CLUSTER: An Approach to Contextual Language Understanding

CLUSTER: An Approach to Contextual Language Understanding

Yigal Arens

ABSTRACT

Understanding natural language in context requires the existence of a model of the surrounding world. Such a model must include representations for the objects and activities present in the language being processed, in the surrounding physical environment, and in relevant past experiences of the understander.

The CLUSTER theory of contextual language understanding addresses this issue. CLUSTER has two main components: a theory of modeling the context of an ongoing conversation, and a theory of language analysis.

The first component of CLUSTER theory concerns the model of the world which an understanding system has: the objects and processes in the world that are to be represented, and the relation of these to the understanding system itself. The model emphasizes those elements that are more significant in the current situation, while ignoring information that is irrelevant. This thesis characterizes a mechanism for maintaining a model of the world, the *Context Modeler,* and the model of the world maintained by it, called the *Context Model.* The Context Modeler constructs the Context Model during the course of interacting with another language user.

The second component of CLUSTER theory concerns language analysis, the production of a representation of the meaning of a given sentence. Natural language analysis provides the basis for constructing the Context Model. The Context Model, in turn, is a major resource used by the language analyzer. This

thesis shows the centrality of the role played by a model of the context to a system's ability to understand natural language input.

In addition to characterizing these mechanisms, this thesis also describes particular implementations of the components of CLUSTER. The implementation of CLUSTER's language analysis component is named PHRAN, for PHRasal ANalyzer. The implementation of CLUSTER's context modeling component is referred to as The Context Modeler. Both components are combined in the *UNIX Consultant (UC)*, a natural language help facility that allows new users of the UNIX operating system to learn about UNIX. Users do so by holding a natural language dialogue with UC.

Acknowledgements

Over the *long* period of time spent on the work described in this thesis many, many, people have had an opportunity to contribute to it. So many, in fact, that I cannot possibly mention them all here. Most of these contributions were so different in kind as to be mutually incomparable. I tried, but found it impossible to arrange them in order of their significance. Therefore, no inferences as to the relative importance of people's contributions should be drawn from the order in which they are listed below.

Thanks go first to the members of my thesis committee: to Leo Harrington who, at untold risk to himself, agreed to serve as chair; to the supportive Lotfi Zadeh, the "outside" member of the committee; to Jack Silver, who at the very last moment was asked to make good on a years-old promise and join in; and to Robert Wilensky, the only *real* supervisor of the work reported here.

The students at the Berkeley AI Research group deserve recognition and warm thanks for their comments, advice, support, and company. It is in great part due to them that the time I spent at Berkeley was so enjoyable. At one time or another during my tenure there this group included Margaret Butler, Dave Chin, Charley Cox, Mike Deering, Joe Faletti, Tom Hahn, Paul Jacobs, Marc Luria, Jim Martin, Jim Mayfield, Mike Morgan, Lisa Rau, and Steve Upstill.

For material support provided at different stages of my Ph.D. studies, I am indebted to the UCB Math Department, to my adviser Robert Wilensky, and to the Computer Science Department at the University of Southern California.

And, finally, special thanks and love to Bosmi and Yael. They had to live with holiday after holiday, summer-vacation after summer-vacation, of finding some place to go to so I could stay alone at home and work. I hope I can make it up to them.

This research was sponsored in part by the Office of Naval Research under contract N00014-80-C-0732, and the Defense Advanced Research Projects Agency (DOD), ARPA order No. 3041, Monitored by the Naval Electronic Systems Command under contract N00039-82-C-0235.

To Yael and Bosmi

Contents

CHAPTER I

OVERVIEW AND BASIC IDEAS

I.1. A World Model

A great deal of effort in Artificial Intelligence has gone into research on the issue of knowledge representation. Possibly as a result of the difficulty encountered in this area, much less work has been done on the question of what structures exist which serve to organize and relate the details of a system's knowledge about the world. This thesis is in part an attempt to deal with this issue. Given a scheme for representing every detail of the world around us, how is this information arranged so that all relevant knowledge is applied in the appropriate circumstances? How are the "appropriate circumstances" determined?

In an attempt to answer these questions this thesis proposes a theory of language understanding in context. This theory is called **CLUSTER***. The CLUSTER theory has two main components: a theory of modeling the context of an ongoing conversation, and a theory of language analysis.

---

* Contextual Language Understanding with Salience Tracking and Environment Recall.

The first component of CLUSTER theory concerns the model of the surrounding world that an understanding system must have: the objects and processes in the world that must be represented, and the relation of these to the understanding system itself. Since this model of the world is continually changing, the model should emphasize those elements that are more significant in the current situation, while ignoring information that is irrelevant.

This thesis characterizes a mechanism for maintaining a model of the world. This mechanism is called a **Context Modeler** and the model of the world maintained by it is called a **Context Model.** The Context Model is a collection of representations of objects, events, and facts considered salient in the world at a particular moment. The Context Modeler constructs the Context Model during the course of interacting with another language user. In doing so the Context Modeler interacts with other processing components, such as a natural language analyzer and a planner. The Context Modeler both contributes to such components and makes use of them.

The second component of the CLUSTER theory concerns language analysis. A language analyzer is a program that, given a natural language sentence, produces a representation of its meaning. Natural language analysis provides the basis for determining the context. The context, in turn, is a major resource used by the language analyzer.

This thesis describes how a natural language analyzer can utilize information present in the Context Model. By doing so, it shows the centrality of the role played by a model of the context to the system's ability to understand new

natural language input. All new information extracted by the language analyzer from the input text will be interpreted relative to the Context Model, which embodies what the system believes exists around it.

In addition to characterizing these mechanisms, this thesis also describes particular implementations of the components of CLUSTER. The implementation of CLUSTER's language analysis component is named PHRAN, for PHRasal ANalyzer. The implementation of CLUSTER's context modeling component is referred to simply as The Context Modeler.

I.2. The UNIX Consultant and PHRAN

An implementation of CLUSTER theory is part of the **UNIX* Consultant (UC)**, currently being developed at the University of California at Berkeley.** The UNIX Consultant is a natural language help facility. It enables a new user of the UNIX operating system to learn about UNIX by interrogating it. This is done by the user holding a natural language dialogue with the UC program.

Following are several examples of exchanges between a user and the UNIX Consultant.

1. User: How do I delete a file?
   UC: Typing 'rm filename' will remove the file with name filename from your current directory.

* UNIX is a registered trademark of Bell Laboratories.

** The UC project is headed by Prof. Robert Wilensky. Many other people at Berkeley have contributed to it, among them Margaret Butler, Dave Chin, Charley Cox, Joe Faletti, Paul Jacobs, Jim Martin, Jim Mayfield, and Lisa Rau. See Wilensky, Arens, and Chin (1984) for a review of the project. A version of UC substantially different from the one described here is now under development at Berkeley.

2. User: How should one edit a file?
   UC: Typing 'ex filename' will get you into the ex editor; typing 'emacs filename' will get you into the emacs editor.

3. User: What is the best way to create a file?
   UC: The best way to create a new file is to use the editor, e. g. – 'ex filename'.

4. User: Do you know how to read a file?
   UC: Typing 'pr filename' will print the file with name filename on your terminal.

5. User: How can I find out how full the disk is?
   UC: Typing 'df' will tell you how full the disk is.

6. User: What is a search path?
   UC: A search path in UNIX is a list of directories in which the operating system searches for programs to execute.

7. User: What do you know about UNIX?
   UC: I currently know about the UNIX file system.

8. User: How do I print the file fetch.l on the line printer?
   UC: To print the file fetch.l on the line printer type 'lpr fetch.l'.

   .
   . (intervening commands and questions)
   .

   User: Has the file fetch.l been printed yet?
   UC: The file fetch.l is in the line printer queue.
   User: How can I cancel it?
   UC: To remove the file fetch.l from the line printer queue you must type 'lprm arens'.

9. User: I can't copy the file tmp1 onto the file args.
   UC: You don't own the file args.

The above examples demonstrate some of the capabilities of UC. UC can understand a wide range of phrasings of questions and requests (examples 1–5); it can identify desired commands (examples 1–5, 8); it can describe operating system terms (example 6); it can respond to queries concerning its own knowledge

base (example 7); it is capable of remembering information and recalling it later (example 8); it can intelligently resolve pronomial references ("How can I cancel *it* .", example 8); and it is able to understand indirect requests for help (example 9). Chapter IV contains a detailed description of how UC achieves all this.

The Context Model is at the center of this version of UC. It contains a representation of what the system considers to be present in its environment, of the files and directories that it knows about, of actions that have been performed or mentioned, etc. The PHRAN (PHRasal ANalyzer) program, a language analyzer described in detail in Chapter II, receives the user's input and produces a representation of its meaning. In doing so, PHRAN both makes use of the information in the Context Model and updates it. The meaning representation arrived at by PHRAN is used as a key for the fetching of related information from UC's long term memory. For example, when the description of a known problem is recognized, its solution is retrieved. The additional structures retrieved in this manner are used to prepare responses for the user. The responses are then generated in English by PHRED (Jacobs, 1983).

I.3. The World Model as the System's Center

A system such as the UNIX Consultant, which is required to incorporate new information about the world, must have a model of the world surrounding it. Such a system needs the ability to recognize a potentially infinite number of objects and actions received as part of the input. To do so, it must identify these inputs with its representation of things it already believes exist. The Context

Model thus functions as the embodiment of the world to the extent that the system is aware of it. It contains entries representing all objects and activities of which the system is aware. This includes actions the system is considering taking. The system is aware only of those things represented in the Context Model.

The Context Model provides a pool of objects which other processes can utilize. For example, a language understanding system may encounter referring expressions such as *"the boy," "he,"* or *"when I tried to delete the file."* Such a system will have to use the information in the Context Model to determine which of the available objects could serve as possible referents of the phrase.

Similarly, processes that must access and update the state of the world will do so via the Context Model. For example, when a planner attempts to choose a plan in a given situation, it needs to consider the availability of objects. To find out what is in fact available, it examines the Context Model. When a certain plan is chosen, a structure representing it is entered in the Context Model. Since the Context Model embodies the system's view of the world, the system is now "aware" of the existence of chosen plan. In particular, references to it can now be made and understood.

In sum, all that an understanding system perceives of the world, either directly or indirectly, making use of information from memory, is recorded in the Context Model.

The contents of the Context Model represent the system's conception of the world. Consequently, the Context Model also forms the basis for the creation of new long term memory structures which will constitute the system's recollection

of current events. The creation and subsequent availability of such new structures form the basis for a system's ability to hold an ongoing conversation. They provide a measure of continuity to the dialogue by permitting the system to recall and be influenced by earlier exchanges with the user.

I.4. Language Analysis and the Context Model.

The term **natural language analysis** was first introduced by Schank and Riesbeck (1975). It is often used in the Artificial Intelligence literature to refer to the process of translating a natural language utterance into a representation of its meaning. Throughout this thesis I will use the term, at times abbreviated simply as 'analysis', in that sense. A computer program which performs the process of natural language analysis will be called a **natural language analyzer.**

The process of natural language analysis is to be distinguished from a purely syntactic investigation of an utterance, such as that resulting in a syntactic parse tree. The latter process is traditionally referred to as *parsing* *.

The language analyzer used with the UC system is called **PHRAN**, for PHRasal ANalyzer. It is described in detail in Chapter II. At PHRAN's core is a large amount of knowledge about the structures of the language and what they mean. This includes knowledge about the words of the language, as well as information about more complex language constructs. This information is stored in the form of **pattern-concept pairs**. The pattern is a phrasal construct, and the

* The distinction made here is not accepted by the entire AI community. Some researchers use the term *parsing* for both processes.

concept is a representation of the meaning of the phrase; the pairing of the two represents an association between a language form and its underlying meaning.

Using this knowledge base, PHRAN tries to find a pattern that matches the input or part of it. When it finds such a pattern, PHRAN passes on the corresponding concept part to the Context Model. It also uses the concept in order to continue constructing the meaning of the complete utterance. The advantages of this approach are discussed in Chapter II.

The CLUSTER system's model of its world has a central and essential role in enabling it to understand new input. When faced with a new piece of information the system must reconcile it with the entries in its Context Model, which represent the complete world to the extent that the system is aware of it at that moment. In particular, new linguistic input must be analyzed in light of the system's Context Model. For example, if input from the user is semantically ambiguous, the CLUSTER system will examine the Context Model in order to determine which interpretation is appropriate (Cf. Example 1 below). Even after the analysis of an utterance is complete, the system must consider the context in order to determine how the addition of an entry representing the meaning of this utterance will cause the current contents of the Context Model to be modified (Cf. Example 5 below).

Investigating language analysis within the framework of a CLUSTER-based system demonstrates that the language analysis process is very closely intertwined with the determination and maintenance of the context. Reading text supplies the CLUSTER system with a continuous stream of new facts about

the world which are inserted in, and used to modify, the Context Model. This model, in turn, influences the process of language analysis.

I.5. A Few Examples and Observations

Below are some examples that demonstrate how maintaining a Context Model is necessary for an understander to interpret new inputs correctly. These examples are in the form of annotated exchanges between a novice and an expert user of UNIX. The annotation points out features of the Context Model relevant to each example and significant aspects of the language analysis taking place. In Chapter IV I describe in detail how the Context Model is constructed and maintained in a manner that enables it to operate as required. Chapter V describes the interactions between the Context Model and PHRAN.

For the purpose of demonstration I will use typical examples of questions a novice would ask an expert when requesting help with UNIX. Moreover, these are questions that a help system like UC must be able to answer. The "Novice" and "Expert" who appear in the following examples are imaginary characters, and many of the examples are rather simple. Their simplicity is by itself indicative of the pervasiveness of the need for a Context Model. In Chapters III, IV, and V, I describe actual cases of what the UC system is capable of doing.

Example 1

Novice: How do I find out who is logged on to the system?
Expert: Give the command 'finger'.

In order to answer this question, the Expert must know which *system* the Novice is talking about. The obvious answer, i. e., the same system that the Novice is using at this moment, is determined by the fact that the Expert is aware of the Novice's surroundings and activities, and a significant activity at this moment is the use of a particular machine.

Note that this is *not* a case of determining the referent of a phrase from the preceding conversation. This question may in fact have been the first utterance made by the Novice to the Expert. It is the fact that there is a system present in the context that enables the Expert to determine which one is meant, not necessarily the previous mention of a system in a conversation. Explicitly mentioning a system is just one way to cause it to be present in the Expert's (and Novice's) representation of the context, so that references to it can later be made and understood. The point here is that a referent for an expression may be present in the context, even if it was not stated verbally at some earlier point in the conversation.

Observation 1. New input is understood in reference to the existing Context Model, as opposed simply to the preceding text.

In UC, the system starts out with an initial Context Model of its surroundings. This model includes representations for the machine, the user, itself, and its capabilities.

We note above that when hearing the question the Expert must determine which system the Novice is talking about. For the purpose of language analysis we are interested in the exact point at which this takes place. Questions of this

type are inherently difficult to answer, but it appears that in this case, for the human reader, the determination is made before the reading of the sentence is complete. Even if the expression "the system" appeared all by itself, the reader would have no difficulty recognizing that system to which it refers.

Observation 2. The production of a representation of the meaning of the input, to the extent that it can be separated from the rest of the system's operation, is not sentence based. I. e., the input is not processed sentence by sentence. Instead, as soon as a known structure is identified in the input, it is relayed on.

For example, each time PHRAN recognizes a phrasal pattern in the text, the associated concept is inserted in the Context Model.

Example 2

Consider the following two exchanges:

(1) Novice: Do you know how to delete a file?
Expert: Give the command 'rm filename'.

And,

(2) Expert: Do you know how to delete a file?
Novice: Yes.

In both cases the initial question is identical. Yet the person questioned interprets it in differently; as evidenced by the two different replies. Nevertheless, both replies are appropriate in their respective contexts.

Questions are not asked in a vacuum. The person being asked has information available to him concerning what he knows, what the purpose of the ques-

tion could be, and, of course, knowledge of the situation surrounding the asking of the question.

In (1), the Expert presumably knows that he is a person knowledgeable about UNIX and is aware of the fact that this is known to the Novice too. He also knows that the Novice is indeed untutored in the use of the operating system. In such a situation it is only reasonable for the Expert to construe the Novice's utterance as a request for information concerning the identity of the appropriate command.

In case (2), it is the Expert who is asking the question. The Novice, aware of the respective levels of knowledge of the participants, correctly interprets the Expert's utterance as a test of his knowledge of the operating system.

We are thus able to explain the difference between the two responses by taking into consideration the different knowledge states of the persons to whom the questions are posed. They each interpret the question within their respective Context Model. Since these are different, they each come up with a different understanding of what they are being asked, which eventually results in a different reply. In this sense the Context Model acts like the *user model* or *student model* often found in intelligent computer aided instruction systems. (For a survey of such models see Barr & Feigenbaum (1982)).

The above example illustrates the fact that the task of the analyzer does not necessarily end at the sentence boundary. Even after the analyzer finishes processing the question, the result of the analysis must still be reconciled with the listener's model of the world. Only in this manner can the listener be provided

with the utterance's full meaning.

Observation 3. The meaning of the input is a function both of the result of its linguistic analysis and other relevant information gleaned from the Context Model.

Example 3

Expert: There is a system called UC available to help you with UNIX.
Novice: How do I use the system?

As in Example 1, the Expert must now figure out which system is being referred to by the expression "the system". While the machine the Novice is working on is still present in his Context Model, this model has been changed by the Expert's mention of an additional system – UC. In the Novice's model of the world, as in that of the Expert, the UC system has become more prominent. Presumably, this is what accounts for the fact that UC, rather than UNIX, is understood to be the referent of the expression in question.

Once the context has been modified, in this case by the Expert's comment, the same expression may take on a completely new meaning. The Novice's question would make perfect sense even in the absence of the Expert's comment. In that case, however, it would most probably mean something different due to a different assignment of meaning to "the system".

Observation 4. The Context Model is constantly being updated and changed.

Observation 5. New input gives rise to additional structures representing that input in the Context Model.

In UC, once PHRAN recognizes a fragment of the input, a new entry representing what PHRAN has understood is created and inserted into the Context Model.

Observation 6. Entries in the Context Model have a degree of salience associated with them. This degree is taken into account when the need arises to make a choice between several available alternatives in the model.

In the formal Context Model, the level of salience mentioned above will be represented as a level of activation of entries in the model. When PHRAN needs to decide which of several objects present in the model is the one meant by a particular reference, it will use this level of activation to distinguish between otherwise comparable alternatives.

The idea of a model of memory incorporating the spreading of activation has been suggested and developed by several other researchers in artificial intelligence and cognitive psychology. For a discussion of other such models and a comparison with CLUSTER theory see section IV.1.1.

Example 4

Novice: How do I get a list of the people using the machine?
Expert: Type 'users'.
Novice: How can I get the output in a file?

The Novice's second inquiry raises the issue of how it is possible for the Expert to understand the reference to "the output" when output was not mentioned before.

An explanation for this is that, when the Novice first asked about a command for listing the users of the machine, he caused the Expert to recall not only the name of the command, but also other circumstances and facts pertaining to its use. These would include what to type, the fact that a carriage return must be hit after the command name, the fact that giving the command would cause output to be printed on the screen of the terminal, etc. The structures representing all this information now become part of the Expert's model of the context. They are now readily available to a process attempting to determine the referent for a phrase like "the output". Upon encountering this phrase in the Novice's next question, the Expert is able to identify the output as that of the command discussed in the previous utterance.

Observation 7. Upon inserting a new entry into the Context Model, the system must identify other structures related to it. These structures must also be added to the Context Model.

In UC, once a new entry has been inserted in the Context Model, the system's long term memory is searched for groups of other related structures. If found, these too are inserted into the Context Model.

The Expert must have some information that enables him to add to his Context Model the fact that the *users* command has output. This information is stored in his long term memory. In general,

Observation 8. The system's long term memory should be organized in a way that facilitates access from any one structure to additional structures related to it.

The above feature of long term memory is achieved in CLUSTER by having its LTM made up of structures called *clusters*. A cluster is a grouping of pieces of information, all of which are brought into the Context Model whenever any one of them is first inserted into it. The criterion for items being grouped together in a cluster is their tendency to co-occur in the world around us. In other words, items form a cluster if they appear to us to be associated with each other. In general, this relationship does not imply any semantic connection between its constituents.

The choice of the relation of *associatedness* as the basis for cluster structure is supported by psychological evidence. Anderson (1983) notes that there is considerable evidence for the existence of an automatic process that makes information available on the basis of associative relatedness. For more detail see Fischler (1977), McKoon and Ratcliff (1979), Meyer and Schvaneveldt (1976), Neely (1977), Warren (1972), and Warren (1977).

Example 5

Novice: How can I print the file test.1 on the line printer?
Expert: Give the command 'lpr test.1'.

. [time passes, during which other things are discussed]

Novice: What has happened to my file?
Expert: Which file?
Novice: The file test.1.
Expert: Oh, you can check the line printer queue to find out.

The Novice starts out by asking a simple question and receiving an appropriate answer from the Expert. The conversation continues on matters unrelated to

the question and answer until the Novice decides to inquire about the fate of the file he printed earlier. The Expert at first has some difficulty in understanding the Novice's question because he cannot find a referent for the phrase "my file". Thus, although this file was at one time represented in the Expert's model of the context, this is apparently no longer the case.

Observation 9. The salience of an entry in the Context Model decreases as time passes. Eventually the entry is dropped from the model. Thereafter, it will no longer be available to other processes.

As mentioned earlier, this salience will be represented as the activation level of an entry in the Context Model. Unless repeated access is made to entries in the Context Model, their activation levels drop with time. A resulting effect is that if an object is not related to the discussion it will be dropped from the context after a while. It will then no longer be available to processes using the Context Model. In effect, the object will be forgotten.

The situation just described does not cause the Novice to abandon his attempt to gain information from the Expert. Rather, he tells the Expert the name of the file he is discussing. This enables the latter to recall other relevant facts about this file, among them that it had been sent to the line printer. In a previous example we saw how information related to that which is being inserted into the Context Model must also be accessible to the system. However, our current case is slightly different in that the information involved is not of general importance, or true of all files, but is unique to this particular file. Only the file test.1 has been printed; and this fact could only remain known to the system if it

were recorded in a more permanent form when the event occurred in the first place.

Observation 10. The information present in the Context Model must be periodically recorded. This should be done in a form that maintains the relationship between the different pieces of information in the model.

In UC, this is accomplished by the user instructing the system to make a copy of the current structures in the Context Model, form a new cluster out of them, and place it in its long term memory. This process has not been automated. Related research on recognizing textual cues for the switching of context might be helpful towards this end (cf. Grosz, 1977).

The above example rather elaborately demonstrates a fairly straightforward phenomenon: when the Expert hears the Novice's mention of the file named *test.1* this causes changes in his knowledge structures to the extent that later on the Novice is able to cause him to recall the whole episode simply by mentioning the name of the file again. In other words, analysis and understanding linguistic input cause lasting changes in the understander's knowledge structures. When these changes are recorded, the understander is later able to make repeated use of the new structures, as we see in the second part of the example. This point has been noted in the past and is central to the work of Lebowitz (1980).

Observation 11. During the process of understanding the text the analyzer is continually causing the Context Model to be modified by the insertion of structures representing the meaning of fragments

of the input. In this manner, earlier input will influence the interpretation of later input.

I.6. Example of PHRAN-Context Model Operation

This section contains an example of the joint operation of PHRAN and the Context Model. Little of the internals of the respective systems will be discussed. Numerous examples later in this thesis will bring those details to light. A trace of UC running on this example is provided in Chapter VI.

I will describe some of the processing involved in engaging in the following dialogue, first presented in section I.2. I will concentrate on the determination of the referent of the word 'it' in line 5 below.

[1] User: How do I print the file fetch.l on the line printer?
[2] UC: To print the file fetch.l on the line printer type 'lpr fetch.l'.

. (intervening commands and questions)

[3] User: Has the file fetch.l been printed yet?
[4] UC: The file fetch.l is in the line printer queue.
[5] User: How can I cancel it?
[6] UC: To remove the file fetch.l from the line printer queue you must type 'lprm arens'.

During the analysis of the user's first question, [1], several patterns match fragments of the input. The matched fragments are, in the order in which they are recognized:*

* Expressions in angular brackets represent previously matched fragments which are part of the larger pattern.

I
the file fetch.l
How do <I> print <the file fetch.l>
the line printer
on <the line printer>
<How do I print the file fetch.l> <on the line printer>

The analysis of each fragment results in a structure representing its meaning. This structure is inserted in the Context Model. The third fragment recognized is a request concerning printing. Its processing results in a whole cluster being retrieved from long term memory and inserted in the Context Model. The cluster contains structures describing the command to print, its effect, the object printed and the fact that a printer is involved. This information is used to determine the answer provided by the system in [2].

After UC's reply, the Context Model contains representations of the following objects and actions:

(1) The user
(2) The file fetch.l
(3) The line printer
(4) A request to print fetch.l
(5) A command to print the file ('lpr fetch.l')
(6) The printing of the file
(7) The user has requested to print the file
(8) UC has informed the user how to print the file

The contents of the Context Model are then preserved in long term memory in the form of a new cluster. (This is done following an explicit request by the user, as this process is not currently automatic.) The new cluster will be recalled when the representation of any of the entries (2), (4), (5), or (7), is encountered again*.

* For further discussion of the creation of new clusters and their recall see sections IV.5. and IV.3.2., respectively.

The conversation now shifts to other topics; the structures previously in the Context Model are gradually deleted. By the time the user asks question [3], there is no trace left in the Context Model of the structures used in the earlier exchange.

When the user's second question is processed by PHRAN, the following fragments are matched by known patterns:

the file fetch.l
Has <the file fetch.l> been printed yet?

When an entry describing the first fragment is created in the Context Model, it serves as a key for retrieving the new cluster created in response to the first transaction. (The concept associated with second language pattern recognized causes the retrieval of other clusters. These clusters enable the system to respond with [4], but are otherwise irrelevant to this example.)

When question [5] is asked by the user, it is analyzed by PHRAN and the following input fragments are match by known patterns:

I
it
How can <I> cancel <it>?

The structure created to describe the first fragment is identified with the structure describing the user, which is already present in the Context Model. When a structure is created in the Context Model to represent the meaning of the second matched fragment, *it*, no immediate identification with a structure in the Context Model is possible. Consequently, it is temporarily identified with a newly created entry containing a list of all possible referents. This list contains essen-

tially all objects currently in the Context Model.

Finally the last fragment is recognized. One of the patterns used to match this input places a restriction on the referent of *it*. In particular, the pattern

<person> cancel <command>

is used in the analysis of this fragment. This pattern requires that its final item be a command. The only command found in the Context Model is the one to print the file fetch.l. This was entry (5) in the cluster formed after processing question [1], which was stored in long term memory and retrieved during the processing of question [3] above. The identification of this entry as the referent enables UC to correctly interpret the user's question and eventually to provide the answer [6].

I.7. The Rest of this Thesis

The rest of this thesis will be divided into four parts.

First PHRAN is described, presented as an independently operating system. I discuss phrasal language constructs and the notation used for the concept parts of pattern-concept pairs, and the algorithm used by PHRAN. I also describe slightly extended notions of pattern and concept that allow the system to identify adverbial phrases and certain relative clauses. Finally, the indexing scheme used to ensure that the appropriate pattern-concept pairs are available to the analyzer when they are needed is discussed. This indexing scheme serves to minimize the number of pairs actively searched for in the input text.

The following two chapters provide a description of the Context Model and the ideas behind it. The first of the two contains an overview and a detailed example, and the second an in-depth examination of the Context Model and the processes that take place within it. It includes a description of the notion of a *cluster* of memory structures, which is the basic construction in the system's long term memory. This structure encodes the relations between various pieces of knowledge the system has available to it. I describe how the Context Model is constructed with the information provided by the input together with what is present in the system's permanent store of knowledge.

Chapter V explains how PHRAN works within the framework provided by the Context Model. It describes how the system's view of the world changes when reading text, and how what is present in the Context Model influences the system's understanding of new text.

Finally, in Chapter VI I provide traces of several sessions with the UNIX Consultant. These detailed examples serve to illustrate the capablities of the Context Modeler described in the rest of this thesis.

CHAPTER II

PHRAN

II.1. Introduction

PHRAN, the PHRasal ANalyzer, is an integral part of the UNIX Consultant. It runs within the framework provided by the CLUSTER theory and in conjunction with the Context Modeler. The UC system understands linguistic input by using PHRAN to analyze the language and interpreting the results in light of what it knows about the surrounding world, as represented in the Context Model. Furthermore, since the Context Model is to reflect what the system has read, the processing of input causes changes to its contents.

In spite of the intrinsic relation of PHRAN to the Context Model, I devote this chapter to a description of PHRAN as a system operating independently. The reason for this is twofold. First, by describing PHRAN in this manner I can concentrate on the theoretical considerations involved in the design and implementation of the processing it performs, and on the nature of the data about language that it employs in the course of this processing. Second, it is possible to use PHRAN as an independent system, in a limited enough context. Therefore, it is useful to examine the abilities and limitations of using PHRAN alone.

In Chapter V I will describe the operation of PHRAN in conjunction with the Context Model. Having by then seen what PHRAN is able and unable to do on its own, the reader will be in a position to understand what exactly is gained

by centering the process of language understanding around the system's Context Model.

II.1.1. Motivation

One of the main motivations of the research that led to PHRAN was the need to determine the form of the information a language analyzer has about the language it is intended to analyze. Due to the potentially large amount of such information, an efficient way must be found to organize it. In addition, since it is impossible for the original system designer to encode *all* the available linguistic information initially, the method of representation must be such that it will be relatively easy for people other than the designer to add knowledge to the system. Throughout all this, we would still like the system to be an acceptable model of a human language user.

Another motivation for PHRAN was the realization that many of the expressions in a language have a meaning that *cannot* be determined from the meaning of the words they comprise. Research in linguistics indicates that such non-productive expressions do not constitute a limited class of exceptions, as was assumed by designers of other language analyzers. (For recent work on the importance of these constructs see Fillmore (1985) on contsruction grammar.) Some rather well-known examples of non-productive expressions are idioms, canned phrases, and lexical collocations. The meaning of these types of utterances can only be determined on the basis of specific prior knowledge of these expressions.

For example, a language analyzer must be able to realize that "French restaurant" refers to a restaurant specializing in French food, as opposed to a restaurant owned by French citizens, say. The information we have that enables us to understand this expression is unique to constructions of the form "*nationality* restaurant". This information is not associated with either of the constituents of the expression separately. This is evidenced by the fact that when any one of the constituents is not of the form described above the resultant meaning is entirely different. For example, a "clean restaurant" does not necessarily specialize in serving clean food; a "French laundry" gives no special status to French clothing.

