1981-9

State University of New York at Buffalo

DEPARTMENT OF COMPUTER SCIENCE

COMBINING PATH-BASED AND NODE-BASED INFERENCE IN SNEPS

by

Rohini K. Srihari

Masters Project

June 1981

Technical Report 183

This work presented here was supported in part by the National Science Foundation under Grant No. MCS80-06314.

ABSTRACT

This paper describes a recent enhancement made to SNePS to include path-based inference. The previous version of SNePS used only node-based inference and was thus limited in its capabilities. Combining path-based inference and node-based inference in SNePS has greatly increased the scope of the system. The paper first motivates the need for such a change and then proceeds to outline the theory as well as the implementation details. Finally, a series of examples are provided which support the claim that the system is indeed more powerful and efficient now.

1. INTRODUCTION

A semantic network is a method of representing knowledge. Usually included in semantic network systems are procedures to carry out inference based on the knowledge that is represented. Two of the more commonly used methods are node-based inference and path-based inference. This paper describes the recent modification made to the SNePS semantic network processing system which previously used only node-based inference, to now allow the combination of node-based and path-based inference. Section 2 briefly discusses the two methods of inference and their differences as well as their respective advantages. The syntax for defining paths appears in Section 3. The cognitive theory involved in combining the two types of inference as well as the actual implementation details are discussed in sections 4 and 5. Finally, a series of examples will be presented which will attempt to

1) illustrate the use of the extensive syntax
2) illustrate some concepts which are difficult if not impossible to implement using only node-based inference but which can be handled easily using path-based inference.

2. PATH-BASED INFERENCE AND NODE-BASED INFERENCE

When discussing path-based inference, it is useful to think of the semantic network as a relational graph. The arcs then represent semantic relationships between the nodes that they connect. Thus, if one is looking for the relationship R between the nodes a and b, one has only to check that the arc labelled R connects these two nodes. Because of this method of representation, relations are restricted to being binary. If a relation does not exist explicitly between two nodes, it may be inferred from a set of rules and other information present in the network. Typically, this other information specifies a "path of arcs" which is semantically equivalent to the original relation being sought. This method of reasoning is known as path-based inference. An example of a path-based inference rule is

Class <-- (Compose Class (Kstar (Compose Sub- Sup)))

This rule has the following interpretation- the arc "Class" is equivalent to the path of arcs which is composed of a "Class" arc followed by zero or more occurences of the set of arcs "Sub-" and "Sup". The idea being expressed here is that "if x is a member of the class y and y is a subset of a larger set say z, then x is also a member of the class z. The specific syntax for describing the paths in SNePS will be given in the next section.

In a system that uses node-based inference, all semantic information (as opposed to structual information) is stored in the nodes i.e. all concepts that are input into or inferred by the system are stored as nodes. The arcs are used only for the purpose of accessing the information as well as for some

implementation details. In order to carry out reasoning, the system must have stored in it, some rules of inference. In SNePS, every rule is costructed using the formalism of the predicate calculus and is stored in the system as a rule node. The antecedents and consequents of a rule node are represented by pattern nodes. Pattern nodes represent a structure of nodes that might occur in the network and use variable nodes to allow generality. When trying to infer a consequent of a rule node, the system tries to find instances of the pattern nodes representing the antecedents. If the matches are successful, then it adds the node representing the consequent to the system. Such a method is known as node-based inference. An example of a rule node appears in figure (2.1). A detailed description of SNePS and its implementation can be found in [2].

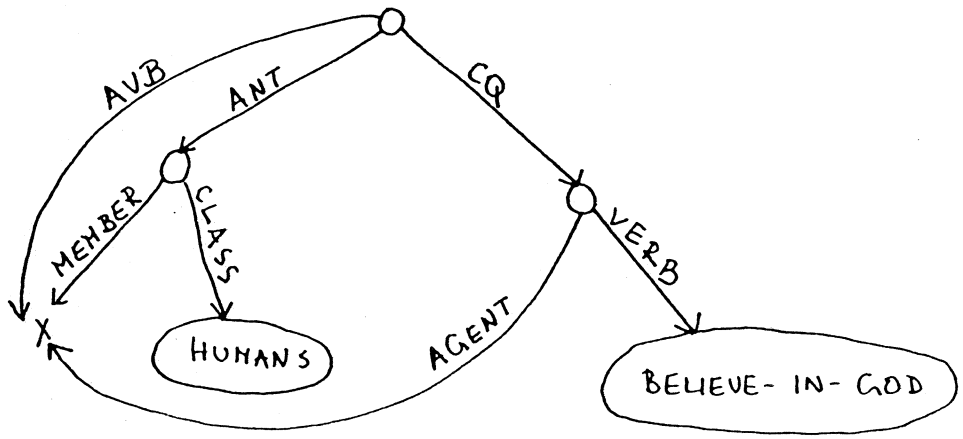

Figure 2.1

Example of a rule-node in SNePS

"IF X IS A HUMAN, THEN X BELIEVES IN GOD"

The primary advantage that path-based inference has over node-based inference is efficiency. In order to infer that a

relation exists between two nodes, one has only to check whether a specified path of arcs goes from one node to the other. An example will be presented later which will demonstrate the efficiency of path-based inference. Node-based inference however, is more general since relationships are not restricted to being binary. For a further discussion of this refer to [1].

3. SYNTAX FOR SPECIFYING PATHS IN SNePS

When specifying alternate path definitions for relations, the user must confine himself to the syntax that is given below. Examples of several of these appear later in the paper. It is important to keep in mind that in SNePS we have three types of relations- ascending, descending and auxiliary. Whenever the relation R is present from node x to node y, the relation R- is present from node y to node x. Auxiliary relations are used strictly for implementation purposes and therefore their converses are not defined. For this reason, we allow only descending and ascending arcs to have alternate path definitions. The syntax for defining paths is given below along with the appropriate SNePSUL commands (SNePSUL being the SNePS user language which permits a user to interact with it). Paths are defined recursively as follows:-

1) A path is either a single relation or a path as described in part 2 of this definition.
2) If P and Q are paths and x,y and z are nodes, paths can be formed in the following manner:-
   a) Converse: If P is a path from x to y, (CONVERSE P) is a path from y to x.
   b) Composition: If $x_1, x_2, \ldots x_n$ are nodes and P is a path from x_1 to x_2, Q is a path from x_2 to x_3,.....Z is a path from x_{n-1} to x_n then (COMPOSE P Q ....Z) is a path from x_1 to x_n.
   c) Composition zero or more times: This corresponds to the Kleene star operator and is defined as follows. If P composed with itself zero or more times is a path from

x to y, then (KSTAR P) is a path from x to y.

d) Composition one or more times: If P composed with itself one or more times is a path from x to y, then (KPLUS P) is a path from x to y.

e) Union: If P is a path from x to y or Q is a path from x to y or R is a path from x to y etc. then (OR P Q R ....) is a path from x to y.

f) Intersection: If P is a path from x to y, and Q is a path from x to y and R is a path from x to y etc. then (AND P Q R ....) is a path from x to y.

g) Complement: This represents universal complement and is described as follows. If there is no path P from x to y, (NOT P) is a path from x to y.

h) Irreflexive restriction: If P is a path from x to y, and x≠y, (IRREFLEXIVE-RESTRICT P) is a path from x to y.

i) Exception: If P is a path from x to y and there is no path Q from x to y with length less than or equal to the length of P, then (EXCEPTION P Q) is a path from x to y.

j) Domain restriction: If P is a path from x to y and Q is a path from x to z, then (DOMAIN-RESTRICT (Q z) P) is a path from x to y.

The extra set of parentheses is to associate the path Q with the node z.

k) Range restriction: If P is a path from x to y and Q is a path from y to z, (RANGE-RESTRICT P (Q z)) is a path from x to y.

The extra set of parentheses is to associate the path

Q with the node z.

1) Relative complement: If P is a path from x to y and there is no path Q from x to y, (RELATIVE-COMPLEMENT P Q) is a path from x to y.

3) In order to specify an alternate path definition for a relation, one issues the command:-

(DEF-PATH PATH-NAME DEFINING-PATH PATH-NAME DEFINING-PATH etc.)

where PATH-NAME is a pre-defined relation and DEFINING-PATH is a path formed by using the above rules.

If a path has already been defined for a given path-name the user will be asked if he wants it redefined. If the user answers negatively, the command will be suspended at that point.

