Transfer Semantics in an Operating System Consultant: The formalization of actions involving object transfer

Paul Mc Kevitt & Yorick Wilks

Computing Research Laboratory, Dept. 3CRL
New Mexico State University
Box 30001, Las Cruces, NM 88003-0001, USA.
CSNET: paul@nmsu & yorick@nmsu [(505) 646-5466]

ABSTRACT

It is a characteristic of computer operating systems that they contain actions or commands which transfer objects such as files and directories from one state to another. In formalizing the domain of operating systems we should build representations of actions which circumscribe the transfer of objects in the system. Transfer Semantics is a knowledge scheme that embodies such representations. Knowledge structures called object frames are used to represent numerous objects. Action frames describe the effects of actions in terms of preconditions, postconditions, actions and actors. Preconditions denote possible or preferred sets of objects that an action will affect. Postconditions relate the state of object sets after an action has occurred. Actions include the particular actions that cause transfer. An actor is the person (or user) who performs some action. The power of Transfer Semantics lies in the inference rules that manipulate action frames. It is applied to the UNIX* and TOPS-20* operating systems in a program called *OS Consultant*. OS Consultant will be used by new users to learn operating system concepts.

I INTRODUCTION

A. Introducing Transfer Semantics

It is our belief that people think (however abstractly) of operating system commands in terms of preconditions and postconditions. Preconditions and postconditions are sets of states of objects before and after a command is executed. Most English queries about operating systems involve users expressing the goal of obtaining some command. Commonly, users will try to describe the affect of a required command on some object(s). For example, in the query, "How do I print out a file with pages?" the user is expressing the need for some command to print files with page-headers.

It is the constraints specified in a user query that enable us to recognize a command. Therefore, it seems useful to build knowledge structures for describing commands so that these structures are closely related to possible natural language expressions of such commands. Natural language queries involving descriptions of commands can be parsed into some high-level meaning representation. To interpret queries effectively we need access to domain-specific knowledge. Such knowledge could be formulated as abstract representations of actions or objects which are matched to natural language representations in order to decipher them.

Transfer Semantics (TS) is a developing knowledge representation scheme used to formalize actions and *preferred* objects affected by actions. By preferred objects we mean objects that are usually affected by some action. We use TS to represent the means by which operating system commands transfer objects from one state to an other. In TS operating system objects are represented by object frames. The object frames are structured in a hierarchic (tree-like) network representation. Action frames are used to specify transfer relations between object frames.

Each action frame is a formal representation of operating system actions or commands. Action frames consist of preconditions, postconditions, actions and actors. Preconditions are sets of states of objects existing before commands take effect. Postconditions involve sets of states of objects after a command is performed. Such conditions specifying states of objects are preferred, i.e. we do not specify all conditions on frames, only those that usually occur. Various rules of inference are used to manipulate preferred conditions in order to expand the meaning of each frame. Actions include the particular command(s) that cause transfer of object states. This representation can then be used effectively to understand natural language expressions describing actions. The ideas herein could be applied to other domains.

B. The OS Consultant and its relation to other work

OS Consultant (OSCON) is a system, programmed in Common Lisp, which will help novice users learn operating system concepts. While

* UNIX is a trademark of AT&T Bell Laboratories.

* TOPS-20 is a trademark of the Digital Equipment Corporation.

the capability of answering the query in terms of UNIX. OSCON is designed in the form of an English interface to a database *. Examples of other interfaces to databases are found in Martin et al. (1983) & Waltz (1975). Although we plan to build an English interface which teaches novice users we do not intend the system to become a command-level interface such as COUSIN (see Hayes 1982).

In OSCON parsed English sentences are translated into formal queries with uninstantiated variables. These formal queries are instantiated by a database of operating system concepts and returned to the interface where answers are produced in English. The formal queries to the database are represented in the form, <{P} A {Q} U>. P and Q represent preconditions and postconditions for any action A. U represents the particular person (user) performing A. Original work on the formal queries is discussed in Douglass & Hegner (1982) and Hegner & Douglass (1984).