If we attached the linguistic knowledge necessary to understand the expression "French restaurant" to one of its constituents, we would be faced with quite an unnatural situation. We would have information attached to one word, or class of words, that required checking if it were followed (or preceded) by some other word. In effect, we would have a process that would have to be attached to one of the constituents so as to be run whenever that constituent is found in the text, and which would depend on which word we chose to attach it to. Gershman (1979) followed this course in extending ELI (Riesbeck and Schank, 1975).

This example demonstrates that the meaning of the expression "French restaurant" is determined from information associated with the pattern

<nationality> restaurant

and *not* with any of the words in it alone. The **pattern-concept pair**, the basic unit of data about the language that PHRAN uses, is an attempt to capture

this property of language.

Most language analyzers work under the assumption that the meaning of an utterance can be computed from the meaning of its constituents by the use of general rules. Thus almost all the knowledge available to other systems is at the word level. However, attaching all knowledge in the system at this low level is restrictive, because it forces decisions about processing strategies that are unrelated to the meaning of the language. Such control structure information would need to be included as part of the meaning of various words, thus making the addition of new linguistic knowledge to the system difficult. Moreover, since the processing involved in understanding language is very different from that which takes place during language production, this information would have to be included twice – once in a form suitable for analysis, and a second time in a form suitable for the production mechanism.

The model proposed here is an attempt to solve the problems indicated above. In PHRAN, the knowledge about the language is not only about the meaning of words, but also about the meaning of larger utterances. In fact, knowledge about words is a relatively small part of the system's total database. This knowledge takes the form of **pattern-concept pairs**. A pattern is a phrasal construct, and a concept is a description of the meaning of the pattern. This pair associates various language forms with their meanings. The knowledge about the language is thus kept separate from the processing strategies that use it. In fact, the same knowledge PHRAN uses to understand language is also usable by a language production mechanism. The former matches utterances in

the text against patterns and uses the associated concept parts to represent the meaning, while the latter performs roughly the reverse.

The notion of pattern used in PHRAN emphasizes the ordering of constituents (see section II.3.3). The PHRAN approach would thus be less successful in more inflected languages with freer word order than English. The representation used by PHRAN has nevertheless been used to understand and generate utterances in Spanish and Chinese, using the same processing strategies but new databases. The databases consist of the pattern-concept pairs appropriate for the respective language. See Wilensky and Morgan (1981) for details.

II.1.2. PHRAN

PHRAN is a multi-language analyzer based on the principles of language understanding and production outlined above. PHRAN reads text and produces structures representing its meaning, using the information about the utterances of the language that is encoded in the form of pattern-concept pairs. A companion program, PHRED (PHrasal English Diction) (Jacobs, 1983), produces natural language utterances when presented with meaning representations, sharing an identical database of pattern-concept pairs with PHRAN. PHRAN may be used independently as a front end for other systems that will receive the results of its operation and reason about them or perform other understanding functions. PHRAN is used in a slightly altered mode of operation in conjunction with the Context Modeler. The latter setup allows a more general language understanding system to overcome some of the difficulties inherent in an analyzer of this sort, as

were illustrated in the first chapter, in order to obtain a better understanding of the input text.

In PHRAN, the ability to analyze both productive and non-productive language forms is totally integrated. It provides an extensible natural language understanding facility which has knowledge about both words and more complex constructs.

Here are a few examples of sentences that PHRAN can analyze:

> Oilmen are encouraged by the amount of natural gas discovered in the Baltimore Canyon, an undersea trough about 100 miles off the New Jersey coast.
> (Newsweek, March 1980)

> Tenneco, one of 39 companies engaged in drilling in the area, thinks its leased tract contains a marketable supply of gas.

> Mary wanted to talk to the man who brought her son home.

> The young man was told to drive quickly over to Berkeley.

> John has gotten into another argument with his boss.

> The man rewarded Bill with a million dollars for saving his life.

> The book marker wanted by your mother is in the red box.

> Willa's best friend is a bum who lives in madison square.

> If John gives Bill the big apple then Bill won't be hungry.

> The school bus was driven by Mary's friend to The Big Apple.

> John has kicked the bucket.

> John kicked the red bucket.

The bucket was kicked by John.

The old French man's brother picked the book up.

If Mary brings John we will go to a Chinese restaurant.

Willa gives me a headache.

The linguistic component of a pattern-concept pair is called a *phrasal pattern.* A phrasal pattern is a description of an utterance at one of various different levels of abstraction. It may be a literal string such as

"The Big Apple",

which can only appear in this form; it may be a pattern with greater flexibility such as

"<nationality> restaurant",

which allows any nationality to appear as the first constituent, or

"<person> <kick> the bucket",

which allows any person to appear as the first constituent and any form of the verb *kick* to appear as the second constituent, with no variation allowed on the last two; or it may be a very general phrase such as

"<person> <kick> <person> <physical object>".

Associated with each phrasal pattern is a *conceptual template.* A conceptual template is description of the meaning of the phrasal pattern, usually with references to constituents of the associated phrase. Each association of a phrasal pattern with a conceptual template encodes a single piece of knowledge about the

language the database is describing. For example, associated with the phrasal pattern

"<nationality> restaurant"

is the conceptual template denoting a restaurant that serves <nationality> type food. Associated with the phrasal pattern

"<person1> <give> <person2> <physical-object>"

is the conceptual template that denotes a transfer of possession of <object> from <person1> to <person2>, initiated by <person1>.

When operating on its own, PHRAN's understanding process reads input text and tries to find phrasal patterns that match fragments of it. In the course of reading the text PHRAN eliminates certain patterns that originally seemed applicable and suggest new ones to be searched for. At some point it may recognize the presence of one or more patterns in the text, at which point it may have to choose among several possible patterns matching the same part of the text. The result of matching a pattern may in turn be present as a constituent in a larger pattern. Finally, the conceptual template associated with a pattern that leaves none of the input unaccounted for is used to generate a structure denoting the meaning of the complete utterance.

II.1.3. Advantages of PHRAN

One of the main advantages of PHRAN is the fact that both productive and non-productive language is handled by one mechanism. Knowledge about non-productive language is encoded in the same type of pattern-concept pair as is

knowledge about productive language, the only difference being the degree of variability permitted in the phrasal pattern. The processing is totally uniform in this respect.

Another major advantage is the system's separation of knowledge about language from processing strategies. The language knowledge is kept in the pattern-concept pairs, while the processing knowledge is embedded in the code of the understanding mechanism.

This separation, in turn, allows the knowledge about the language to be entirely declarative, and thus sharable by the language production mechanism, PHRED. When a new pattern-concept pair is added to the knowledge base, it automatically becomes available both to the analyzer and to the generator, thus extending the capabilities of both simultaneously. It is also possible to use PHRAN as an analyzer for other languages simply by providing it with a database of pattern-concept pairs appropriate for these languages. This has been done for Spanish, and to a more limited extent for Chinese (Wilensky and Morgan, 1981). As evidenced by the additional databases built by people other than the original designer of the system, adding information in the form of pattern-concept pairs is relatively easy. In particular, adding information about the language does not require constructing any new routines.

Finally, it would seem that PHRAN provides a better model of human language understanding than previously available parsers. It provides a uniform treatment to different forms of language without treating the less productive ones a special cases and requiring a separate mechanism for their processing. Most

other systems that bother with these forms of language at all, do precisely that, and do not consider them an essential part of their model of language understanding.

Having a single store of knowledge about language, usable both for analysis and for generation, also seems cognitively correct. Our experience as language users indicates that the use of an expression does not need to be learned separately for understanding it and for being able to express it. The greater ease involved in the understanding of a new expression, as opposed to its generation, can be accounted for by a difference in the knowledge indexing mechanisms used in the processes of analysis and generation, or by differences in the procedures used for the two tasks.

There also exists psychological evidence suggesting that people attach special importance to semantically meaningful phrasal structure when they read (Just and Carpenter, 1980) (Carpenter and Daneman, 1981) (McDonald and Carpenter, 1981). This fact may be accounted for by having a phrase-based database of linguistic information, as PHRAN does.

II.2. Phrasal Language Constructs

In this section I will examine those constructs that are present in the English language and to which I refer when speaking of "*phrasal constructs*". This list is neither an exhaustive listing of such constructs, nor is there anything distinctive about the way these are handled. In PHRAN, all phrasal language constructs are handled in the same way. No claim is made that there is a small number of

different types of constructs, each with its own special processing rules. PHRAN allows one to define independently all varieties of patterns; they are all treated uniformly by the analysis process. Rather, the purpose of this section is to demonstrate the pervasiveness and importance of these forms in the language. It is an attempt to justify the work involved in developing a system that would afford these constructs the same status as has traditionally been granted to more productive language forms.

The term **phrasal language constructs** refers to those language units of which the language user has specific knowledge. Many different forms come under this heading. In particular, non-productive language constructs, those utterances whose meaning is not determined simply by combining the meaning of the constituents making them up, form an obvious subclass. However, as mentioned above, strictly non-productive constructs are by no means the only type of utterance that PHRAN deals with.

The phrasal constructs that allow the least variability are *lexical collocations* and "*canned phrases*". Some examples of the former are

The Big Apple,
Hush money,
Prime minister,
Pipe dream,
Head hunter,

and many other noun-noun compounds.

Many examples of "canned phrases" are presented by Becker (1975). He lists various classes of phrasal structures and proposes the use of a *phrasal lexicon.* These include *situational utterances*, which the hearer and user associate

with a particular situation, e. g.

It only hurts when I laugh!

and *verbatim texts*, e. g.

99 and 44/100 percent pure.

Both of the above expressions can be understood in a productive manner. Yet the complete phrase conveys a meaning beyond the one that could be determined by one who was not familiar with the phrase beforehand.

The meaning, or some aspect of the meaning, of all the above expressions seems to be associated directly with them in their particular form, and not computed as a result of the words from which they are made up. They function very much like entries in a dictionary, like multi-word "words", allowing no variation in the form of their constituent parts, or their order.

Chafe (1968) notes that some such expressions, e. g. "by and large", would be considered ungrammatical if it were not for the fact that they are recognized as phrasal constructions. The fact that as a rule the hearer is not even aware of this fact is evidence that he has knowledge of the meaning of such utterances associated directly with their complete form. It would appear that no attempt is made to analyze the constituent structure of such an expression once it is recognized.

Expressions that allow a bit more variability in their constituents are idiomatic phrases like

Kick the bucket,
Throw the book at,
Bite the dust,
Sweep under the rug.

In these, the subject of the sentence in which they are used can be any noun group designating a person, and the verb involved may appear in any form. For example, the verb *kick* in the first expression may appear as *kick*, *has kicked*, *would have kicked*, etc., with the meaning of the expression as a whole still being maintained, with the appropriate changes.

A language form that allows a similar degree of generalization is the one used as an example in an earlier section,

<nationality> restaurant

Here the condition on the expression that may appear as the first constituent is semantic. Any word or words designating a nationality may be present in the above expression, and the whole form would then have associated with it a concept denoting a restaurant serving food characteristic of that nationality. The pattern indicated will include "Vietnamese restaurant", and "Hungarian restaurant", etc.

In the most general case a phrasal pattern may be used to express a "normal", productive, form of using the word sense associated with a word. For example, the phrase

<person> <eat> <food>

would be used to express a "normal" use of the verb *to eat*. This phrasal pattern indicates that an expression denoting a person is followed by a form of the verb

eat, which in turn is followed by an expression denoting an item of food. Associated with this phrasal pattern is a concept denoting the ingestion of some food. More specifically, the concept will denote the ingestion of the food appearing in the place marked by <food> by the person appearing in the place marked by <person>.

Phrasal constructs vary greatly in the how specific or general a set of utterances they may correspond to. I consider these constructs to be phrasal in that the language user has special knowledge of the construct as a whole, and uses this knowledge to understand it and produce utterances based on it. In the case of non-productive utterances there is no general rule that determines the meaning of the phrasal construct from the meaning of its constituents. In other cases such a rule may exist. The point is that the knowledge of the language contained in the system is centered around these structures and their associated meaning representations. A general rule is simply a very general phrasal pattern.

In the operation of PHRAN the only difference between productive and non-productive forms of language is a difference of degree in the variability allowed in the constituents of the form. While productive language forms allow greater variability, non-productive forms allow less. Thus, these two kinds of linguistic structures are not so much two discrete types, but rather, the two endpoints of a continuous spectrum.

The criterion for determining whether a set of language forms should be accommodated with a single phrasal pattern is conceptual. Thus, if the meaning of phrases is similar and their structure is too, they should be considered

instances of the same underlying phrasal pattern. The goal of this approach is to express the relation of language patterns to meaning, and as a result, little importance is given to syntactic parallelism between expressions that is not accompanied by semantic similarity. (Cf. Fillmore, 1985).

II.3. How PHRAN works

Having discussed in detail some of the motivations for phrasal-based language processing, I now examine some of the details of the implementation. The basic intent of this section is to show how the claims for the model have been realized, how some of the problems that arise once language processing issues are formulated in this model have been solved, and what problems still remain to be solved. While improvements to the program are still being made, it is far enough along to allow critical examination.

II.3.1. Overall Algorithm

PHRAN is made up of three parts:

- A database of pattern-concept pairs,
- A set of comprehension routines,
- A routine which suggests appropriate pattern-concept pairs.

PHRAN takes as input an English sentence, and as it reads it from left to right, PHRAN compares the sentence against patterns from the database. Whenever a matching pattern is found, PHRAN interprets that part of the sentence that matched the pattern as describing the concept associated with the pattern in

the pattern-concept pair. Periodically throughout this process, the pattern suggesting mechanism offers PHRAN patterns that enable the analysis to continue.

PHRAN's knowledge base contains many of the types of patterns described previously. They range in specificity from literal strings to sequences of concepts described in a certain order. Patterns vary in size from those that match individual words to those that match entire sentences.

As PHRAN reads an input sentence, the words appearing in the sentence, along with patterns that have already matched parts of it, drive the pattern suggesting routine in offering PHRAN patterns to be considered for matching fragments of the sentence. The patterns PHRAN considers "active" (i.e. those that have matched up to the point where PHRAN has read) are used to interpret the remaining words in the sentence by giving PHRAN a reasonable idea of what may conceivably appear later on. The process of suggesting new patterns and matching them against the input continues until the end of the sentence is reached and all suggested patterns have been tried. At this point PHRAN uses the pattern that matches the whole sentence to determine the meaning of the entire utterance.

II.3.1.1. Overview of PHRAN Patterns

A pattern-concept pair consists of a specification of the phrasal unit, an associated concept, and some additional information about how the two are related. PHRAN instantiates the concept, i. e., fills in the specifics from the matched pattern, when the pattern match is successful. It is often necessary to

carry around more information, however. For example, it may be important to know that a concept denoting a person came from a noun phrase. Thus when PHRAN instantiates a concept, it actually creates an item called a **term** that includes the concept as well as some additional information.

A pattern is a sequence of conditions that must hold true for a sequence of terms. The conditions on a term may refer to lexical, syntactical, and conceptual categories. They range from the requirement that the word giving rise to the term be identical to a given word, to having the term represent a particular kind of action.

A pattern may specify optional terms too, the place where these may appear, and what effect (if any) their appearance will have on the properties of the term formed if the pattern is matched. For example, consider the following informal description of one of the patterns suggested by the mention of the verb 'to eat' in certain contexts.

```
{ pattern to recognize -
        [<first term: represents a person>

         <second term: an active form of eat>

         <OPTIONAL third term: represents food>
        ]

  term to form -
        (ingest (actor <first term>)
                (object <third term, if present, else food>))
}
```

This pattern directs PHRAN to ask if the term preceding the verb is a person, if the term following that is a form of *to eat*, and if the following term could

be an item of food. If so, then the first term will be used to fill the *actor* slot of the *ingest* conceptualization, and the third term to fill the *object* slot. The third term is marked as optional. If it is not present in the text, PHRAN will fill the *object* slot with a default representing generic food.

II.3.1.1.1. Pattern Generation

Not all patterns PHRAN needs are explicitly present in its database. For instance, consider the pattern of the previous example,

[<person> <root eat> <food (*optional*)>]

This pattern will match sentences like "John ate the apple". But it seems as if the same information should be usable in sentences like "John wanted to eat the apple" as well, in conjunction with a pattern like

[<person> <root want> <*event*>].

The problem is that here the verb appears preceded by 'to' and not a <person> as before.

PHRAN does not have each of these cases stored explicitly in its database, but rather it has only the 'basic' patterns (i. e. '<subject> <verb> <object>' type patterns) stored, and the pattern suggesting routine is able to generate a new pattern to use to match the input. PHRAN can generate patterns that recognize passive forms of verbs, various types of relative clauses, phrase of the form

<person> <do> <verb>,
<person> <do> not <verb>,
<person> <modal-verb> <verb>,

etc., and phrases with infinitive forms of verbs (e. g., "John likes to go fishing"). In all, PHRAN can generate twelve types of modified patterns, although each form is not applicable to all verb-based patterns.

II.3.2. Processing Overview

When PHRAN analyzes a sentence, it reads the words one at a time, from left to right. It does a little morphological analysis on the sentence while it is being read, just enough to recognize contractions and "'s"s. The pattern suggesting routine determines if any new patterns should be tried, and PHRAN checks all the new patterns to see if they agree with that part of the sentence already analyzed, discarding those that don't. A word's meaning is determined simply by its matching a pattern consisting of that literal word. Then a new term is formed with the properties specified in the concept associated with the word, and this term is added to a list PHRAN maintains. PHRAN then checks if the term it just added to the list completes or extends patterns that had already been partially matched by the previous terms. If a pattern is matched completely, the terms matching that pattern are removed and a new one, specified by the concept part of the pattern-concept pair, is formed and replaces the terms the pattern matched.

When PHRAN finishes processing one word it reads the next, iterating this procedure until it reaches the end of a sentence. At this point, it should end up

with a single term on its list. This term contains the conceptualization representing the meaning of the whole sentence.

II.3.2.1. Simple Example

The following is a highly simplified example of how PHRAN processes the sentence "John dropped out of high school":

First the word "John" is read. "John" matches the pattern consisting of the literal "John", and the concept associated with this pattern causes a term to be formed that represents a noun phrase and a particular male person named John. No other patterns are suggested. This term is added to *PHRAN-BUF*, the list of terms PHRAN maintains and which will eventually contain the meaning of the sentence. Thus *PHRAN-BUF* looks like

< [john1 - person, np] >

"Dropped" is read next. It matches the literal "dropped", and an appropriate term is formed. The pattern suggesting routine instructs PHRAN to consider the 'basic' pattern associated with the verb 'to drop', which is:

{ [<person> <root drop> <object>] [...] }.

Its initial condition is found to be satisfied by the first term in *PHRAN-BUF* - this fact is stored under that term, and succeeding ones will be checked to see if this partial match continues. The term that was formed after reading "dropped" is now added to the list. *PHRAN-BUF* is now:

< [john1 - person, np] , [drop - verb] >

PHRAN now checks to see if the pattern stored under the first term matches the term just added to *PHRAN-BUF* too, and it does. This new fact is now stored under the last term.

Next the word "out" is read. The pattern suggestion mechanism is alerted by the occurrence of the verb 'drop' followed by the word 'out', and at this point it instructs PHRAN to consider the pattern

{ [<person> <root drop> "out" "of" <school>] [...] }

The list in *PHRAN-BUF* is checked against this pattern to see if it matches its first two terms, and since that is the case, this fact is stored under the second term. A term associated with 'out' is now added to *PHRAN-BUF*:

< [john1 - person, np] , [drop - verb] , [out] >

The two patterns that have matched up to *drop* are checked to see if the new term extends them. This is true only for the second pattern, and this fact is stored under the next term. The pattern

[<person> <root drop> <object>]

is ignored from this point on.

Now the word "of" is read. A term is formed and added to *PHRAN-BUF*. The pattern that matched up to *out* is extended by *of* so the pattern is moved to the next term.

The word "high" is read and a term is formed and added to *PHRAN-BUF*. Now the pattern under *of* is compared against *high*. It doesn't satisfy the next condition. PHRAN reads "school", and the pattern suggestion routine

presents PHRAN with two patterns:

1. { ["high" "school"] [representation denoting a school for 10th through 12th graders] }

2. { [<adjective> <noun>] [representation denoting noun modified by adjective] }

Both patterns are satisfied by the previous term and this fact is stored under it. The new term is added to *PHRAN-BUF*, now:

< [john1 - person, np] , [drop - verb] , [out] ,
[of] , [high - adj] , [school - school, noun] >

The two patterns are compared against the last term, and both are matched. The last two terms are removed from *PHRAN-BUF*, and the patterns under *of* are checked to determine which of the two possible meanings should be chosen. Patterns are suggested such that the more specific ones appear first, so that the more specific interpretation will be chosen if all patterns match equally well. Only if the second meaning (i. e. a school that is high) were explicitly specified by a previous pattern, would it have been chosen.

A term is formed and added to *PHRAN-BUF*, which now contains

< [john1 - person, np] , [drop - verb] , [out] ,
[of] , [high-school1 - school, np] >

The pattern under *of* is checked against the last term in *PHRAN-BUF*. PHRAN finds a complete match, so all the matched terms are removed and replaced by the concept associated with this pattern.

PHRAN-BUF now contains this concept as the final result:

```
< [ ($schooling (student john1)
                (school high-school1)
                (termination premature)) ] >
```

II.3.3. Pattern-Concept Pairs in More Detail

II.3.3.1. The Pattern

The pattern portion of a pattern-concept pair consists of a sequence of predicates. These may take one of several forms:

1) A word; which will match only a term representing this exact word.

2) A class name (in parentheses); will match any term representing a member of this class (e. g. "(food)" or "(physical-object)").

3) A pair, the first element of which is a property name and the second is a value; will match any term having the required value of the property (e.g. "(Part-Of-Speech verb)").

In addition, it is possible to negate a condition or specify that a conjunction or disjunction of several must hold. There are also some less often used special conditions, e. g. requiring that the term describe a concept of a certain type, such as a PTRANS.

The following is one of the patterns which may be suggested by the occurrence of the verb 'give' in an utterance:

[(person) (root give) (person) (physob)]

Each condition aside from the second is a call to memory to determine if the

object denoted by the term belongs to the specified category.

II.3.3.1.1. Optional Parts

Not all terms a pattern calls for are required to be present for PHRAN to consider the pattern successfully matched. To indicate the presence of such optional terms, a list of pattern-concept pairs is inserted into the pattern at the appropriate place. These pairs have as their first element a sub-pattern that will match the optional terms. The second part describes how the new term to be formed if the main pattern is found should be modified to reflect the existence of the optional sub-pattern.

The concept corresponding to the optional part of a pattern is treated in a form slightly different from the way regular concept parts of pattern-concept pairs are treated. As usual, it consists of pairs of expressions. The first of each pair will be placed as is at the end of the properties of the term to be formed, and the second will be evaluated first and then placed on that list, which will be described in the next section.

For example, another pattern suggested when 'give' is seen is the following:

```
[(person) (root give) (physob) ([(to (person))
                                 (to (opt-val 2 cd-form))])]
```

The terms of this pattern describe a person, the verb *give*, and then some physical object. The last term describes the optional terms, consisting of the word *to* followed by a person description. Associated with this pattern is a concept part that specifies what to do with the optional part if it is there. Here it specifies that the second term in the optional pattern should fill in the *TO* slot in the

conceptualization associated with the whole pattern. This associated conceptualization is not shown here, but will be described below.

It should be noted that the particular pattern shown above need not be a separate pattern in PHRAN from the one that looks for the verb followed by the recipient followed by the object transferred. Patterns are often shown without all the alternatives that are possible for expositional purposes. Sometimes it is simpler to write the actual patterns separately, although no theoretical significance is attached to this disposition.

II.3.3.2. The Concept

When a pattern is matched, PHRAN removes the terms that match it from *PHRAN-BUF* and replaces them with a new term, as defined by the second part of the pattern-concept pair. For example, here is a complete pattern that may be suggested when the verb 'eat' is encountered:

```
([(person) (root eat) ([[(food))
                        (food (opt-val 1 cd-form))]])]

 [p-o-s 'sentence
  cd-form '(ingest (actor ?actor) (object ?food))
  actor (value 1 cd-form)
  food 'food)])
```

The concept portion of this pair describes a term covering an entire sentence, and whose meaning is the action of *ingest*ing some food. The next two descriptors specify how to fill in variable parts of this action. These begin with '?'. The expression *(value n prop)* specifies the 'prop' property of the n'th term in the matched sequence of the pattern (not counting optional terms). *Opt-val* does

the same thing with regards to a matched optional sub-pattern. Thus the concept description above specifies that the actor of the action is to be the term matching the first condition. The object eaten will be either the default concept *FOOD*, or, if the optional sub-pattern was found, the term corresponding to this sub-pattern.

Some parts of a conceptualization may be left empty, to be filled at a later point; a slot in the conceptualization can be filled by a term in a higher level pattern of which this one is an element. For example, when analyzing "John wanted to eat a cupcake" a slight modification of the previous pattern is used to find the meaning of "to eat a cupcake". No subject appears in this form, and the higher level pattern specifies where it may find it. That is, a pattern associated with "want" looks like the following:

```
{ [<person> <WANT> <infinitive>]
  [infinitive-subject (value 1 cd-form)
  .....] }
```

This specifies that the subject of the clause following *want* is the same as the subject of the phrase.

II.3.4. Extended Notions of Pattern and Concept

Phrasal units involving adverbs, relative clauses, and prepositional phrases are recognized through the use of patterns in the manner just described. However, these forms differ from those presented so far both in how they may appear in an utterance and in their associated semantic function. A slightly modified

notion of a pattern-concept pair is required in order to handle them.

The semantic function associated with the patterns previously encountered is to denote a concept. The semantic function of an adverb or the like is to modify some existing concept rather than simply denote one of its own. The former patterns of speech are therefore often referred to as *concept builders*, and the latter as *concept modifiers*.

When a pattern matches a concept builder, the concept template associated with the pattern is used to create a structure denoting a specific instance of that concept. The resulting concept may match a constituent of another, higher-level pattern, and eventually become part of the conceptual structure that that pattern is responsible for denoting. For example, "home run" is a concept builder that denotes a certain kind of baseball feat. If it is matched in a sentence, a structure is built denoting this idea. If the phrase occurred in the sentence "John hit a home run", the home run structure will match a constituent of a pattern for hit, and thus end up as part of a concept denoting something John did.

Concept modifiers do not work this way. For example, consider the sentence "John ran quickly." If *quickly* just built a structure denoting a faster than normal activity, then the *run* pattern would have to have a constituent that matches it in order for the structure to find its way into the structure that the *run* pattern builds. The problem is that concept modifiers may occur in too many places in too many different phrasal patterns. For example, *quickly* may appear in any position in any phrase of the form

"<person> <transitive verb> ... ":

"John ate quickly", "Quickly, John left", "John quickly spoke up", etc. It would therefore be unwise to require that patterns matching modifiers explicitly appear as optional constituents in all the places in all patterns in which they might appear. This would be unmanageable and would fail to capture a useful generalization.

To handle concept modifiers, therefore, an extended notion of a pattern-concept pair is used. Rather than specify what to build, the concept part associated with a concept modifier pattern specifies where to look for a concept to modify, and how to modify it. The pattern parts are matched just like any other pattern. But once it has been determined that such a pattern has been matched, PHRAN uses the associated concept part to find and change a structure created from some other pattern-concept pair.

II.3.4.1. Adverbs and Adverbial Phrases

Adverbs are generally concept modifiers, and do not as a rule appear as constituents in patterns. Instead, upon recognizing an adverb, PHRAN is instructed to search within the active patterns for an action that it can modify. When such an action is found the concept part of the pair associated with the adverb is used to modify the concept of the original action. An adverb will be used to modify the next appropriate term seen if a modifiable one has not occurred yet. In this manner PHRAN is able to correctly interpret adverbs appearing in a variety of places within a pattern.

For example, the adverbs "slowly" and "quickly" operate in this way, and

PHRAN can handle them in constructs like the following:

John ate slowly.
Quickly, John left the house.
John left the house quickly.
John slowly ate the apple.
John wanted slowly to eat the apple.

Some special cases of negation are handled by specific patterns. For example, the negation of the verb *want* is usually interpreted as meaning "want not" – "Mary didn't want to go to school" means the same thing as "Mary wanted not to go to school". Thus PHRAN contains the specific pattern

[<person> <do> "not" <want> <inf-phrase>]

which is associated with this interpretation.

In general, most of the difficulty in handling adverbs is representational rather than procedural – PHRAN can figure out what the adverb is modifying, but it is often difficult to describe how the appearance of the adverb affects the meaning of the sentence, e. g. in "John walked carefully".

II.3.4.1.1. Adverbs and Their Relation to the Structure of a Phrase

As mentioned above, when an adverb is encountered PHRAN searches within patterns that are active at that time for the action that the adverb modifies. In fact, the exact patterns which are examined are determined by the structure of the patterns being used to understand the utterance. For example, in a sentence like "John thought Bill quickly won the match", PHRAN will try to modify the action associated with the "win" pattern rather that associated with the "think" pattern because the former pattern is the one most immediate

to the phrase containing the adverb.

Thus the internal structure of the patterns of the system imposes a kind of constituent structure on the sentences of the input. Although no effort is expended in producing this constituent structure per se, the presence of structuring in the patterns themselves provides PHRAN with information about how to carry out certain kinds of processes. This structuring takes the place of the grammatical knowledge that is used in other systems for similar functions by generating explicit syntactic parses of the input.

II.3.4.2. Relative Clauses and Prepositional Phrases

When analyzing utterances containing relative clauses, PHRAN must also search for the term which the clause refers to. PHRAN must find the appropriate term preceding the clause, a task which requires consulting a special list of the objects mentioned in the sentence. While reading a sentence, whenever a new object or person is recognized, PHRAN adds it to a list it maintains. Then, when the need arises, PHRAN can quickly determine whether a term referred to actually exists.

For example, when processing the sentence "John saw the man who was walking towards him" PHRAN first processes "John saw the man" and forms a term denoting its meaning. When the relative clause "who was walking towards him" is processed, PHRAN must find the person that the clause refers to. This is done by checking the last term on the above mentioned list to see if it indeed represents a person.

Following is an example of a pattern-concept pair of the previous type. It is suggested when an utterance such as "the man who saw the gun" is encountered. If the required referent is not found when this pattern is suggested, the pattern is immediately discarded.

```
{ [ (*referent immediately-preceding (person))
    who (root see) (physob) ]

  [ p-o-s 'rel-clause
    cd-form '(attend (actor ?actor) (object eyes) (to ?to))
    actor (value 1 cd-form)
    to (value 4 cd-form)] }
```

The previous pattern-concept pair does not exist explicitly in the system. Instead, when the pattern suggesting mechanism recognizes the sequence

<person> who <verb>,

PHRAN generates a pattern-concept pair of this type, from the "basic" patterns associated with the verb.

Prepositional phrases often appear as constituents in larger phrasal patterns, such as the one associated with the verb "to give" that was seen earlier. When prepositional phrases appear and they are not part of another pattern, PHRAN is capable of processing them in a manner similar to that in which it deals with relative clauses.