Two things should be mentioned at this point. Whenever an alternate path definition is given, the converse of that path is automatically stored as an alternate path definition for the converse of that relation. To remind the users of this, the path-definitions for both the relation and the converse of the relation are echoed back to the user. Secondly, whenever an atom appears in a path-definition, it is taken to represent the relation of that name and not the alternate path definition for that relation. This is to prevent users from specifying recursive path definitions.

4. COMBINING PATH-BASED AND NODE-BASED INFERENCE

SNePS implements node-based inference through the use of two subsystems. The first is a sophisticated pattern-matching routine (MATCH) which when given a pattern node, scans the network of asserted nodes to find instances of it. The second is the inference subsystem (INFER) which takes as input a command to deduce instances of a given pattern node. It calls on MATCH to locate relevant rule nodes and depending on the logical connectives and quantifiers being used, find instances of antecedents, consequents etc. INFER operates in a multi-processing environment enabling it to deduce the required node much faster.

In order to implement path-based inference, the MATCH routine has been modified so that when searching for an instance of a pattern node, it considers alternate path definitions. That is, when searching for all nodes with the relation R to a node x, it first checks to see if the user has specified an alternate path definition for the relation R. Instead of retrieving all nodes that have the arc R to x, it finds and returns all nodes that have the path of arcs as identified by the name R to x. The path has to conform to the syntax given in section [3] and the path-name is required to be a descending or ascending relation. The path defintions can then be considered to be inference rules and the match routine can be said to perform path-based inference.

Whenever a pattern node representing a node structure to be inferred (e.g. the pattern node in a DEDUCE command) is

succcessfully inferred the following action takes place. The matching node (or nodes depending on how many were found) representing an instance of the pattern node is asserted into the network if it is not already present. By the word "instance" we mean a copy of the pattern node with all variable nodes in it bound to constant nodes. However, the results of intermediary calls to MATCH by INFER are not asserted into the network. Typically, these would involve calls to MATCH in order to match pattern nodes representing antecedents, arguments to numeric quantifiers etc. The point of this is to show how path-based inference and node-based inference can work in unison.

Path-based inference represents the subconcious reasoning that humans do. There are some logical truths that we take for granted and it is not unreasonable to have them built into SNePS so that the system can do "implicit reasoning" where necessary. Node-based inference on the other hand, represents the concious reasoning that we do. This involves a systematic thought procedure which enables us to infer the validity or non-validity of propositions. SNePS has the ability to perform this explicit type of reasoning through the use of rule nodes. It is often the case that we combine the two types of reasoning. For example, we would like to know who believes in God. We are given the rule (by a Sunday-school teacher) stating that "if x is a human, then x believes in God." We know that Tom is a boy and that all boys are humans. If we were to do explicit reasoning only, this information would not enable us to conclude that Tom believes in God. Our subconcious tells us that since Tom is a boy and boys are humans, Tom is a human. This implicit reasoning is expressed

in SNePS as the path-based inference rule

(DEF-PATH Class (Compose Class (Kstar (Compose Sub- Sup))))

We can now go back to explicit reasoning to infer that since Tom is a human, he believes in God. Figure (4.1) is an illustration of this example. Note that a node representing the concept that Tom believes in God has been asserted into the network. However, the intermediate result that Tom is a human is not asserted. If we were to carry out this example through node-based inference only, we would need another rule stating that "If x is a member of the class y and y is a subclass of the class z, then x is also a member of the class z. If we had a long chain of set inclusions separating the source node and the node that we were seeking, the deduction procedure would be slowed down considerably since for each (SUB SUP) relationship, INFER would have to be called. By using path-based inference, we are able to obtain all nodes in the chain at once without having to call on INFER.

This example illustrates that combining the two types of inference might be very beneficial for the overall efficiency of the system.

The implementation of path-based inference adheres to the notion of implicit reasoning since modifications have only been made to the MATCH routine and INFER remains unaltered. The only place where path-definitions are used (or their existence even recognized) is within the MATCH routine. When INFER calls on MATCH, it is oblivious of the fact that some of the resultant matching nodes may have been inferred to exist by using

```
$*:$$$$$ (DEF-PATH CLASS (COMPOSE CLASS (KSTAR (COMPOSE SUB- SUP))))

(CLASS DEFINED ALTERNATELY AS PATH (COMPOSE CLASS (KSTAR (C
OMPOSE SUB- SUP))))

(CLASS- DEFINED ALTERNATELY AS PATH (COMPOSE (KSTAR (COMPOS
E SUP- SUB)) CLASS-))
(DEFINED)
34 MSECS

$*:$$$$$ (BUILD MEMBER TOM CLASS BOYS)
(M1)
29 MSECS

$*:$$$$$ (BUILD SUB BOYS SUP HUMANS)
(M2)
29 MSECS

$*:$$$$$ (BUILD MEMBER JANE CLASS GIRLS)
(M3)
29 MSECS

$*:$$$$$ (BUILD SUB GIRLS SUP HUMANS)
(M4)
29 MSECS

$*:$$$$$ (BUILD AVB ($X)
$*:$$$$$ ANT (BUILD MEMBER *X CLASS HUMANS)
$*:$$$$$ CQ  (BUILD AGENT *X VERB BELIEVE-IN-GOD))
(M7)
111 MSECS

$*:$$$$$ (BUILD MEMBER SPOT CLASS DOGS)
(M8)
29 MSECS

$*:$$$$$ (DEDUCE AGENT %X VERB BELIEVE-IN-GOD)

 SINCE
(M5 (CLASS (HUMANS))
    (:SVAR (V1 (:VAR (T)) (:VAL (TOM))))
    (MEMBER (V1 (:VAR (T)) (:VAL (TOM)))))

 WE INFER
(M6 (VERB (BELIEVE-IN-GOD))
    (:SVAR (V1 (:VAR (T)) (:VAL (TOM))))
    (AGENT (V1 (:VAR (T)) (:VAL (TOM)))))
```

Figure 4.1

COMBINING THE TWO TYPES OF INFERENCE

path-based inference rules as opposed to those that are

```
 SINCE
(M5 (CLASS (HUMANS))
    (:SVAR (V1 (:VAR (T)) (:VAL (JANE))))
    (MEMBER (V1 (:VAR (T)) (:VAL (JANE)))))

 WE INFER
(M6 (VERB (BELIEVE-IN-GOD))
    (:SVAR (V1 (:VAR (T)) (:VAL (JANE))))
    (AGENT (V1 (:VAR (T)) (:VAL (JANE)))))


(M9 M10)
2214 MSECS

$*:$$$$$ (DUMP *NODES)
(M10 (VERB (BELIEVE-IN-GOD)) (AGENT (TOM)))
(M9 (VERB (BELIEVE-IN-GOD)) (AGENT (JANE)))
(SPOT (MEMBER- (M8)))
(DOGS (CLASS- (M8)))
(M8 (CLASS (DOGS)) (MEMBER (SPOT)))
(M7 (CQ (M6)) (ANT (M5)) (AVB (V1)))
(BELIEVE-IN-GOD (VERB- (M10 M9 M6)))
(M6 (VERB (BELIEVE-IN-GOD)) (:SVAR (V1)) (AGENT (V1)) (CQ-
(M7)))
(M5 (CLASS (HUMANS)) (:SVAR (V1)) (MEMBER (V1)) (ANT- (M7))
)
(V1 (:VAR (T)) (MEMBER- (M5)) (AGENT- (M6)) (AVB- (M7)) (:V
AL (JANE)))
(X (:VAL (Q76)))
(M4 (SUP (HUMANS)) (SUB (GIRLS)))
(JANE (MEMBER- (M3)) (AGENT- (M9)))
(GIRLS (CLASS- (M3)) (SUB- (M4)))
(M3 (CLASS (GIRLS)) (MEMBER (JANE)))
(HUMANS (SUP- (M4 M2)) (CLASS- (M5)))
(M2 (SUP (HUMANS)) (SUB (BOYS)))
(TOM (MEMBER- (M1)) (AGENT- (M10)))
(BOYS (CLASS- (M1)) (SUB- (M2)))
(M1 (CLASS (BOYS)) (MEMBER (TOM)))
(DUMPED)
188 MSECS
```

Figure 4.1 contd.