An important distinction of OSCON, as an interface to a database, is that Transfer Semantics is used to formalize abstractions of database detail in the interface itself. The interface contains abstract knowledge about the relationships between UNIX objects and actions and includes four levels of meaning representation. Initially, a shallow representation is produced by a syntactico-semantic parser. Examples of such parsers are described by Ball and Huang in Wilks (1986). The shallow representation is translated into a meaning representation of embedded concepts where case labels are attached to various items. The embedded concept representation is replaced by a domain-specific structure after processing with Transfer Semantics. This domain-specific structure is further translated into a formal query with uninstantiated variables. Finally, the formal query is passed to a database where instantiation occurs.

C. Relation to Other Work

Wilensky et al. (1984, 1986) are also working on building an understanding system called Unix Consultant (UC) which processes natural language queries about UNIX. The Unix Consultant embodies a knowledge representation called KODIAK. The central theme of KODIAK is that it is a relation-based system. KODIAK relations have a fixed number of argument positions and each argument position of a relation is itself a full-fledged object. The meaning of argument-objects is derived from the named relation that holds between them. KODIAK has a wide representational scope and still maintains the possibility of conforming to a canonical form. At the action frame level Transfer Semantics is also a relation-based system where actions are described in terms of precondition-postcondition correspondence. In Transfer Semantics the meaning of any action is the precondition and postcondition set for that action. Wilensky decides to represent all concepts with relations. We only see the need to represent actions which manipulate objects in terms of relations. Many objects are not defined by relations in Transfer Semantics although there may be relations between them.

The Unix Consultant system is not intended to handle queries on operating systems other than UNIX. In OSCON we are putting more effort into understanding complex queries where there are a number of operating system commands interrelated with each other, to denote some higher level process. It seems that Transfer Semantics which captures the meaning of commands, is a suitable formalism for abstracting operating system behavior.

Our object frames are similar to the frames proposed by Minsky (1975). Yet, Minsky decides (p. 234), "...that any event, action change, flow of material or information can be represented by a two-frame generalized event." This is in contrast to our system where single action frames are used to represent state changes of objects. Wilks (1978) describes semantic structures called pseudo-texts for natural language understanding. A pseudo-text is a structure of factual and functional information about some concept and is intended to fall broadly within the notion of frame in the sense of Minsky, Charniak, and Schank. Pseudo-texts are also similar in function to the object frames we describe herein. Our action frames have similarities with the "scripts" discussed by Schank & Abelson (1977). Action frames could be interpreted as structured scripts for various operating system commands.

The arrangement of object frames is based on many semantic network and frame systems. Examples are Bobrow & Winograd (1977) and Brachman (1979). Our network structure is closely related to that of Fass (1986) where he uses dictionary entries called "sense-frames" to define word senses in a sense network for Collative Semantics.

II TRANSFER SEMANTICS

A. Object Frames

Various operating system objects such as "files", "protection", "commands", "last-read-time", "creation-time", and "password" are represented by object frames. Object frames exist statically in the system before any processing begins. Each object frame has a set of arcs and nodes. Arcs specify types of relations between some object and other related objects in a network hierarchy. Nodes define characteristics of the particular object represented by a frame. Object frames are a refinement of more detailed information about operating system objects residing in the static knowledge base of the database (see Hegner 1987).

Presently, in OSCON there are three types of arc relation linking objects. These are **type-of**, **part-of** and **instance-of** relations. It may be necessary to define other types of relation as research continues. The type-of arc relation is used to specify one object as a type of another. So, a plain file is a type of non-directory-file

* A complete formal database is being built by Dr. Steve Hegner at the University of Vermont.

and a non-directory file is a type of file. A part-of arc relation indicates that one object frame is part of another. For example, "creator" and "last-tape-read-time" are parts of files. Each instance-of relation indicates that an object is an instance of another. The commands "lpr", "cat", and "cp" are related to the "command-name" object frame by this relation.

Each node is a set of attributes characterizing an object frame. Nodes in object frames are specified using the **has** relation. Has relations usually contain other object frames. In Figure 1 below there is an example of the object frame for protection-type.

```
(o-frame protection-type
    (arcs (part-of protection))
    (node (has user-designator)
          (has access-privilege)
          (has file-access)))
```

Figure 1.