An example of such a phrase is the following:

```
{  [ (*referent immediately-preceding (physob)) in (container) ]

   [ p-o-s 'prep-phrase
     cd-form '(contains (container ?container) (object ?object))
     container (value 3 cd-form)
     object (value 1 cd-form) ]  }
```

II.3.5. Pattern Manipulation In More Detail

II.3.5.1. Reading A Word

After a new word is read, PHRAN receives a list of relevant patterns from the pattern suggesting mechanism. These patterns include patterns that are used to interpret the word just read – since words in PHRAN are just short patterns and like other patterns they are associated with a concept. PHRAN compares the suggested patterns with the list *PHRAN-BUF* discarding those that conflict with it and retaining the patterns that match the list up to the point PHRAN has read. The partially matched patterns are stored in a *pattern-list* associated with the last term in *PHRAN-BUF*. Now PHRAN uses the associated concepts in the pattern-concept pairs whose pattern consists of the literal word to construct a new term, denoting the word's meaning. If there is more than one associated concept (i. e. more than one possible meaning) PHRAN tries to determine which meaning is appropriate in the current context, using the "active" patterns (those that have matched up to the point where PHRAN has read). It checks if there is a particular meaning that will match the next slot in some pattern or, if no such definition exists, if there is a meaning that might be the beginning of a sequence of terms whose meaning, as determined via a pattern-concept pair, will

satisfy the next slot in one of the active patterns. If this is the case, that meaning of the word is chosen. Otherwise PHRAN defaults to the first of the meanings of the word.

A new term is formed and if it satisfies the next condition in one of the active patterns, the appropriate pattern is moved to the pattern-list of the new term. If the next condition in a pattern indicates that the term specified is optional, then PHRAN checks for the optional terms, and if it is convinced that they are not present, it checks to see if the new term satisfies the condition following the optional ones in the pattern. If, at a later time, there is an indication that the decision that the optional terms were not present may have been erroneous, PHRAN will undo some of its processing and search for them again.

II.3.5.2. A Pattern Is Matched

At some point in this process PHRAN will check an incoming term against the active pattern-list and will find that, not only does the new term satisfy the next condition of a particular pattern, but that the condition is the last one specified. When a pattern has matched completely, PHRAN still continues checking all the other patterns on the pattern-list. When it has finished, PHRAN will take the longest pattern that was matched and will consider the concept of its pattern-concept pair to be the meaning of the sequence. If there are several patterns of the same length that were matched PHRAN will group all their meanings together. New patterns are suggested and a disambiguation process follows, exactly as in the case of a new word being read.

For example, the words "the big apple", when recognized, will have two possible meanings; one being a large fruit, the other being New York City. PHRAN will check the patterns active at that time to determine if one of these two meanings satisfies the next condition in one of the patterns. If so, then that meaning will be chosen. Otherwise 'a large fruit' will be the default, as it happens to be the first in the list of possible meanings.

When a meaning is chosen, PHRAN replaces the terms that matched the pattern with a term denoting this meaning.

II.3.6. Indexing and Pattern Suggestion

Retrieving the phrasal pattern matching a particular utterance from PHRAN's knowledge base is an important problem that has not yet been solved in a completely satisfactory manner. One may find some consolation in the fact that the problem of indexing a large data base is a necessary and familiar problem for all knowledge based systems.

Two pattern suggestion mechanisms were tried with PHRAN:

1) Keying patterns off *individual words* or previously matched patterns.

2) Indexing patterns under ordered *sequences of cues* gotten from the sentence and phrasal patterns recognized in it.

The first indexing mechanism works, but it requires that any pattern used to recognize a phrasal expressions be suggested by some word in it. This is unacceptable because it will cause the pattern to be suggested *whenever* the word it is triggered by is mentioned. The difficulties inherent in such an indexing scheme

can be appreciated by considering which word in the phrase "by and large" should be used to trigger it. Any choice one makes will cause the pattern to be suggested very often in contexts when it is not appropriate. In this form, PHRAN's processing roughly resembles ELI's (Riesbeck, 1975) (Riesbeck and Schank, 1975).

The second mechanism was therefore developed. The patterns-concept pairs of the database are indexed in a tree. As words are read, the pattern suggestion mechanism travels down this tree, choosing branches according to the meanings of the words and of larger units that have been recognized. It suggests to PHRAN the patterns found at the nodes it has arrived at. The list of nodes is remembered, and when the next word is read the routine continues to branch from them, in addition to starting from the root. In practice, the number of nodes in the list is rather small.

For example, whenever a noun-phrase is followed by an active form of some verb, the suggesting routine instructs PHRAN to consider the simple declarative forms of the verb. When a noun-phrase is followed by the verb 'to be' followed by the perfective form of some verb, the routine instructs PHRAN to consider the passive uses of the last verb. The phrasal pattern that will recognize the expression "by and large" is found at the node reached only after seeing those three words consecutively. In this manner this pattern will be suggested only when necessary.

This mechanism is designed so that it also suggests patterns that would match the sentence under a re-interpretation of some of the words already read.

In other words, PHRAN may become aware, at certain points, that a different interpretation of fragments of the sentence analyzed previously will lead to a different meaning being given to a larger segment of the text. It should be possible for PHRAN to decide, when such a situation arises, whether to re-analyze this part of the sentence. However, such a procedure has not yet been implemented.

The main problem with this scheme is that it does not lend itself well to allowing contextual cues not associated with a single word or term to influence the choice of patterns PHRAN should try. This is one area where future research will be concentrated.

II.3.7. A Detailed Example

Following is the detailed PHRAN analysis of the sentence

John gave Mary a piece of his mind.

This example is a fairly typical demonstration of how PHRAN works. However, it illustrates only the more basic features of the program. For example, the example contains no instances of pattern generation, relative clause or prepositional phrase attachment, or handling of complicated noun phrases. In addition, this particular example was chosen to minimize the number of patterns suggested by PHRAN that later end up being rejected.

In this example I will refer to several patterns with symbols P1, P2, ..., and OPT. The correspondence between symbols and pattern-concept pairs is given at the end of the example.

Word Read	Properties of Term Formed	*PHRAN-BUF*	Patterns Suggested	Results of Checking Patterns & Comments
John	T1 (john1,person noun-phrase)	(T1)	none	
gave	T2 (root give, verb)	(T1,T2)	P1,P2,P3	all are consistent with T1,T2.

In the pattern suggestion tree, PHRAN is on a branch that will eventually suggest the correct pattern to interpret the sentence (P8). It will only be suggested after more information is available, however.

Mary	T3 (mary1,person noun-phrase)	(T1,T2,T3)	none	P1,P3 satisfied by T3. P2 is not, and is left on T2's
a	T4 (article, indefinite)	(T1,T2,T3, T4)	P4	P4 consistent with *PHRAN-BUF*. P1,P3 left on T3.
piece	T5 (measure)	(T1,T2,T3 T4,T5)	P5	P4 fails. PHRAN now expects, e. g., "a piece of rock"
of	T6 (of, pre-position)	(T1,T2,T3, T4,T5,T6)	OPT (optional sub-pattern from P5)	OPT satisfied, in part, by T6.
his	T7 (possessive pronoun)	(T1,T2,T3, T4,T5,T6, T7)	P6,P7	both ok on T7.

Up to this point PHRAN believes (i. e. has a main active pattern whose associated concept indicates) that the sentence is something of the form "<person> gave <person> <some object>". During the course of analyzing the input PHRAN's suggestion mechanism has been moving down the nodes of the indexing

tree. A particular pattern will be triggered after reading the next word that will supply PHRAN with the true meaning of the utterance.

mind	T8 (organ, mind, noun)	(T1,T2,T3, T4,T5,T6, T7,T8)	P8	P7,P8 are both matched completely. P8 is longer and so it dominates. All terms matching P8 are removed and replaced by a new term, T9.
---	T9 (sentence, mtrans)	(T9)	none	

The end marker is seen and there are no further patterns to consider, so PHRAN's work is complete. PHRAN now outputs information it has collected about objects mentioned in the sentence:

```
((person (object john1))
 (person (object mary1)))
```

Finally, PHRAN outputs its representation for the complete utterance:

```
(mtrans
   (actor john1)
   (mobject
      (causation
         (antecedent (do (actor mary1)))
         (consequent
            (state-change
               (actor john1)
               (state-name anger)
               (to (level 7)))))
   (from john1)
   (to mary1))
```

The following are patterns and the optional sub-pattern used in the example. The 'concept' parts of the pattern-concept pairs are omitted when they are deemed unnecessary or clear from the context.

Patterns:

```
P1:   { [ (person) (root give) (person) (physob) ]

        [ p-o-s 'sentence
          cd-form '(atrans (actor ?actor) (object ?object)
                           (from ?from) (to ?to))
          actor (value 1 cd-form)
          object (value 4 cd-form)
          from '?actor
          to (value 3 cd-form) ] }

P2:   { [ (person) (root give) (physob)
          ([(to (person))
            (to (opt-val 2 cd-form)]) ]

        [ .... ] }

P3:   { [ (person) (root give) (person) (action) ]

        [ p-o-s 'sentence
          cd-form (value 4 act)        ; this will have slots to be
          actor (value 1 cd-form)      ; filled by actor and
          to (value 3) ] }             ; to.

P4:   { [ a (p-o-s noun) ]

        [ .... ] }

P5:   { [ a piece ([(of (physob))
                    (substance (opt-val 2 cd-form))]) ]

        [ p-o-s 'noun-phrase
          cd-form (newsym piece)
          do (add-to-*supplementary-concepts*
                '(amount-of (substance ?substance)
                            (object @(oldsym piece)))) ] }

P6:   { [ (p-o-s possessive-pronoun) (physob) ]

        [ p-o-s 'noun-phrase
          description (value 2 description)
          cd-form (value 2 cd-form)
          do (add-relation-to-*supplementary-concepts*
                'poss (value 1 person) (value 2 cd-form)) ] }
```

```
P7:   { [ (p-o-s possessive-pronoun) (or (organ) (limb)) ]

        [ p-o-s 'noun-phrase
          description (value 2 description)
          cd-form (value 2 cd-form)
          do (add-relation-to-*supplementary-concepts*
                'part-of (value 1 person) (value 2 cd-form)) ] }

P8:   { [ (person) (root give) (person) a piece of
          (p-o-s possessive-pronoun) mind ]

        [ p-o-s 'sentence
          cd-form '(mtrans (actor ?actor) (mobject ?mobject)
                           (from ?actor) (to ?to))
          actor (value 1 cd-form)
          to (value 3 cd-form)
          mobject '(causation
                      (antecedent (do (actor ?to)))
                      (consequent (state-change
                                    (actor ?actor)
                                    (state-name anger)
                                    (to (level 7))))) ] }
```

Optional Sub-Pattern:

```
OPT:  { [ option              ; in P5
          of (physob) ]

        [ .... ] }
```

II.3.8. Technical Data

PHRAN has been implemented in UCI-LISP on a KL-10 and in FRANZ LISP on a VAX 11/780 at the University of California at Berkeley. Using an interpreted version of PHRAN, sentences of the type described in this chapter take from 0.5 to 4 seconds of CPU time to analyze on the KL10, and the processing uses up 500 - 2500 words from free storage and 20 - 105 words from the free word list. A typical sentence will be analyzed in 2 seconds, using 1100 words of

free storage and 40 from the free word list. Using a compiled version of PHRAN in FRANZ LISP on the VAX, these sentences take roughly the same amount of time, i. e. approximately 0.2 CPU seconds per word for sentences ranging from one word to 25 words long.

In its current state PHRAN knows about 765 "basic" patterns of varying length and abstraction. About 540 of these patterns are individual words. Of these, about 60 are verbs. PHRAN knows both the roots of these verbs, as well as all the morphological variations in which each verb may be found. Of the 220 patterns containing more than one word, about 90 are verb based, i. e., they indicate the way a particular verb is used. This latter group of patterns can be used by the program, when the need arises, to generate an additional 800 patterns, approximately.

Two thirds of all these patterns are currently used by PHRAN as part of the UC system. The others are needed by PHRAN in order to understand texts on other topics.

Adding knowledge of more words to PHRAN will not increase processing time, since words and concepts that are not mentioned in a sentence have no bearing on its processing. What will cause an increase in processing time is an increase in the number of patterns suggested when the same words or concepts are encountered. Such an increase will cause PHRAN to have to consider more patterns and will require more time to be spent checking those patterns against the sentence being analyzed. The amount of additional time spent may not be very significant, however, because most of the irrelevant patterns will be ruled

out by the time only a few more words are read. For example, whether the system knows about 1 or 4 basic patterns associated with "give" makes only a 12% difference in the time spent on analyzing the sentence

John told Bill to give the book to Mary.

Of course, this will make no difference in the time needed to analyze a sentence not containing any mention of "give".

II.3.9. Other versions of PHRAN

PHRAN knowledge bases have also been designed for Spanish and Chinese. Both can be used, without modification, by the same version of PHRAN that analyzes English. The effort of using PHRAN for these languages is described in Wilensky and Morgan (1981), and the Spanish version has been further extended by Paul Jacobs.

A version of PHRAN was implemented by Fred Mueller using the INGRES data base system. INGRES is a data base management system based on the relational model, and was developed by Held, Stonebraker, and Wong (1975). It runs on top of the UNIX operating system on a VAX 11/780.

Since a great deal of effort was put into separating PHRAN's knowledge of language from its procedural knowledge, it is possible to move PHRAN's entire pattern-concept pair knowledge base out of LISP and into the relational structures provided by INGRES. Accessing the appropriate patterns then becomes a matter of making a data base query to INGRES. Some of PHRAN's processing have been changed somewhat to accommodate and take advantage of INGRES'

searching and querying facilities. For example, the INGRES version tends to suggests more patterns at fewer points than does the standard LISP implementation.

Preliminary benchmarks suggest that, while the INGRES version is far slower than the LISP version on a small knowledge base, it is more efficient when the data base reaches several thousand entries in size. Thus there would appear to be some possibility of making use of relational data base technology as the volume of knowledge contained in a natural language interface becomes significant.

II.3.10. A Comment about Asymmetries and the Shared Knowledge Base

One argument that could be advanced against the psychological validity of a common knowledge base for the understanding and production mechanisms is that there are observable asymmetries between what people produce and what they are capable of understanding. For example, a person may be able to understand an utterance using a certain word or phrase, although he would never use that phrase himself.

One explanation for this asymmerty is that learning, i. e., analysis, is necessary before production is possible. However, other natural ways to account for such asymmetries exist within the propopsed model. The simplest involves indexing. This model suggests that the understanding and production mechanisms use different indexing schemes: The understander must be able to react to any phrase it hears, as it has no control over its input. In contrast, the language generator

need only have a way of producing some utterance for each idea it wishes to express; if there is more than one phrasal pattern that can be used to express this idea, only one of them needs to be accessible to the producer for it to function adequately. For example, the generator need only index the "<person> kill <person>" phrasal-concept pair in order to have a way of expressing that someone died. However, the understander must index both this pattern and "<person> kick the bucket", as it has no way of knowing which it will hear.

This predicts that a person should able to understand (or at least recognize) what he produces, but that he may be able to understand many utterances he would not produce. This is precisely what is observed.

This kind of asymmetry in task requirements carries over into other aspects of production and understanding. For example, in the case of concept modifiers, PHRAN must often search around to find an appropriate concept to modify, whereas the analogous generation task need only find a place in a phrase to insert a modifier. Thus the generator need not have all the flexibility of placement required by the understander. This would be consistent with a finding that individuals tend to insert modifiers in fewer places than they are capable of understanding them in.

II.4. Comparison to other Systems

Much work has been done in the area of natural language understanding and the design of natural language front ends, and I will not attempt to survey all of it here. Rather, I will compare PHRAN primarily to other systems that use some

form of a "patterns of speech" approach to language processing.

One of the earliest such systems is Colby's PARRY. PARRY is a simulation of a paranoid mental patient that contains a natural language front end (Parkinson *et al*, 1977). It receives a sentence as input and analyzes it in several separate "stages". PARRY makes several passes over the input in which it identifies and brackets off noun phrases, replaces idiomatic phrases with simpler one word expressions, replaces verbs used in the passive with active ones In some cases PARRY even rearranges the sentence using the same, or sometimes another, verb, and performs other tasks as well.

In effect, PARRY replaces the input with sentences of successively simpler form. In the simplified sentence PARRY searches for patterns, of which there are two basic types: patterns used to interpret the whole sentence, and those used only to interpret parts of it (relative clauses, for example). In matching these patterns, PARRY is allowed to ignore unrecognized words.

For PARRY, the purpose of the natural language analyzer is only to translate the input into a simplified form that a model of a paranoid person may use to determine an appropriate response. No attempt is made to model the analyzer itself after a human language user, as is being done here, nor are claims made to this effect. A system attempting to model human language analysis could not permit several unrelated passes, the use of a transition network grammar to interpret only certain sub-strings in the input, or a rule permitting it to simply ignore parts of the input.

This additional theoretical shortcoming of PARRY – having separate gram-

mar rules for the complete sentence and for sub-parts of it – is shared by Hendrix's LIFER (Hendrix, 1977). LIFER is designed to enable a database to be queried using a subset of the English language. This subset can be enlarged rather easily by the user, who can type in new patterns and instruct the system on how to interpret them.

As is the case for PARRY, the natural language analysis done by LIFER is not meant to model humans. Rather, its function is to translate the input into instructions and produce a reply as efficiently as possible. Thus nothing resembling a *representation* of the meaning of the input is ever formed. Not wishing to breathe new life into the procedural-declarative controversy, I will just note that the whole view of language understanding underlying this thesis takes it for granted that a declarative representation of meaning is necessary for the inference processes that begin once natural language analysis is completed.

Of course, the purpose of LIFER is not to be the front end of a system that understands coherent texts, and which must therefore perform subsequent inference processes. While LIFER provides a workable solution to the natural language problem in a limited context, many general problems of language analysis are not addressed in that context.

SOPHIE (Burton, 1976) was designed to assist students in learning about simple electronic circuits. It can conduct a dialogue with the user in a restricted subset of the English language, and it uses knowledge about patterns of speech to interpret the input. SOPHIE accepts only certain questions and instructions concerning a few tasks. As is the case with LIFER, the language utterances accept-

able to the system are restricted to such an extent that many natural language processing problems need not be dealt with and other problems have solutions appropriate only to this context.

SOPHIE also does not produce any representation of the meaning of the input. For example, the fragment "the voltage at the collector of Q5" is always interpreted as a call to a function that measures the appropriate voltage. In addition, SOPHIE makes more than one pass on the input and ignores unknown words, practices that have already been criticized.

It should be noted here that the notion of a *semantic grammar* which underlies SOPHIE, is shared by PHRAN too. PHRAN builds on the basic idea of a semantic grammar, attempting to provide the answers and improvements necessary to allow it to cover as wide a range of linguistic structures as PHRAN does. In the "grammar" of PHRAN, as explained earlier, information about the language is only half of what is provided. The fundamental element in it is the pairing of such information with a conceptual representation of the meaning of the phrasal language construct.

The augmented finite state transition network (ATN) has been used by a number of researchers to aid in the analysis of natural language sentences (for example, see Woods 1970). The ATN itself is a very general formalism, and in recent years has been extended to include enough alternative control structures that it may be considered more of a programming language than a theory of language understanding (e.g., see Waltz *et al* (1976)).

However, most systems that use ATNs incorporate one feature which is

objectionable on both theoretical and practical grounds. This is the separation of analysis into syntactic and semantic phrases. Natural language analyzing programs utilizing ATNs parse an input according to given grammar rules. The only role semantics plays is in the classification of grammatical nodes. Later on, special semantic routines are used to analyze the content of the parse created by the ATN. It would seem unlikely that such a separation occurs in human understanders; and there is no compelling evidence to justify such a separation.

Of course, most ATN advocates do not make claims about the psychological validity of their model. As to the efficacy of the separation of syntactic and semantic processing, this has been argued at length elsewhere (see Schank 1975 for example). Let it suffice to say that using all forms of knowledge together would seem to have a better chance of constraining the entire process than would using these forms of knowledge in separate phases. In addition, most ATN based systems (for example Woods' LUNAR program) do not produce representations, but rather, query a data base.

None of the previously mentioned systems contains a component that generates English sentences from a representation of a concept. The responses they do give are either completely canned or else arrived at by filling slots in some fixed pattern. While these systems were not designed with this task in mind, it would seem that a reasonable model of human natural language understanding should in principle be capable of sharing most of its knowledge about the meaning of the utterances of its language with a language generation mechanism. However, it would not seem possible to use most of the knowledge in the data

bases of the analysis programs described above for the generation task. This is particularly true in systems espousing some form of procedural semantics – if the meaning of the utterance is thought to be the execution of some routine, then it is hard to see how semantic knowledge that results in a process being executed can be used to produce a natural language utterance.

In contrast to the systems just described, Wilks' English-French machine translator (Wilks, 1973) (Wilks, 1975a) (Wilks, 1975b) is free of several of the shortcomings listed above. It produces a representation of the meaning of an utterance, and it attempts to deal with unrestricted natural language. The main difference between Wilks' system and PHRAN is that Wilks' patterns are matched against concepts mentioned in a sentence. To recognize these concepts he attaches representations to words in a dictionary.

The problem is that this presupposes that there is a simple correspondence between the form of a concept and the form of a language utterance. However, it is the fact that this correspondence is not simple that leads to the difficulties being addressed in this chapter.

For example, Wilks' systems is forced to make an initial pass using "fragmentation functions" which separate the sentence into segments using key words (and sometimes rearrange the segments) so that the concepts in the sentence may be identified. This solution would appear to be entirely unsatisfactory for processing non-productive language forms in which meanings cannot be closely associated with individual words. In general, since the correspondence of words to meanings is complex, it would appear that a program like Wilks' translator will

eventually need the kind of knowledge embodied in PHRAN to complete its analysis.

One recent attempt at natural language analysis that radically departs from pattern-based approaches is Rieger and Small's system (Small, 1978) (Small, 1980). This system uses *word experts* rather than patterns as its basic mechanism. The idea of putting as much information as possible under individual words is about as far from PHRAN's conception of language analysis as one can get, and I would argue, would exemplify all the problems just described in word-based systems.

However, there are actually a number of important similarities between the two systems. First, Rieger and Small seem to combine syntactic and semantic knowledge in their experts. Since more traditional pattern-based systems are generally pattern-based insofar as syntactic analysis is concerned, the word expert idea is closer to PHRAN's approach in this respect. In addition, Rieger and Small's system acknowledges the fact that a natural language analyzer must contain a great deal of knowledge about its language, and that representing and organizing this knowledge is a major problem. Thus while I do not feel that the use of word experts is the best way to resolve this problem, it at least seems to address the crucial issue.

A special mention should be given to the conceptual analyzer developed by Riesbeck (1974), which maps English language utterances into Conceptual Dependency meaning structures. The best known version of this system is ELI (English Language Interpreter) (Riesbeck, 1975), which was further extended to include

additional language constructs by Gershman (1977). Many of the ideas present in PHRAN were developed while investigating ELI and attempting to correct some of its shortcomings.

Riesbeck's system works by attaching routines to individual words. These routines are responsible for building pieces of a meaning representation for the utterance being analyzed. When a word is read by the system, the routines associated with this word are used to build up a meaning structure that eventually denotes the entire utterance.

While being similar to PHRAN in producing declarative meaning representations and not having separate syntactic and semantic parsing phases, there are major differences between the two approaches. The routines attached to words in ELI are not usable for the process of comprehension, and all knowledge about the language would have to be duplicated for a production mechanism. In addition, ELI suffers from all the problems detailed earlier which stem from the assumption that all knowledge of the language can be stored at the single word level.

CHAPTER III

INTRODUCTION TO THE CONTEXT MODELER

III.1. Introduction

This chapter presents some of the basic concepts underlying the Context Modeler component of CLUSTER. It begins with a description of the memory structures used by it, and continues with an overview of the operations performed by the Context Modeler. The description will be illustrated with the detailed example at the end of the chapter. Chapter IV contains an in-depth description of the Context Modeler and its implementation.

III.2. Memory Structures

In CLUSTER, memory to which the Context Modeler has access is built of two parts:

- **Long Term Memory**, a permanent long term store of knowledge about the world.
- The **Context Model**, a model of the changing context in which the understanding process is taking place.

The long term memory is a permanent collection of static information about the environment within which the system is operating. This includes facts about the situations, objects, rules of behavior and discourse, and various associations.

These together constitute the total knowledge the system has of itself, the world and other actors in it; they establish the system's relation to the world and ability to modify it.

The Context Model is a structure representing the world as it is viewed at a certain point in a conversation. It is a representation of the system's awareness of particular aspects of the surrounding environment. At any given time only a few of the large number of things known about the world are actually being considered. And of those, some are more central than others to the discussion. The Context Model is constructed and modified in the course of a conversation to reflect these facts.

For example, upon receiving as input the sentence

I want to delete the file

a system should not consider all files it has ever encountered as candidates for the interpretation of the words "the file". The Context Model approach presents a systematic way to define the limits of the appropriate context. In cases such as the above, referents will be searched for only from among the files currently available within the current Context Model.

To facilitate such efficient memory management, Long Term Memory is made up of a collection of **clusters**. A cluster is a fragment of a network. In this network each node represents a fact about the world. The term **entry** refers to a node in a cluster. An arc connecting two nodes represents the strength of the inference that can be made from the presence of one node in a context to the presence of the other. These network fragments are assembled in the Context

Model during a conversation, in a manner explained below.

A collection of entries is arranged in a cluster if those entries tend to be found together in the real world. Following are several types of relationships that may exist among the constituents of a cluster:

- A complex action and its several steps.

  E. g., logging on, which includes typing the user's name, and typing the password.

- A complex object and its various parts.

  E. g., a terminal, which includes a screen, a keyboard, etc.

- An activity and its associated actors and objects, or an object and its intended, or associated, use.

  E. g., compiling, with its associated compiler and the file to be processed, as well as the resulting file of compiled code; or a particular file and the operation of the program stored in it.

- Standard associations between certain stimuli and responses.

  E. g., the association between being asked a question and answering it.

- Conventionalized relationship between actions.

  E. g., a cluster relating an expression of need for help to a request for help, or one relating a failure in the performance of an action to an attempt to determine the cause of the failure.

- Repeated simultaneous occurrences of concepts for reasons unknown to the system, i. e., for reasons not represented in the cluster.

For example, the execution of a certain program with input from a certain file may be associated with a core dump, without there being any additional information about the reasons for this failure.

The only difference between the last type of clusters and the previous ones is that clusters of the former types contain entries describing the relationship between their other entries. If such "explanatory" entries are missing from a cluster, then the system will have no information about the reason for such relationships. Nevertheless, the entire cluster will still be loaded into the Context Model and become available to the understanding system upon the mentioning of one of the entries, as we see below.

As an illustration, a cluster relating the making of a file executable to the command that must be issued in order to do so will include the following entries*:

?F is a file
?U issues command 'chmod' with argument '755'
?F changes its 'executability' state to 'yes'

The existence of this cluster in LTM asserts that the last two facts mentioned above are related, i. e., issuing the command 'chmod 755 <filename>' is associated with making a file executable, and vice versa.

Generally speaking, the Context Model is constructed by loading into it all those clusters found in long term memory which contain an entry matching a newly encountered input. Thus, mentioning any one of these two facts in a

* Entries are presented in a simplified notation. ?F and ?U are variables.

conversation causes the second to become present in the Context Model, and thus in focus.

For example, when the user types

The file init.l

a cluster describing the file init.l and its attributes will be loaded into the Context Model. Depending on what precisely is known about the file, the cluster may look something like the following:

{ **init.l** is a file

init.l has protection 644

init.l is owned by user 'arens'

init.l has name 'init.l'

init.l has been sent to the line printer }

When a cluster becomes part of the Context Model, the system immediately becomes capable of accessing all the other information represented in it. In this case, the fact that this particular file has been sent to the line printer is now accessible. This piece of information might prove useful in understanding future input concerning the file init.l.

In general, clusters may be formed because of known semantic relationships among their entries, arbitrary co-occurrences, or a combination of both. In this example, it is only a coincidence that the file in question was sent to the line printer. On the other hand, every file has some protection, a name and an owner. All these properties of init.l are reflected in the cluster.

Further discussion of the advantages of this way of structuring memory is provided in section IV.1.

III.3. Entries

In the Context Model, knowledge about the world is built up of units called **entries**. Each entry represents a unit of information, and has a certain level of activation. Entries typically represent a piece of information concerning some property of a known object, e. g., its size or contents, or some state known to exists, e. g., the fact that a user is logged on to the system. The choice of which actions and objects are to be considered atomic, and hence represented by single entries in the UNIX Consultant, has been done on an *ad hoc* basis. Decisions were made based on the semantics of the domain of expertise of the system.

The structure of entries is explained in section IV.2.

Entries are divided into three classes: *Objects*, *Assertions*, and *Intentions*.

The first class is composed mainly of entries representing objects known to exist in the world.

The second class of entries, *assertions*, consists of entries that represent states of the world and statements of facts about the world that are known to the system.

An entry of class *assertion* may be of one of several subtypes. These describe assertions, requests, failures and other events the user or the system may experience or initiate.

The third class of entries, *intentions*, comprises entries representing intentions that the system has of taking certain actions. The action associated with such an entry will be performed if and when the activation level of the entry is sufficiently high.

An entry of class *intention* may also be of several subtypes. These include intentions to output utterances to the user, to find a referent for a particular expression, intentions to find a plan to achieve some goal, and others.

A full description of entry classes and types is provided in section IV.2.1.

III.4. Processing and Activation

Long term memory is composed of a collection of clusters, which act as network fragments. Upon processing input, appropriate clusters are fetched from long term memory and instantiated. They are then assembled in the Context Model to form a memory network of nodes (i. e., the entries) which is used when the understanding of further input requires information concerning the context. As time passes and new input is processed, this network is modified by the addition of new fragments and the deletion of old, unused, ones. Spreading activation and decay are used to control the evolution of the network.

A Context Model consists, therefore, of a subcollection of long term memory clusters, which are linked by the identification of an entry (or entries) in one cluster with an entry (or entries) in another. In the process of this identification, unification is performed and variables in the clusters may be assigned certain values or identified with others. Such values propagate to the rest of the cluster

to which the entry belongs.

The Context Modeler operates as follows: When the language understanding system processes a new input, semantically meaningful language fragments in it are found. If an entry matching the meaning of the new language fragment is not already found in the Context Model, long term memory is searched for clusters that contain entries structurally similar to the meaning representation of the language fragment. These clusters are recalled and the entry representing the new input is unified with the one similar to it in every cluster. If in the course of unification variables become bound, those bindings are propagated to other instances of that variable in the cluster. The instantiated clusters are attached into the network in the Context Model by unifying entries in the existing network with entries in the new clusters where possible. Each new unified entry is assigned alevel of activation.

