER LOOKING SUPERMARKET FRAME

So far I have argued for the shared FS model primarily on the basis of its ability to handle stories where mistakes occurred, but it is also the case that it allows us to solve one of the "aesthetic" problems mentioned earlier, namely the constant repetition of "IF CART then . . ." in (47). What we find is that all occurrences of this phrase in (47) can be replaced by either an explicit call to cart-carry (as in (50)) or an identity pointer to an FS in cart-carry, as in (53). To give another example of this, consider line (47 c), repeated here:

(47) c) If CART then SHOPPER obtain BASKET

This is clearly another example of a pre-requisite of cart-carry, and hence should be considered shared with cart-carry in the same way that ITEM in BASKET is shared. Furthermore, we can now remove the "If CART then" portion of (47 c) by assuming the eminently reasonable convention that a shared node in frame-1 which has a pointer to frame-2 is only applicable to the action in frame-1 if frame-2 is activated in the sense that we have created a frame image for frame-2. When performing the action in real life this means that upon deciding to use a cart, one sets up a cart-carry image and this in turn makes the various FS's in the supermarket frame dealing with carts relevant to one's activities. While reading a story the general rule will be that any SS instantiating an FS which is shared with cart-carry will be sufficient to create an FI for cart-carry. With this convention (47 c) becomes:

(56) SHOPPER obtain BASKET $* \xrightarrow[\text{pointer to cart-carry}]{\text{identity}}$

Furthermore, since I have argued that these pointers are needed on independent grounds, we have received this simplification for free.

With both this simplification, and the use of cart-carry, our supermarket frame now looks like:

```
(57)  a)   Goal:  SHOPPER owns PURCHASE-ITEMS
      b)   SHOPPER  decide if to use basket, if so set up cart-carry FI
      c)   SHOPPER obtain BASKET *cart-carry
      d)   SHOPPER obtain PURCHASE-ITEMS
      e)   | method - suggested
      f)   └→ Do for all ITEM ∈ PURCHASE-ITEMS
      g)         SHOPPER choose ITEM ∈ PURCHASE-ITEMS - DONE
      h)         SHOPPER at ITEM
      i)            side-condition DONE at ITEM also
      j)               | method - suggested
      k)               └→cart-carry (SHOPPER, BASKET, DONE, ITEM)
      l)         SHOPPER hold ITEM
      m)         ITEM in BASKET *cart-carry
      n)         DONE ←  DONE + ITEM
      o)      End
      p)   SHOPPER at CHECK-OUT-COUNTER
      q)      side-condition PURCHASE-ITEMS at CHECK-OUT-COUNTER also
      r)         | method - suggested
      s)         └→cart-carry (SHOPPER, BASKET, PURCHASE-ITEMS, CHECK-
                                                        OUT-COUNTER)
      t)   SHOPPER pay for PURCHASE-ITEMS
      u)   SHOPPER leave SUPERMARKET
```

I have here adopted the convention of indicating an identity pointer to another frame by a "*" followed by the name of the second frame.

In spite of the "simplification" introduced, (57) is considerably longer than (47). But on the other hand, (57) shows much more of the structure of shopping at supermarkets than does (47). To point out only one way where (57) is superior, it, but not (47), states that it is necessary to get one's groceries to the checkout counter, whether or not one uses a cart. Of course, (57) still does not contain more than a fraction of our knowledge of supermarkets, and in fact many of its particulars are clearly wrong, but it's a start.

INFERENCE ON FRAME BASED KNOWLEDGE

So far I have been discussing the organization of knowledge and have suggested that a large portion of it is stored in frames which connect up to other frames by either sub-frame relations or identity pointers. But a quick look back will reveal that at the same time several issues of inference have crept in. For example near the very beginning we stated that if an SS instantiates an FS then any modification of the FS within the frame is true of the SS also unless there is explicit information to the contrary. This, you will remember, allowed the system to conclude that when Jack got the shopping cart he was most likely already at the supermarket, but had yet to begin the actual act of collecting the groceries. Or again, when I mentioned that an FI had one or more progress pointers, I implicitly assumed that one could infer what actions had already taken place using the time sequence information in the frame plus the progress pointer in the FI.

This is, of course, as it should be. One cannot, or at least should not, discuss structure independently of use and the use of frames is to allow us to make inferences about the stories we read. Nevertheless, there remain many issues of inference left untouched by the previous discussion and I will cover one or two of them here.