Objects are related in a complex hierarchy. In the diagram below there is a description of some of the hierarchy:

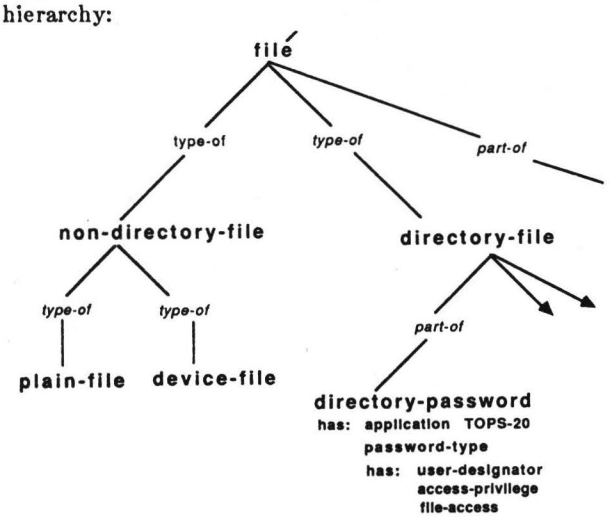


Figure 2.

So, from this diagram we note such relations as:

plain-file
is a type of **non-directory file**
is a type of **file**
is a type of **container.**

We note above that "directory-password" (a concept from the TOPS-20 operating system) is defined in terms of UNIX concepts. This will be particularly useful for helping some user who is confused as to which operating system he/she is using.

B. Action Frames

Operating system actions such as "printing", "listing", "moving", "deleting" and "mailing" can be represented by action frames. Action frames also exist statically in the system before any processing begins.

Preconditions and postconditions for any action denote sets of preferred conditions on objects. Not only are the conditions on objects preferred, but the actual objects themselves are also preferred. It is important that we specify preferred sets of conditions because there are many possible conditions for any action. Preconditions are mentioned in most planning literature and have been used for specifying plans and goals. For example in Wilensky (1987) there is a description of concerns which are preconditions particularly relevant to a given plan. The term concern is synonymous with our concept of preferred conditions.

Preferences are used in frame selection processes where the frame with the maximum number of preferences satisfied is probably the best frame for interpreting the input. For example, the print frame will have more preferences satisfied than the list frame from the query, "How do I list a file on the line printer". Of course, that is because one usually associates line printers with printing rather than listing. The idea of preference is not new to Artificial Intelligence. It has been used by Wilks (1978) in Preference Semantics and Fass (1986) in Collative Semantics to formulate correct interpretations of natural language sentences.

1. Preconditions

Each action frame precondition set contains various conditions related by the logical operands AND, OR, and NOT. For example, the precondition set for the action frame "print" is:

```
(preconditions
    (AND (NOT (o-frame directory-file))

        (OR (AND ((o-frame file)
                  ((o-frame contents)
                   = (o-frame visible-byte-sequence))))

            (AND ((o-frame file)
                  ((o-frame contents)
                   = (o-frame visible-byte-sequence)))
                  (o-frame print-queue))

            (AND ((o-frame file)
                  ((o-frame contents)
                   = (o-frame visible-byte-sequence)))))))
```

Figure 3.

The initial logical operand in any precondition set is usually AND. The reason for this is that mandatory conditions must be ANDed to other conditions in the set. The optional conditions in each precondition set are ORed together. The final ORed condition is a default. In Figure 3 there are three ORed preconditions which are ANDed to one mandatory precondition. Of course, the third ORed condition is the default.

Interpreting the above set, it is noted that the mandatory condition specifies that a directory file should not be printed. Of course, it is possible to print a directory by first listing it and then printing it. Yet, one does not usually print directories themselves, and this is what we are concerned with here. The first optional condition specifies a preference that files are printed and their contents are preferably visible byte sequences. The second optional condition declares in

addition the existence of a printer queue. In order to print a file on the printer it is certainly useful to have a printer queue. Finally, the third condition in the set is a default, and is the same as the first condition. We do not worry about preconditions such as the system being up, the terminal working or keyboard on-line. These are simply assumed.

It is important that we represent the "weakest" precondition set for any action. By weakest, we mean the least number of (or least constraining) preconditions necessary to characterize some action sufficiently. For example, we know non-directory files to be types of files (Figure 2) and that either can be preconditions for printing. The use of files (weak) as a precondition for printing will do just as well as non-directory files (stronger). That is exactly why we reflect files in the precondition set rather than say non-directory files (or device files or plain files) which are types of file.