Figure 4.1
COMBINING THE TWO TYPES OF INFERENCE

explicitly present in the network. When the pattern node in a DEDUCE command can be matched directly in the network (without the use of any node-based inference rules), the system returns the message "WE KNOW" followed by the descriptions of the matching nodes. This if true even if the MATCH routine had to make use of some path-based inference rules. On the other hand, if the system has to use node-based deduction rules for a successful match, it responds with the message "WE INFER". This reflects the difference between the two types of reasoning and can be observed in some of the examples that appear later.

5. IMPLEMENTATION DETAILS

The SNePS source program has been modified to accept the new command DEF-PATH. LISP functions have been added which check the validity of the path-definition and print out appropriate error mesages in case of invalid path definitions.

MATCH makes frequent use of the function GET in order to retrieve all nodes that have a specified relation to a given node. The function GET is used since the required nodes are stored as the value of the property which is the name of the relation on the property list of the given node. MATCH has been modified so that all calls to GET are substituted by calls to a new function GET-NODES. The function GET-NODES is similar to GET since it also takes a node and a relation as arguments but it differs in that it also considers the alternate path definition (if it exists) for that relation. If there is no path definition for a relation, then GET-NODES is equivalent to GET. Otherwise, it returns all nodes that have the specified path to or from the node in question depending on whether the relation is ascending or descending. In order to do this, it makes use of the key function PATHGET.

PATHGET takes as arguments a single node and a path and returns the set of nodes that can reached by traversing the given path starting at the original node. While traversing the paths, the function takes into account the syntax for paths given in section [3]. In order to ensure that unasserted nodes are not matched, PATHGET makes use of the function TOP? while traversing paths. Whenever PATHGET encounters an ascending relation (i.e.

whenever PATHGET is required to find all nodes that can be reached from the current nodeset NODE by following an ascending relation), it returns only nodes that satisfy the TOP? requirement. This way no unasserted nodes can be matched (figure 6.5). An exception to this is made for variable nodes in the nodeset NODE. Here, the system is allowed to retrieve nodes other than top-level nodes. This is done to allow matching of nodes representing possible antecedents, consequents etc. of a rule node.

When GET-NODES is given a node and relation and asked to find all nodes having that relation to the given node, it operates in the following way. After verifying that the relation is not an auxiliary one, it checks to see if there is an alternate path definition for that relation. As mentioned before, if it does not find one, the result is the same as calling GET. Otherwise GET-NODES calls on PATHGET to find all nodes that can be reached by traversing the specified path starting at the given node.

At this point it should be mentioned that while a reasonable amount of error-checking is done on the syntax of the paths (see function VALID-PATH), the system cannot detect all errors. Thus if the user is not getting the results he expected, it would be beneficial to trace the function PATHGET (and the function LENGTH-PATH if the exception feature is being used).

A documented listing of all the new functions that have been added to SNePS in order to implement path-based inference can be found in the Appendix.

6. SOME ILLUSTRATIVE EXAMPLES

In this section, several examples of path-based inference will be presented, some of which are used to illustrate the use of the extensive syntax and others which are used to illustrate the many advantages that path-based inference has over node-based inference. In many of the examples, the arc labels are more "meaningful" then they normally would be. This was done to avoid repeated use of domain and range restriction since they were not the primary features being illustrated. For example, an arc labelled "intelligent" is substituted for the case frame (HAVE-PROP x PROP intelligent) since the latter would call for domain restriction. The change has been made for the sake of convenience and in no way affects the validity of the examples.

Figure (6.1) is an example that illustrates the domain restriction feature. The rule that is being expressed here is that "A brother is a male sibling".

Figure (6.2) is an example that illustrates the range

```
$*:$$$$$ (DEF-PATH BROTHER-OF (DOMAIN-RESTRICT (GENDER MALE) SIBLING-OF))

(BROTHER-OF DEFINED ALTERNATELY AS PATH (DOMAIN-RESTRICT (G
ENDER MALE) SIBLING-OF))

(BROTHER-OF- DEFINED ALTERNATELY AS PATH (RANGE-RESTRICT SI
BLING-OF- (GENDER MALE)))
(DEFINED)
29 MSECS

$*:$$$$$ (BUILD SIBLING-OF TOM NAME BILL GENDER MALE)
(M1)
38 MSECS

$*:$$$$$ (BUILD SIBLING-OF TOM NAME NANCY GENDER FEMALE)
(M2)
38 MSECS

$*:$$$$$ (BUILD SIBLING-OF TOM NAME JOHN GENDER MALE)
(M3)
38 MSECS

$*:$$$$$ (BUILD NAME ROBERT GENDER MALE)
(M4)
29 MSECS

$*:$$$$$ (DEDUCE BROTHER-OF TOM NAME %X)

 WE KNOW
(T61 (:SVAR (Q60 (:VAR (T)) (:VAL (JOHN))))
     (NAME (Q60 (:VAR (T)) (:VAL (JOHN))))
     (BROTHER-OF (TOM)))

 WE KNOW
(T61 (:SVAR (Q60 (:VAR (T)) (:VAL (BILL))))
     (NAME (Q60 (:VAR (T)) (:VAL (BILL))))
     (BROTHER-OF (TOM)))
(M5 M6)
464 MSECS
$*:$$$$$ (DUMP *NODES)
(M6 (NAME (JOHN)) (BROTHER-OF (TOM)))
(M5 (NAME (BILL)) (BROTHER-OF (TOM)))
(X (:VAL (Q60)))
(ROBERT (NAME- (M4)))
(M4 (GENDER (MALE)) (NAME (ROBERT)))
(JOHN (NAME- (M6 M3)))
(M3 (GENDER (MALE)) (NAME (JOHN)) (SIBLING-OF (TOM)))
(NANCY (NAME- (M2)))
(FEMALE (GENDER- (M2)))
(M2 (GENDER (FEMALE)) (NAME (NANCY)) (SIBLING-OF (TOM)))
(TOM (SIBLING-OF- (M3 M2 M1)) (BROTHER-OF- (M6 M5)))
(BILL (NAME- (M5 M1)))
(MALE (GENDER- (M4 M3 M1)))
(M1 (GENDER (MALE)) (NAME (BILL)) (SIBLING-OF (TOM)))
```

Figure 6.1

restriction feature. We use the case-frame (HAS-PARENT HAS-PARENT NAME) to indicate that the person represented by the node at the terminal end of the NAME arc has as parents the people represented by the nodes at the terminal ends of the HAS-PARENT arcs. We then use range-restriction to express the rule that a father is a male parent. The system can then infer that in the example of figure (6.2), Tommy has Tom as a father.

The relation "sibling" is "almost" transitive since if x is

```
$*:$$$$$ (DEF-PATH HAS-FATHER (RANGE-RESTRICT HAS-PARENT
$*:$$$$$                        ((COMPOSE MEMBER- CLASS) MALES)))


(HAS-FATHER DEFINED ALTERNATELY AS PATH (RANGE-RESTRICT HAS
-PARENT ((COMPOSE MEMBER- CLASS) MALES)))


(HAS-FATHER- DEFINED ALTERNATELY AS PATH (DOMAIN-RESTRICT (
(COMPOSE MEMBER- CLASS) MALES) HAS-PARENT-))
(DEFINED)
33 MSECS

$*:$$$$$ (BUILD MEMBER TOM CLASS MALES)
(M1)
29 MSECS

$*:$$$$$ (BUILD MEMBER MARY CLASS FEMALES)
(M2)
29 MSECS

$*:$$$$$ (BUILD HAS-PARENT TOM HAS-PARENT MARY NAME TOMMY)
(M3)
37 MSECS

$*:$$$$$ (DEDUCE NAME TOMMY HAS-FATHER %X)

 WE KNOW
(T61 (:SVAR (Q60 (:VAR (T)) (:VAL (TOM))))
     (HAS-FATHER (Q60 (:VAR (T)) (:VAL (TOM))))
     (NAME (TOMMY)))


(M4)
290 MSECS

$*:$$$$$ (DUMP *NODES)
(M4 (HAS-FATHER (TOM)) (NAME (TOMMY)))
(X (:VAL (Q60)))
(TOMMY (NAME- (M4 M3)))
(M3 (NAME (TOMMY)) (HAS-PARENT (TOM MARY)))
(MARY (MEMBER- (M2)) (HAS-PARENT- (M3)))
(FEMALES (CLASS- (M2)))
(M2 (CLASS (FEMALES)) (MEMBER (MARY)))
(TOM (MEMBER- (M1)) (HAS-PARENT- (M3)) (HAS-FATHER- (M4)))
(MALES (CLASS- (M1)))
(M1 (CLASS (MALES)) (MEMBER (TOM)))
(DUMPED)
76 MSECS
```