If the same language fragment is encountered again, its representation will be found in the Context Model. Clusters containing it will not be recalled from LTM a second time. Instead, the activation of the existing entry will be increased, and this increase will spread to other entries linked to it, with the increase declining as the 'distance' from the original entry increases.

The activation levels of all entries in the Context Model are assumed to decay gradually and in parallel as time passes. When an entry's activation decays below a certain threshold, it is deleted from the network in the Context Model and all links to it are severed.

III.5. The Monitor

The **Monitor** is a demon which is awakened when a new entry is introduced into the Context Model. It makes sure construction of the Context Model takes place as needed. The Monitor has several functions:

1. Retrieve appropriate clusters from LTM, if necessary.
2. Attach new clusters to those in the Context Model at matching entries.
3. Increase activation of appropriate existing entries.
4. Perform the actions appropriate for highly enough activated entries of class *intention* .

Entries of the *intention* class always have actions associated with them that the Monitor executes when the activation of the entry rises to a high enough level. The actions may involve requesting information from other systems, e. g., from a planner working in conjunction with the language understanding system. In the case of UC, for example, the Monitor may request information from UNIX. Actions may also involve the deletion of entries from the Context Model, or the addition of entries. In the latter case the whole process described in this section is performed again for every new entry inserted.

The Monitor behaves as follows:

1. Awake upon receiving entry e.
2. Insert e in the Context Model.
3. Inspect Context Model, perform appropriate actions.
4. Decrease activation of all entries.
5. If actions in (3) caused change in Context Model, go back to (3).
6. Quit.

III.6. Detailed Example

The operation of the Context Modeler is illustrated below with a detailed example of the processing of the following user's statement:

"I can't copy the file com1 to the file com2"

In order to provide an appropriate response, UC must first realize that there is a request for help underlying this statement; the user probably wishes that the system aid in overcoming some obstacle. UC must then determine what the user was trying to do and why this did not succeed. The system must then decide to communicate this information to the user.

The processing of this sentence eventually results in UC explaining to the user what the cause of the problem is. It should be noted that little problem solving in the traditional sense takes place here. This is not a program specifically intended to solve problems. The solution is provided entirely by relying on remembered information concerning the process of copying files.

In order to increase the clarity of the trace, PHRAN's trace output is not included in it. The '&'s appearing in some of the entries are symbols the pretty-printer inserts when the level of nesting is above 6. This is often the case, since entries routinely contain cycle-forming pointers to other members of their clusters.

I have added lines of explanation to the output of the program. These paragraphs of comments are indented and preceded by a '•'.

```
# I can't copy the file com1 to the file com2
```

- To UC's prompt, '#', the user typed the above sentence. Notice that there is no explicit question posed or request made by the user. The user is simply making a statement. As we will see below, there exist clusters in long term memory associating the assertion of failure of the user at some task with an intention on the part of the system to solve the problem.
- As the input sentence is read, PHRAN recognizes first the word, "I", and then the fragments "the file com1" and "the file com2". A representation for the complete sentence is produced later.

```
Inserting in *CONTEXT-MODEL*:        (base level -- 100)
(refdef (activation 100)
        (meaning ((p-o-s noun-phrase
                   cd-form *ego*
                   description (person)))))

No matching entry.
```

- An entry of type *refdef* is created to represent the user. This entry is of the category of *intentions* (as distinct from *objects* and *assertions*). Its name reflects the fact that it describes a definite reference to an object probably known to the system. When this entry is inserted in the Context Model, the Monitor searches the context for a matching entry which may already be present. Such an entry is not found.

```
Monitor -- inspecting context-model

Warning: No good referent for
(refdef (activation 100)
        (meaning ((p-o-s noun-phrase
                   cd-form *ego*
                   description (person)))))
```

- Since there is no more activity in the Context Model, the Monitor looks over the contents. An entry of type *refdef* (the one just dealt with) is found to be highly enough activated to cause the Monitor to perform its associated action. (The various entries of class *intention* and their associated actions will be described in full in the next chapter.) In the case of a 'refdef' the action is to find another entry in CM that this one is a reference to, if possible, and to create an appropriate entry otherwise. This is done by first

comparing this entry to all others of type *object* and ranking them according to how closely they match. Ranking is done on the basis of the objects' 'description' property and other known characteristics of the entries, such as their syntactic properties and adjectival modifiers used in the language describing them. The top ranked entry is chosen, if it is a close enough match. In this case, no match is found and a warning is printed. A new entry of type *mention* is created. It replaces the original 'refdef' entry. Its presence asserts the existence of the object it describes. It is now inserted.

- Control now passes from the Monitor back to PHRAN, which analyzes the sentence further. Having processed the words "the file com1", PHRAN hands CM an entry representing the file com1, another one of the type *mention*.

```
Inserting in *CONTEXT-MODEL*:        (base level -- 100)
(mention (activation 100)
         (cd com1)
         (meaning ((cd-form com1
                    description (file container physob)
                    p-o-s noun-phrase))))
```

No matching entry.

Monitor -- inspecting context-model

- Again, no matching entry is found. Once this entry is inserted, the Monitor again inspects CM. There is nothing there requiring action so the Monitor returns control to PHRAN, which analyzes the sentence further and delivers to the Context Model an entry representing the file com2.

```
Inserting in *CONTEXT-MODEL*:        (base level -- 100)
(mention (activation 100)
         (cd com2)
         (meaning ((cd-form com2
                    description (file container physob)
                    p-o-s noun-phrase))))
```

No matching entry.

Monitor -- inspecting context-model

- Finally the end of the input sentence is reached, and an entry representing its complete interpretation by PHRAN is handed to CM.

```
Inserting in *CONTEXT-MODEL*:        (base level -- 100)
(assertion (activation 100)
          (cd (not
                (concept
                  (able (actor *ego*)
                        (perform (duplicate (oldfile com1)
                                            (newfile com2))))))))

No matching entry.
```

- There is no other entry in the Context Model matching this one. However, there is a cluster in long term memory that contains an entry matching it*. (The cluster appears in the segment of the trace following this comment.) This cluster is put in a queue of clusters to be inserted, and its entries will be inserted into CM when this cluster's turn in the queue is reached. In this particular case this happens immediately, since this cluster is the only one in the queue. The queue's purpose is to allow the Context Model to emulate parallel insertion of entries.
- The new cluster found in LTM represents the association of the user's failure to perform an action with the system's intention to find its own plan for performing that action. The existence of this cluster causes UC to identify a request for help on the part of the user, the user's failure to perform a task, and an intention on the system's part to perform the user's intended task. In the current example, this is UC's main driving force. It might also be viewed as an encoding of the fact that a statement of failure is a speech act (Austin, 1962) indicating a request for help. The cluster below is more than that, however, since it not only associates a user's request for help with the statement, but also an intention on the part of the system to help.
- Unification was done during the search for this cluster, and so the details of the particular task the user was attempting to perform have already been made explicit in it. The entry responsible for the retrieval of this cluster, i. e., the one stating the failure to copy the files, is not listed again. To save space, throughout this example clusters will be listed without repeating the entry which caused their recall from LTM.

* In fact, the cluster in LTM is *indexed* under the matching entry – a distinction that is explained in the next chapter.

```
Adding associated clusters.          (base level -- 100)
(((out-planfor (activation 100)
               (cd (duplicate (oldfile com1) (newfile com2))))
  (question (activation 100)
            (cd (planfor
                  (concept
                     (duplicate (oldfile com1) (newfile com2)))
                  (is *unknown*))))
  (fail (activation 100)
        (cd (planfor
              (result (duplicate (oldfile com1) (newfile com2)))
              (method ?plan))))
  (mention (activation 50)
           (cd (planfor
                 (result (duplicate (oldfile com1) (newfile com2)))
                 (method ?plan))))
  (output (activation 100) (text ?any))))

Inserting in *CONTEXT-MODEL*:        (base level -- 100)
(out-planfor (activation 100)
             (cd (duplicate (oldfile com1) (newfile com2))))

No matching entry.
```

- The first member entry of the (single) cluster associated with the user's assertion of failure is inserted into the Context Model. This is another entry of class *intention*. More specifically, it is of type *out-planfor*, which the Monitor recognizes as indicating that it is an intention of the system to find a plan for the task the user intended to perform.

  (The details of how to execute this intention are found in the specification of the action that the Monitor performs when it notices that an entry of this type is highly enough activated. It is at that point that the Context Model will be searched for knowledge of a plan to pursue. This will be elaborated on further when it actually takes place below. At this point, the new entry is simply inserted.)

Inserting in *CONTEXT-MODEL*: (base level -- 100)

```
(question (activation 100)
        (cd (planfor
              (concept
                 (duplicate (oldfile com1) (newfile com2)))
              (is *unknown*))))
```

No matching entry.

- Another member of the cluster is added. This one is of the type *question*, another member of the *assertion* class of entries. Its presence in the cluster reflects the system's understanding that a question by the user is associated with the assertion of failure, i. e., that an assertion by the user that he/she has failed at a task may be a request for help in performing that task. Its presence in the cluster also signifies, conversely, that a request for help in performing a task is indicative of a failure in executing it.
- The question the system considers as having been asked may be paraphrased as "how do I copy the file com1 to the file com2?".
- Again, there is no existing entry matching this one, but two clusters are found to contain it in long term memory. They are retrieved and printed by the trace next.

Adding associated clusters. (base level -- 50)

```
(((output (activation 100) (text ?any)))
 ((out-planfor (activation 100) (cd ?conc))))
```

- The first of these clusters associates with having been asked a question an intention to *output*, that is, to generate some statement in English. The second cluster represents the association between the question above, requesting a way to perform some action, and UC's intention to find such a plan.
- The first cluster represents an association between being asked a question and intending to reply – an association that could be considered a conversational convention.
- In both cases, as with the cluster discussed earlier, only the entries other than the one under which the cluster was indexed are shown. These clusters are added to the queue and their entries will be inserted into the Context Model shortly.

```
Inserting in *CONTEXT-MODEL*:        (base level -- 100)
(fail (activation 100)
      (plan
        (planfor
          (result (duplicate (oldfile com1) (newfile com2)))
          (method ?plan))))

No matching entry.
```

- This entry is of class *assertion*, describing the failure of the user's plan (the identity of which is not yet known) for copying the file. It is originally from the cluster first introduced on page 88. This entry, in turn, has yet another cluster containing it in long term memory, displayed below.

```
Adding associated clusters.        (base level -- 100)
(((check-preconds (activation 100)
                  (plan
                    (planfor (result
                               (duplicate (oldfile com1)
                                          (newfile com2)))
                             (method ?plan))))
  (output (activation 100) (text ?any))))
```

- This cluster represents the association between a failure of some plan and an *intention* to *check-preconditions* of the plan in question, and also to *output* something. This cluster is added to the cluster queue and will be dealt with in its turn.

```
Inserting in *CONTEXT-MODEL*:        (base level -- 100)
(mention (activation 50)
         (cd (planfor
               (result (duplicate (oldfile com1)
                                  (newfile com2)))
               (method ?plan))))

No matching entry.
```

- This is an entry of class *object*. It is from the cluster first introduced on page 88. Its presence in the context reflects the fact that a particular object is considered to exist. In this case, the object is a plan for the duplication of the file com1. The existence of this entry will permit the understanding of a future reference, if any, to such a plan.

```
Adding associated clusters.        (base level -- 50)
(((preconds (activation 100)
            (plan
              (planfor
                (result
                   (duplicate (oldfile com1) (newfile com2)))
                (method
                    (use-command
                       (command (cp (file1 com1) (file2 com2)))))))
          (conds (own (actor *ego*) (object com2))))))
```

- The plan for copying a file has a cluster indexed by it. This cluster associates the copying of a file with known preconditions for it. The *preconds* entry is of class *assertion* . This cluster is also added to the cluster queue.
- As the trace indicates, in the process of retrieving the cluster above the particular plan used for copying a file has been determined. This is reflected in the filler of the "method" slot in the *preconds* entry above. Consequently, the *mention* entry listed immediately before this one – the cause of the recall of this cluster – now becomes fully specified, including the method used.

```
Inserting in *CONTEXT-MODEL*:      (base level -- 100)
(output (activation 100) (text ?any))

No matching entry.
```

- The last of the entries in the first cluster, the one associated with PHRAN's analysis of the input, is inserted in the Context Model. It represents the *intention* to output something. Next, another *output* entry is inserted.

```
Inserting in *CONTEXT-MODEL*:          (base level -- 50)
(output (activation 100) (text ?any))

Matching entry found.
(output (activation 100)
        (text ?any)
        (parents (activation 100)
                 (cd (not (concept &)))
                 (clusters
                   ((out-planfor & & &)
                    (question & & & &)
                    (fail & & & &)
                    (mention & & & &)
                    (output & & &))))))

Reinforcing from it
```

- This entry is the first one from the second cluster in the queue, the one associated with the *question* entry in the first cluster. This time a matching assertion *is* found; it is a member of the first cluster retrieved, the one displayed on page 88. The structure of this entry appears complicated because of all the pointers it contains to the other members of its cluster, links that are used for the passing of activation. Because of the match, instead of a search for additional clusters in long term memory, reinforcement of activation levels takes place. It begins with this matched entry and activation spreads to other entries linked to it, the increase becoming progressively smaller as the "distance" from the original matching entry increases.

Inserting in *CONTEXT-MODEL*: (base level -- 50)
(out-planfor (activation 100) (cd ?conc))

Matching entry found.
(out-planfor (activation 100)
(cd (duplicate (oldfile com1) (newfile com2)))
(parents
(assertion (activation 100)
(cd (not (concept &)))
(clusters
((out-planfor & & &)
(question & & & &)
(fail & & & &)
(mention & & & &)
(output & & &))))))

Reinforcing from it

- Now the third cluster in the queue is dealt with. Its single entry (in addition to the one it was indexed under) is inserted and also found to match an existing one (the first member of the cluster on page 88.)

Inserting in *CONTEXT-MODEL*: (base level -- 100)
(check-preconds (activation 100)
(plan
(planfor (result (duplicate (oldfile com1)
(newfile com2)))
(method ?plan))))

No matching entry.

- The preceding entry is the first in the cluster associated with the *fail* entry. Next is the second entry in that cluster, which will match an existing one in CM.

```
Inserting in *CONTEXT-MODEL*:        (base level -- 100)
(output (activation 100) (text ?any))

Matching entry found.
(output (activation 100)
        (text ?any)
        (parents
           (question (activation 100)
                     (cd (planfor (concept &) (is *unknown*)))
                     (parents
                        (assertion (activation 100)
                                   (cd &)
                                   (clusters &)))
                     (clusters
                       ((output & & &)) ((out-planfor & & &))))
           (assertion (activation 100)
                     (cd (not (concept &)))
                     (clusters ((out-planfor & & &)
                                (question & & & &)
                                (fail & & & &)
                                (mention & & & &)
                                (output & & &))))))

Reinforcing from it
```

- Next, the *preconds* entry in the cluster associated with the *mention* of the plan for copying is inserted.

```
Inserting in *CONTEXT-MODEL*:        (base level -- 50)
(preconds (activation 100)
          (plan (planfor (result (duplicate (oldfile com1)
                                            (newfile com2)))
                         (method
                            (use-command
                               (command (cp (file1 com1)
                                            (file2 com2)))))))
          (conds (own (actor *ego*) (object com2))))

No matching entry.
```

- The above entry is the last one in the last cluster that was present in the queue. Now that the queue is empty and activity in the Context Model has ceased, the Monitor will look over the entries present to see if any *intentions* that require action on its part are highly enough activated.

- In doing so, the Monitor first finds the *out-planfor* entry, which has the action of searching the Context Model for an appropriate plan (in this case, a plan for copying files) associated with it. The Monitor finds the *mention* of such a plan. If that entry had not been present, the Monitor would have sent a request to a planner working in conjunction with the UNIX Consultant for a method for copying.
- The next entry requiring action on the part of the Monitor is the *check-preconds* entry, which is an *intention* class entry with the associated action of testing the validity of any known preconditions. To discover which preconditions are present in the environment, the Context Model is inspected. One such condition is found in the entry stating that the copying of one file to another requires ownership of the second. This is checked and found not to hold. As a result, an entry of type *fail-precond*, another entry of class *assertion*, describing this fact is inserted in the Context Model.

Monitor -- inspecting context-model

Inserting in *CONTEXT-MODEL*: (base level -- 100)
(fail-precond (activation 100)
(cond (own (actor *ego*) (object com2))))

No matching entry.

Adding associated clusters. (base level -- 100)
(((output (activation 100)
(text |You do not own the file com2|)))))

- The *fail-precond* entry is inserted and no matching one is found. A cluster is fetched with it as an index: one associating a certain output with the knowledge that this precondition has failed. In this version of UC the output is canned. In more recent versions this is no longer the case and the output is generated from a representation of the concept UC intends to express (Wilensky, Arens, and Chin, 1984).

Inserting in *CONTEXT-MODEL*: (base level -- 100)
(output (activation 100)
(text |You do not own the file com2|))

- The new entry is inserted and is found to match one present in the context already. Due to the Monitor having already gone through several cycles, activation has started to decay and smaller activation levels than were originally given to the entries may now be observed.

```
Matching entry found.
(output (activation 100)
        (text |You do not own the file com2|)
        (parents
           (fail (activation 98)
                 (plan (planfor (result &) (method &)))
                 (parents
                    (assertion (activation 98) (cd &) (clusters &)))
                 (clusters
                   ((check-preconds & & & &) (output & & & &))))
           (question (activation 98)
                     (cd (planfor (concept &) (is *unknown*)))
                     (parents
                        (assertion (activation 98) (cd &) (clusters &)))
                     (clusters
                       ((output & & & &))
                       ((out-planfor & & & &))))
           (assertion (activation 98)
                      (cd (not (concept &)))
                      (clusters
                        ((out-planfor & & & &)
                         (question & & & &)
                         (fail & & & &)
                         (mention & & & &)
                         (output & & & &)))))
        (done no))

Reinforcing from it

Monitor -- inspecting context-model
```

- Activity in the Context Model has now abated, and the Monitor inspects it again to see if there is anything requiring action. Such is again the case, this time it is the *output* entry. Since it is highly enough activated and contains some text, the text is actually output to the user. The following is printed:

You do not own the file com2

- The action associated with the *output* entry also involves modifying it to reflect the fact that the text specified in it has been printed. The slot "done" is modified to have the value "yes", thus preventing the Monitor from executing the output action a second time. (The entry in its final form is not displayed).

- The Monitor inspects the Context Model once again, but there is no further action required, so it returns control to the language analyzer. There is no further text to be analyzed.

Processing the above sentence takes 16.5 seconds of CPU time in verbose mode, and 8.5 CPU seconds without the trace.

CHAPTER IV

THE CONTEXT MODELER

In this chapter I discuss the Context Model and the ideas underlying it. After describing what can be gained by this approach and how it compares to previous ones having a similar aim, I give the details of its theory and implementation.

I set forth the components of the Context Model and explain how it is constructed from input received from outside the system together with information in the system's long term memory. I also show how the understanding of new input is influenced by what is present in the Context Model.

A Note about "Understanding"

The word *understanding* is used throughout this chapter to describe what takes place when the system under discussion processes newly received input.

The term *understanding* has often been used to denote the process of extracting information from input text on the basis of a database of knowledge about the language used. This was also the case with early descriptions of the operation of PHRAN (Wilensky and Arens, 1980a) (Wilensky and Arens, 1980b). This usage of the word tends to obscure the fact that the employment of language is a communicative act that involves a speaker attempting to create a change in the hearer's conception of the world (cf. Austin (1962) and Searle

(1969)). Language cannot be understood solely on the basis of information pertaining to the words and constructions used in it. Rather, the lasting effect of the processing of language on the knowledge structures of the hearer is an integral part of the understanding process.

It is in this latter sense that the word *understanding* is used here. In this chapter, and in this thesis as a whole, I attempt to give an idea of how this kind of understanding can be performed and in what ways it supplements understanding in the more limited sense.

The language analyzer PHRAN is a powerful tool for the analysis and understanding of input text. However, its database consists almost entirely of information about language, and it is thus inherently limited in its ability to understand in the sense being described here. The framework of the Context Model is supposed to provide the equivalent of the hearer's world view, thereby making it possible to model the effect of the language on it.

IV.1. Previous Work and Basic Ideas

IV.1.1. Models of Memory

The notion that language understanding is a process that consists of analyzing the linguistic input in a way that is dependent on the view of the world held by the understander, and affects this view, is rather straightforward. It is surprising therefore to find that very few systems exist which incorporate this idea.

Of the numerous existing paradigms for language analysis, only Quillian's TLC (Quillian, 1969), Winograd's SHRDLU (Winograd, 1970) (Winograd, 1972), and Lebowitz's IPP (Lebowitz, 1980) can be said to have parsers working within a model that the system has of the surrounding world. Even these systems are found to be wanting in this respect, however, when one considers them as prototypes for more general understanding systems.

Quillian's model of memory consists of a semantic network of nodes representing abstract and concrete objects, which are linked according to the various semantic relationships between these objects. While reading the input text his system marks the objects mentioned in it and traverses the links emanating from those nodes, marking others along the way. Great importance is assigned to nodes at which paths emanating from different objects intersect, since they are considered to represent additional relationships between the original objects that the text is attempting to convey.

TLC is primarily a model of memory. One of the central features that it lacks is a powerful language analyzing facility. Built into TLC are all the assumptions about the role of words in the transmission of meaning that were criticized in Chapter II. The role of TLC's memory is to assist in determining the relationships between words in the input sentence, and once a sentence is parsed by the system there appears to be no information retained which would influence future input processing. In other words, the system's view of the world is essentially static and not affected by the contents of text it is analyzing. Once a sentence is parsed, memory returns to its original state.

Quillian's model of memory was a pioneering one for its time, and much of the criticism just directed at it is purely a reflection of the fact that it was a very early effort in this direction. Its basic thrust is very close to that which will be presented here, and it has inspired some of the ideas present in this work.

SHRDLU did contain a complete model of the blocks world with which it dealt. Input sentences were parsed in reference to the objects present in this world, and the result of parsing a sentence was an immediate change in the system's model of the world reflecting the execution of any request conveyed by the input. In these strengths, however, lie SHRDLU's weaknesses, from my point of view. SHRDLU did not have a *model* of the world – it contained an *exact replica* of its world within it. Every object considered to exist and every aspect of it were constantly available to the system. It is instructive to note that it was not considered necessary for SHRDLU to actually manipulate real objects, since its representation of things mapped in such an obvious and natural way to a possible real world.

Such an approach is practicable only when the world the system is supposed to deal with is limited to the extent that it was for SHRDLU. It thus sidesteps the problem of establishing the focus of a conversation, whereas much of the use of language is aimed precisely at this end. It also essentially eliminates the need to deal with the structure of the system's knowledge about the world around it, since the system knows only of such a limited universe.

If SHRDLU captured more of its world than we would like a system to, IPP captures much less.

IPP reads stories directly off a newswire. It identifies one of several memory structures, called MOPs (Schank, 1980), representing the type of activity or event described in the news item. Once that has been done, the parser's processing of the rest of the story is directed towards identifying missing elements in the relevant memory structure, while ignoring other material. A central aspect of IPP is its ability to form new memory structures capturing generalizations observed by the system in stories it has read.

IPP's model of what was discussed consists, therefore, of that single MOP which is a generalization of the event being described in the story. This is a good choice for short newswire stories that, more often than not, do describe a single event. But it does not address the problems facing a system like UC which is supposed to hold a conversation with a user, shifting its focus continually in the process. In general, a text is not meant to describe in full a *single* memory structure.

The work presented here is an attempt to offer a direction towards solving some of the problems exhibited by the work reviewed above. This thesis presents a model of memory and several ideas for the organization and utilization of memory structures which enable the UC system to perform in ways in which previous systems were not capable. Building on this effort, additional work can be done that will lead towards more powerful language understanding systems. I will note possible avenues for such research as they come up.

IV.1.1.1. The ACT Theory of Factual Memory

The theory being proposed here is most similar to the memory and activation scheme of Anderson's ACT theory of memory (Anderson, 1983). ACT theory attempts to explain memory phenomena observed by experimental psychologists. Anderson proposes a network memory structure where activation of nodes determines the speed of their recall. ACT theory successfully explains interference, judgement of associative relatedness, the influence of practice on recall, the difference between the recall and recognition paradigms, and the influence of elaboration on recall.

The ACT and CLUSTER theories were developed independently. ACT theory is concerned primarily with explaining the encoding and recall of symbols and facts, while CLUSTER theory is concerned with explaining the processing performed in the course of understanding natural language. The fact that by independently attempting to account for two different cognitive phenomena models so similar are arrived at, could be taken as evidence of the validity of both theories.

In some cases practically identical solutions are proposed to account for observations in the two domains. For a striking example, compare the discussion of "the file" and "the execution" in IV.1.3. with Anderson's explanation of thematic judgments. Anderson proposes the existence of subnodes in memory to stand for themes. These subnodes organize facts in memory in exactly the same way as clusters do here. Also in the same section, Anderson comments:

> Level of activation can also provide information in and of itself. The more active a particular part of the network is, the more it

must be related to the current context.

This is precisely the interpretation of the weight of an entry in a cluster, provided in section IV.3.2.

There are some differences between the memory structures proposed by the two theories, however. One is in the interpretation given to the activation level of a node. ACT theory explicitly states that it is a measure of the speed of a fact's recall. CLUSTER theory, on the other hand, only states that it is a measure of an entry's salience in the context. It is up to any process that makes use of information in the Context Model to decide how an entry's salience should affect it.

Another difference is in how activation decays. ACT states that this occurs with the passage of time, while in CLUSTER it is a function of the amount of processing that takes place. This discrepancy can be explained as a consequence of the nature of the problems the two theories deal with. The theory proposed in this thesis attempts to account for interactions between a human typing language and a computer printing information on a video display terminal. A user is free to wait as long as he/she wishes before responding to the system's output, and can reread the output later. The mere passage of time is not correlated to a loss of salience of information conveyed in the conversation.

ACT theory ignores the issue of how information stored at a memory node is represented. In particular, there is no discussion of the use of general information in specific situations. ACT accounts for people's behavior in situations where they are presented with specific facts and are later asked to recall them. In this

context, there is no need for general knowledge to be applied. Such an ability is important for us in this thesis, and is discussed in IV.3.2.

In sum, although they attempt to account for the performance of different tasks, there is significant similarity between the structures proposed by the ACT and CLUSTER theories. At the very least, this should convince us that a spreading activation memory model is worthy of further investigation.

IV.1.2. The Structure of Memory

The cluster is a collection of entries which share the property of co-occurring often in real life situations. This includes entries that occur together because they are all part of a sequence of frequently executed steps in a familiar plan, (a script (Schank and Abelson, 1977)), or because they describe all the aspects of a familiar event (a MOP), etc., but it is not limited to these. In particular, the mere fact that several entries belong to the same cluster does not imply that there is any structural relationship among them. Therefore, any relations which do exist, such as the temporal ordering of a script, must be made explicit if the system is to have knowledge of them.

In previous systems, where such information was not made explicit in the memory structures, it had to be built into the processors. The result is that separate systems had to be constructed to process various types of information, despite the overall similarity in the kind of memory structures involved (cf. SAM (Cullingford, 1978), PAM (Wilensky, 1978), IPP, and Dyer's AFFECTs (Dyer, 1982)).

Using an abstract notion of "association" as the basis for a cluster enables the system to reflect our intuition that at times pieces of information are associated with each other without there being any explicit mechanism that could account for it. For example, most users of the UNIX operating system will respond to the question

How can I delete a file?

By saying something like

Use the 'rm' command.

This is true even for novice users who have little or no understanding of how UNIX works or of the technical details of what actually happens when the 'rm' command is used. People will suggest using this command even if they have no clear knowledge of what is actually involved in the deletion of a file from a directory. In such a case it would be difficult to argue that the user arrives at the answer by using a problem solver, or some reasoning mechanism, with which he figures out a way to achieve the desired goal state. The user cannot do so without detailed knowledge of the system, which we assume is not available.

This is more easily accounted for by postulating that the user has a cluster that includes the following information:

{ 1) use the 'rm' command
2) delete a file
3) (1) causes (2)
}

Entry (3) represents a piece of knowledge of the user, which is associated with the other two entries. With many UNIX users this knowledge is not based

on an understanding of the program run by typing 'rm'. Such associations need not necessarily be reflections of the true state of affairs in the world. E. g., a belief that chicken soup cures colds would be represented in a similar cluster and cause a similar kind of behavior. TLC, in contrast, allowed only links based on certain known conceptual relations between the nodes in its semantic net.

In addition, the cluster structure allows associations to be used in either direction, so that the use of 'rm' may be interpreted as an attempt to delete a file.

The phenomenon of recognizing facts as 'related' at a high level, without having any knowledge about *how* this relationship comes about, appears to be a very common experience among people. The lack of structuring in clusters allows us to capture it within the same types of knowledge structures that we use for entries whose relationship to each other is known. The only difference is that in the latter case the cluster contains entries representing the nature of the relationship.

Clusters can just as easily encode information about relationships that are not causal. Sensations such as fear and pleasure are linked in people's minds to certain situations. The characterization of a situation as fearful, pleasurable, etc., varies from person to person. The relationship is very often not causal or rational, in the sense that we are unable to predict or even explain it. Although such associations were not of interest in the design of the UNIX Consultant, they could nevertheless be encoded in clusters*, allowing the prediction of people's behavior in the appropriate situations.

* Provided, of course, that one knows how to represent the concepts to be associated.

IV.1.2.1. Examples of Clusters

Following are several examples of clusters. These are presented in a very simplified form, since the details of the language used to represent entries and clusters will only be described in section IV.2.

* An object and its attributes -

  { **OB1** is a file

    OB1 has protection 644

    OB1 is owned by user 'arens'

    OB1 has name 'init.l'

    OB1 has been sent to the line printer }

* An action, related preconditions and consequences -

  { **FILE1** is a file

    User types 'rm **FILE1**'

    FILE1 used to exist but no longer will

    FILE1 is in a directory where user has write-permission

    User wants to delete **FILE1** }

* A convention of discourse -

  { **QUEST1** is a question

    ANS1 is output

    User asks **QUEST1**

    System produces **ANS1** }

* A behavior rule -

{ **STATE1** is a state

User asks how to achieve **STATE1**

User wishes to achieve **STATE1**

System tells user how to achieve **STATE1** }

IV.1.3. Spreading Activation

The usefulness of spreading activation in understanding natural language was recognized by Quillian (1969). In order to permit the Context Modeler to distinguish between more and less central entries, it associates with entries in a Context Model a **level of activation,** a number on a scale from 0 to 100. In this respect CLUSTER differs both from TLC and from Charniak's marker passing model (1983), in which relevant information is simply marked as such, with no gradation possible.

The processing of input is accompanied by the activation of concepts mentioned in it and the flow of activation from those entries to others linked to them. It is possible for an entry to receive contributions from several other entries. As with other activation based language understanding systems, CLUSTER thus has the ability to identify entries that are contextually related to several concepts in the input, although not closely associated with any single one.