2. Postconditions

Postconditions for any action denote preferred conditions on objects resulting from the execution of that action. In all action frames the postconditions represent changes in state of the precondition set. The postconditions for the action frame "print" are shown in Figure 4.

In the postcondition set below there are no mandatory conditions. Hence, the initial logical operand is OR. There are three ORed conditions, the final one delimiting a default. The first condition declares that the file which we saw in the precondition set also exists in the postcondition set. The file doesn't disappear after printing as would be the case with a delete frame. The file still contains visible byte sequences although a filter is now also applied. Filters are items such as page-headers, line numbers and dates. Also a device file exists to denote default standard output which is the terminal screen.

The second ORed condition tells us that a print queue exists and has a print queue entry. Also, a filter may be applied to the contents of the file. The third postcondition in the set is again a default and specifies output to a device file.

```
(postconditions
  (OR (AND ((o-frame non-directory-file)
            ((o-frame contents)
              = (o-frame visible-byte-sequence)
              (o-frame filter)))
            (o-frame device-file))

      (AND ((o-frame non-directory-file)
            ((o-frame contents)
              = (o-frame visible-byte-sequence)
              (o-frame filter)))
            (o-frame print-queue)
            (o-frame print-queue-entry))

      (AND ((o-frame non-directory-file)
            ((o-frame contents)
              = (o-frame visible-byte-sequence)))
            (o-frame device-file))))
```

Figure 4.

We try to represent the "strongest" postcondition set for any action. By strongest we mean the maximum number of (or most constraining) postconditions necessary to characterize some action sufficiently. We know device files to be types of non-directory file (Figure 2) and that either could denote postconditions for printing files. However, the use of device files (strong) as a postcondition for printing rather than non-directory files (weaker) is a more precise definition about the effects of printing. That is why we reflect device-file in the postcondition set rather than say file or non-directory file. There is no harm in weakening the postcondition set. We will see in section four that the ability to weaken postconditions is, in fact, an advantage.

3. Precondition-Postcondition Correspondence

Now, one may think there is some redundancy in the condition sets for print. For example, one condition occurs twice in the precondition set. However, we have done this because there is an exact one-to-one correspondence between the ORed conditions in the precondition and postcondition sets. Say, $\{ P_0, P_1, \ldots P_n \}$ denote the ORed preconditions for some action A. Then, these are related to the ORed postconditions $\{ Q_0, Q_1, \ldots Q_n \}$ so that $P_0 \Rightarrow Q_0$, $P_1 \Rightarrow Q_1$, ... $P_n \Rightarrow Q_n$ for action A. So, the first ORed precondition in the precondition set corresponds to the first in the postcondition set, the second to the second, and so on. As there are no mandatory postconditions there is no correspondence for them at all.

The one-to-one correspondence between preconditions and postconditions is implicit: it is the position of a particular condition in its precondition/postcondition set that determines correspondence. If it turns out that many some conditions need to be repeated exhaustively (it only happened once above) for action frames we can represent the correspondence explicitly by placing a marker on each condition. One may wonder what's the use of all this correspondence. Correspondence aids in predicting the most likely postcondition (or precondition) for some explicitly mentioned precondition (or postcondition) in a user query. It is the ability to predict preconditions and postconditions for user queries that gives added power to the system.

4. Actions and Actors

It is also necessary to specify the possible actions that cause transfer between preconditions and postconditions. Associated with each action will be a number of options. So, for the print frame actions are represented as:

```
(actions
  (OR (o-frame cat)
      (o-frame more)
      (o-frame lpr)
      (o-frame pr)
      (o-frame print)
      (o-frame option-list)))
```

Figure 5.

Printing can be completed with any of the commands in the ORed set of actions and their respective options. Finally, in Figure 6 we specify the actor performing the action or transfer. Any user can print a file and this is represented in the actor set.

```
(actor
   (OR (o-frame user)))
```

Figure 6.

To summarize, operating system actions are defined in terms of preconditions, postconditions, actions, and actors. Action frames reflect the behavior of operating system actions in terms of the effect of these actions on objects. We use the notation,

$$\left\{ \{P\}\ A\ \{Q\} \right\} : U$$

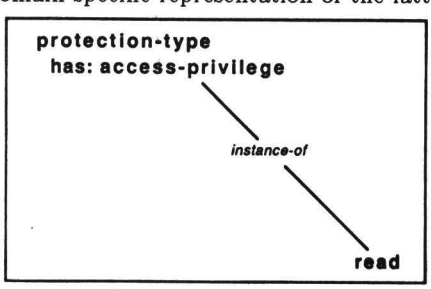
to denote the fact that some user U can execute the action A to transfer the precondition set P to the postcondition set Q. Now that we have described object frames and action frames it is possible to show how they may be used to interpret natural language queries about operating systems.