Figure 6.2

example to show use of range-restriction

a sibling of y and y is a sibling of z, x is a sibling of z. The qualification "almost" is used since "sibling" is not reflexive. The example in figure (6.3) was originally chosen to illustrate the irreflexive restriction feature of the syntax. Originally, the author envisioned a (SIBLING SIBLING) case-frame to denote that the nodes at the terminal ends of the SIBLING arcs were siblings. An alternate path definition for sibling would be given as follows:-

```
(DEF-PATH SIBLING (IRREFLEXIVE-RESTRICT (COMPOSE
      SIBLING (KSTAR (COMPOSE SIBLING- SIBLING)))))
```

In attempting to infer a node which has the relation SIBLING to another node, it is necessary to select an assertion node and starting there follow the alternate path given above to see which nodes can be reached. Clearly the only nodes that can be found by following this path are base nodes. Therefore there is never the possibility of reaching the original node and thus the "IRREFLEXIVE-RESTRICT" used in the path definition becomes redundant. But we are still left with the undesirable possibility of inferring a node representing the assertion that some person is a sibling of himself.

This particular problem was solved using the relative

```
$*:$$$$$ (DEF-PATH SIBLING2 (RELATIVE-COMPLEMENT (COMPOSE SIBLING2
$*:$$$$$    (KSTAR (COMPOSE SIBLING1- SIBLING2))) SIBLING1))


(SIBLING2 DEFINED ALTERNATELY AS PATH (RELATIVE-COMPLEMENT
(COMPOSE SIBLING2 (KSTAR (COMPOSE SIBLING1- SIBLING2))) SIB
LING1))
(SIBLING2- DEFINED ALTERNATELY AS PATH (RELATIVE-COMPLEMENT
 (COMPOSE (KSTAR (COMPOSE SIBLING2- SIBLING1)) SIBLING2-) S
IBLING1-))
(DEFINED)
45 MSECS

$*:$$$$$ (BUILD SIBLING1 JANE SIBLING2 TOM)
(M1)
29 MSECS

$*:$$$$$ (BUILD SIBLING1 TOM SIBLING2 BILL)
(M2)
29 MSECS

$*:$$$$$ (BUILD SIBLING1 BILL SIBLING2 JANE)
(M3)
28 MSECS

$*:$$$$$ (DEDUCE SIBLING1 JANE SIBLING2 %X)

 WE KNOW
(T61 (:SVAR (Q60 (:VAR (T)) (:VAL (BILL))))
     (SIBLING2 (Q60 (:VAR (T)) (:VAL (BILL))))
     (SIBLING1 (JANE)))

 WE KNOW
(T61 (:SVAR (Q60 (:VAR (T)) (:VAL (TOM))))
     (SIBLING2 (Q60 (:VAR (T)) (:VAL (TOM))))
     (SIBLING1 (JANE)))


(M1 M4)
459 MSECS

$*:$$$$$ (DUMP *NODES)
(M4 (SIBLING2 (BILL)) (SIBLING1 (JANE)))
(X (:VAL (Q60)))
(M3 (SIBLING2 (JANE)) (SIBLING1 (BILL)))
(BILL (SIBLING2- (M4 M2)) (SIBLING1- (M3)))
(M2 (SIBLING2 (BILL)) (SIBLING1 (TOM)))
(JANE (SIBLING1- (M4 M1)) (SIBLING2- (M3)))
(TOM (SIBLING2- (M1)) (SIBLING1- (M2)))
(M1 (SIBLING2 (TOM)) (SIBLING1 (JANE)))
```

Figure 6.3

USE OF RELATIVE COMPLEMENT TO IMPLEMENT IRREFLEXIVITY

complement feature of the syntax as illustrated in figure (6.3). Here we use the case frame (SIBLING1 SIBLING2) to indicate that the nodes at the terminal ends of these arcs are siblings. We give an alternate path definition for the relation SIBLING2 which represents the rule that starting at node x, a SIBLING2 arc can be inferred to any node which has the path of arcs (COMPOSE SIBLING2 (KSTAR SIBLING1- SIBLING2)) as long as that node does not already have the relation SIBLING1 from x to itself (indicating irreflexivity).

A more general solution to the problem described above is to allow the path-name to be a path itself. Hitherto we have stated that a path-name must be a single ascending or descending relation. Consider a path definition of the form

(DEF-PATH (Sibling- Sibling) (Irreflexive-restrict
(Kstar (Compose Sibling- Sibling))))

MATCH would then have to be modified to accept a path as a path-name in an alternate path definition. Furthermore, whenever a node y can be found by following the defining-path from a node x, a node would have to be asserted with arc-names corresponding to those in the path-name and terminal nodes corresponding to x, y and any other intermediary nodes that were found. We could now use the path-definition given above to find all pairs of siblings. Further discussion of this can be found in [1].

Figure (6.4) is an implementation of the penguin example which is mentioned in [1]. It has been chosen since it clearly illustrates the use of the exception principle which in the syntax is represented as (EXCEPTION P Q). The need for the

exception principle arises when we specify a general rule but later want to add instances of exceptions to the rule. A hierarchy consisting of birds and penguins along with their properties is first built into the network. A rule for inheritance of properties in this hierarchy is then stated which can be summarized as "If there is a path from node x to a property of any node that x is a subclass of, then x inherits that property- unless there is a path of equal or shorter length from node x to the opposite property". This concept is impossible to represent using only node-based inference. At best you could assert a node representing the exception but this would be inconsistent with what the rule is saying. Returning to the example of figure (6.4), the test cases are MALE-FLYING-PENGUINS, EMPEROR-PENGUINS and CANARIES since we do not explicitly associate any property with them. The example illustrates that MALE-FLYING-PENGUINS and CANARIES correctly inherit the property "CAN-FLY" and that EMPEROR-PENGUINS inherit the property "CANNOT-FLY".

The example in figure (6.5) has been chosen to illustrate

```
$*:$$$$$ (DEF-PATH HAVE-PROP (EXCEPTION (OR HAVE-PROP
$*:$$$$$                                 (COMPOSE PROP PROP- HAVE-PROP
$*:$$$$$                                  (KSTAR (COMPOSE SUP- SUB))))
$*:$$$$$                                (COMPOSE PROP OPA- OPB PROP-
$*:$$$$$                                 HAVE-PROP (KSTAR (COMPOSE SUP-
$*:$$$$$                                                  SUB)))))


(HAVE-PROP DEFINED ALTERNATELY AS PATH (EXCEPTION (OR HAVE-
PROP (COMPOSE PROP PROP- HAVE-PROP (KSTAR (COMPOSE SUP- SUB
)))) (COMPOSE PROP OPA- OPB PROP- HAVE-PROP (KSTAR (COMPOSE
 SUP- SUB)))))


(HAVE-PROP- DEFINED ALTERNATELY AS PATH (EXCEPTION (OR HAVE
-PROP- (COMPOSE (KSTAR (COMPOSE SUB- SUP)) HAVE-PROP- PROP
PROP-)) (COMPOSE (KSTAR (COMPOSE SUB- SUP)) HAVE-PROP- PROP
 OPB- OPA PROP-)))
(DEFINED)
82 MSECS

$*:$$$$$ (BUILD SUB MALE-FLYING-PENGUINS SUP FLYING-PENGUINS)
(M1)
33 MSECS

$*:$$$$$ (BUILD SUB FLYING-PENGUINS SUP PENGUINS)
(M2)
30 MSECS

$*:$$$$$ (BUILD SUB EMPEROR-PENGUINS SUP PENGUINS)
(M3)
33 MSECS

$*:$$$$$ (BUILD SUB PENGUINS SUP BIRDS)
(M4)
30 MSECS

$*:$$$$$ (BUILD SUB CANARIES SUP BIRDS)
(M5)
28 MSECS

$*:$$$$$ (BUILD HAVE-PROP BIRDS PROP CAN-FLY)
(M6)
29 MSECS

$*:$$$$$ (BUILD HAVE-PROP PENGUINS PROP CANNOT-FLY)
(M7)
29 MSECS
```