For example, consider the following utterance:

User: The file I sent to the printer –
what happened to it?

As an understanding system reads this input from the user it first encounters

a mention of the concept of a *file*. A file has many actions associated with it, i. e., it belongs to many clusters. The notion of a file is integral to copying, deleting, reading, printing, editing, and more. We would not expect the understanding system to attach particular importance to any one of these simply because a file has been mentioned.

On the other hand, after the system is told that the file in question was sent to the printer, identification of the relevant context becomes possible. The additional information allows the system to identify the desired file, and also to bring into focus the circumstances of the original printing. The printing action now stands out among the numerous possible file manipulations, since it is the only action related both to files and to printers.

This effect is achieved in CLUSTER by analyzing input for entries describing fragments of the input. For each such entry all clusters which contain it are retrieved from long term memory, and a fixed amount of activation is divided between them. Each entry in each cluster is then activated by that amount, regardless of the number of entries involved. In the situation described above, due to the large number of clusters that the "file" concept is a member of, very little activation is given to each one. The resulting effect is that none of their entries have much significance in the Context Model.

When later in the input the printer is mentioned, activation is added only to the entries of the cluster describing the printing of a file. This cluster thus becomes the only one whose entries have a high enough level of activation to be considered salient in the present context. For this reason, other processes which

use the Context Model as a pool of potential candidates (e. g., reference identification), will choose an entry in this cluster over one in any of the others.

A simple network like Quillian's, in which activation spreads, often does not contain enough information about real world situations to permit identification of the context. The cluster-based scheme of spreading activation described above gives the Context Modeler additional abilities in this regard. The clusters of which an entry is a member describe the different situations of which it is a part. The amount of activation spread from an entry depends on the number of clusters to which it belongs, and not simply on the number of links emanating from it. Therefore, the amount of activation transmitted from an entry to others linked to it is a function of the number of possible situations which might be present when that entry is seen.

An entry may be relevant in only a few situations, or else in many. In the former case it would probably be appropriate to try to determine the context immediately after receiving the input; this would probably not be appropriate in the latter case.

For example, when an input to the system begins with

User: The file . . .

it is wasteful to try to determine what the context is after processing only those two words. (I am assuming the absence of any previous context.) Files are part of many different actions and scenarios, and the system does not yet have a basis for preferring one to the others.

On the other hand, if the user were to start input with the words

User: The execution . . .

we would like our system to conclude that either a command or a program is being discussed, since these are the only two objects it knows about which can be executed.

CLUSTER is able to distinguish between the two cases since it organizes all entries related to the same situation into one cluster. In the case of "The file", which is contained in many clusters, each receives very little activation. The Context Modeler will recognize that no particular cluster is known to be relevant at that point. In the case of "The execution", the two clusters to which it belongs become moderately activated and will be considered possibly relevant.

IV.1.3.1. Decay

The activation of entries in the Context Model increases with continued mention of the entry in the input, either explicitly or by being a member of clusters retrieved from long term memory in the course of processing the new input. A parallel process is the continuous **decay** of each entry's activation level. Unless it is reinforced, the activation level of an entry will decrease to the point of it being removed from the Context Model.

As a result, we find that when the discussion shifts to another topic, all information pertaining to the earlier exchanges eventually disappears from the context. It is then no longer available to processes that access the Context Model.

Decay occurs by the activation of all entries in the Context Model being reduced by a certain fixed proportion once every "cycle". A cycle of the operation of the Context Model is the complete processing of a single new entry. This includes the retrieval of any additional clusters in long term memory that the entry is a member of, and their insertion in the Context Model.

IV.1.4. Summary of Processing

When a new entry is passed to the Context Model the following **insertion process** takes place:

1. Does matching entry already exist in CM?

Yes	*No*
2a. Increase its activation	2a. Insert new entry
2b. Spread activation to its other clusters	2b. Retrieve indexed clusters from LTM
	2c. Run insertion process on new entries (recursively)

3. Inspect Context Model, perform appropriate actions
4. Decay: decrease activation of all entries.
5. If actions in 3. caused changes in CM, go back to 3.

A complete description of this process is the subject of the rest of this chapter, following a comment on the nature of definitions.

IV.1.5. A Note on Definitions

I have explained previously that one of the principles underlying the notion of a cluster is the lack of any distinguished "slots" in it. In the CLUSTER model there is no room for attributes distinguished as "defining properties". In fact, we

can provide a better explanation for this notion, which may also help in understanding what it means to be part of the definition of a concept.

In the terms of the CLUSTER model of memory, an entry is considered to represent a defining characteristic of a concept if it has a high cumulative weight in the clusters indexed by the concept in question. *The cumulative weight of the an entry X as a defining characteristic of a concept C* could be defined as the level of activation X would eventually receive if the concept C were inserted into an empty Context Model with an initial activation level of 100. If we let n be the number of clusters indexed by C, and X_i be the weight X has in the ith cluster, and if we further assume that there is no interaction between the various clusters (i. e., no other shared entry and no pair of entries one from each cluster which belong to a third cluster*) then the cumulative weight of X as a defining characteristic of C would be given by the formula,

$$100 \times \left[1 - \prod_{i=1}^{n} (1 - \frac{X_i}{100 \times n}) \right]$$

Consequently, the property of being a defining characteristic is now viewed as a point along a continuous scale. An entry may be more, or less, a defining characteristic of a concept represented by some other entry. This is determined by the amount of activation it receives whenever the other entry is activated.

The notion of being a defining characteristic of a concept to a high degree is somewhat wider in its scope than the traditional notion of the definition of a con-

* Although this assumption does not commonly hold, the contributions to activation from transfer through more than one link are relatively small.

cept. Besides entries representing statements about the nature of the defined concept it also includes entries representing other concepts consistently and closely associated with it.

IV.2. Entries

The language used in UC to describe the content of an entry is a variant of Schank's Conceptual Dependency (Schank, 1975). An entry has an associated level of activation, and may belong to one of several types, as explained below. The choice of which actions and objects are to be considered atomic in the UNIX Consultant has been done on an *ad hoc* basis, taking into consideration the semantics of the domain.

For example, an entry representing the observed fact that the user has taken the action of logging in looks like the following:

```
(assertion (activation 80)
        (cd (state-change (actor *user*)
                          (state-name login)
                          (from out)
                          (to in))))
```

The term *assertion* is the entry type, in this case an entry representing an asserted fact gained from the user's input or from the system's observation. The *activation* slot contains the level of activation of this entry, a number between 1 and 100, whose significance will be explained later in this chapter. The last component is the *cd*, a Conceptual Dependency diagram, describing the concept being asserted. In this case this is the fact that a *state-change* has taken place, with the *actor* being the user, the *state-name* being the login state, and this

state has changed *from* 'out' *to* 'in'.

The Context Modeler has no knowledge built into it concerning the semantics of 'state-change', or 'login'. The knowledge that exists is in the form of clusters in the system's long term memory. For example, there could be a cluster in LTM associating the entry described above with another representing the assertion that the user is now in the state of being logged in, and another one associating the assertion of the state of being logged in with an assertion of the user's ability to perform actions on the system. If both of these clusters are indeed present, then upon receiving input concerning the original state change, the first cluster will be loaded into the Context Model from long term memory, causing the system's model of the world to contain an entry representing the fact that the user is now logged in. As I describe in section IV.3., the insertion of this last entry into the Context Model will cause the loading of the second cluster, and make the system "aware" of the user's ability to perform actions.

IV.2.1. Classes and Types of Entries

In the example above I gave an entry of type *assertion*. This is one of many types of entries. They fall into three classes, reflecting the different kinds of processes and constructions that take place in the Context Model.

There are numerous types of entries falling into each of the three classes, but no formal classification of them has been attempted. The most common types used in the UNIX Consultant are described below. Several additional examples of entries are presented throughout the rest of this chapter.

The three classes are:

1. **Object**: Entries belonging to this class stand for objects. The described objects may be either physical objects, or more abstract ones, e. g., events and actions. Actions are included in the same category as objects since they are treated similarly. In both cases the Context Model must contain a token representing the concept so that the relationships between it and other concepts can be explicitly indicated. The existence of an entry of this type is a prerequisite for the ability to understand references to it. References can be made to the real world objects that such entries represent, and they will be matched to these entries. When input is interpreted as referring to an entry of type *object* (in the simplest case, when the user employs a direct reference), the system immediatedly attempts to identify the object referred to, and will note if it cannot. (See discussion of *intentions* of type *refdef* below.)

The reading of a noun-phrase, such as "my home directory", "a file without read protection", or "the removal of directory X", will give rise to an entry of this class.

A simple entry describing a file represented by the token "file1" looks as follows:

```
(mention (activation 100)
         (cd file1)
         (meaning ((cd-form file1
                    description (file container physob)
                    p-o-s noun-phrase))))
```

This entry is of type *mention*, reflecting the fact that it stands for an object asserted as being present in the real-world context of some conversation. The

most common way for this to happen is to have the object actually mentioned in the text.

The *meaning* slot contains the description of the object represented by the token "file1". The description is in the form that PHRAN produces. This slot is necessary in order to provide a full description of the token, since the *cd* slot of such entries contains only the token itself. The information in the *meaning* slot is gathered by PHRAN in the course of processing the language fragment that gives rise to the entry.

The only type of entry in the *object* class is:

Mention – Description of a token representing an object believed to exist, along with the object's basic properties.

2. **Assertion**: Entries in this class describe states of the world, state-changes, or actions known to the system. For example, the fact that the user has deleted a file represented by the token "file1" will be represented as the following assertion:

```
(assertion (activation 100)
        (cd (causation
                (antecedent (do (actor *user*)))
                (consequent (state-change
                                (actor file1)
                                (state-name physical-state)
                                (from existing)
                                (to non-existing))))))
```

An entry of class *assertion* may be of one of several subtypes, the most common being:

Assertion – Description of an event, e. g., the user having issued a command.

Question - Description of a request made by the user.

Fail - Description of an apparent state of failure of an attempted plan.

Preconds - Description of the preconditions necessary for a particular action to be executed successfully.

There may be entries of type *object* and *assertion* describing the same event. For example, the Context Model may include an entry of type *object* representing "the removal of directory X by the user", and also one of type *assertion* representing "the user removed directory X". These are not the same. Only the first, being of type *object*, may be referred to, and only the second implies the actual occurrence of the event. If the first entry alone is present, the *hypothetical* consideration of the event is possible, without its consequences, if any, being asserted.

The reading of a complete declarative sentence will, as a rule, give rise to an entry of type assertion representing the event or action described by the sentence.

3. **Intention**: An entry of this class describes an intention the system has of performing a particular action. There is a variety of actions that UC is able to perform, among them producing an output utterance, searching the Context Model for a desired entry, and querying the UNIX operating system. An entry of this type is called "intention" as opposed to "action", say, since its presence does not necessarily imply that it will be performed. Later in this chapter I explain how the level of activation of such an entry determines whether or not this takes place.

An example of an entry representing the system's intention to output to the user the statement "The file func2 does not exist"* is:

```
(output (activation 100)
        (text |The file func2 does not exist|))
```

The most common types of *intentions* are:

Output - Intention to inform the user of something. When the activation level is high enough the required information gets printed on the terminal.

Refdef - Intention to find a referent for a particular linguistic fragment. To carry out this intention, the information obtainable from the given linguistic expression (including selectional restrictions gotten from higher level phrasal patterns in the input) is matched against existing entries of class *object* .

The matching function increases the degree of matching whenever agreement is found between attributes of the two entries. The attributes compared include syntactic and semantic ones, and they are determined both from information in long term memory and from the utterance itself. The object-matching function is described in the next section.

If a good enough match is found the 'refdef' is replaced with the matching entry. Otherwise, a new entry of type *mention* is created to represent a conjectured object.

Out-planfor - Intention to find a plan for a achieving a given goal. To

* The text to be output is *canned* in this example, as in the rest of this thesis. In the UNIX Consultant system itself, the text is generated by the natural language production program PHRED (Jacobs, 1983). In that case, the text will be replaced by a conceptual representation of the meaning of the utterance to be generated.

carry out this intention the Context Model is searched for entries of class *object* representing plans. These are checked to see if the plan one of them represents has the desired goal listed as its own goal. It should be noted that *the Context Model,* and not LTM, is searched. This means that the necessary plan must already have been loaded into the context for this search to be successful. This will often be the case, since other information about the desired goal will have been used for fetching the cluster including the needed plan. If an appropriate plan cannot be found then a search for one will be conducted, possibly requesting help from a separate planning module.

Again, entries of the first two classes may exist and represent the same action that an entry of type *intention* stands for. This enables the system to understand references to the intention and its consequences.

Since entries of the class *intention* stand for intentions of the understanding system, they are not described in the user's speech. They are given rise to internally, by being present in clusters associated with entries derived from the input. For example, the user's asking

How can I change the protection of a file?

will eventually give rise to intentions in UC to find the answer to the question and reply to the user. The analysis of the question itself, however, will only provide UC with an entry representing the assertion that the user has asked how to perform this action.

IV.2.2. The Object-Matching Function

The object-matching function is used to find a possible referent for a *refdef* entry. It compares each entry of class *object* in the Context Model to the reference, and ranks them according to how close they match it.

The function works by comparing attributes of the two entries and increasing the degree of matching whenever agreement is found. The attributes compared are the syntactic part of speech, the semantic categories the objects represented by the entries belong to, and the adjectives used in the descriptions of the objects. The semantic categories are obtained from the "description" list of the terms the entries represent, and the information about adjectives is that collected by PHRAN during its analysis. The total number of categories in the "description" and the total number of adjectives used for each entry are taken into account during the comparison, so as not to bias the comparison in favor of object about which much happens to be known.

The degree of match is further increased in proportion to the activation of the entry being considered as the referent. The matching process is thus weighted in favor of those entries with a higher level of activation.

Several numerical constants are used by the matching function. They are: the amount by which the degree of matching is increased when both entries share a consistent property; the increase for having consistent adjectives describing the two objects; and the increase added due to the activation level of the matching entry. In the UC system, values were chosen so that reasonable results would be obtained during matching. No theoretical importance is claimed for these partic-

ular values. The effects of these choices are discussed further in section V.2.2.

For examples of matching of a reference to objects see the treatment of references in the program sessions in Chapter VI.2.

IV.3. Clusters and Entry Weights

IV.3.1. What Constitutes a Cluster?

Clusters are memory structures reflecting previous observations about collections of concepts that are found to be present simultaneously in the system's surroundings. This is the case for a group of entries that together describe one situation, or for a pair that represent a cause and its effect. Typically, clusters indicate a particular relationship among the structures, or else they are created as the result of such a relationship, which may be unknown to the system.

A list of the types of relationships that may exist between constituents of a cluster was provided in section III.2.

There are two types of clusters, **static** and **active**. All clusters described up to this point are of the first type. These clusters are present in the system's long term memory. Each encodes the fact that the entries in it tend to appear together in the world within which the system operates. Whenever, as a result of analyzing input, a new entry is inserted into the Context Model, those static clusters in long term memory that contain this entry are copied into the Context Model. These instantiated clusters then become *active* ones. Their number, and the content of their constituent entries, may change from that point on in ways

that will be described below. Such changes will be the result of the specifics of the conversation taking place.

IV.3.2. Static Clusters and Their Activation

In a static cluster there is a numerical weight associated with each of the constituent entries. The weight is an indication of how central an entry is to the whole cluster. These weights are integers between 1 and 100. Entries with high weights are given a higher activation level when the cluster is eventually instantiated in the Context Model. The weights are used when the cluster is retrieved after one of its entries is identified by the system in the context. In the discussion that follows, I will refer to that entry as the **base entry.** At that point a certain amount of activation is allocated to the about-to-be-activated cluster. This amount is called the **base level of activation** of the cluster, and is computed by dividing the activation level of the base entry by the total number of clusters indexed by it. If the base level of activation is below some fixed threshold, none of the retrieved clusters are activated. Otherwise, they are all activated and inserted into the Context Model; the base level is used to determine the activation level given to each of the entries. If a given entry has weight **W** in a cluster with a base level of activation **B**, then the activation level assigned to the entry is

$$\mathbf{W} \times \mathbf{B} \,/\, 100 \,.$$

The weight of each of the other entries in the cluster influences, then, how much of the base activation it will receive.

If an entry inserted into the Context Model is a totally new one, the activation level computed above will be assigned to it. Otherwise, i. e., if a matching entry already exists, an increase in the existing entry's activation level will follow.

The spreading of activation reflects the intuition that if too many clusters contain a certain entry, no cluster should be viewed by the system as part of the context solely on the basis of this entry's appearance.

For example, consider a static cluster describing the relationship between the user's inability to perform an action, **action1**, the user's request for help, and the system's intention to solve the problem and notify the user.

```
{   (assertion    (weight 100)
                  (not (concept (able (actor *user*)
                                      (perform action1)))))
    (out-planfor  (weight 100)
                  (cd action1))
    (question     (weight 100)
                  (cd (planfor (concept action1)
                               (is *unknown*))))
    (fail         (weight 100)
                  (plan (planfor (result action1)
                                 (method plan1))))
    (mention      (weight 50)
                  (cd (planfor (result action1)
                               (method plan1))))
    (output       (weight 100)
                  (text text1))      }
```

The 'out-planfor' entry is an intention to find a plan for the action being considered; the 'question' entry is the assertion that the user has asked a question requesting to know what the plan for **action1** is; the 'fail' entry is the assertion that the user's original, unidentified, plan has failed to produce the desired results; the 'mention' is an entry of class object representing this unidentified

plan the user is believed (by the system) to have originally tried; and the 'output' is an intention to express something to the user.

This cluster expresses the relationship among all these entries. Next, suppose the 'assertion' entry is introduced into the Context Model. Typically this would occur if the user said:

I can't do <some action>

Then the entire cluster would be retrieved and assigned some activation level. The exact value assigned is a function of the precise circumstances. For the purpose of this example let us assume the level assigned is 60. Each entry with a weight of 100 will then receive an activation of level 60. The 'mention' type entry only has a weight of 50, and so will receive only 50% of the available activation, for a total of 30 units. Next, each entry would be matched against those already present in the Context Model. If a matching one was found the new entry's activation would contribute to that of the existing one. Otherwise the new entry would simply be inserted.

When a new entry introduced into the Context Model matches an existing one, the existing one is *reinforced*; its activation level is increased by an amount which depends on the activation of both matching entries. If the activation levels of the two entries are A1 and A2 then the final activation level of the entry in the Context Model is given by the formula

$$A1 + A2 - A1 \times A2 / 100$$

This formula is symmetric in A1 and A2. Hence it is of no consequence

which entry we consider to be the matching one and which the matched. Since both A1 and A2 are between 0 and 100, that will also be true for the resulting level of activation. Furthermore, 100 serves as an upper bound. Continual reinforcement of an entry will only make its activation level closer and closer to 100 but will never actually equal it, unless the activation of one of the entries was 100 to begin with.

An illustration of reinforcement is provided in the following section, IV.3.3.

Generalized clusters

Often the information encoded in a cluster is of a general nature. Consider the simplified cluster used earlier to exemplify one describing a system behavior rule:

{ **STATE1** is a state

User asks how to achieve **STATE1**

User wishes to achieve **STATE1**

System tells user how to achieve **STATE1** }

This cluster encodes a certain relationship among several entries representing facts associated with **STATE1**. A similar relationship, however, holds for *all* states.

The required generalization is handled by allowing tokens that begin with a '?' in clusters, to stand for variables. The generalized form of the above cluster will thus be:

{ **?STATE** is a state

User asks how to achieve **?STATE**

User wishes to achieve **?STATE**

System tells user how to achieve **?STATE** }

This cluster will be retrieved whenever an entry is introduced into the Context Model which matches any of its members. The variable **?STATE** matches any structure. The retrieved cluster will have all the variables replaced by the structure corresponding to **?STATE** in the matched entry. The matching procedure is described in the section on unification below.

IV.3.3. Unification

A process of unification is used in two places in the processing the CLUSTER system performs, and is identical in both cases. One case is that of retrieving clusters of which a given new entry is a member; the second is the matching of a new entry against existing ones in the Context Model, for the purpose of checking if it is already present there.

Entries have a "slot-filler" form, as do the fillers. During unification, the two entries being processed are compared slot by slot; the fillers in each slot being compared recursively. An empty slot in one structure is considered to match anything in the other, while filled slots in both are required to be filled by matching structures. A variable in one structure matches anything in the other, except that the same variable appearing more than once in an entry must match the same structure in the other entry. During the process, a new structure is constructed containing all the fillers present in at least one of the entries being

matched. This new entry replaces the matched one in the Context Model.

The *activation* slot is the only slot treated differently. The activation of the new structure determined according to the rule described in the previous section.

Examples

1.
```
(assertion (activation 80)
           (cd (ptrans (actor john1)
                       (object john1)
                       (from new-york-city))))
```

will match

```
(assertion (activation 40)
           (cd (ptrans)))
```

and result in the following:

```
(assertion (activation 88)
           (cd (ptrans (actor john1)
                       (object john1)
                       (from new-york-city))))
```

2.
```
(assertion (activation 88)
           (cd (ptrans (actor john1)
                       (object john1)
                       (from new-york-city))))
```

will **not** match

```
(assertion (activation 30)
           (cd (ptrans (actor mary1))))
```

And

3. (assertion (activation 80)
(cd (ptrans (actor john1)
(object john1)
(from new-york-city))))

will match

(assertion (activation 30)
(cd (ptrans (actor ?a)
(object ?a)
(from new-york-city))))

and will result in

(assertion (activation 86)
(cd (ptrans (actor john1)
(object john1)
(from new-york-city))))

IV.3.4. An Alternative Approach to Cluster Structure

IV.3.4.1. Bi-Directional Links

An alternate approach to the structure of clusters in LTM and the Context Model would link every entry to every other entry in a cluster with bi-directional links. That is, every pair of entries would be linked by two links, one going in each direction, with an independent weight assigned to each of the links. The weight assigned to the link in each direction is to be a measure of the strength of the association between the two entries in that direction.

For example, consider the cluster representing the association between removal of a file, its preconditions and its consequences, described earlier in this chapter, in section IV.1.2.1. Here it is again in its simplified form:

{ **FILE1** is a file [1]

User types 'rm **FILE1**' [2]

FILE1 used to exist but no longer will

FILE1 is in a directory where user has write-permission

User wants to delete **FILE1** }

Consider the directed links that would have to exist between the first and second entries in this cluster, [1] and [2]. The weight of the link [1]→[2] is a measure of the strength of the association between something being recognized as a file, and the user's removing it by using the 'rm' command. From the point of view of the processing routines, it is a measure of how strongly activated the entry representing the deletion of file will become upon the insertion of an entry representing the file itself. The weight of the link [2]→[1] is a measure of the strength of the association between identifying a user input as having the form 'rm *string*' and the realization that *string* is, in fact, the name of a file. The processing routines will assign a level of activation to the assertion that *string* is a filename, dependent on the weight of this link.

In general, the weights associated with the links in the two directions would not be identical. In the present example, the link [1]→[2] should be weak (i. e., have low attached weight), since we would not expect that whenever a file is mentioned the possibility of the user deleting it be considered. This will only rarely be the case, and in general this data will be useless. On the other hand, when a user types 'rm *string*' it is quite certain that the *string* is the name of a file, and we would expect any understanding system to be aware of that. The

link in this direction should be heavily weighted.

In this design, as in the current one, when a new entry is created and inserted in the Context Model, LTM is searched for clusters of which it is a member. In the current design it is required that this matching entry have a high enough weight associated with it, but in the bi-directional link design no weight is associated with individual entries. Instead, once a matching entry is found in some cluster, all links from that entry to the others would be checked and a sub-cluster created of all those entries linked strongly enough with the original one. The cluster eventually activated is created as a copy of this sub-cluster. This mechanism allows for a finer tuning of the extent to which insertion of an entry in the Context Model causes the insertion and activation of additional entries from its cluster.

When an entry in the Context Model is reinforced, usually as a result of being mentioned again in the text, activation spreads from it to all other entries in clusters it belongs to. With the weighted bi-directional link design of LTM it is necessary to maintain all the links when the cluster is copied over to the Context Model, for the purpose of controlling the spreading of activation there. More activation should flow through a strong link than through a weak one.

Despite its intuitive appeal and the added control it gives to the spreading of activation, the approach described above was abandoned in favor of the simpler one currently used. This was done for several reasons.

First, this bi-directionally linked structure is much more complicated. The number of links is on the order of the square of the number of elements in a clus-

ter, and even the task of entering a single new cluster into LTM while establishing all the connections and their weights involves considerable work. Activating a cluster means copying over an instantiated version of this richly linked structure from LTM to the Context Model while maintaining all the links between its entries. The inherently large number of links makes this process complex and very time consuming. These problems are technical in nature, however, and an efficient solution to them might exist.

The second reason is that it is not clear how assignment of all the weights of the links should be made. While the notion that a particular entry is more "central" to a cluster is intuitively fairly clear, the relative strengths of the various connections are not. In practice, it was found that weight assignment could only be done by considering the relative salience of entries in the cluster. Recognition of this fact led to the development of the current individual-salience-based structure.

In addition, placing heavy importance on the links between the individual entries seems to be contrary to the intended notion of a cluster. Entries are to be interpreted within the framework of a cluster describing the context in which they appear. This should also be the case for the associative connections between them. It would be better to have a method for determining the strength of the link between entries as a function of their status in a cluster, as opposed to storing a fixed value of this parameter for every pair of entries. As it turns out, this effect is actually achievable in the current implementation. It is described in the next section.

IV.3.4.2. Bi-Directional Link Effect in Current Model

In the current model of LTM each entry in a static cluster has a numerical weight (between 0 and 100) associated with it. The underlying intuition is that this weight is a reflection of the salience of the entry in its cluster. As a rule, when the system designer defines a new cluster, heavy weights are assigned to those entries which appear to be central and more uniquely associated with the situation described; smaller weights are assigned to those entries that are common, appearing in many other clusters.

For example, the following cluster from a previous example shows the weights associated with its entries:

{ <40> **STATE1** is a state

<90> User asks how to achieve **STATE1**

<90> User wishes to achieve **STATE1**

<60> System tells user how to achieve **STATE1** }

During processing these weights are expressed in two ways: in the choice of clusters to activate, and in the degree of activation the entries in an activated cluster receive initially.

When a new entry is introduced into the Context Model (usually as a result of the concept it represents being mentioned in the input), only clusters in which this entry appears with a high enough weight are actually activated. This procedure allows only clusters in which the new entry is prominent to be activated. In effect, it establishes an associative link between heavily weighted entries and all others in the cluster.

The *strength* of the link between the key entry and each other entry depends also on the weight of the second entry. This is achieved by making the activation level given tc entries inserted in the Context Model proportional to the their weights in the static cluster*. In effect we get strong links from heavily weighted entries to other heavily weighted entries, and weaker links, i. e., less activation transfer, from heavily weighted entries to less weighted entries.

No record is kept of the original weights when a cluster is activated, so to a great extent this implicit bi-directional link structure is lost. However, if, as a result of a combination of low weights, too little activation is passed from the original key entry to some other entry in its cluster, the latter entry will not be included in the activated cluster. This entails that in the future no activation will be passed from the original entry to the the second one, since the latter will have been deleted from the Context model. Thus a limited bi-directional link structure is nevertheless preserved in the form of 0/100 weight links, i. e., links that transmit either no activation at all, or the full amount of activation passed through them.

IV.4. The Context Model

The Context Model is made up of instantiated clusters, and the details of the cluster structure were explained in section IV.3. This section is devoted to expanding on several issues that were mentioned only in passing, or else not at all, earlier. I will discuss issues related to the structure of the Context Model and

* An example of this process was provided in section IV.3.2.

the mechanism of spreading activation.

IV.4.1. Structure of the Context Model

The Context Model constitutes an *evolving partial network* of entries. This notion was mentioned briefly in Chapter III. Below I discuss it in more detail.

The clusters in long term memory may be viewed as network fragments, with the entries being the nodes. Retrieving a cluster from LTM and inserting its entries in the Context Model is, in effect, attaching the network fragment from LTM to the existing network in the Context Model. Attachment is done at shared nodes - a node in the existing Context Model and a node in the new fragment that are unifiable. With the inclusion of each new node in the network it is given a level of activation. As explained earlier in this chapter, with the passage of time this activation may increase or decrease. In fact, it may decrease to the extent that the node is deleted from the network. We thus find that the network in the Context Model evolves as the conversation progresses, with new pieces being added and nodes being deleted occasionally. This network is *constructed as needed* during the processing of input. Only clusters related to the input (i. e., containing entries that appear in it) are retrieved and added to the Context Model, and those entries not made reference to are eventually deleted.

For the purposes of the Context Model, "understanding" a new input consists of loading the appropriate network pieces - instantiating static clusters from Long Term Memory - and connecting them to the existing network by finding which new entries match ones already present.

The structure of the network constructed during the reading of a sentence might itself be capable of giving us information in certain situations. For example, if we end up having several unconnected clusters, i. e., clusters sharing no common entries, this may be an indication that the input has not been understood. This situation arises when no cluster is found that integrates all of the concepts appearing in the input. We thus have a simple notion of *incoherence*, which is exhibited here by the Context Model network being poorly connected.

IV.4.2. Activation in the Context Model

After the process of matching and unification determines which clusters in long term memory will be activated and loaded into the Context Model, a new process takes over. The spreading and decay of activation determines which entries will continue to be part of the Context Model and which will be removed. The spreading and decay of activation are also responsible for maintaining the measure of relative salience of the various entries, expressed as their respective levels of activation.

There are five situations during which activation of an entry may change. They are described in the following five subsections.

IV.4.2.1. Insertion

A new entry about to be inserted into the Context Model has an associated level of activation. This is (always) 100 for entries resulting from the analysis of the input text. The level of activation associated with other entries is determined

by the process described in section IV.3.2.

A new entry inserted into the Context Model either does or does not match one already existing there. If it does not, then it is given its associated level of activation. If it does match an existing entry then instead of a new one being created, the activation of the old one is increased. This is done using the formula:

$$(1) \quad A1 + A2 - A1 \times A2 / 100$$

where A1 and A2 stand for the activation levels of the first and second entry respectively.

This latter process is called *reinforcement*.

IV.4.2.2. Membership in a Cluster in Which a New Entry Matched

When a new entry is reinforced the increase in its activation is spread to all other members of the cluster. If we call the measure of the increase I, then the activation of each other member of the cluster is given by

$$(2) \quad A + I - A \times I / 100$$

where A is the activation level of the particular entry being modified.

IV.4.2.3. Membership in a Cluster in Which an Entry Was Reinforced

When an entry is reinforced through the simple spreading of activation, activation also spreads to all other clusters in which such an entry is a member. If the entry in question was reinforced by measure I, and the entry is a member

of N clusters in addition to the one through which the initial reinforcement was received, then the quantity M determines the activation passed on.

$$(3) \quad M = I \times K / N$$

Here K is a constant less than 1. It determines how much activation dampens when passing from one cluster to another.