III LANGUAGE & TRANSFER SEMANTICS

In this section we will show how various queries could be interpreted using object and action frames. We adopt a distinction between concept description and dynamic queries. This distinction has been emphasized by Hegner (1987). Concept description queries are simple queries about objects involving no manipulation of those objects. Dynamic queries are those which involve actions transferring objects.

A. Concept Description Queries

In handling concept description queries such as, "What is read protection?" the hierarchy of object frames becomes very useful. From the network it is possible to locate relevant object frame relations. The following section of network is used in generating a static domain-specific representation of the latter query:



Now, say some user has used the TOPS-20 operating system for most of his computer time and decides to use UNIX for a change. Then he/she is likely to assume that UNIX is similar to TOPS-20. One could expect queries such as, "What is the permanent storage limit?". The following relations from the object network are used:

directory-file
has **permanent-storage-limit**
has **application TOPS-20**
has **similarity disk-space-hard-limit.**

The above relations denote the similarity between concepts from two operating systems. The similarity between disk-space-hard-limit and permanent-storage-limit is marked using has relations. This mechanism is especially useful if a user thinks in terms of one operating system but is using another.

B. Dynamic Queries

The object hierarchy is availed of again for dynamic queries. However, as dynamic queries involve actions, action frames must be referenced. Say, for example we want to interpret the query, "What is the option on the cat command which numbers lines?". Looking at Figure 3, the first ORed condition would be matched. This condition constitutes the precondition for this particular query. The relevant postcondition is specified by the first ORed condition (Figure 4). This is done by moving down the object hierarchy from filter (Figure 4) to numbered-lines which are a type of filter. Also, the first action in Figure 5 is marked because "cat" was mentioned in the query. From Figure 6 we mark the user as being the relevant actor. Similarly, the query, "How do I print a file on the line printer?" matches ORed precondition two (Figure 3), ORed postcondition two (Figure 4), and no match is found in Figure 5. The actor is again "user" from Figure 6.

C. Rules of Consequence

As we mentioned earlier, it is possible to "strengthen" preconditions and "weaken" postconditions for action frames without affecting the truth of the frame. Say, we have the query, "How do I print a device file?". Even though we only represent files in the precondition set (Figure 3) it is possible to use inference rules to infer more specific preconditions. This inference capacity is implemented by moving down the object frame hierarchy from file to non-directory file to device file.

In this example the inference processes are rather straightforward. They simply involve moving down the object hierarchy from one object frame to another. In the example "How do I print a device file which has pageheaders?" the system should recognize pageheaders as being a precondition. Now, in the object network we can derive the relations:

pageheaders
is a type of **filter**
is a type of **visible-byte-sequence.**

and from Figure 3 we know:

(o-frame contents
= (o-frame visible-byte-sequence))

The inference processes used here involve comparison of objects such as pageheaders and visible-byte-sequences. We use a process of projection to derive pageheaders as a precondition when they are not already specified.

It is also possible to "weaken" postconditions and still preserve the truth of {P} A {Q}. So, the query, "How do I print files on the printer?" will still be interpreted from the postcondition set in Figure 4 by weakening the first ORed condition so that non-directory file becomes file.

The processes of strengthening and weakening above are definable by logical inference rules. More specifically, they are called the *Rules of Consequence.* The above rules and other inferencing techniques are described more completely in Mc Kevitt (1986b). In that paper there is a description of various rules and how they specify manipulation of action frames. The rules allow the system to directly infer new object preferences for action frames from the object hierarchy. Minsky (1975) also notes that we need some method of applying transformations between frames in a system. He says, "I do not understand the limitations of what can be done by simple processes working on frames. One could surely invent some "inference-frame technique" that could be used to rearrange terminals of other frames so as to simulate deductive logic."