Figure 6.4

EXCEPTION FEATURE

```
$*:$$$$$ (BUILD HAVE-PROP FLYING-PENGUINS PROP CAN-FLY)
(M8)
29 MSECS

$*:$$$$$ (BUILD OPA CANNOT-FLY OPB CAN-FLY)
(M9)
29 MSECS

$*:$$$$$ (BUILD OPB CANNOT-FLY OPA CAN-FLY)
(M10)
31 MSECS

$*:$$$$$ (DEDUCE HAVE-PROP %X PROP CAN-FLY)

 WE KNOW
(T61
 (PROP (CAN-FLY))
 (:SVAR (Q60 (:VAR (T)) (:VAL (MALE-FLYING-PENGUINS))))
 (HAVE-PROP (Q60 (:VAR (T)) (:VAL (MALE-FLYING-PENGUINS))))
)

 WE KNOW
(T61 (PROP (CAN-FLY))
     (:SVAR (Q60 (:VAR (T)) (:VAL (CANARIES))))
     (HAVE-PROP (Q60 (:VAR (T)) (:VAL (CANARIES)))))

 WE KNOW
(T61 (PROP (CAN-FLY))
     (:SVAR (Q60 (:VAR (T)) (:VAL (BIRDS))))
     (HAVE-PROP (Q60 (:VAR (T)) (:VAL (BIRDS)))))

 WE KNOW
(T61
 (PROP (CAN-FLY))
 (:SVAR (Q60 (:VAR (T)) (:VAL (FLYING-PENGUINS))))
 (HAVE-PROP (Q60 (:VAR (T)) (:VAL (FLYING-PENGUINS)))))

 WE KNOW
(T61
 (PROP (CAN-FLY))
 (:SVAR (Q60 (:VAR (T)) (:VAL (MALE-FLYING-PENGUINS))))
 (HAVE-PROP (Q60 (:VAR (T)) (:VAL (MALE-FLYING-PENGUINS))))
)

 WE KNOW
(T61 (PROP (CAN-FLY))
     (:SVAR (Q60 (:VAR (T)) (:VAL (CANARIES))))
     (HAVE-PROP (Q60 (:VAR (T)) (:VAL (CANARIES)))))

 WE KNOW
(T61
 (PROP (CAN-FLY))
 (:SVAR (Q60 (:VAR (T)) (:VAL (FLYING-PENGUINS))))
 (HAVE-PROP (Q60 (:VAR (T)) (:VAL (FLYING-PENGUINS)))))
```

Figure 6.4
Contd.

```
 WE KNOW
(T61 (PROP (CAN-FLY))
     (:SVAR (Q60 (:VAR (T)) (:VAL (BIRDS))))
     (HAVE-PROP (Q60 (:VAR (T)) (:VAL (BIRDS)))))


(M6 M8 M11 M12)
2869 MSECS

$*:$$$$$ (DEDUCE HAVE-PROP %Y PROP CANNOT-FLY)

 WE KNOW
(T66
 (PROP (CANNOT-FLY))
 (:SVAR (Q65 (:VAR (T)) (:VAL (EMPEROR-PENGUINS))))
 (HAVE-PROP (Q65 (:VAR (T)) (:VAL (EMPEROR-PENGUINS)))))

 WE KNOW
(T66 (PROP (CANNOT-FLY))
     (:SVAR (Q65 (:VAR (T)) (:VAL (PENGUINS))))
     (HAVE-PROP (Q65 (:VAR (T)) (:VAL (PENGUINS)))))


(M7 M13)
793 MSECS

$*:$$$$$ (DUMP *NODES)
(M13 (PROP (CANNOT-FLY)) (HAVE-PROP (EMPEROR-PENGUINS)))
(Y (:VAL (Q65)))
(M12 (PROP (CAN-FLY)) (HAVE-PROP (MALE-FLYING-PENGUINS)))
(M11 (PROP (CAN-FLY)) (HAVE-PROP (CANARIES)))
(X (:VAL (Q60)))
(M10 (OPA (CAN-FLY)) (OPB (CANNOT-FLY)))
(M9 (OPB (CAN-FLY)) (OPA (CANNOT-FLY)))
(M8 (PROP (CAN-FLY)) (HAVE-PROP (FLYING-PENGUINS)))
(CANNOT-FLY (PROP- (M13 M7)) (OPA- (M9)) (OPB- (M10)))
(M7 (PROP (CANNOT-FLY)) (HAVE-PROP (PENGUINS)))
(CAN-FLY (PROP- (M12 M11 M8 M6)) (OPB- (M9)) (OPA- (M10)))
(M6 (PROP (CAN-FLY)) (HAVE-PROP (BIRDS)))
(CANARIES (SUB- (M5)) (HAVE-PROP- (M11)))
(M5 (SUP (BIRDS)) (SUB (CANARIES)))
(BIRDS (SUP- (M5 M4)) (HAVE-PROP- (M6)))
(M4 (SUP (BIRDS)) (SUB (PENGUINS)))
(EMPEROR-PENGUINS (SUB- (M3)) (HAVE-PROP- (M13)))
(M3 (SUP (PENGUINS)) (SUB (EMPEROR-PENGUINS)))
(PENGUINS (SUP- (M3 M2)) (SUB- (M4)) (HAVE-PROP- (M7)))
(M2 (SUP (PENGUINS)) (SUB (FLYING-PENGUINS)))
(MALE-FLYING-PENGUINS (SUB- (M1)) (HAVE-PROP- (M12)))
(FLYING-PENGUINS (SUP- (M1)) (SUB- (M2)) (HAVE-PROP- (M8)))
(M1 (SUP (FLYING-PENGUINS)) (SUB (MALE-FLYING-PENGUINS)))
(DUMPED)
```

Figure 6.4 contd.

two features. One of these is the AND feature of the syntax (representing intersection of paths) and the other is that only nodes that are asserted may be returned by MATCH. In figure (6.5) a node (M5) is included which meets the "AND" requirement but does not meet the requirement that it be asserted. As expected, it is not returned as the result of the DEDUCE command.

Figure (6.6) is the implementation of the Maureen Gelt example discussed in [1] and shows how path-based inference can be used to infer that two intensional concepts are extensionally equivalent. We are given two sets of facts about two individuals Dr. Smith and John Smith and later told that Dr. Smith is John Smith. It is then the case that anything true about Dr. Smith is true about John Smith and vice versa. To relate the two sets of information, the case frame (EQUIV EQUIV) is used and paths are re-defined to include the segment (EQUIV- EQUIV) so that all information pertaining to Dr. Smith can be inferred to be true about John Smith and vice versa. Using node-based inference only, this would be at least very difficult to implement if not impossible since it involves building rules with antecedents that quantify over higher-order case-frames such as (MEMBER CLASS). Further discussion of this can be found in [1].

```
$*:$$$$$ (DEFINE SUCCEED SUCCEED- HARD-WORKING HARD-WORKING-
$*:$$$$$  INTELLIGENT INTELLIGENT-)
(SUCCEED SUCCEED-)
(HARD-WORKING HARD-WORKING-)
(INTELLIGENT INTELLIGENT-)
(DEFINED)
24 MSECS

$*:$$$$$ (DEF-PATH SUCCEED (AND HARD-WORKING INTELLIGENT))


(SUCCEED DEFINED ALTERNATELY AS PATH (AND HARD-WORKING INTE
LLIGENT))


(SUCCEED- DEFINED ALTERNATELY AS PATH (AND HARD-WORKING- IN
TELLIGENT-))
(DEFINED)
29 MSECS

$*:$$$$$ (BUILD HARD-WORKING TOM INTELLIGENT TOM)
(M1)
29 MSECS

$*:$$$$$ (BUILD INTELLIGENT MARY)
(M2)
20 MSECS

$*:$$$$$ (BUILD HARD-WORKING BILL)
(M3)
20 MSECS

$*:$$$$$ (BUILD INTELLIGENT MICHAEL HARD-WORKING MICHAEL)
(M4)
30 MSECS

$*:$$$$$ (BUILD MIN 0 MAX 0
$*:$$$$$  ARG (BUILD INTELLIGENT ROBERT HARD-WORKING ROBERT))
(M6)
61 MSECS

$*:$$$$$ (DEDUCE SUCCEED %X)
```