Each of the entries in each of the other clusters now has its activation level increased to

$$(4) \quad A + M - A \times M / 100$$

from its initial A.

IV.4.2.4. Decay

With each cycle of processing, the activation of all entries in the Context Model is decreased by a constant factor. A cycle is defined as the complete processing of a single new entry. This includes its insertion in the Context Model (if it is not already there), reinforcement of all other entries linked to it, and the loading and processing of any other clusters in long term memory indexed by it, or by the other entries in the added clusters, and so forth.

IV.4.2.5. Dropping Below Minimum Threshold

Whenever the activation level of a particular entry drops below a fixed minimum level, it is removed from the Context Model. The rest of its cluster remains intact. This usually happens as a result of decay. Consequently, indivi-

dual entries and whole clusters that are no longer related to the input are dropped from the context.

The effect of dropping below the threshold for removal from the Context Model is the equivalent of forgetting things that were mentioned earlier in a conversation. In the final section of this chapter I comment further on this point.

IV.5. Creating and Storing New Clusters

For the Context Model to be useful, a large number of static clusters must exist in long term memory. These clusters are needed in order to encode knowledge about the domain, about conversational conventions, about how to perform various actions, and more, as explained in section III.2. All this information must be written by the system designer prior to the operation of the Context Model.

However, a system designer cannot create in advance clusters encoding all the knowledge the system will need in order to handle input. In the course of an exchange between a system like UC and a user, new contexts are created which may be referred to again at a later point. The system must have the ability to record such information so that it will be capable of understanding and reacting to a future reference to it correctly. For example, at one point a user might ask the UNIX Consultant the following question:

(1) How can I change the protection of the file 'data' so that everyone can read it?

UC should respond with something like,

(2) Give the command 'chmod 644 data'.

The user might then go on and ask other questions and discuss other issues. At a later point he/she may pose the following question:

(3) How do I change the protection of 'data' back to the way it was?

A Context Model related difficulty arises with the request to change the protection "back to the way it was". This makes reference to knowledge which of necessity was gathered in the course of an earlier part of the conversation. A correct reply to the question will be possible only if the mention of the file 'data' causes UC to 'recall' a trace of the previous exchange together with all other facts pertaining to the file. This will be possible only if such a trace were indeed stored earlier. Using the terminology of the Context Model, this will be possible only if at that earlier stage a cluster describing the situation at the time were created and added to long term memory.

An example of understanding input with the use of clusters created at an earlier point was provided in section I.6.

IV.5.1. Creating New Clusters

In the CLUSTER model we have the capability to create new clusters reflecting the state of the Context Model at any point.

Such a new cluster is essentially a copy of the Context Model at a particular point in time. As is the case with the creation of any cluster, the following questions must be answered when it is formed: Which entries should it contain, what weights should they be given, and which entries should serve as the indices for

this new cluster in LTM. In the current implementation, all these questions are answered purely on the basis of the level of activation of the various entries that are present in the Context Model at the moment the copy of it is created.

Specifically, a single new cluster is created to reflect the state of the Context Model. Every entry in the context with a level of activation above a fixed minimum threshold becomes a member of the new cluster. Each entry with a high enough level of activation becomes an index for the new cluster when it is added to long term memory. In the new static cluster, each member is given a weight equal to half its current activation level. This keeps the contribution of entries in such clusters lower, so that the permanent clusters in LTM maintain a greater influence on future processing in the Context Model. This is done for purely technical reasons. Since there is no mechanism for removing such new clusters from LTM, they might end up being recalled at inappropriate times. Giving entries in them lesser activation assures us that if this occurs, conflicting information from permanent clusters will be preferred.

IV.5.2. Problems with Creating Clusters

Currently, the user must explicitly initiate the recording of the contents of the Context Model. The problem of how to have this procedure invoked automatically is still open.

There are numerous interesting questions that need to be solved before this process can be automated, and doing so is beyond the scope of this thesis. These include the following issues:

- Should *every* change in the context cause a copy of it to be made, or should copying be done less frequently?
- If not every change is recorded, when should copies be made? Should it be based upon certain cues in the environment (e. g., in the text), some property of the Context Model's structure at a particular point, or the mere passage of a certain length of time?
- How will the relationship between new clusters formed in this manner and representing only slightly differing states be indicated? (This would be a particularly acute problem if the copying process were performed after every change in the Context Model.)
- How long should such clusters be kept as part of long term memory?
- How and when should such clusters be generalized? (Unless this is done, any cluster created in this way can only contain references to the specific objects present in the Context Model at the time it was created.)

In the current version of the UNIX Consultant some of the above questions are given *ad hoc* answers, and others are not addressed at all.

The process of creating a copy of the state of the Context Model in UC is user initiated. The user issues the required command when it appears to be needed. The basis for deciding to create a copy is the user's realization that the current situation is one that they might wish to refer to later. When a new cluster is created it is added to long term memory and from that moment it is indistinguishable from any other cluster. It remains there until the end of the current session. Like other clusters, it is not generalized beyond its current form. As an

unfortunate result, the user may cause the creation of a cluster describing a situation where a certain command was used to modify one particular file in some way, and when the need to modify *another* file in precisely the same way arises, this existing cluster will be of no use.

IV.6. Details of the Insertion Process

This section contains flowcharts describing the process of inserting a new entry in the Context Model. It complements verbal descriptions of this process earlier in this chapter.

The process of inserting a new entry into the Context Model is quite complex. The course taken differs considerably depending on whether an entry unifiable with the one to be inserted is already present in the context or not. If one is found, then all static clusters of which the entry is a member must have already been fetched from long term memory. Then the activation of the matching entry must be increased and the increase spreads to the other entries linked to it. If no matching entry is found then the appropriate clusters are fetched from LTM and their entries inserted. These processes were explained earlier in this chapter.

Diagram 1 describes the former process and diagram 2 describes the latter. Both diagrams are slightly simplified; the actual insertion process contains some additional checking to ensure that unnecessary work is not performed.

Diagram 1 – Matching Entry Found

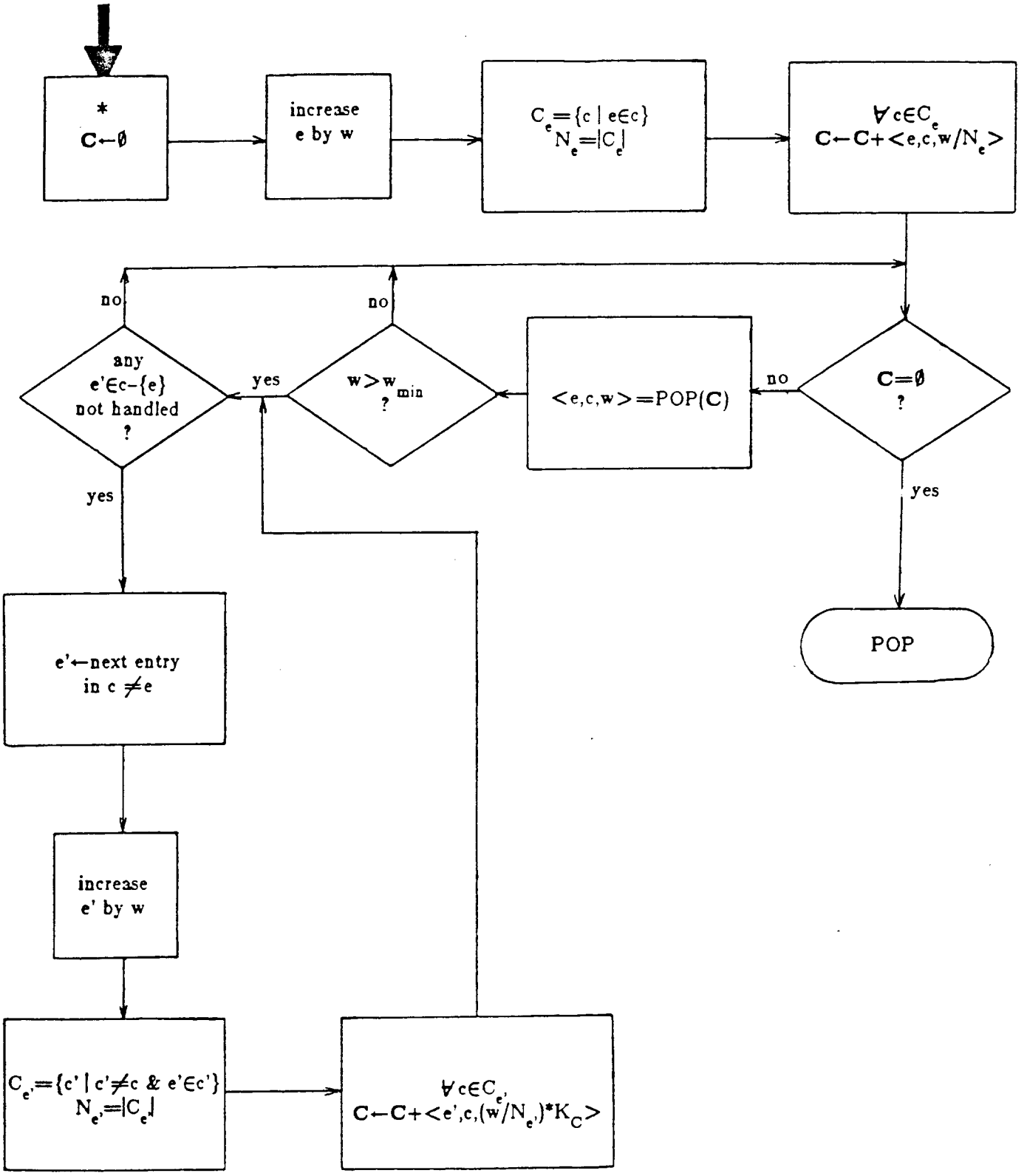
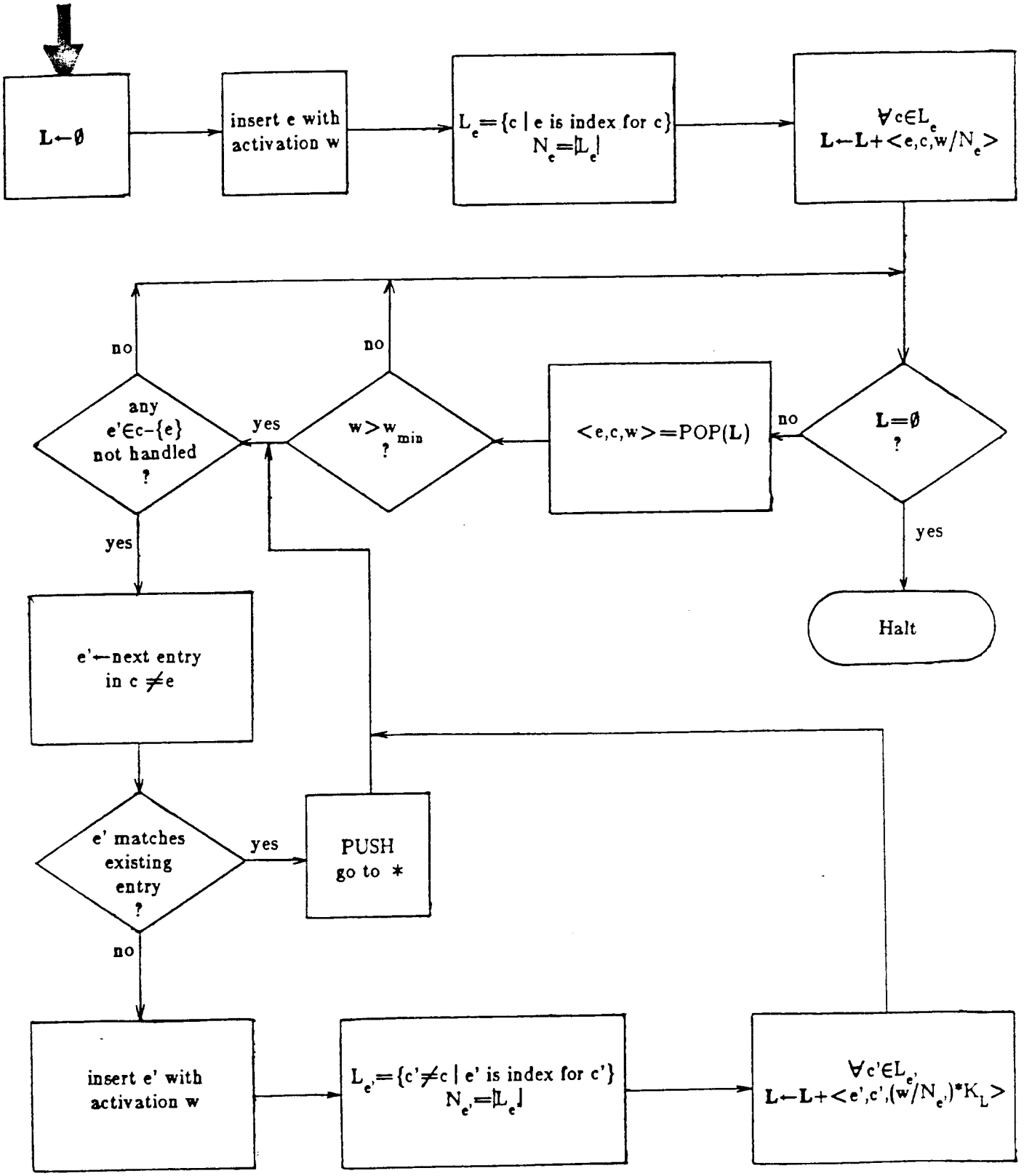


Diagram 2 – No Matching Entry Found



Notation:

The entry being inserted is 'e' and its weight, or base level of activation, is 'w'.

C is a FIFO queue of triples of the form <e,c,w>, each consisting of an entry e, a cluster c of which it is a member, and an activation level w. Each such triple represents a cluster that has had activation spread to it through entry e. The increase to each entry in c should be w. e is kept track of so that activation will not feed back into the cluster from which it came to that entry. The queue **C** is used to imitate parallel spreading of activation.

L is a queue similar to **C**, except that it is used to keep track of clusters that have been retrieved from LTM, where they were indexed under e.

POP is a function that returns the first triple in the queue, removing it in the process. For any set S, |S| is its cardinality.

w_{min}, K_L, and K_C, are constants of the Context Model. They are the minimum weight or increase in weight dealt with, and the factors by which activation dampens when spreading to an additional static or active cluster, respectively. $0<w_{min}<100$, $0<K_L,K_C<1$.

When entering the box labeled PUSH in diagram 2, control jumps to the box marked with a '*' in diagram 1. Upon entering the oval labeled POP, control returns to the original location and processing continues from there. A POP on level 0 brings the process to a halt, as does arriving at the position labeled Halt.

It should be noted that while the process of determining C_e requires only examination of the Context Model for all clusters containing the entry e, the

determination of L_e actually involves a process of unification, as described in the section pertaining to the fetching of clusters from long term memory (section IV.3.3.)

IV.7. Miscellaneous Comments

IV.7.1. Clusters and Scripts

A cursory examination might suggest that clusters are very much like scripts. This, however, is not accurate. Scripts exhibit a problem characteristic of most data structures used in language understanding programs: There are implicit, built in relationships among the conceptualizations contained in it. In the case of a script the relation is that of temporal ordering. The script structure is said to describe a sequence of actions which form a familiar and common chain of events, such as eating at a restaurant, or traveling to another city. The temporal ordering that is described is not fully represented in an explicit manner, but is assumed to be present from the fact that the structure is claimed to be a "script".

An inevitable result of such an approach is that one must have a special program to process scripts. This program has encoded into it the consequences of the ordering of the script's elements, and is able to make the appropriate inferences. A script must be identified as such to the program. The Script Analyzer Mechanism, SAM (Cullingford, 1978), is not capable of handling other similar data structures.

On the other hand, in the CLUSTER model information which is implicit in a script, i. e., that its elements are temporally ordered in a particular way, should be represented explicitly. A processor of greater generality would then handle these data structures just as it does many others. Since all relationships must be made explicit, the problem of representation becomes a major one. In fact, due to the lack of adequate facilities for representing temporal information in UC, it does not currently fully capture the order relationships in its equivalent of a script.

The cluster structure is also used to encode relationships other than of the type represented in a script.

IV.7.2. Reminding

The choice of which clusters to load into the Context Model is based on structural similarity between entries. This accounts for the Context Model's ability to be "reminded" of certain situations. For example, assume the following cluster is present in LTM:

```
{   demo.l is a file
    demo.l is owned by arens
    demo.l was printed on device LP
    LP is a line printer        }
```

Further, assume that it is indexed both under the entry describing the file "demo.l", and under the one describing the printing of demo.l.

In this case, any mention of the file "demo.l", or of printing it, will cause the activation of this whole cluster. However, under our assumptions about how the

cluster is indexed, the mention of printing another file will not cause the cluster to be recalled. Following recall, the Context Model again contains an entry denoting the fact that this file has been printed. This means that any other process that consults the Context Model for information it needs will have access to this information too. In other words, upon hearing the file demo.l mentioned, the system is reminded that it was printed.

Once new clusters are formed to represent interactions with the user, as described in section IV.5, the system may be reminded of these past events. UC is currently unable to generalize clusters created in the course of processing (see section IV.7.4 below). It can therefore only be reminded of clusters containing the specific objects and events in the created cluster, e. g., "the file demo.l" and "the printing of demo.l on device LP".

IV.7.3. Forgetting

The processing of an input sentence having multiple possible interpretations is costly in its effect on the activation of entries in the Context Model. This is due to the large number of entries representing the possible interpretations of text fragments. The insertion of each such entry causes the passage of an additional cycle of processing, bringing about further decay of the activation of all entries in CM. If pre-existing entries are not reinforced due to being related to the new input, their activation levels may be reduced to the extent that they will be removed from the Context Model. This will effectively cause them to be forgotten.

It remains to be seen whether human readers exhibit such behavior. It is, however, well known (e. g., Peterson and Peterson, 1959) that performance of tasks unrelated to items read earlier does interfere with their retention.

IV.7.4. Learning

An ability to generalize clusters is central to the ability of a CLUSTER model based language understanding system to learn. With this ability it would be possible to use information gathered about one vparticular situation when dealing with other similar situations. How this could be done, however, is a major unsolved problem. It is similar to the problem of generalizing MOPs in (Lebowitz, 1980), where a special case is solved. The generalization of several similar instances of a known MOP is made possible, creating a new MOP. The new MOP describes a subset of the situations described by the higher level MOP, where particular slots contain certain fixed fillers. The slots that may be fixed in such a generalization process must be designated beforehand by the system designer as permitting such a process.

Nothing equivalent exists in the CLUSTER system at this time. A major factor contributing to the difficulty of implementing such a process here is the lack of any way to note relationships *between clusters.* In particular, it is difficult to recognize that the current contents of the Context Model are a specific instance of some more general cluster. As a rule, the Context Model contains numerous entries left over from earlier exchanges, and it is not known how to determine which of these are germane to the current context, and which are not.

CHAPTER V

INTERACTION BETWEEN THE LANGUAGE ANALYZER AND THE CONTEXT MODELER

V.1. Introduction

As a separate module, PHRAN processes one sentence at a time, using its knowledge base of pattern-concept pairs to produce a representation of each sentence's meaning.

In a conversational context, PHRAN cannot treat sentences as independent entities. Rather, it must relate their content and import to information gathered during the processing of previous utterances. In particular, objects, actions, and any other concepts mentioned in or whose existence is implied by an utterance must be disambiguated, and their interpretations identified from among those concepts believed to be present in the context. For this purpose, the Context Model must be consulted during the processing of an utterance. Further, the results of this processing must in turn influence the contents and structure of the Context Model.

Achieving these goals required some changes in the design and operation of PHRAN.

This chapter first describes how PHRAN was modified to operate with the Context Model in the UNIX Consultant, and how it makes use of the information present in the Context Model in the course of processing input. Following that,

it describes how PHRAN helps construct the context while analyzing its input. Finally, I will briefly discuss how concurrency is achieved in the UNIX Consultant.

V.2. PHRAN Modified to Work with the Context Model

When working together with the Context Model, it is necessary for PHRAN to consider the contents of the Context Model and the activation levels of entries there. This information must taken into account while processing input. The information about the Context Model affects the choices of meanings for ambiguous words and phrases, the identities of objects mentioned in the text, and the referents of anaphoric expressions.

PHRAN and the processes operating in the Context Model communicate by using and modifying the same memory structures. In this sense, the Context Model acts like a *blackboard* (Lesser and Erman, 1977), in addition to serving as a model of the context of the conversation. When PHRAN recognizes the presence of a language pattern it creates a structure representing the meaning of the pattern, using the concept part of the appropriate pattern-concept pair. The structure created by PHRAN is passed on to the Context Model and becomes part of the entry representing the meaning of the original language pattern there. Any modifications to this structure, e. g., changes in its level of activation or changes that result from its unification with other entries, also cause immediate changes in the structure as viewed by PHRAN.

V.2.1. Ambiguous Phrases

When PHRAN processes an input sentence it often encounters ambiguous language fragments. Fragments are considered ambiguous when they are matched by more than one pattern, or when the pattern matching them has more than one concept part associated with it.

For example, the fragment "The Big Apple" is matched by the two patterns

1. "The <adjective> <noun>"
2. "The Big Apple"

The first is associated with the concept of a definite reference to the noun modified by the adjective, in this case to a certain large fruit of the genus *Malus*. The second pattern is associated with the concept of the City of New York.

When operating independently, PHRAN constructs a term with more than one associated meaning. The new term has all its possible meanings associated with it. These meanings are all considered equally possible until there is a basis for preferring one of them. This happens when the term fits into a larger pattern, and only one of its possible meanings is appropriate in that pattern. When such a situation arises, that particular meaning is chosen as the disambiguated meaning of the term.

For example, this is the case if the above fragment is read as part of the sentence

John ate the Big Apple

Here the larger pattern is

<person> <eat> <food>

and only the first interpretation of the fragment in question is an item of food.

If the larger pattern is such that more than one meaning fits it, one of the various meanings is arbitrarily chosen, and PHRAN proceeds with the analysis. If there is no larger pattern to use for the purpose of disambiguation, e. g., if the complete utterance typed in by the user is ambiguous, PHRAN makes no choice and displays all meanings as the result of the parse. The manner in which input is handled by stand-alone PHRAN is explained in detail in Chapter II.

While operating in conjunction with the Context Model PHRAN has additional information at its disposal. Namely, it can consult the activation levels of the various interpretations of the phrase in question. This ability is used, and makes a difference in two cases:

1. When more than one interpretation satisfies the condition in a pattern in which the term in question is a constituent.
2. When there is no pattern to provide a linguistic context for interpretation.

In both cases, instead of arbitrarily choosing an interpretation, PHRAN chooses the one with the highest activation level.

Example

When PHRAN operates independently, processing the phrase "The Big Apple" leads to a term with two possible meanings – a large apple and New York City. In that mode PHRAN is not able to make an intelligent choice between the two. For a human reader, however, a simple additional

sentence may be enough to establish a context that makes the correct interpretation obvious. For example, suppose a person is presented with the following two utterances:

1. John ate something.
2. The big apple.

Despite the fact that this pair of utterances is somewhat awkward, an intelligent reader could reasonably be expected to understand the second sentence as a specification of the object referred to as having been eaten in the first sentence.

When PHRAN operates in conjunction with the Context Model the above utterances are processed as follows. While processing the first utterance, the pattern "<person> <eat> <food>" is found to match the sentence. The concept of 'eating' is thus recognized as present and a cluster related to eating is retrieved and loaded into the Context Model. A simplified version of that cluster (e. g., without weights) is the following:

{ **?P** ingested **?O**
?P is animate
?O is food
?P is not hungry }

An entry representing an item of food is thus inserted in the Context Model, and is eventually identified with that "something" John ate, although its precise nature remains unknown. (This process is described in detail in the previous Chapter, section IV.3.3.)

When the second sentence is read and processed, PHRAN determines that it is ambiguous. Both meanings are passed to the Context Model and the

entries representing each of the two meanings are given an initial activation level of 50. In inserting these two possible meanings in the Context Model, the one representing a large apple is found to be unifiable with the entry describing **?O**, the object of Food. The activation of that entry is therefore increased to a higher level than that of the entry representing NYC, which does not match any other entry. The meaning of the latter sentence is therefore interpreted to be equivalent to "The apple which is large". Furthermore, that apple is identified as the object which John consumed. Note that this identification takes place even though the anaphor "something" was read *before* the phrase that described the real world object it refers to. A trace of the program running this example is provided in section VI.2.3.

V.2.2. Definite References

By itself, an analyzer like PHRAN is incapable of determining precisely which real world objects definite references refer to. This is because the understanding of most definite references inherently involves the use of information not present in the sentence being processed. When PHRAN operates independently, it merely creates a new token for each definite reference. For example, when the phrase "the large file" is encountered, PHRAN creates a term with a *CD-form* which is a new token, 'file27', say. This token has a list of semantic categories attached to it, including *file*, and *container*, and a notation indicating that it has been modified by the adjective 'large'. However, if it processes the same expression again, PHRAN will create an entirely new term to represent the

expression's meaning.

In a more realistic conversation, every such reference may be assumed to have an interpretation consisting of a concept already represented by some entry in CM. To accommodate this assumption, the reference is analyzed by PHRAN into a term that is inserted into the Context Model in the form of an entry of type *refdef*. As explained in Chapter IV, this is an entry of class *intention*, which causes the Monitor to compare it to all entries of class *object* in an attempt to find a match. The Monitor uses an object-matching function which determines a ranking of the degree of fit between the 'refdef' and each of the 'object's. This function is described in section IV.2.2. The object that matches best is chosen as the interpretation of the original 'refdef' entry.

The relative activation levels of the various 'object's are influential in determining how good a match is, with a higher activated entry being preferred to a lesser activated one, all other things being equal. Adjectives used in the description of the original object and in the referring phrase also influence the comparison. In a manner similar to what takes place in the case of ambiguity, the structure representing the meaning of the entry of class 'object' replaces the structure of type 'refdef', causing the meaning of the reference as viewed by PHRAN to be its new interpretation in the Context Model.

The treatment of pronouns and demonstratives is similar.

Example

Consider the following situation. A user of UC is in a directory including several files, some of them executable. The user asks the UC system:

User: List all files of form Lisp.*

to which the system responds:

UC: Lisp.l Lisp.pat Lisp.exe

Now suppose the user types

User: How big is the executable file?

Assume that only 'Lisp.exe' is executable. UC must determine which file is being referred to in order to respond to the question. PHRAN initially analyzes the fragment "the executable file". This causes to be inserted into the Context Model an entry of type *refdef* which describes a file that is executable. The Monitor, when it inspects the Context Model, attempts to match this entry to all others representing objects. There are likely to be numerous such entries, representing the files in the directory, several of which will be executable. The ones which match the pattern Lisp.*, however, are much more highly activated, having just been mentioned. Since there is only one executable file among those, it is the best match and the 'refdef' entry is replaced with it.

Suppose that instead of the question above, the user asks the following:

User: How big is the *large* executable file?

If we further assume that there is a single large executable file in this directory and that it is not Lisp.exe, then the Monitor will identify that file with the 'refdef'. This file will be considered the referent instead of the more recent executable one, since the reference explicitly describes a *large* file,

and consequently the match will be closer.

The behavior described above is not always observed, however. This happens since both the closeness of the descriptions and the activation level of the candidate objects are taken into account in calculating the degree of matching. The example above was given under the assumption that all the entries representing the files in the directory are moderately activated, so that the addition of the close description would be enough to overcome the contribution to the degree of matching of the very high activation of the file Lisp.exe. If, on the other hand, the other files are only very slightly activated, the system will still prefer the only executable file which appears relevant – Lisp.exe. Such behavior is a feature of human discourse and it is desirable that the Context Model imitates it. In practice, however, fine tuning the activation thresholds and transfer mechanism so that the intended interpretation is always arrived at is difficult. The difficulty involved is probably due to the inadequacy of the current model of activation transfer. More work towards refining the model is needed. However, even at this stage it is very rare for the Context Model to come up with a completely unreasonable intepretation of a reference.

Sometimes no entry is found that matches a definite reference. For example, no appropriate objects may exist in the Context Model, or, more likely, the degree of match may be too low. In these cases the Monitor will create a new entry of class *object* and endow it with the properties of the object presumably being referred to. These properties are determined by examining the referring

expression and any phrasal patterns of which it is a part.

For a traces of the UC program interpreting referring expressions see Chapter VI. In particular, consider the treatment of "the file fetch.l" and the pronoun "it" in the example in section VI.2.2.

V.3. How PHRAN Helps Construct the Context Model

With the exception of decay of activation levels, no activity takes place in the Context Model unless triggered by the introduction of a new entry. If no new entries are inserted, the activation levels of existing entries eventually drop to the level where they are deleted. Processing that leads to retrieval of clusters, inferences, interpretation of references, and to responses, is initiated by outside "stimuli". In the current configuration, a natural language analyzer working together with a memory mechanism, outside stimuli can only be provided through the interpretation of the user's utterances.

In this section, I describe how information extracted by PHRAN when it analyzes input is used to produce new entries for the Context Model, and how their activation level is set. When operating in conjunction with the Context Model, PHRAN relinquishes control whenever a pattern has been recognized in the input. Just prior to doing so, it passes on the concepts found in the text for insertion in the Context Model. These are the contents of the concept part of the pattern-concept pair that matched the input fragment. The nature of these concepts determines the precise types of entries that are formed, and their number determines the amount of initial activation given to each. The entries formed

will be described in the following four sections. How their initial level of activation is determined will be explained in the last section. The insertion of these new entries triggers all the processing described in Chapter IV.

V.3.1. Types of Entries Created

After a new term is created by PHRAN, an entry is formed for each of the term's possible meanings. Each entry is inserted into the Context Model. The precise entries created are a function both of the syntactic properties of the language fragment and the semantics of the concept identified from it.

For every concept conveyed by the input, an appropriate entry must be constructed. However, no simple correspondence exists between the representation of concepts in PHRAN and the types of entries in the Context Model. This is due to the fact that a satisfactory general representation language has yet to be developed. Both in PHRAN and in the Context Model new representational classes are added as needed, often in an *ad hoc* fashion and without maintaining consistency with the other system. When enough becomes known about knowledge representation it should be possible to include the Context Model information in the PHRAN concepts. This would be more consistent with the intention behind the definition of the pattern-concept pair, and would eliminate the need for this translation process altogether.

Presently the system designer must define which concepts translate into which entries. For most concepts, however, the process follows simple general rules. Below I provide several examples of entry formation. Each example illus-

trates one of the general cases. Together these cases cover the great majority of expressions encountered by UC.