It is important to note that only the "best" conditions are selected while matching a frame to an initial meaning representation of some query. For each condition we determine the ratio of matched to non-matched predicates. The best condition is the one with the highest ratio. For any condition to be best not all its preferences have to be satisfied. Indeed, we saw above that the process of weakening postconditions is required because local preferences in conditions are not satisfied.

D. The Rule of Composition

Many queries about UNIX involve more than one action to complete some process. For example, the query, "How do I stop a listing of my directory, which is printing on the line printer?" involves three actions: "removing", "listing" and "printing". We call these queries *embedded queries.* The previous query is an an example of *explicit embedding* where three actions are explicitly mentioned. Other types of embedding are described in (Mc Kevitt 1986a).

We use the notation $[A_1 < A_2 < ...A_n]$ to denote an embedding set where action A_1 is embedded inside action A_2, and so on. One can think of embedding in terms of a stack where A_n is pushed on top of A_{n-1} and so on. Interpreting the stack, the postcondition {Q} from performing A_1 is passed as a precondition to A_2 and so on until we reach the top of the stack. For the previous query we have the embedding set, [LIST < PRINT < REMOVE] and for the query, "How do I print a listing of my directory on the line printer?" we get, [LIST < PRINT]. In the latter example a directory is initially listed and then printed. In effect, the concept of listing is embedded inside printing. Certainly, in order to interpret queries involving embedding, we need to use some other inference rule to process action frames. We describe such an inference rule using the notation below:

$$\left\{ \frac{\{P\}\ A_1\ \{Q\}\ ,\ \{Q\}\ A_2\ \{R\}}{\{P\}\ [A_1 < A_2]\ \{R\}} \right\} : U$$

This general formula states that if {P} A_1 {Q} is true and {Q} A_2 {R} is also true then we can infer {P} $[A_1 < A_2]$ {R} to be true too. We call this inference rule the *Rule of Composition.* A more specific formula for the example query, "How do I print a listing of my directory on the line printer?" is:

$$\left\{ \frac{\{P\}\ LIST\ \{Q\}\ ,\ \{Q\}\ PRINT\ \{R\}}{\{P\}\ [\text{LIST} < \text{PRINT}]\ \{R\}} \right\} : U$$

Interpreting this inference rule we deduce that if the postcondition of LIST is applied as the precondition of PRINT then it is inferred that the postcondition of PRINT is the postcondition of executing both actions.

We can formulate the domain specific information needed for the query, "How do I print a listing of my directory on the line printer?", in terms of {P} LIST {Q}, {Q} PRINT {R}. Our inference rule tells us that this is equivalent to {P} [LIST < PRINT] {R}. Note that the system must derive the new postcondition set {R}. The techniques for developing interpretations of other queries involving embedding are aspects of ongoing research.

IV CONCLUSION

It is concluded that Transfer Semantics is an appropriate mechanism for describing actions and how these actions transfer objects. It seems a particularly effective mechanism for abstracting characteristics of various computer operating system actions in a concise formalism. The use of Transfer Semantics in OSCON enables the production of complex formal queries to be instantiated by a fully formalized database. Sets of conditions for action frames are only preferences in the system which are typical of some action. We use preferences for two reasons: (1) in order to select the correct frame (2) if we specified all possible transfer conditions on frames they would certainly become very large. Yet, the system is not restricted to preferred conditions due to the presence of various inference rules.

It is a significant feature of Transfer Semantics that there exists a number of inference rules enabling manipulation of action frames. Therefore, by using the object frame hierarchy and these inference rules an action frame can circumscribe a large quantity of domain-specific relations. In this paper we have shown the usefulness of logical inference rules of consequence and composition. The consequence rules enable the system to infer more detailed or less specific objects from an object hierarchy. Embedded queries involving many

concepts can be interpreted effectively on application of the composition rule.

A particularly useful feature of Transfer Semantics is that similarities between object frames are marked. Therefore, even though a query may be presented to OSCON with TOPS-20 lingo, that query can be interpreted and answered in terms of UNIX. It is hoped that Transfer Semantics will be used to model other operating systems as research continues.

We are continuing to build action frames for other actions such as "mailing", "moving", and "creating". Of particular interest is the possibility of recognizing user misconceptions in queries. For example, say a user asks the query, "How do I print a file with the -Z option?". "-Z" is not an option on printing. Nor, can -Z be inferred for printing. So, the action frame for "printing" does not specify a formula of the form <{P} A {Q} U> because A is not satisfied. We also hope to investigate the possibility of recognizing ill-formed embedding. For example, the query, "How do I delete my files and then list them?" doesn't make much sense at all.