Figure 6.5

"AND" and TOP-LEVEL NODE FEATURES

Figure 6.5

```
 WE KNOW
(T61 (:SVAR (Q60 (:VAR (T)) (:VAL (MICHAEL))))
     (SUCCEED (Q60 (:VAR (T)) (:VAL (MICHAEL)))))

 WE KNOW
(T61 (:SVAR (Q60 (:VAR (T)) (:VAL (TOM))))
     (SUCCEED (Q60 (:VAR (T)) (:VAL (TOM)))))


(M7 M8)
488 MSECS

$*:$$$$$ (DUMP *NODES)
(M8 (SUCCEED (MICHAEL)))
(M7 (SUCCEED (TOM)))
(X (:VAL (Q60)))
(M6 (ARG (M5)) (MAX (0)) (MIN (0)))
(ROBERT (HARD-WORKING- (M5)) (INTELLIGENT- (M5)))
(M5 (HARD-WORKING (ROBERT)) (INTELLIGENT (ROBERT)) (ARG- (M
6)))
(MICHAEL (HARD-WORKING- (M4)) (INTELLIGENT- (M4)) (SUCCEED-
 (M8)))
(M4 (HARD-WORKING (MICHAEL)) (INTELLIGENT (MICHAEL)))
(BILL (HARD-WORKING- (M3)))
(M3 (HARD-WORKING (BILL)))
(MARY (INTELLIGENT- (M2)))
(M2 (INTELLIGENT (MARY)))
(TOM (INTELLIGENT- (M1)) (HARD-WORKING- (M1)) (SUCCEED- (M7
)))
(M1 (INTELLIGENT (TOM)) (HARD-WORKING (TOM)))
(DUMPED)
119 MSECS
```

Figure 6.5 contd.

Finally we present an example figure (6.7) which illustrates

```
$*:$$$$$ (DEF-PATH AGENT (COMPOSE AGENT (KSTAR (COMPOSE EQUIV- EQUIV)))
$*:$$$$$          MEMBER (COMPOSE MEMBER (KSTAR (COMPOSE EQUIV- EQUIV)))
$*:$$$$$       HAS-PROP (COMPOSE HAS-PROP (KSTAR (COMPOSE EQUIV- EQUIV))))

(AGENT DEFINED ALTERNATELY AS PATH (COMPOSE AGENT (KSTAR (C
OMPOSE EQUIV- EQUIV))))

(AGENT- DEFINED ALTERNATELY AS PATH (COMPOSE (KSTAR (COMPOS
E EQUIV- EQUIV)) AGENT-))

(MEMBER DEFINED ALTERNATELY AS PATH (COMPOSE MEMBER (KSTAR
(COMPOSE EQUIV- EQUIV))))

(MEMBER- DEFINED ALTERNATELY AS PATH (COMPOSE (KSTAR (COMPO
SE EQUIV- EQUIV)) MEMBER-))

(HAS-PROP DEFINED ALTERNATELY AS PATH (COMPOSE HAS-PROP (KS
TAR (COMPOSE EQUIV- EQUIV))))

(HAS-PROP- DEFINED ALTERNATELY AS PATH (COMPOSE (KSTAR (COM
POSE EQUIV- EQUIV)) HAS-PROP-))
(DEFINED)
97 MSECS

$*:$$$$$ (BUILD MEMBER DR.SMITH CLASS SURGEONS)
(M1)
32 MSECS

$*:$$$$$ (BUILD OBJ MAUREEN-GELT VERB SAVED AGENT DR.SMITH)
(M2)
39 MSECS

$*:$$$$$ (BUILD HAS-PROP JOHN-SMITH PROP FIVE-FEET-SIX-INCHES-TALL
$*:$$$$$                            PROP BLACK-HAIR)
(M3)
40 MSECS

$*:$$$$$ (BUILD EQUIV DR.SMITH EQUIV JOHN-SMITH)
(M4)
31 MSECS

$*:$$$$$ (DEDUCE HAS-PROP DR.SMITH PROP ?X)
```

Figure 6.6

EXTENSIONAL EQUIVALENCE OF INTENSIONAL CONCEPTS

```
 WE KNOW
(T61
 (:SVAR (Q60 (:VAR (T)) (:VAL (FIVE-FEET-SIX-INCHES-TALL)))
 )
 (PROP (Q60 (:VAR (T)) (:VAL (FIVE-FEET-SIX-INCHES-TALL))))
 (HAS-PROP (DR.SMITH)))

 WE KNOW
(T61 (:SVAR (Q60 (:VAR (T)) (:VAL (BLACK-HAIR))))
     (PROP (Q60 (:VAR (T)) (:VAL (BLACK-HAIR))))
     (HAS-PROP (DR.SMITH)))


(M5 M6)
478 MSECS

$*:$$$$$ (DEDUCE OBJ MAUREEN-GELT VERB SAVED AGENT %Y)

 WE KNOW
(T66 (:SVAR (Q65 (:VAR (T)) (:VAL (JOHN-SMITH))))
     (AGENT (Q65 (:VAR (T)) (:VAL (JOHN-SMITH))))
     (VERB (SAVED))
     (OBJ (MAUREEN-GELT)))

 WE KNOW
(T66 (:SVAR (Q65 (:VAR (T)) (:VAL (DR.SMITH))))
     (AGENT (Q65 (:VAR (T)) (:VAL (DR.SMITH))))
     (VERB (SAVED))
     (OBJ (MAUREEN-GELT)))


(M2 M7)
540 MSECS

$*:$$$$$ (DEDUCE MEMBER JOHN-SMITH CLASS SURGEONS)

 WE KNOW
(T70 (CLASS (SURGEONS)) (MEMBER (JOHN-SMITH)))


(M8)
260 MSECS
```

```
$*:$$$$$ (DUMP *NODES)
(M8 (CLASS (SURGEONS)) (MEMBER (JOHN-SMITH)))
(M7 (AGENT (JOHN-SMITH)) (VERB (SAVED)) (OBJ (MAUREEN-GELT)
))
(Y (:VAL (Q65)))
(M6 (PROP (FIVE-FEET-SIX-INCHES-TALL)) (HAS-PROP (DR.SMITH)
))
(M5 (PROP (BLACK-HAIR)) (HAS-PROP (DR.SMITH)))
(X (:VAL (Q60)))
(M4 (EQUIV (DR.SMITH JOHN-SMITH)))
(JOHN-SMITH (HAS-PROP- (M3)) (EQUIV- (M4)) (AGENT- (M7)) (M
EMBER- (M8)))
(FIVE-FEET-SIX-INCHES-TALL (PROP- (M6 M3)))
(BLACK-HAIR (PROP- (M5 M3)))
(M3 (PROP (FIVE-FEET-SIX-INCHES-TALL BLACK-HAIR)) (HAS-PROP
 (JOHN-SMITH)))
(MAUREEN-GELT (OBJ- (M7 M2)))
(SAVED (VERB- (M7 M2)))
(M2 (AGENT (DR.SMITH)) (VERB (SAVED)) (OBJ (MAUREEN-GELT)))
(DR.SMITH (MEMBER- (M1)) (AGENT- (M2)) (EQUIV- (M4)) (HAS-P
ROP- (M6 M5)))
(SURGEONS (CLASS- (M8 M1)))
(M1 (CLASS (SURGEONS)) (MEMBER (DR.SMITH)))
(DUMPED)
1018 MSECS
```

Figure 6.6 contd.