As a rule, nouns and noun phrases give rise to entries of the class *object*, and complete sentences, relative clauses, etc., give rise to entries of the class *assertion*. Adjectives, prepositional phrases, and other types of structures are ignored, since in PHRAN they are either integrated into larger patterns or else they are used to modify the meaning of nouns or noun phrases directly.

The major exception to this rule are the definite references. Pronouns and demonstratives cause the creation of entries of type *refdef*. These are of the class *intention*.

Following are several examples of typical entries created during the PHRAN analysis process.

V.3.2. Noun Phrases – Objects

When PHRAN reads the phrase "a file", for example, it finds that the phrase matches a known pattern-concept pair. The associated concept part is used to create a token, say 'file14', for the file mentioned, and a term with a single meaning component (assuming for the moment only one meaning of *file* is known to the system). The term has the following properties:

```
p-o-s   noun-phrase
ref   indef
cd-form   file14
description   (file container physob)
```

This term is used, in turn, to form the following Context Model entry of type *mention*, which is an *object* class entry.

```
(mention (activation 100)
         (cd-form file14)
         (meaning ((p-o-s noun-phrase)
                   cd-form file14
                   description (file container physob)))))
```

This entry will be inserted in the Context Model, at which time an attempt will be made to unify it with other entries there. The existence of this entry indicates that the system recognizes the existence of a new object in the context. This object has the properties inherited from the representation provided by PHRAN, and receives a high level of activation since it was explicitly mentioned in a user's utterance.

V.3.3. Sentences – Assertions

When reading the question "How do I delete a file?" a term is created by PHRAN representing the meaning of the question. This term has a meaning component with the following properties:

```
p-o-s  sentence
cd-form  (planfor (result (causation
                           (antecedent (do (actor *ego*))
                           (consequent (state-change
                                        (actor file14)
                                        (state-name physical-state)
                                        (from existing)
                                        (to non-existing)))))
                  (method *unknown*))
actor *ego*
file file14
question  t
```

When this term is passed on to the Context Model it causes the creation and insertion of the following entry of type *assertion* .

```
(question (activation 100)
          (cd (planfor
                (result (causation
                          (antecedent (do (actor *ego*))
                          (consequent (state-change
                                        (actor file14)
                                        (state-name physical-state)
                                        (from existing)
                                        (to non-existing)))))
                (method *unknown*))))
```

The presence of this entry in the Context Model indicates the fact that the user has made a statement expressing a question. The user's desire was to know what method to use in order achieve the result of deleting a file. An entry of type *question* was created since PHRAN explicitly indicated that that was its interpretation of the user's utterance. If PHRAN interprets a user's input as an imperative, or as a simple assertion, an entry of the corresponding type will be created instead.

Once the entry described above has been inserted in the Context Model, the Monitor will become aware of the desire of the user and will attempt to find an answer to the question.

V.3.4. References – Refdefs

The processing of the expression "The large file" by PHRAN will cause the creation of a temporary token, say file15, and the formation of a term with the following meaning component:

```
p-o-s  noun-phrase
ref  def
cd-form  file16
description  (file container physob)
adjs  ((size large))
```

The discovery of this term in the text will cause the formation of an entry of type *refdef* and its insertion into the Context Model.

```
(refdef (activation 100)
        (meaning ((p-o-s noun-phrase
                   ref def
                   cd-form file16
                   description (file container physob)
                   adjs ((size large))))))
```

As we see, the entry includes a pointer to the original meaning representation of the term. This structure will be modified if and when a matching existing entry of type *object* is found, as explained earlier in this chapter. This entry is of the class *intention*, as are all *refdef*s. The appropriate matching will be performed by the Monitor when the contents of the Context Model are inspected by it.

V.3.5. Assignment of Activation Level

A concept unambiguously referred to in the input gives rise to an entry in the Context Model with the highest possible level of activation. That is, if only a single pattern is matched by the input and if there is only one concept part associated with it, then the entry created on the basis of this concept will be given an activation level of 100.

If there is more than one concept part associated with the pattern, say **n**, then an entry is created for each. They all receive an equal amount of initial

activation, i. e., 100 / **n**.

V.4. Simulated Concurrency

One of the basic assumptions underlying the view of understanding presented in this thesis is that the manipulation of the context happens concurrently with other understanding processes that are taking place. Accordingly, PHRAN's operation should in principle be taking place in parallel with the construction of the Context Model. This could not be done in a truly concurrent manner with the computing facilities available. Instead, the standard compromise was achieved, with control passing back and forth between PHRAN and the Monitor in CM. In order to do this in a manner as close to concurrent as possible, while at the same time acting in an intuitively reasonable way, it was necessary to determine minimal "indivisible" steps in PHRAN and in the operation of the Context Model. Control is to be relinquished at the end of each such step.

A solution to this problem is provided by the structure of PHRAN's knowledge base, i. e., the pattern-concept pair. As explained in Chapter II, a pattern-concept pair describes the relationship between a particular linguistic pattern and its meaning. Consequently, the patterns parts of pattern-concept pairs match semantically meaningful fragments of text. It is thus only natural that for our purposes here we choose the matching of a pattern as indicative of the completion of such a minimal "indivisible" step. There is also psychological evidence that the recognition of the type of pattern used by PHRAN is followed by a pause in the reading of the input sentence (Just and Carpenter, 1980) (Car-

penter and Daneman, 1981) (McDonald and Carpenter, 1981).

Together with the passing of control, PHRAN passes the entry representing the meaning of the matched pattern to the Monitor. The new entry is then inserted in the Context Model.

The Context Model lacks the kind of structure present in the database of PHRAN, and a different approach was taken with it. The processes in the Context Model themselves simulate parallelism – the spreading of activation and the insertion of entries are supposed to be taking place concurrently. It would therefore make no sense to interrupt the processes taking place in the Context Model until all activity in it ceases. In other words, once a change is made in the context and control passes to the Monitor, control remains there until activation stabilizes again throughout the network of entries that constitutes the Context Model.

To summarize: in order to simulate concurrency, control passes back and forth between PHRAN and the Context Model's Monitor. Whenever PHRAN finds the end of a pattern in the text, an entry representing the meaning of the matched pattern, as embodied in the concept part of the appropriate pattern-concept pair, is inserted in the Context Model and control is relinquished. When activity in the Context Model eventually subsides, control is returned to PHRAN, which then continues processing the input.

CHAPTER VI

SESSIONS WITH THE UNIX CONSULTANT

VI.1. State of the Program

This section (and this thesis as a whole) discusses only one of several currently existing implementations of the UNIX Consultant system. Several people at the University of California at Berkeley have modified and extended the original implementation, and have created versions that operate differently from the one described here.

VI.1.1. Current Capabilities of the UC Program

In its current state, the version of UC described here handles routinely all tasks involving simple interfacing between the Context Model and the Phrasal Analyzer. This includes:

- Referent identification. This is performed both for definite references and pronouns, and works whether the referring expression precedes or succeeds the referent. (See section VI.2.3. for an example of the latter, and VI.2.2. for the former).
- Context based disambiguation. The Context Modeler is able to make an appropriate context induced choice among multiple meanings of expressions. (See section VI.2.3. for an example.)

The ability of the program to behave in ways that involve more complicated reasoning is more limited. By "complicated reasoning" I mean situations where an entry necessary for the triggering of some behavior can be found only after a chain of clusters is recalled. An example of such a chain is provided by the example in section III.6. In that example, a statement by the user that some task cannot be performed is associated with a request for help, which is associated with an intention on the part of UC to find the reason, which is associated with an intention to check the prerequisites for the action. The failure of one of the prerequisites is then associated with a statement to that effect which is eventually output. The difficulties involved in such processes are discussed below. However, once they are overcome for a single example, all examples involving a similar chain of reasoning work as well. For example, adding the right clusters to permit UC to respond correctly to the statement in section III.6. took a whole afternoon, but making UC respond correctly also to "I can't remove the file foo" took only several more minutes. Specifically, it required adding only a single cluster describing the preconditions of the command 'rm'.

In sum, UC can currently respond to any number of question and statements of the forms of the samples presented in section I.2. and in the examples later in this chapter. While doing so, UC can perform the disambiguation tasks described above. The number of different types of questions and more involved interactions that UC can perform is limited however to those illustrated in this thesis.

VI.1.1.1. Technical Data

The UNIX Consultant system is running on a VAX 11/780 at the University of California at Berkeley, and is written in FRANZ LISP. The number of clusters in the UC system is currently 75. UC is capable of answering questions relating to the UNIX file system, and has knowledge of common file-related operations such as copying, editing, printing, etc. Processing time for questions ranges from 2 to 6 seconds of CPU time, and averages about 4 CPU seconds.

VI.1.2. Prospects and Difficulties of Extending the Program

It is inherently difficult to anticipate the effects of spreading activation from numerous sources through a network. The problem becomes exponentially more difficult as the size of the network increases. Ascertaining that the system behaves as desired after increasing its size involves, therefore, much testing and revision; it is time consuming and tedious.

Extending the program to handle a new type of input requires writing clusters describing previously non-existent associations. The system designer must decide what each cluster should contain and what the salience of each entry should be. It is impossible to anticipate what the precise effect of some entry being present, or of it having one particular level of salience or another, will have in every context in which such a cluster may be retrieved. An incorrect decision concerning the content of a cluster may prevent a necessary entry from being retrieved, or may cause too many clusters to be retrieved. An incorrect decision concerning the level of activation may cause entries not to be inserted even

though the cluster containing them was retrieved or to be dropped from the Context Model prematurely. An extension to the program thus involves numerous trials with different cluster contents and salience levels.

Consequently, a program based on the CLUSTER model will be successful in a domain where the number of different types of interactions between the user and the system is limited. Extending the system simply to handle different subject matter is not difficult.

VI.2. Running Examples

This section contains traces of several sessions with the UNIX Consultant. I have omitted a trace of the operation of PHRAN since the input sentences are relatively simple. The trace displays the entries formed as a result of PHRAN's analysis, the clusters retrieved based on them, entries inserted in the Context Model, and most other facets of the Context Modeler. When relevant, the contents of the Context Model are also printed.

In these examples the text output by the program is "canned". Input from the user and the output of the program are set in bold type. UC's processing of each input sentence ends with it printing the representation of the meaning of the input.

VI.2.1. Request for Help

User: How do I delete a file?
UC: Use the command 'rm'.

In this example the user's question is interpreted by UC as a request for the name of the appropriate command. A cluster exists in which the association between the deletion of a file and the appropriate command is encoded. This cluster is used in order to produce the answer.

 # How do I delete a file?

- The first fragment for which the program creates an entry is the word 'I'. It is inserted.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 100)
               (cd *ego*)
               (meaning
                 ((p-o-s noun-phrase cd-form *ego* person first
                   number singular description (person)))))

No matching assertion.


Monitor -- inspecting context-model
```

- Next, an entry is formed to represent the result of analyzing the words 'a file.'

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 100)
               (cd file8)
               (meaning
                 ((cd-form file8 ref indef p-o-s noun-phrase description (file container physob)
                   %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2))))))

No matching assertion.


Monitor -- inspecting context-model
```

- An entry is formed representing the meaning of the fragment '[I] delete a file'.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (assertion (activation 100)
                 (cd
                   (causation (antecedent (do (actor *ego*)))
                              (consequent
                                (state-change (actor file8) (state-name physical-state)
                                              (from existing) (to non-existing)))))))

No matching assertion.


Monitor -- inspecting context-model


Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (question (activation 100)
                (cd
                  (planfor (result
                             (state-change (actor file8) (state-name physical-state)
```

```
                                    (from existing) (to non-existing)))
                          (method *unknown*))))
```

No matching assertion.

- At this point PHRAN has completed the analysis of the input. It is interpreted as a request by the user for the identity of a method for making an existing file no longer exist.
- Next, clusters associated with the above question are retrieved from long term memory. They include a cluster associating the question with an intention of UC to find a plan with the desired effect and describe it to the user.

```
Adding associated clusters.  (base level -- 50)
      (((output (activation 100) (text ?any)))
       ((out-planfor (activation 100)
                     (cd
                       (planfor (result
                                  (state-change (actor file8) (state-name physical-state)
                                                (from existing) (to non-existing)))
                                (method *unknown*))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (output (activation 100) (text ?any))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (out-planfor (activation 100)
                   (cd
                     (planfor (result
                                (state-change (actor file8) (state-name physical-state)
                                              (from existing) (to non-existing)))
                              (method *unknown*))))

No matching assertion.


Adding associated clusters.  (base level -- 50)
      (((output (activation 100) (text |Use the command 'rm'.|)))))
```

- A cluster exists which associates the intention to find and express the plan for removing a file with an intention to output the appropriate explanation to the user. Its new element will be inserted next, eventually causing UC to output this explanation.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (output (activation 100) (text |Use the command 'rm'.|))

Matching assertion found.
(output (activation 25) (text |Use the command 'rm'.|) (parents ...))
Reenforcing from it


Monitor -- inspecting context-model
```

Use the command 'rm'.

VI.2.2. Context Storing and Recall, Referent Identification

User: How do I print the file fetch.l on the line printer?
UC: Use the command 'lpr'.
.
.
.
User: Has the file fetch.l been printed yet?
UC: The file fetch.l is in the line printer queue.
User: How can I cancel it?
UC: Use the command 'lprm arens'.

This is a trace of the example outlined in section I.6. It demonstrates the creation of new clusters to permit later recall of a context (section IV.5.), and the identification of the pronoun "it" in circumstances where otherwise this would be impossible. After the first question and answer the context is stored and the Context Model is emptied. Both operations are performed by the user manually.

 # How do I print the file fetch.l on the line printer?

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 100)
               (cd *ego*)
               (meaning
                 ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)))))
No matching assertion.

Monitor -- inspecting context-model

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 100)
               (cd fetch.l)
               (meaning
                 ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
                   %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2))))))
No matching assertion.

Monitor -- inspecting context-model

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (refdef (activation 100)
              (meaning
                ((cd-form lineprinter1 p-o-s noun-phrase ref def description (lineprinter printer physob)
                  %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2))))))
```

```
No matching assertion.

Monitor -- inspecting context-model
```

- The Monitor notices that no object currently exists in the Context Model that might be the referent of the expression "the line printer". Consequently, it will replace the 'refdef' with an entry representing a line printer about which nothing else is currently known.

```
Warning: No good referent for
(refdef (activation 100)
        (meaning
          ((cd-form lineprinter1 p-o-s noun-phrase ref def description (lineprinter printer physob)
            %do ((add-adjs-to-*sc* & &) (copy-term 2)) activation (activation 100)))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (assertion (activation 100)
                (cd (causation (antecedent (do (actor *ego*)))
                               (consequent
                                 (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))

No matching assertion.
```

- The fragment "I print the file fetch.l on the line printer" is analyzed by PHRAN and recognized as asserting the existence of a possible printing event. This assertion triggers the recall of a cluster that associates with it a printing event as a 'mention', i. e., as something that may be referred to.

```
Adding associated clusters.  (base level -- 100)
        (((mention (activation 75)
                   (cd printing-event)
                   (meaning
                     ((cd-form printing-event p-o-s noun-phrase description (mentob command printing)))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (mention (activation 75)
                 (cd printing-event)
                 (meaning
                   ((cd-form printing-event p-o-s noun-phrase description (mentob command printing))))))

No matching assertion.

Monitor -- inspecting context-model
```

- The analysis of the input by PHRAN is complete. It is interpreted as a request for a method for "moving" the file fetch.l to the printer.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (question (activation 100)
                  (cd (planfor (result
                                 (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                               (method *unknown*)))))

No matching assertion.
```

- Next, clusters associated with the above question are retrieved from long term memory. As in the previous example, they include a cluster associating the question with an intention of UC to find and express a plan with the desired effect.

```
Adding associated clusters.  (base level -- 50)
        (((output (activation 100) (text ?any)))
         ((out-planfor (activation 100)
```

```
                (cd (planfor (result
                              (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                             (method ?method))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (output (activation 100) (text ?any))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (out-planfor (activation 100)
                   (cd (planfor (result
                                 (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                                (method ?method))))

No matching assertion.
```

- The intention to describe this plan is associated with an intention to output the approporiate text.

```
Adding associated clusters.  (base level -- 50)
      (((output (activation 100) (text |Use the command 'lpr'.|)))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (output (activation 100) (text |Use the command 'lpr'.|))

Matching assertion found.
(output (activation 25) (text |Use the command 'lpr'.|) (parents ...))
Reinforcing from it


Monitor -- inspecting context-model
```

Use the command 'lpr'.

- Below are the current contents of the Context Model:

```
# (pprint *context-model*)
((question (activation 98) (cd (planfor (result (ptrans & & & &)) (method *unknown*))) (clusters ...))
 (assertion (activation 96) (cd (causation (antecedent (do &)) (consequent (ptrans & & & &)))) (clusters ...))
 (mention (activation 95)
          (meaning
            ((cd-form printing-event p-o-s noun-phrase description (mentob command printing))))
          (cd printing-event) (parents ...))
 (mention (activation 92)
          (cd fetch.l)
          (meaning
            ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
              %do (& &) activation (activation 92)))))
 (mention (activation 90)
          (cd *ego*)
          (meaning
            ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)
              activation (activation 90)))))
 (output (activation 98)
         (text |Use the command 'lpr'.|) (parents ...) (done yes))
 (out-planfor (activation 33)
              (cd (planfor (result (ptrans & & & &)) (method ?method)))
              (parents ...) (clusters ...) (done yes)))
```

- Following an explicit command, the contents of the Context Model are stored in the form of a new cluster. This will permit their later recall. The retrieval keys are listed. The process of constructing new clusters in this manner is described in section IV.5.

```
# (store-context)

Storing context;  Indexed under --
   (question (activation 49)
             (cd (planfor (result
                            (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                          (method *unknown*))))
   (assertion (activation 48)
              (cd (causation (antecedent (do (actor *ego*)))
                             (consequent
                               (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))
   (mention (activation 47)
            (cd printing-event)
            (meaning
              ((cd-form printing-event p-o-s noun-phrase description (mentob command printing)))))
   (mention (activation 46)
                          (cd fetch.l)
            (meaning
              ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
                %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2)) activation (activation 46)))))
   (mention (activation 45)
            (cd *ego*)
            (meaning
              ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)
                activation (activation 45)))))
nil
```

- Next, the Context Model is emptied. This simulates the passage of time and the continuation of the conversation along lines irrelevant to the previous exchange.

```
# (setq *context-model* nil)

nil
```

- The conversation continues with a new question by the user. This question, however, makes reference to a file and an event which took place earlier.

 # Has the file fetch.l been printed yet?

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 100)
               (cd fetch.l)
               (meaning
                 ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
                   %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2))))))

No matching assertion.
```

- Although there is no longer any entry representing the file fetch.l in the current Context Model, there is a cluster in long term memory with which it is associated. This cluster essentially describes the earlier exchange.

```
Adding associated clusters.  (base level -- 100)
      (((question (activation 49)
                  (cd (planfor (result
                                 (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                               (method *unknown*))))
        (assertion (activation 48)
                   (cd
                     (causation (antecedent (do (actor *ego*)))
                                (consequent
                                  (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))
        (mention (activation 47)
```

```
            (cd printing-event)
            (meaning
              ((cd-form printing-event p-o-s noun-phrase description (mentob command printing)))))
    (mention (activation 45)
            (cd *ego*)
            (meaning
              ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)
                activation (activation 45)))))
    (out-planfor (activation 16)
            (cd (planfor (result
                          (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                         (method ?method))))))
```

- Each of the entries in the above cluster is inserted in turn. Most of them cause the retrieval of the same cluster again, since it was indexed under them too. A process of mutual reinforcement follows after which the activation levels in the Context Model finally stabilize.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
    (question (activation 49)
            (cd (planfor (result
                          (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                         (method *unknown*)))))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 100)
    (assertion (activation 48)
            (cd
              (causation (antecedent (do (actor *ego*)))
                         (consequent
                           (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 100)
    (mention (activation 47)
            (cd printing-event)
            (meaning
              ((cd-form printing-event p-o-s noun-phrase description (mentob command printing)))))

No matching assertion.


Adding associated clusters.  (base level -- 47)
    (((question (activation 49)
            (cd (planfor (result
                          (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                         (method *unknown*)))))
     (assertion (activation 48)
            (cd
              (causation (antecedent (do (actor *ego*)))
                         (consequent
                           (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))
     (mention (activation 46)
            (cd fetch.l)
            (meaning
              ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
                %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2))
                activation (activation 46)))))
     (mention (activation 45)
            (cd *ego*)
            (meaning
```

```
                ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)
                  activation (activation 45)))))
        (out-planfor (activation 16)
                (cd (planfor (result
                              (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                             (method ?method))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (mention (activation 45)
                 (cd *ego*)
                 (meaning
                   ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)
                     activation (activation 45)))))

No matching assertion.


Adding associated clusters.  (base level -- 45)
        (((question (activation 49)
                    (cd (planfor (result
                                  (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                                 (method *unknown*))))
          (assertion (activation 48)
                     (cd
                       (causation (antecedent (do (actor *ego*)))
                                  (consequent
                                    (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))
          (mention (activation 47)
                   (cd printing-event)
                   (meaning
                     ((cd-form printing-event p-o-s noun-phrase description (mentob command printing)))))
          (mention (activation 46)
                   (cd fetch.l)
                   (meaning
                     ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
                       %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2)) activation (activation 46)))))
          (out-planfor (activation 16)
                       (cd (planfor (result
                                     (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                                    (method ?method))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (out-planfor (activation 16)
                     (cd (planfor (result
                                   (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                                  (method ?method)))))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 47)
        (question (activation 49)
                  (cd (planfor (result
                                (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                               (method *unknown*)))))

Matching assertion found.
(question (activation 24)
          (cd (planfor (result
                        (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                       (method *unknown*)))
          (parents ...))
Reinforcing from it
```

```
Inserting in *CONTEXT-MODEL*:  (base level -- 47)
        (assertion (activation 48)
                (cd (causation (antecedent (do (actor *ego*)))
                               (consequent
                                 (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))

Matching assertion found.
(assertion (activation 23)
          (cd (causation (antecedent (do (actor *ego*)))
                         (consequent
                           (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))))
          (parents ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 47)
        (mention (activation 46)
                 (cd fetch.l)
                 (meaning
                   ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
                     %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2)) activation (activation 46)))))

Matching assertion found.
(mention (activation 100)
          (cd fetch.l)
          (meaning
            ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
              %do ((add-adjs-to-*sc* & &) (copy-term 2)) activation (activatio.. 46))))
          (clusters ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 47)
        (mention (activation 45)
                 (cd *ego*)
                 (meaning
                   ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)
                     activation (activation 45)))))

Matching assertion found.
(mention (activation 20)
          (cd *ego*)
          (meaning
            ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)
              activation (activation 45))))
          (parents ...) (clusters ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 47)
        (out-planfor (activation 16)
                     (cd (planfor (result
                                    (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                                  (method ?method)))))

Matching assertion found.
(out-planfor (activation 2)
            (cd (planfor (result
                           (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                         (method ?method)))
            (parents ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 45)
```

```
(question (activation 49)
        (cd (planfor (result
                      (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                     (method *unknown*)))))

Matching assertion found.
(question (activation 41)
        (cd (planfor (result
                      (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                     (method *unknown*)))
        (parents ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 45)
      (assertion (activation 48)
                 (cd
                   (causation (antecedent (do (actor *ego*)))
                              (consequent
                                (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))

Matching assertion found.
(assertion (activation 39)
        (cd (causation (antecedent (do (actor *ego*)))
                       (consequent
                         (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))))
        (parents ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 45)
      (mention (activation 47)
               (cd printing-event)
               (meaning
                 ((cd-form printing-event p-o-s noun-phrase description (mentob command printing)))))

Matching assertion found.
(mention (activation 22)
        (cd printing-event)
        (meaning
          ((cd-form printing-event p-o-s noun-phrase description (mentob command printing))))
        (parents ...) (clusters ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 45)
      (mention (activation 46)
               (cd fetch.l)
               (meaning
                 ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
                   %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2)) activation (activation 46)))))

Matching assertion found.
(mention (activation 100)
        (cd fetch.l)
        (meaning
          ((cd-form fetch.l p-o-s noun-phrase description (file container physob)
            %do ((add-adjs-to-*sc* & &) (copy-term 2)) activation (activation 46))))
        (clusters ...) (parents ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 45)
      (out-planfor (activation 16)
                   (cd (planfor (result
```

```
                                (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                               (method ?method))))

Matching assertion found.
(out-planfor (activation 8)
             (cd (planfor (result
                            (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))
                           (method ?method)))
             (parents ...))
Reinforcing from it


Monitor -- inspecting context-model
```

- The retrieval by its several keys of the cluster formed after the first exchange is complete. PHRAN proceeds to analyze the input question to its end. The input is understood as a request for information about the status of the printing event.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (question (activation 100)
                (cd (current-status (event
                                      (causation (antecedent
                                                   (do (actor *unspecified*)))
                                                 (consequent
                                                   (ptrans (actor *Unix*) (object fetch.l)
                                                           (from *unspecified*) (to printer)))))
                                    (status *unknown*))))

No matching assertion.
```

- The request is associated with, among other things, an intention on the part of UC to determine and inform the user of the status of the event.

```
Adding associated clusters.  (base level -- 50)
      (((output (activation 100) (text ?any)))
       ((out-status (activation 100)
                    (cd
                      (current-status-of
                        (action
                          (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))
        (output (activation 100) (text ?any))
        (mention (activation 50)
                 (cd
                   (current-status-of
                     (action
                       (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (output (activation 100) (text ?any))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (out-status (activation 100)
                  (cd
                    (current-status-of
                      (action
                        (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
```

```
(output (activation 100) (text ?any))

Matching assertion found.
(output (activation 25) (text ?any) (parents ...))
Reinforcing from it


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (mention (activation 56)
               (cd
                 (current-status-of
                   (action
                     (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer))))))

No matching assertion.
```

- The mention of the status of this event causes the retrieval of a cluster which associates it with the intention of running a certain UNIX command. The Monitor eventually interprets this intention and runs the command.

```
Adding associated clusters.  (base level -- 28)
      (((run (activation 100) (use-command (command (unix (run lpq) (find fetch.l))))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 28)
      (run (activation 100) (use-command (command (unix (run lpq) (find fetch.l))))))

No matching assertion.


Monitor -- inspecting context-model
```

- The Monitor executes the required command an inserts an entry in the Context Model representing the intention to inform the user of the result.

```
Call to Unix:  Check if 'fetch.l' is in output of 'lpq' -- True


Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (output (activation 25) (text |The file fetch.l is in the line printer queue.|))

Matching assertion found.
(output (activation 62) (text |The file fetch.l is in the line printer queue.|) (parents ...) (done no))
Reinforcing from it


The activation level of the following has dropped below cutoff level.
It is being removed from *CONTEXT-MODEL* --
(mention (activation 6)
         (cd
           (current-status-of
             (action
               (ptrans (actor *Unix*) (object fetch.l) (from *unspecified*) (to printer)))))
         (parents ...) (clusters ...))

The activation level of the following has dropped below cutoff level.
It is being removed from *CONTEXT-MODEL* --
(run (activation 6) (use-command (command (unix (run lpq) (find fetch.l)))) (parents ...) (done yes))

Monitor -- inspecting context-model
```

The file fetch.l is in the line printer queue.

- The user asks another question:

 # How can I cancel it?

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (mention (activation 100)
                 (cd *ego*)
                 (meaning
                   ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)))))
```

- The mention of the user ('I') is identified with an existing entry in the Context Model.

```
Matching assertion found.
(mention (activation 33)
         (cd *ego*)
         (meaning
           ((p-o-s noun-phrase cd-form *ego* person first number singular description (person))))
         (parents ...) (clusters ...))
Reinforcing from it

Monitor -- inspecting context-model
```

- PHRAN reads the words 'it' and interprets it as a reference either to some object or to some mental object. Due to this ambiguity two terms are created. The semantic category 'command' is added to the 'description' property of each, since that is what is required in the active phrasal pattern. This will not necessarily prevent matching with something that is not a command (see below), but it will cause an entry that does not represent a command to be a poor match. ·

```
Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (refdef (activation 100)
                (meaning
                  ((p-o-s noun-phrase cd-form object1 description (physob command))
                   ...)))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (refdef (activation 100)
                (meaning
                  ((p-o-s noun-phrase cd-form mobject1 description (mentob command)))))

No matching assertion.


Monitor -- inspecting context-model

Found referent for
(refdef (activation 25)
        (meaning
          ((p-o-s noun-phrase cd-form object1 description & activation &)
           ...)))
in the mention
(mention (activation 97)
         (cd fetch.1)
         (meaning
           ((cd-form fetch.1 p-o-s noun-phrase description & %do & activation &)))
         (clusters ...) (parents ...))
Found referent for
(refdef (activation 25)
        (meaning ((p-o-s noun-phrase cd-form mobject1 description & activation &))))
```

```
in the ment
(mention (activation 95)
        (cd printing-event)
        (meaning ((cd-form printing-event p-o-s noun-phrase description &)))
        (parents ...) (clusters ...))
```

- UC determines that a reference to a real object would be to the file fetch.l, and a reference to a mental object must be to the printing of the file. The Monitor determines this by comparing the 'refdef's to the 'mention's existing in the Context Model.
- When PHRAN continues with the interpretation of the user's question, it chooses the command, the mental object, as a constituent in the phrasal pattern used. It thus ends up understanding the user's question as referring to the previously discussed printing command.
- UC determines the response in the usual manner.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (assertion (activation 100)
                (cd (causation (antecedent (do (actor *ego*)))
                               (consequent (stop-action (command printing-event))))))

No matching assertion.

Monitor -- inspecting context-model

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (question (activation 100)
                (cd (planfor (result (stop-action (command printing-event))) (method *unknown*))))

No matching assertion.

Adding associated clusters.  (base level -- 50)
        (((output (activation 100) (text ?any)))
         ((out-planfor (activation 100)
                       (cd (planfor (result (stop-action (command printing-event)))
                                    (method ?method)))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (output (activation 100) (text ?any))

Matching assertion found.
(output (activation 92) (text |The file fetch.l is in the line printer queue.|) (parents ...) (done yes))
Reinforcing from it

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (out-planfor (activation 100)
                     (cd (planfor (result (stop-action (command printing-event)))
                                  (method ?method))))

No matching assertion.

Adding associated clusters.  (base level -- 50)
        (((output (activation 100) (text |Use the command 'lprm arens'.|)))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (output (activation 100) (text |Use the command 'lprm arens'.|))
```

```
No matching assertion.