We have not described the meaning representations of English queries before the frames are matched to them. These representations are discussed in Mc Kevitt (1986a). In this paper we are interested only in the frames themselves. Further research includes developing robust matching processes that determine the right frame for some query.

ACKNOWLEDGEMENTS

We wish to acknowledge the Natural Language Group at the Computing Research Laboratory (CRL) for valuable discussion on the content of this paper. We are indebted to the referees for suggested revisions.

REFERENCES

Bobrow, D.G. & Winograd, T. 'An overview of KRL, a knowledge representation language.' *Cognitive Science*. 1:1 (1977) 3-46.

Brachman, R.J. 'On the epistemological status of semantic networks.' In *Associative Networks: Representation and use of knowledge by computers*, N.V. Findler (Ed.). New York: Academic Press, 3-50, 1979.

Douglass, Robert J. & Hegner, Stephen J. 'An expert consultant for the UNIX operating system: Bridging the gap between the user and command language semantics.' *Proc. Fifth National Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI)/SCIEO Conference*. Saskatoon, Saskatchewan, May, 1982.

Fass, D.C. 'Collative Semantics: an approach to coherence.' Memoranda in Computer and Cognitive Science, Memorandum MCCS-86-56, Rio Grande Research Corridor, Computing Research Laboratory, New Mexico State University, Box 30001, Las Cruces, NM 88003-0001, USA, 1986.

Hayes, Philip J. "Uniform help facilities for a cooperative user interface." *Proc. National Computer Conference*, Houston, 1982, 469-474.

Hegner, Stephen J. & Douglass, Robert J. "Knowledge base design for an operating system expert consultant." *Proc. of the Fifth National Conference of the Canadian Society for Computational Studies of Intelligence (CSCSI)*. London, Ontario, December, 1984, pp. 159-161.

Hegner, Stephen J. "Representation of command language behavior for an operating system consultation facility." Technical Report CS/TR87-02, CS/EE Department, University of Vermont, USA, 1987.

Martin, Paul; Appelt, Douglas & Pereira, Fernando "Transportability and generality in a natural-language interface system". *In Bundy, Alan (Ed.) Proc. IJCAI-8*. Karlsruhe, West Germany, August, 1983, pp. 573-581.

Mc Kevitt, Paul "Parsing embedded queries about UNIX." Memoranda in Computer and Cognitive Science, MCCS-86-72, Rio Grande Research Corridor, Computing Research Laboratory, New Mexico State University, 1986a.

Mc Kevitt, Paul 'Formalization in an English interface to a UNIX database'. Memoranda in Computer and Cognitive Science, MCCS-86-73, Rio Grande Research Corridor, Computing Research Laboratory, New Mexico State University, 1986b.

Minsky, Marvin "A framework for representing knowledge." In *The psychology of computer vision, P.H. Winston (Ed.)*. New York: McGraw-Hill, 1975.

Schank, R.C. & Abelson R.P. "Scripts, plans, goals and understanding: an enquiry into human knowledge structures." Hillsdale, New Jersey: Lawrence Erlbaum Associates, 1977.

Waltz, David "Natural language access to a large database: an engineering approach." *Advance papers IJCAI-4*. Tbilisi, Georgia, USSR, Sept, 1975, 868-872.

Wilensky, Robert; Mayfield, Jim; Albert, Anthony; Chin, David; Cox, Charles; Luria, Marc; Martin, James and Wu, Dekai 'UC — a progress report.' Report No. UCB/CSD 87/303, Computer Science Division (EECS), University of California, Berkeley, California 94720, July, 1986.

Wilensky, Robert "Some complexities of goal analysis." *Preprints of Conference on Theoretical Issues in Natural Language Processing-3 (TINLAP-3)* Computing Research Laboratory, New Mexico State University, January, 1987, pp. 97-99.

Wilks, Yorick "Making preferences more active." *Artificial Intelligence* 11, (1978) 197-223.

Wilks, Yorick "Projects at CRL in Natural Language Processing." Memoranda in Computer and Cognitive Science, MCCS-86-58, Rio Grande Research Corridor, Computing Research Laboratory, New Mexico State University, 1986.