the efficiency that path-based inference has over node-based inference. A hierarchy of automobiles is built into the network along with a node representing the fact that automobiles are fuel-powered. A path-based rule is specified which states that "if x is a subclass of y and y has the property z, then x also has this property. We also build rule nodes (M8 M11 M14) which when combined allow us to conclude that Harrys car has the property of being fuel-powered. After going through a long chain of set inclusions, we successfully infer (using path-based inference) that the Dodge is fuel-powered. Compare the execution time of this deduction (348 msecs) to the execution time involved in inferring that Harrys car is fuel-powered (3944 msecs). For this last deduction, the system is forced to use the rule nodes since Harrys car is not linked to the automobile hierarchy with the use of the (SUB SUP) case frame. Clearly path-based inference is more efficient and should be used whenever possible.

```
$*:$$$$$ (DEF-PATH HAVE-PROP (COMPOSE HAVE-PROP (KSTAR (COMPOSE SUP- SUB))))


(HAVE-PROP DEFINED ALTERNATELY AS PATH (COMPOSE HAVE-PROP (
KSTAR (COMPOSE SUP- SUB))))


(HAVE-PROP- DEFINED ALTERNATELY AS PATH (COMPOSE (KSTAR (CO
MPOSE SUB- SUP)) HAVE-PROP-))
(DEFINED)
36 MSECS

$*:$$$$$ (BUILD SUB DODGE-AUTOMOBILES SUP CHRYSLER-AUTOMOBILES)
(M1)
31 MSECS

$*:$$$$$ (BUILD SUB CHRYSLER-AUTOMOBILES SUP AMERICAN-AUTOMOBILES)
(M2)
30 MSECS

$*:$$$$$ (BUILD SUB AMERICAN-AUTOMOBILES SUP AUTOMOBILES)
(M3)
31 MSECS

$*:$$$$$ (BUILD HAVE-PROP AUTOMOBILES PROP FUEL-POWERED)
(M4)
29 MSECS

$*:$$$$$ (BUILD MEMBER HARRYS-CAR CLASS MERCEDES)
(M5)
31 MSECS

$*:$$$$$ (BUILD AVB ($X)
$*:$$$$$   ANT (BUILD MEMBER *X CLASS MERCEDES)
$*:$$$$$   CQ  (BUILD MEMBER *X CLASS GERMAN-AUTOMOBILES))
(M8)
110 MSECS

$*:$$$$$ (BUILD AVB ($X)
$*:$$$$$   ANT (BUILD MEMBER *X CLASS GERMAN-AUTOMOBILES)
$*:$$$$$   CQ  (BUILD MEMBER *X CLASS AUTOMOBILES))
(M11)
115 MSECS

$*:$$$$$ (BUILD AVB ($X)
$*:$$$$$   ANT (BUILD MEMBER *X CLASS AUTOMOBILES)
$*:$$$$$   CQ  (BUILD HAVE-PROP *X PROP FUEL-POWERED))
(M14)
835 MSECS
```

Figure 6.7

EFFICIENCY OF PATH-BASED INFERENCE

```
$*:$$$$$ (DEDUCE HAVE-PROP DODGE-AUTOMOBILES PROP FUEL-POWERED)

 WE KNOW
(T60 (PROP (FUEL-POWERED)) (HAVE-PROP (DODGE-AUTOMOBILES)))


(M15)
348 MSECS

$*:$$$$$ (DEDUCE HAVE-PROP HARRYS-CAR PROP FUEL-POWERED)

 SINCE
(M6 (CLASS (MERCEDES))
    (:SVAR (V1 (:VAR (T)) (:VAL (HARRYS-CAR))))
    (MEMBER (V1 (:VAR (T)) (:VAL (HARRYS-CAR)))))

 WE INFER
(M7
 (CLASS (GERMAN-AUTOMOBILES))
 (:SVAR (V1 (:VAR (T)) (:VAL (HARRYS-CAR))))
 (MEMBER (V1 (:VAR (T)) (:VAL (HARRYS-CAR)))))

 SINCE
(M9
 (CLASS (GERMAN-AUTOMOBILES))
 (:SVAR (V2 (:VAR (T)) (:VAL (HARRYS-CAR))))
 (MEMBER (V2 (:VAR (T)) (:VAL (HARRYS-CAR)))))

 WE INFER
(M10
 (CLASS (AUTOMOBILES))
 (:SVAR (V2 (:VAR (T)) (:VAL (HARRYS-CAR))))
 (MEMBER (V2 (:VAR (T)) (:VAL (HARRYS-CAR)))))

 SINCE
(M12
 (CLASS (AUTOMOBILES))
 (:SVAR (V3 (:VAR (T)) (:VAL (HARRYS-CAR))))
 (MEMBER (V3 (:VAR (T)) (:VAL (HARRYS-CAR)))))

 WE INFER
(M13
 (PROP (FUEL-POWERED))
 (:SVAR (V3 (:VAR (T)) (:VAL (HARRYS-CAR))))
 (HAVE-PROP (V3 (:VAR (T)) (:VAL (HARRYS-CAR)))))


(M16 M17 M18)
3944 MSECS
```

Figure 6.7

Contd.

```
$*:$$$$$ (DUMP *NODES)
(M18 (PROP (FUEL-POWERED)) (HAVE-PROP (HARRYS-CAR)))
(M17 (CLASS (AUTOMOBILES)) (MEMBER (HARRYS-CAR)))
(M16 (CLASS (GERMAN-AUTOMOBILES)) (MEMBER (HARRYS-CAR)))
(M15 (PROP (FUEL-POWERED)) (HAVE-PROP (DODGE-AUTOMOBILES)))
(M14 (CQ (M13)) (ANT (M12)) (AVB (V3)))
(M13 (PROP (FUEL-POWERED)) (:SVAR (V3)) (HAVE-PROP (V3)) (C
Q- (M14)))
(M12 (CLASS (AUTOMOBILES)) (:SVAR (V3)) (MEMBER (V3)) (ANT-
 (M14)))
(V3 (:VAR (T)) (MEMBER- (M12)) (HAVE-PROP- (M13)) (AVB- (M1
4)) (:VAL (HARRYS-CAR)))
(M11 (CQ (M10)) (ANT (M9)) (AVB (V2)))
(M10 (CLASS (AUTOMOBILES)) (:SVAR (V2)) (MEMBER (V2)) (CQ-
(M11)))
(M9 (CLASS (GERMAN-AUTOMOBILES)) (:SVAR (V2)) (MEMBER (V2))
 (ANT- (M11)))
(V2 (:VAR (T)) (MEMBER- (M10 M9)) (AVB- (M11)) (:VAL (HARRY
S-CAR)))
(M8 (CQ (M7)) (ANT (M6)) (AVB (V1)))
(GERMAN-AUTOMOBILES (CLASS- (M16 M9 M7)))
(M7 (CLASS (GERMAN-AUTOMOBILES)) (:SVAR (V1)) (MEMBER (V1))
 (CQ- (M8)))
(M6 (CLASS (MERCEDES)) (:SVAR (V1)) (MEMBER (V1)) (ANT- (M8
)))
(V1 (:VAR (T)) (MEMBER- (M7 M6)) (AVB- (M8)) (:VAL (HARRYS-
CAR)))
(X (:VAL (V3)))
(HARRYS-CAR (MEMBER- (M17 M16 M5)) (HAVE-PROP- (M18)))
(MERCEDES (CLASS- (M6 M5)))
(M5 (CLASS (MERCEDES)) (MEMBER (HARRYS-CAR)))
(FUEL-POWERED (PROP- (M18 M15 M13 M4)))
(M4 (PROP (FUEL-POWERED)) (HAVE-PROP (AUTOMOBILES)))
(AUTOMOBILES (SUP- (M3)) (HAVE-PROP- (M4)) (CLASS- (M17 M12
 M10)))
(M3 (SUP (AUTOMOBILES)) (SUB (AMERICAN-AUTOMOBILES)))
(AMERICAN-AUTOMOBILES (SUP- (M2)) (SUB- (M3)))
(M2 (SUP (AMERICAN-AUTOMOBILES)) (SUB (CHRYSLER-AUTOMOBILES
)))
(DODGE-AUTOMOBILES (SUB- (M1)) (HAVE-PROP- (M15)))
(CHRYSLER-AUTOMOBILES (SUP- (M1)) (SUB- (M2)))
(M1 (SUP (CHRYSLER-AUTOMOBILES)) (SUB (DODGE-AUTOMOBILES)))
(DUMPED)
```

Figure 6.7

Contd.

7. CONCLUSIONS

A modification to SNePS has been described which consists of adding path-based inference to a system which previously used only node-based inference. A brief discussion of the two types of inference is given followed by the formal syntax for defining paths in SNePS. Section [4] discussed the theory involved in combining the two types of inference and the signficance of the fact that all the changes are restricted to the MATCH subsystem. The section on implementation details appears for those who are familiar with MATCH and the LISP functions that it is comprised of. Finally a series of examples are presented which not only illustrate the use of the syntax, but also illustrate concepts that currently cannot be implemented using node-based inference alone.