Monitor -- inspecting context-model
```

Use the command 'lprm arens'.

VI.2.3. Disambiguation in Context – I

User: John ate something.
User: The big apple.

This example is discussed in section V.2.1. Processing the first input establishes a context that causes one of the two possible meanings of the second to be preferred. Consequently, after processing the second input the Context Modeler will cause "something" to be reinterpreted as referring to the apple mentioned there.

 # John ate something.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (mention (activation 100)
                 (cd john1)
                 (meaning ((p-o-s noun-phrase cd-form john1 description (person male name)))))

No matching assertion.

Monitor -- inspecting context-model
```

- PHRAN interprets the word 'something' just like it does the word 'it' in the previous example. It considers it to be an ambiguous word referring to either a real or mental object, the two kinds of objects it knows about. In each case, the semantic category needed to match the active phrasal pattern at that point is added to the 'description' property of the term. PHRAN is not equipped to determine the inherent contradiction between being an item of food and being a mental object. This causes no problem for UC, since such contardictory entries are always poor matches for existing ones.
- In our case, PHRAN produces one term for New York City and one for a big apple, and 'refdef's for both are inserted in the Context Model. PHRAN chooses the one representing the apple as the correct meaning, due to its appropriateness in the pattern.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (refdef (activation 100)
                (meaning ((p-o-s noun-phrase cd-form object8 description (physob food))
```

```
...)))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (refdef (activation 100)
                (meaning ((p-o-s noun-phrase cd-form mobject8 description (mentob food)))))

No matching assertion.


Monitor -- inspecting context-model

Warning: No good referent for
(refdef (activation 25)
        (meaning ((p-o-s noun-phrase cd-form object8 description (physob food) activation (activation 25))
                 ...)))

Warning: No good referent for
(refdef (activation 25)
        (meaning ((p-o-s noun-phrase cd-form mobject8 description (mentob food) activation (activation 25)))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (assertion (activation 100) (cd (ingest (actor john1) (object object8))))

No matching assertion.


Monitor -- inspecting context-model
```

- UC has finished processing the user's input. In the course of doing so it has named the object eaten *object8*. As seen in the appropriate 'refdef' above, UC has added to the description of object8 a notation indicating that it is an item of food. Naturally, this was not part of the original semantics of the word "something". UC tried to find a referent for this word in the Context Model, but was unsuccessful.

 # The big apple.

- The two possible meanings of this new input are determined by PHRAN and appropriate entries are inserted in the Context Model.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (refdef (activation 100)
                (meaning
                  ((cd-form apple9 p-o-s noun-phrase ref def description (apple physob food)
                    %do ((add-adjs-to-*sc* '((size large)) (terms cd-form))))
                   ...)))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (mention (activation 100)
                 (cd new-york-city)
                 (meaning ((cd-form new-york-city p-o-s noun-phrase description (location city)))))


No matching assertion.
```

```
Monitor -- inspecting context-model

Found referent for
(refdef (activation 25)
        (meaning ((cd-form apple9 p-o-s noun-phrase ref def description & %do & activation &)
                  ...)))
in the mention
(mention (activation 32)
         (meaning ((p-o-s noun-phrase cd-form object8 description & activation &)
                   ...)))
```

- The phrase "the big apple" was analyzed and recognized as ambiguous by PHRAN. However, with the help of the Context Modeler, it was determined that *apple9* is coreferent with *object8*. Consequently, PHRAN's analysis results in the following structure (which is not part of the trace):

```
(object8 new-york-city)
```

- PHRAN's analysis output lists object8 and new-york-city as the two interpretations of the phrase. In addition, the mention of a large apple is considerably more activated than that of New York City, as can be seen from the contents of the Context Model, below.

```
# (pprint *context-model*)
((assertion (activation 96) (cd (ingest (actor john1) (object object8))))
 (mention (activation 92)
          (cd john1)
          (meaning ((p-o-s noun-phrase cd-form john1 description (person male name)
                     activation (activation 92)))))
 (mention (activation 39)
          (meaning ((p-o-s noun-phrase cd-form object8 description (physob food)
                     activation (activation 39))
                    ...)))
 (mention (activation 24)
          (cd new-york-city)
          (meaning ((cd-form new-york-city p-o-s noun-phrase description (location city)
                     activation (activation 24))))))
```

VI.2.4. Disambiguation in Context – II

User: How do I use this editor?
UC: Type 'ex filename'.

User: How do I use this editor?
UC: Type 'vi filename'. Your terminal must support cursor addressing.

The question "How do I use this editor?" is meaningful only in a context which contains a possible referent for "the editor". Following are traces of UC's processing of this query within environments provided by two different Context Models. In both cases the Context Model contains entries for the editors 'ex' and 'vi'. In the first case the entry for 'ex' is highly activated and the one for 'vi'

only mildly so. In the second case the situation is reversed.

The answers provided differ accordingly.

In either case, by the time UC begins searching for a referent the two entries displayed below are no longer the only ones in the Context Model. However, they are the only ones representing editors.

VI.2.4.1. Ex

- The initial Context Model is displayed.

```
# (pprint *context-model*)
((mention (activation 98)
          (cd ex4)
          (meaning ((cd-form ex4 ref indef p-o-s noun-phrase description (ex editor physob) %do (& &)
                    activation (activation 98) associated-action (modify-content & & &) number singular))))
 (mention (activation 47)
          (cd vi4)
          (meaning ((cd-form vi4 ref indef p-o-s noun-phrase description (vi editor physob) %do (& &)
                    activation (activation 47) associated-action (modify-content & & &) number singular)))))
```

 # How do I use this editor?

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (mention (activation 100)
                 (cd *ego*)
                 (meaning ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)))))

No matching assertion.


Monitor -- inspecting context-model


Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (refdef (activation 100)
                (meaning ((cd-form editor4 ref def p-o-s noun-phrase description (editor physob)
                           %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2)))))))

No matching assertion.


Monitor -- inspecting context-model
```

- A referent is sought for the expression "this editor". Entries representing two editors are present in the Context Model. The more highly activated one is chosen.

```
Found referent for
(refdef (activation 100)
        (meaning ((cd-form editor4 ref def p-o-s noun-phrase description & %do & activation &))))
in the mention
(mention (activation 96)
```

```
        (cd ex4)
        (meaning ((cd-form ex4 ref indef p-o-s noun-phrase description & %do & activation &
                  associated-action & number singular))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (assertion (activation 100)
                 (cd (causation (antecedent (do (actor *ego*) (instrument ex4)))
                                (consequent (modify-content (actor *ego*) (instrument ex))))))

No matching assertion.

Monitor -- inspecting context-model
```

- The user's input is interpreted as a request for the identity of a method for modifying the contents of a file using the editor 'ex'. The question is answered in the same manner as similar ones seen earlier in this chapter.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (question (activation 100)
                (cd (planfor (result (modify-content (actor *ego*) (instrument ex)))
                             (method *unknown*)))))

No matching assertion.

Adding associated clusters.  (base level -- 50)
      (((output (activation 100) (text ?any!))
        ((out-planfor (activation 100)
                      (cd (planfor (result (modify-content (actor *ego*) (instrument ex)))
                                   (method ?method)))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (output (activation 100) (text ?any))

No matching assertion.

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (out-planfor (activation 100)
                   (cd (planfor (result (modify-content (actor *ego*) (instrument ex)))
                                (method ?method))))

No matching assertion.

Adding associated clusters.  (base level -- 50)
      (((output (activation 100) (text |Type 'ex filename'.|)))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (output (activation 100) (text |Type 'ex filename'.|))

Matching assertion found.
(output (activation 25) (text |Type 'ex filename'.|) (parents ...))
Reinforcing from it

Monitor -- inspecting context-model
```

Type 'ex filename'.

VI.2.4.2. Vi

The Context Model is now changed to present a different view. Again entries for both the editors are present, but now the one for 'vi' is more highly activated than the one for 'ex'. Similar processing takes place, but a different reply is provided.

```
# (pprint *context-model*)
((mention (activation 98)
          (cd vi3)
          (meaning ((cd-form vi3 ref indef p-o-s noun-phrase description (vi editor physob) %do (& &)
                    activation (activation 98) associated-action (modify-content & & &) number singular))))
 (mention (activation 47)
          (cd ex3)
          (meaning ((cd-form ex3 ref indef p-o-s noun-phrase description (ex editor physob) %do (& &)
                    activation (activation 47) associated-action (modify-content & & &) number singular)))))
```

 # How do I use this editor?

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (mention (activation 100)
                 (cd *ego*)
                 (meaning ((p-o-s noun-phrase cd-form *ego* person first number singular description (person)))))

No matching assertion.


Monitor -- inspecting context-model


Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (refdef (activation 100)
                (meaning ((cd-form editor3 ref def p-o-s noun-phrase description (editor physob)
                           %do ((add-adjs-to-*sc* (value 2 adjs) (terms cd-form)) (copy-term 2))))))

No matching assertion.


Monitor -- inspecting context-model
```

- The reference to 'the editor' is interpreted, this time, as a reference to the editor 'vi'.

```
Found referent for
(refdef (activation 100)
        (meaning ((cd-form editor3 ref def p-o-s noun-phrase description & %do & activation &))))
in the mention
(mention (activation 96)
         (cd vi3)
         (meaning ((cd-form vi3 ref indef p-o-s noun-phrase description & %do & activation &
                    associated-action & number singular))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (assertion (activation 100)
                   (cd (causation (antecedent (do (actor *ego*) (instrument vi3)))
                                  (consequent (modify-content (actor *ego*) (instrument vi)))))))
```

```
No matching assertion.

Monitor -- inspecting context-model

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (question (activation 100)
                (cd (planfor (result (modify-content (actor *ego*) (instrument vi)))
                        (method *unknown*))))

No matching assertion.

Adding associated clusters.  (base level -- 50)
        (((output (activation 100) (text ?any)))
         ((out-planfor (activation 100)
                (cd (planfor (result (modify-content (actor *ego*) (instrument vi)))
                        (method ?method))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (output (activation 100) (text ?any))

No matching assertion.

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (out-planfor (activation 100)
                (cd (planfor (result (modify-content (actor *ego*) (instrument vi)))
                        (method ?method))))

No matching assertion.

Adding associated clusters.  (base level -- 50)
        (((output (activation 100)
                (text |Type 'vi filename'.  Your terminal must support cursor addressing.|))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
        (output (activation 100)
                (text |Type 'vi filename'.  Your terminal must support cursor addressing.|))

Matching assertion found.
(output (activation 25)
        (text |Type 'vi filename'.  Your terminal must support cursor addressing.|)
        (parents ...))
Reinforcing from it

Monitor -- inspecting context-model
```

Type 'vi filename'. Your terminal must support cursor addressing.

VI.2.5. Disambiguation in Context – III

User: John had 1000 dollars.
User: He went to the bank.

User: The Mississippi was wide.
User: John went to the bank.

In each of these examples a context is set up in which the words "the bank" have a preferred interpretation. This is not done by mentioning the interpretation explicitly. Rather, a statement is made which reminds the reader of a situation where one particular meaning of 'bank' is more relevant than the other.

In each case, upon processing the first sentence the program retrieves a related cluster which contains an entry corresponding to one meaning of bank.* The definite reference is later considered to refer to this entry.

VI.2.5.1. Commercial Bank

In this example, the system also determines that the pronoun 'he' in the second sentence refers to the person named John mentioned in the first one.

 # John had 1000 dollars

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 100)
               (cd john1)
               (meaning ((p-o-s noun-phrase cd-form john1 description (person male name)))))

No matching assertion.

Monitor -- inspecting context-model

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 100)
               (cd (money (object money2) (level 1000) (unit dollars)))
               (meaning ((cd-form (money (object money2) (level 1000) (unit dollars)) p-o-s noun-phrase
                         name money2 level 1000 description (money physob)))))

No matching assertion.
```

* Only the relevant entry in each cluster will be noted in the traces below.

- Money is associated with a commercial bank.

```
Adding associated clusters.  (base level -- 100)
      (((mention (activation 75)
                 (cd commercial-bank)
                 (meaning ((cd-form commercial-bank p-o-s noun-phrase number singular
                            description (commercial-bank location financial-inst)))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 75)
               (cd commercial-bank)
               (meaning ((cd-form commercial-bank p-o-s noun-phrase number singular
                          description (commercial-bank location financial-inst)))))

No matching assertion.


Monitor -- inspecting context-model


Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (assertion (activation 100)
                 (cd (poss (actor john1) (object (money (object money2) (level 1000) (unit dollars))))))

No matching assertion.
```

- The program has finished analyzing the first sentence and an entry representing its meaning has been inserted into the Context Model.

```
Monitor -- inspecting context-model
```

- Next, the second sentence is input.

 # He went to the bank

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (refdef (activation 100)
              (meaning ((p-o-s noun-phrase cd-form human2 description (person male subject)))))

No matching assertion.


Monitor -- inspecting context-model
```

- The pronoun 'he' is recognized as referring to John.

```
Found referent for
(refdef (activation 100) (meaning ((p-o-s noun-phrase cd-form human2 description & activation &))))
in the mention
(mention (activation 94) (cd john1) (meaning ((p-o-s noun-phrase cd-form john1 description & activation &))))

Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (refdef (activation 100)
              (meaning ((cd-form bank1 p-o-s noun-phrase ref def
                         description (commercial-bank financial-inst location) number singular)
                        ...)))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (refdef (activation 100)
              (meaning ((cd-form bank2 p-o-s noun-phrase ref def
                         description (river-bank location river-part) number singular))))

No matching assertion.
```

```
Monitor -- inspecting context-model

Found referent for
(refdef (activation 25)
        (meaning ((cd-form bank1 p-o-s noun-phrase ref def description & number singular activation &)
                  ...)))
in the mention
(mention (activation 50)
         (cd commercial-bank)
         (meaning ((cd-form commercial-bank p-o-s noun-phrase number singular description &)))
         (parents ...))
```

- The first possible interpretation of the words "the bank" is determined to refer to the commercial bank associated with the earlier mention of money.
- The program is unable to find a reasonable referent for the second interpretation of bank, a river bank.

```
Warning: No good referent for
(refdef (activation 25)
        (meaning ((cd-form bank2 p-o-s noun-phrase ref def description (river-bank location river-part)
                  number singular activation (activation 25)))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (assertion (activation 100)
                   (cd (ptrans (actor john1) (object john1) (from *here*) (to commercial-bank))))

No matching assertion.

Monitor -- inspecting context-model
```

- We see above that PHRAN's analysis of the sentence reflects the Context Modeler's correct interpretation of the word 'bank' in this context.

VI.2.5.2. River Bank

The Mississippi was wide

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (mention (activation 100)
                 (cd mississippi)
                 (meaning ((p-o-s noun-phrase cd-form mississippi description (mississippi river location)
                           number singular))))

No matching assertion.
```

- The Mississippi, being a river, has a cluster associated with it that contains the concept of a river bank. An entry representing this concept is added to the Context Model.

```
Adding associated clusters.  (base level -- 100)
        (((mention (activation 75)
                   (cd river-bank)
                   (meaning ((cd-form river-bank p-o-s noun-phrase number singular
                             description (river-bank location river-part)))))))

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (mention (activation 75)
                 (cd river-bank)
```

```
        (meaning ((cd-form river-bank p-o-s noun-phrase number singular
            description (river-bank location river-part)))))
```

No matching assertion.

Monitor -- inspecting context-model

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (assertion (activation 100)
            (cd (state (actor mississippi) (state-name width) (value large))))
```

No matching assertion.

- PHRAN's analysis of the input is complete, and an entry representing it has been inserted into the Context Model.

Monitor -- inspecting context-model

- The user now types the second sentence. The existing entry for the concept of a river bank will be determined to be the referent of the words "the bank".

 # John went to the bank

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
      (mention (activation 100)
            (cd john1)
            (meaning ((p-o-s noun-phrase cd-form john1 description (person male name)))))
```

No matching assertion.

Monitor -- inspecting context-model

```
Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (refdef (activation 100)
            (meaning ((cd-form bank1 p-o-s noun-phrase ref def
                  description (commercial-bank financial-inst location) number singular)
                  ...)))
```

No matching assertion.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (refdef (activation 100)
            (meaning ((cd-form bank2 p-o-s noun-phrase ref def
                  description (river-bank location river-part) number singular))))
```

No matching assertion.

Monitor -- inspecting context-model

```
Warning: No good referent for
(refdef (activation 25)
      (meaning ((cd-form bank1 p-o-s noun-phrase ref def description (commercial-bank financial-inst location)
            number singular activation (activation 25))
            ...)))
```

- No reasonable interpretation is found for the first meaning of bank, above, but one exists for the second meaning, below.

```
Found referent for
(refdef (activation 25)
        (meaning ((cd-form bank2 p-o-s noun-phrase ref def description & number singular activation &))))
in the mention
(mention (activation 50)
         (cd river-bank)
         (meaning ((cd-form river-bank p-o-s noun-phrase number singular description &)))
         (parents ...))


Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (assertion (activation 100)
                   (cd (ptrans (actor john1) (object john1) (from *here*) (to river-bank))))

No matching assertion.


Monitor -- inspecting context-model
```

- PHRAN's analysis of the sentence, displayed in the entry above, reflects its determination of the referent of "the bank".

VI.2.6. Disambiguation in Context – IV

User: A pen is in a box.

In this well known example the words "pen" and "box" are both ambiguous. The sentence describes a spacial relationship between two objects, thus placing constraints on their relative sizes. This additional information can be used to disambiguate both words.

The fact that if one object contains another then the container must be 'large' and the contained object must be 'small' is described in a cluster associating the description of the spacial relationship with these characteristics of the two objects involved. This cluster will be recalled upon reading the sentence, and the appropriate meanings of pen and box will receive an increase in their activation levels.

For the purpose of this example I will assume that each of the nouns has two

possible interpretations. The pen may be a 'large' pen, e. g., a playpen, or a 'small' pen, a writing implement. The box may be either a 'large' box, large enough to fit the pen, or a 'small' box. While the meaning representation scheme used here leaves much to be desired, it is sufficient to demonstrate the point.

For reasons explained in the comments in the trace below, PHRAN does *not* arrive at a correct analysis of the utterance. Nevertheless, after the processing is complete all entries in the Context Model are activated at the appropriate levels.

 # A pen is in a box

- After processing the fragment "a pen", two entries are created – one for each meaning.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (mention (activation 100)
               (cd pen-small)
               (meaning ((cd-form pen-small p-o-s noun-phrase ref indef description (writing-pen small physob)
                         number singular)
                        ...)))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (mention (activation 100)
               (cd pen-large)
               (meaning ((cd-form pen-large p-o-s noun-phrase ref indef description (playpen large physob)
                         number singular))))

No matching assertion.


Monitor -- inspecting context-model
```

- The fragment "a box" is read, and two entries are created representing its two meanings.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (mention (activation 100)
               (cd box-small)
               (meaning ((cd-form box-small p-o-s noun-phrase ref indef description (box-small small physob)
                         number singular)
                        ...)))

No matching assertion.


Inserting in *CONTEXT-MODEL*:  (base level -- 50)
      (mention (activation 100)
               (cd box-large)
               (meaning ((cd-form box-large p-o-s noun-phrase ref indef description (box-large large physob)
                         number singular))))
```

```
No matching assertion.

Monitor -- inspecting context-model

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (assertion (activation 100) (cd (contains (outer box-small) (inner pen-small))))
```

- PHRAN has completed its analysis of the input and an entry representing its meaning is about to be inserted in the Context Model. The insertion of the entry will cause the cluster placing restrictions on the size of the objects to be recalled. But PHRAN had to complete the analysis without the benefit of this information. It was therefore forced to chose meanings for the pen and the box arbitrarily.

```
No matching assertion.

Adding associated clusters.  (base level -- 100)
        (((refdef (activation 100)
                (meaning ((p-o-s noun-phrase cd-form container-large description (physob large) number singular))))
          (refdef (activation 100)
                (meaning ((p-o-s noun-phrase cd-form contained-small description (physob small) number singular))))))
```

- At this point the necessary cluster is recalled. (The indications that the first entry concerns the box and the second concerns the pen are not present in the trace). The entries will be inserted next.

```
Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (refdef (activation 100)
                (meaning ((p-o-s noun-phrase cd-form container-large description (physob large) number singular))))

No matching assertion.

Inserting in *CONTEXT-MODEL*:  (base level -- 100)
        (refdef (activation 100)
                (meaning ((p-o-s noun-phrase cd-form contained-small description (physob small) number singular))))

No matching assertion.

Monitor -- inspecting context-model
```

- The Monitor's processing of the two refdef's will bring about the identification of the required interpretations of the ambiguous words.

```
Found referent for
(refdef (activation 100)
        (meaning ((p-o-s noun-phrase cd-form container-large description & number singular)))
        (parents ...))
in the mention
(mention (activation 24)
          (cd box-large)
          (meaning ((cd-form box-large p-o-s noun-phrase ref indef description & number singular
                    activation &))))


Found referent for
(refdef (activation 100)
        (meaning ((p-o-s noun-phrase cd-form contained-small description & number singular)))
        (parents ...))
in the mention
(mention (activation 23)
```

```
(cd pen-small)
(meaning ((cd-form pen-small p-o-s noun-phrase ref indef description & number singular
          activation &)
         ...)))
```

- Despite the ability to identify the correct meanings in the Context Model, PHRAN is not capable of reconsidering its original decisions. PHRAN's analysis of the sentence is thus unchanged.
- The decision not to have PHRAN reconsider its analyses in light of subsequent modifications to the Context Model was taken in order to minimize the interface between PHRAN and the Context Modeler. As a result PHRAN is truly an independent sub-module of UC.
- A look at the contents of the Context Model proves that the words "pen" and "box" were disambiguated correctly.

```
# (p *context-model*)
((assertion (activation 98) (cd (contains (outer box-small) (inner pen-small))) (clusters ...))
 (mention (activation 60)
          (cd pen-small)
          (meaning ((cd-form pen-small p-o-s noun-phrase ref indef description (writing-pen small physob)
                     number singular activation (activation 60))
                    ...)))
 (mention (activation 60)
          (cd box-large)
          (meaning ((cd-form box-large p-o-s noun-phrase ref indef description (box-large large physob)
                     number singular activation (activation 60)))))
 (mention (activation 23)
          (cd box-small)
          (meaning ((cd-form box-small p-o-s noun-phrase ref indef description (box-small small physob)
                     number singular activation (activation 23))
                    ...)))
 (mention (activation 22)
          (cd pen-large)
          (meaning ((cd-form pen-large p-o-s noun-phrase ref indef description (playpen large physob)
                     number singular activation (activation 22))))))
```

REFERENCES

Anderson, J. R., 1983. A Spreading Activation Theory of Memory. *Journal of Verbal Learning and Verbal Behavior,* 22, pp. 261-295.

Austin, J. L., 1962. *How to Do Things with Words.* Oxford University Press, London.

Barr, A., and Feigenbaum, E. A., Editors, 1982. *The Handbook of Artificial Intelligence.* Volume II, pp. 231-233. William Kaufmann, Inc.

Becker, J. D., 1975. The Phrasal Lexicon. In *Proceedings of Interdisciplinary Workshop on Theoretical Issues in Natural Language Processing.* R. Schank and B. L. Nash-Webber (eds.). Cambridge, Mass. June 1975.

Burton, Richard R. 1976. Semantic Grammar: An Engineering Technique for Constructing Natural Language Understanding Systems. BBN Report No. 3453. December, 1976.

Carpenter, P. A., and Daneman, M., 1981. Lexical Retrieval and Error Recovery in Reading: A Model Based on Eye Fixations. In *Journal of Verbal Learning and Verbal Behavior,* 20, pp. 137-160, 1981.

Chafe, W. 1968. Idiomaticity as an Anomaly in the Chomskyan Paradigm. *Foundations of Language* 6.1.22-42.

Charniak, Eugene. 1983. Passing Markers: A Theory of Contextual Influence in Language Comprehension. *Cognitive Science,* 7 (3), 1983.

Chin, D. N. 1983a. Knowledge Structures in UC, the UNIX Consultant. In the *Proceedings of the 21st Annual Meeting of the Association for Computational Linguistics.* Boston, MA. June, 1983.

Chin, D. N. 1983b. A Case Study of Knowledge Representation in UC. In the *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* Karlsruhe, Germany. August, 1983.

Cullingford, R. E., 1978. Script Application: Computer Understanding of newspaper stories. Research Report #116. Dept. of Computer Science, Yale University.

Deering, M., Faletti, J., and Wilensky, R. 1981. PEARL: An Efficient Language for Artificial Intelligence Programming. In the *Proceedings of the Seventh International Joint Conference on Artificial Intelligence.* Vancouver, British Columbia. August, 1981.

Deering, M., Faletti, J., and Wilensky, R. 1982. The PEARL Users Manual. Berkeley Electronic Research Laboratory Memorandum No. UCB/ERL/M82/19. March, 1982.

Dyer, M. G., 1982. In-Depth Understanding: A Computer Model of Integrated Processing for Narrative Comprehension. Research Report #219. Computer Science Department, Yale University.

Fillmore, C. 1985. Thoughts on Grammatical Constructions. Unpublished manuscript. Linguistics Department, University of California at Berkeley.

Fischler, I., 1977. Semantic Facilitation Without Association In A Lexical Decision Task. *Memory and Cognition,* 5, pp. 335-339.

Gershman, A. V. 1979. Knowledge-Based Parsing. Ph.D. Dissertation. Research Report #156, Dept. of Computer Science, Yale University.

Grosz, B. J. 1977. The Representation and Use of Focus in a System for Understanding Dialogs. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.* Carnegie-Mellon University, Pittsburgh, PA.

Held, G. D., Stonebraker, M. R., and Wong, E. 1975. INGRES – A relational data base system. AFIPS Conference Proceedings Vol. 44, NCC.

Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., and Slocum, J. 1978. Developing a Natural Language Interface to Complex Data. In *ACM Transactions on Database Systems* Vol. 3 No. 2. June, 1978.

Jacobs, P. 1983. Generation in a Natural Language Interface. In the *Proceedings of the Eighth International Joint Conference on Artificial Intelligence.* Karlsruhe, Germany. August, 1983.

Just, M. A., and Carpenter, P. A., 1980. A Theory of Reading: From Eye Fixations to Comprehension. In *Psychological Review,* volume 87, number 4, July 1980.

Lebowitz, M. 1980. Generalization and Memory in an Integrated Understanding System. Yale University Department of Computer Science Technical Report 186.

Lesser, V. R., and Erman, L. D. 1977. A Retrospective View of the Hearsay-II Architecture. In *Proceedings of the Fifth International Joint Conference on Artificial Intelligence.* Cambridge, MA, 1977.

McKoon, G., and Ratcliff, R., 1979. Priming in Episodic and Semantic Memory. *Journal of Verbal Learning and Verbal Behavior,* 18, pp. 463-480.

McDonald, J. L., and Carpenter, P. A., 1981. Simultaneous Translation: Idiom Interpretation and Parsing Heuristics. In *Journal of Verbal Learning and Verbal Behavior,* 20, pp. 231-247, 1981.

Meyer, D., and Schvaneveldt, R., 1976. Meaning, Memory, and Mental Processes. *Science,* 192, pp. 27-33.

Neely, J. H., 1977. Semantic Priming and Retrieval from Lexical Memory: Roles of Inhibitionless Spreading Activation and Limited-capacity Attention. *Journal of Experimental Psychology: General,* 106, pp. 226-254.

Parkinson, R. C., Colby, K. M., and Faught, W. S. 1977. Conversational Language Comprehension Using Integrated Pattern-Matching and Parsing. *Artificial Intelligence* 9, pp. 111-134.

Peterson, L. R., and Peterson, M. J. 1959. Short-term Retention of Individual Verbal Items. *Journal of Experimental Psychology* 58, pp. 193-198.

Quillian, M. R., 1969. The Teachable Language Comprehender: A Simulation Program and Theory of Language. In *Communications of the ACM,* 12(8), August 1969, pp. 459-476.

Riesbeck, C. K. 1975. Conceptual analysis. In R. C. Schank, *Conceptual Information Processing.* American Elsevier Publishing Company, Inc., New York.

Riesbeck, C. K., and Schank, R. C., 1975. Comprehension by Computer: Expectation-Based Analysis of Sentences in Context. Yale University Research Report #78.

Schank, R. C. 1975. *Conceptual Information Processing.* American Elsevier Publishing Company, Inc., New York.

Schank, R. C., 1980. Language and Memory. In *Cognitive Science,* 4(3), 1980.

Schank, R. C., and Abelson, R., 1977. *Scripts, Plans, Goals, and Understanding.* Lawrence Erlbaum Associates, Hillsdale, NJ 1977.

Searle, J. R., 1969. *Speech Acts: An Essay in the Philosophy of Language.* Cambridge University Press, 1969.

Small, S. 1978. Conceptual language analysis for story comprehension. Technical Report No. 663, Dept. of Computer Science, University of Maryland, College Park, Maryland.

Small, S. 1980. Word Expert Parsing: A Theory of Distributed Word-Based Natural Language Understanding. Ph.D. Dissertation. Technical Report No. 954, Dept. of Computer Science, University of Maryland, College Park, Maryland.

Waltz, D. L., Finin, T., Green, F., Conrad, F., Goodman, B., and Hadden, G. 1976. The PLANES system: natural language access to a large data base. Coordinated Science Lab., University of Illinois, Urbana, Tech. Report T-34.

Warren, R. E., 1972. Stimulus Encoding and Memory. *Journal of Experimental Psychology,* 94, pp. 90-100.

Warren, R. E., 1977. Time and Spread of Activation in Memory. *Journal of Experimental Psychology: Learning and Memory,* 3, pp. 458-466.

Wilensky, R., 1978. Understanding Goal Based Stories. Research Report #140. Dept. of Computer Science, Yale University.

Wilensky, R., and Arens, Y., 1980a. PHRAN: A Knowledge-Based Approach to Natural Language Analysis. Memo UCB/ERL M80/34, Electronic Research Laboratory, UC Berkeley, August 1980.

Wilensky, R., and Arens, Y., 1980b. PHRAN: A Knowledge-Based Natural Language Understander. In *Proceedings of the 18th Annual Meeting of the Association for Computational Linguistics,* Philadelphia, PA. June 1980.

Wilensky, R., Arens, Y., and Chin, D., 1984. Talking to UNIX in English: An overview of UC. In *Communications of the ACM.* June, 1984.

Wilensky, R. and Morgan, M. 1981. One Analyzer for Three Languages. Berkeley Electronic Research Laboratory Memorandum No. UCB/ERL/M81/67. September, 1981.

Wilks, Y. 1973. An AI Approach to Machine Translation. In *Computer Models of Thought and Language*. R. C. Schank and K. M. Colby (eds.), W.H. Freeman and Co., San Francisco, 1973.

Wilks, Y. 1975a. An Intelligent Analyzer and Understander of English. in *Communications of the ACM*. Volume 18, number 5, May 1975.

Wilks, Y. 1975b. A Preferential, Pattern-Seeking, Semantics for Natural Language Inference. In *Artificial Intelligence* 6 (1975), pp. 53-74.

Winograd, T., 1970. Procedures as a Representation for Data in a Computer Program for Understanding Natural Language. Ph.D. Dissertation. Department of Mathematics, MIT.

Winograd, T., 1972. *Understanding Natural Language*. Academic Press, New York, 1972.

Woods, W. A. 1970. Transition Network Grammars for Natural Language Analysis. *Comm. ACM,* Vol. 13, pp. 591-606.