One problem is the inference triggering question of which inferences should be made at read time. It is my impression that the frame-like system just outlined has several nice properties in this regard.

For one thing, many inferences which would have to be made in a demon system need not be made in the system we have just outlined. To take again the example of Jack getting a shopping cart, I pointed out that the supermarket frame allowed the system to make several inferences about the statement, like why he did it, and that he had yet to start the shopping, but I left it vague as to whether these inferences should actually be made at read time, or only if a question was asked. In fact, there seems to be little reason for actually making most of these inferences at read time. Since the SS will have a pointer to the FS it instantiates, we may assume that a standard tactic for answering questions about a particular SS, like why the action was performed, or where, or when, would be to look in the frame for the answer.* To put this slightly differently, when an SS instantiates an FS it is not necessary to put into the data base instantiations of all the modifications of the FS.

Looked at in this light, it is interesting to ask under what circumstances one *would* want to instantiate an FS and put it in the data base. This is, after all, a large class of potential inferences and some restrictions on them would be at least a start on the problem of which inferences need be made at read time. The best answer I currently have to this question is summed up in the following rule:

(58) The Dual Usage Rule: If X is an FS in an active frame (one which has an FI) then X will only appear instantiated in the data base if it has two purposes.

Some typical purposes are:

a) appearing in an active frame
b) appearing in the semantic representation of the text
c) updating older statements in the data base

(this list will surely be expanded)

*I might point out in passing that the use of the modifying information in such a manner is a major reason why frames as I describe them look like "data" rather than "program". Traditionally programs have the property that they are not meaningful "locally" whereas we want to be able to use a modifying FS to answer a question about an SS without going through the entire frame. Hence my description of frames earlier as "a static data structure".

So the typical example of an FS which appears instantiated in the data base is one which appeared in the semantic representation of the text and was found to instantiate some FS. A more interesting case is exhibited by:

(59) Jack was going to Bill's birthday party. He thought Bill would like a kite. Jack then went to the store.

In (59) we should expect Jack to be buying a kite at the store. But consider the statement

(60) Jack1 own kite1

On one hand we can only predict that this is likely to occur using information from the birthday frame (perhaps in conjunction with the give sub-frame). On the other hand, this statement serves as the goal statement in the store frame. To enable (60) to act as the link between frames in this fashion it must appear explicitly in the data base with frame pointers to two different frame statements in two different frames. That is to say, (60) appears in two active frames, and hence is a justified instantiation.

It seems to me that (58) is a fairly strong rule, and it will be interesting to see if it can be maintained. Of course, given that the list of uses can be expanded, it is not completely clear what would serve as a counter example to the rule, but it seems to me that the general intent should be clear. For example, (58) would prohibit many of the inferences made by Rieger's system (Rieger, 1974). To give only one example, given statement (61) Rieger's system would infer statements like (62)-(65) none of which would qualify by the standard set up by (58).

(61) John told Mary that Bill wants a book
(62) John believes that Bill wants a book
(63) Mary now knows that Bill wants a book (assuming that the representation of "tell" is something like "cause to know by word of mouth")
(64) Bill might get himself a book
(65) John may want Mary to give Bill a book

(This is one of Rieger's own examples.)

COMPARISON WITH THE DEMON APPROACH

To get some perspective on frames as presented here let me compare them to the demon based model.

We already saw how demons (but not frames) had problems with our fact about umbrella using. To make further comparisons it is necessary to note the correspondence between frame statements and demons. This analogy works because frame statements accomplish precisely what demons were designed to accomplish—assign significance to a line due to the context it is in. So we notice how lines like (66)-(69) in the context of supermarket will instantiate FS's in the supermarket frame and hence (66)-(69) will be given more significance than they have out of context.

(66) Jack got a cart.
(67) Jack picked up a carton of milk.
(68) Jack walked further down the aisle.
(69) Jack walked to the front of the store. He put the groceries on the counter.

What is interesting in this comparison is that one demon usually has a minimum of three or four statements, whereas obviously a single FS is only one statement. FS's seem then to have a considerable conciseness to them, at least when compared to demons. The reason for this is not hard to see. Demons, being independent facts, must bind their own variables, and much of the size of a demon is due to checks to make sure that the variable bindings are correct (e.g., BASKET must be a basket, and not a carton of milk). These same things must be checked in a frame, but since the scope of the variables is the entire frame, rather than a single FS, the overhead, so to speak, is shared. Furthermore, the inferences about a given FS are stored implicitly in the structure of the frame, whereas they had to be stated explicitly in the demon. So a second advantage of the frames approach over demons is the conceptual economy one obtains in the expression of facts.

The analogy between FS's and demons also points to a third way in which the frames approach seems superior. One problem which bothers many people (including myself) about the demon approach is that it seemingly calls for large numbers of demons to be activated every time a given topic is mentioned in the story, although it is unlikely that more than a small fraction of the demons will ever be used. There are two possible reasons why people feel this is a problem. One is that so many active demons might make it hard to locate those demons which really *should* apply. Using frames does not help with this problem since there will be equal numbers of FS's.

To see the second reason why activating large numbers of demons is problematic, note that if it took no time at all to set up a demon, setting up many of them would seem less bad. But of course it does take time to set up a demon, and it becomes a problem to justify this computation in light of the unlikeliness of the demon every being used. Frames do offer a potential solution to this second problem because with frames, rather than "supermarket" activating many demons, we need only create a frame image for one frame (i.e., supermarket). This would take much less time, and hence would be better, but it should be noted that we pay a price. In particular most of the work involved in setting up a demon is to index our storage of active demons so that retrieving the ones needed will be reasonably easy. By comparison looking through frames to find matching FS's promises to be a time consuming task unless we do something similar. This is what I meant earlier when I said that perhaps each frame would have its own index to its contents. On the other hand, the approach presented here allows one to trade more time for locating an FS in return for less time to set up a new topic (frame), and the spectre of all those never-to-be-used demons makes me inclined to accept this trade.

Finally, the frames presented here have no problem handling time relations between FS's as we saw earlier in the paper. The same cannot be said of demons. We saw earlier how we might use a progress pointer to allow the program to notice actions which were out of sequence. What could be the equivalent in the demon model of the progress pointer? For one thing, where would such a pointer be stored? Short of giving every demon a pointer to the progress pointer, an inelegant solution at best, it is not clear what one could do. Furthermore, where would the time ordering information be stored? Notice that time ordering information is much more complex than a simple string, or even lattice which indicates the time orderings of actions. For example some time orderings are "strict" in the sense that one cannot possibly do things any other way, while others are "suggested" in the sense that it is a good idea to do the actions in a given order, but possible to do them some other way, while yet others are "regulatory" in the sense that it is possible, but illegal to do the actions in the opposite order (Charniak, 1975a). In the frames model one can store time ordering statements in the frame along with the rest. It is by no means obvious what to do in the demon model. This is not to say that one could not do it, but rather that having done it one would be left with something of little resemblance to the original demon model, and even less aesthetic appeal.

REFERENCES

Charniak, E. (1972) *Toward a model of children's story comprehension.* AI-TR266, MIT Artificial Intelligence Laboratory.

Charniak, E. (1975) *A partial taxonomy of knowledge about actions.* Paper submitted to IJCAI4.

Hewitt, C. (1969) *Planner: A language for proving theorems in robots.* (Eds. D. Walker and L. Norton) *Proc. IJCAI.*

Minsky, M. (1974) *A framework for representing knowledge.* AI Memo 306, MIT Artificial Intelligence Laboratory.

Raphael, B. (1968) Sir: A computer program for semantic information retrieval. (Ed. M. Minsky) *Semantic information processing.* Cambridge Massachusetts: M.I.T. Press.

Rieger, C.J. (1974) *Conceptual memory.* Unpublished Ph.D. Thesis, Stanford University.

Schank, R.C. (1973) *The fourteen primitive actions and their inferences.* AI Memo 183, Stanford Artificial Intelligence Laboratory.

Schank, R.C. and Abelson, R.P. (1975) *Scripts, plans, and knowledge.* Paper submitted to IJCAI4.

Sussman, G., Winograd, T., and Charniak, E. (1971) *Micro-planner reference manual.* AI Memo 203A, MIT Artificial Intelligence Laboratory.

Wilks, Y.A. (1975) *Notes for course "parsing english".* Lecture notes for the Tutorial on Computational Semantics, Institute for Semantic and Cognitive Studies.