8. APPENDIX

DEF-PATH

THIS FUNCTION ENABLES THE USER TO SPECIFY ALTERNATE PATH-DEFINITIONS FOR PREVIOUSLY DEFINED RELATIONS. IF AN ALTERNATE PATH-DEFINITION ALREADY EXISTS FOR THAT RELATION, THE USER WILL BE ASKED WHETHER HE WANTS IT REDEFINED AND ACTION IS TAKEN ACCORDINGLY. THE SYNTAX FOR DEFINING PATHS APPEARS IN THE SNEPS USER MANUAL.

```
     DEF-PATH
VALUE
(FLAMBDA ////L
   (REPEAT (PATH-NAME PATH-SPEC)
     BEGIN (IF (NULL ////L) (RETURN '(DEFINED)))
           (SETQ PATH-NAME (CAR ////L))
           (IF (NULL (CDR ////L))
               (RETURN (LIST 'PATH-DEFINITION 'EXPECTED)))
           (SETQ PATH-SPEC (CADR ////L))
           (IF (NOT (OR (DNREL PATH-NAME) (UPREL PATH-NAME)))
               (RETURN (LIST PATH-NAME
                             'NOT
                             'A
                             'DESCENDING
                             'OR
                             'ASCENDING
                             'RELATION)))
           (COND
            ((ISPATH PATH-NAME)
             (PRINT (LIST PATH-NAME
                          'ALREADY
                          'DEFINED
                          'AS
                          'PATH
                          (GET PATH-NAME 'PATHDEF)))
             (PRINT (LIST 'DO
                          'YOU
                          'WANT
                          'IT
                          'REDEFINED?
                          'TYPE
                          'YES
                          'OR
                          'NO))
```

APPENDIX

```
            (IF (EQ (READ) 'NO) (RETURN (LIST 'COMMAND 'CANCELLED)))
            T)
           (T T))
          (IF (NOT (VALID-PATH PATH-SPEC))
              (RETURN (LIST 'INVALID 'PATH 'SPECIFIED)))
          (PUT PATH-NAME 'PATHDEF PATH-SPEC)
          (PUT (CONV PATH-NAME) 'PATHDEF (CONV-PATH PATH-SPEC))
          (PRIN3 <> <>)
          (PRINT (LIST PATH-NAME
                       'DEFINED
                       'ALTERNATELY
                       'AS
                       'PATH
                       PATH-SPEC))
          (PRIN3 <> <>)
          (PRINT (LIST (CONV PATH-NAME)
                       'DEFINED
                       'ALTERNATELY
                       'AS
                       'PATH
                       (GET (CONV PATH-NAME) 'PATHDEF)))
          (SETQ ////L (CDDR ////L))))
```

PLIST
NIL

ISPATH

THIS FUNCTION RETRIEVES THE PATH-DEFINITION (IF SPECIFIED EARLIER BY THE USER) WHICH IS ASSOCIATED WITH THE RELATION "ARC". IF THERE IS NO PATH-DEFINITION FOR THIS RELATION, THEN IT RETURNS NIL.

ISPATH
VALUE

```
(LAMBDA (ARC) (GET ARC 'PATHDEF))
```

PLIST
NIL

APPENDIX

VALID-PATH

THIS FUNCTION TAKES A PATH AS AN ARGUMENT AND RETURNS T IF THE PATH CONFORMS TO THE SYNTAX GIVEN IN THE SNEPS USER MANUAL. A LIMITED AMOUNT OF ERROR-CHECKING IS DONE (E.G. CHECKING FOR CORRECT NUMBER AND TYPE OF ARGUMENTS) BUT NO CHECKING IS DONE TO TEST THE SEMANTIC CORRECTNESS OF THE PATH. VALID-PATH RETURNS NIL IF ANY ERROR IS FOUND IN THE PATH-DEFINITION.

```
     VALID-PATH
VALUE
(LAMBDA (ARC-LIST)
   (COND
    ((NULL ARC-LIST) NIL)
    ((ATOM ARC-LIST) (OR (DNREL ARC-LIST) (UPREL ARC-LIST)))
    ((EQ (CAR ARC-LIST) 'COMPOSE)
     (REPEAT (CPATH)
             (SETQ CPATH (CDR ARC-LIST))
       BEGIN (IF (NOT (VALID-PATH (CAR CPATH))) (RETURN NIL))
             (SETQ CPATH (CDR CPATH))
             (IF (NULL CPATH) (RETURN T))))
    ((EQ (CAR ARC-LIST) 'CONVERSE)
     (COND ((NOT (EQ (CDDR ARC-LIST) NIL)) NIL)
           (T (VALID-PATH (CADR ARC-LIST)))))
    ((OR (EQ (CAR ARC-LIST) 'OR) (EQ (CAR ARC-LIST) 'AND))
     (REPEAT (PATH-LIST)
             (SETQ PATH-LIST (CDR ARC-LIST))
             (IF (NULL (CDR PATH-LIST)) (RETURN NIL))
       BEGIN (IF (NOT (VALID-PATH (CAR PATH-LIST))) (RETURN NIL))
             (SETQ PATH-LIST (CDR PATH-LIST))
             (IF (NULL PATH-LIST) (RETURN T))))
    ((OR (EQ (CAR ARC-LIST) 'NOT)
         (EQ (CAR ARC-LIST) 'IRREFLEXIVE-RESTRICT))
     (COND ((NOT (EQ (CDDR ARC-LIST) NIL)) NIL)
           (T (VALID-PATH (CADR ARC-LIST)))))
    ((OR (EQ (CAR ARC-LIST) 'KSTAR) (EQ (CAR ARC-LIST) 'KPLUS))
     (VALID-PATH (CADR ARC-LIST)))
    ((EQ (CAR ARC-LIST) 'RELATIVE-COMPLEMENT)
     (AND (VALID-PATH (CADR ARC-LIST)) (VALID-PATH (RAC ARC-LIST))))
    ((EQ (CAR ARC-LIST) 'EXCEPTION)
     (AND (VALID-PATH (CADR ARC-LIST)) (VALID-PATH (RAC ARC-LIST))))
    ((EQ (CAR ARC-LIST) 'DOMAIN-RESTRICT)
     (AND (VALID-PATH (CAADR ARC-LIST))
          (VALID-PATH (RAC ARC-LIST))
          (ATOM (CADR (CADR ARC-LIST)))))
    ((EQ (CAR ARC-LIST) 'RANGE-RESTRICT)
     (AND (VALID-PATH (CADR ARC-LIST))
          (VALID-PATH (CAR (RAC ARC-LIST)))
          (ATOM (CADR (RAC ARC-LIST)))))))

PLIST
NIL
```

APPENDIX

CONV-PATH

THIS FUNCTION TAKES A PATH (WHICH HAS BEEN DETERMINED TO BE VALID) AS ARGUMENT AND RETURNS THE CONVERSE OF THAT PATH. THE CONVERSE OF A NULL PATH IS ASSUMED TO BE NIL. THE USER SHOULD KEEP IN MIND THAT THE CONVERSE OF PATHS IS NOT THE SAME AS COMPLEMENTS OF SETS- E.G. THE CONVERSE OF THE PATH (OR P Q) WHERE P AND Q ARE PATHS IS (OR (CONVERSE OF PATH P) (CONVERSE OF PATH Q)) AND NOT (AND (CONVERSE OF PATH P) (CONVERSE OF PATH Q)).

```
CONV-PATH
VALUE
(LAMBDA (PATH)
   (COND
    ((NULL PATH) NIL)
    ((ATOM PATH) (CONV PATH))
    ((EQ (CAR PATH) 'COMPOSE)
     (CONS 'COMPOSE (MAPCAR (REVERSE (CDR PATH)) CONV-PATH)))
    ((EQ (CAR PATH) 'CONVERSE)
     (CONS 'CONVERSE (LIST (CONV-PATH (CADR PATH)))))
    ((OR (EQUAL (CAR PATH) 'KSTAR) (EQUAL (CAR PATH) 'KPLUS))
     (CONS (CAR PATH) (MAPCAR (REVERSE (CDR PATH)) CONV-PATH)))
    ((EQ (CAR PATH) 'OR) (CONS 'OR
                               (MAPCAR (CDR PATH)
                                       (LAMBDA (X) (CONV-PATH X)))))
    ((EQ (CAR PATH) 'AND) (CONS 'AND
                                (MAPCAR (CDR PATH)
                                        (LAMBDA (X) (CONV-PATH X)))))
    ((EQ (CAR PATH) 'NOT) (LIST 'NOT (CONV-PATH (CADR PATH))))
    ((EQ (CAR PATH) 'RELATIVE-COMPLEMENT)
     (LIST 'RELATIVE-COMPLEMENT
           (CONV-PATH (CADR PATH))
           (CONV-PATH (RAC PATH))))
    ((EQ (CAR PATH) 'IRREFLEXIVE-RESTRICT)
     (LIST 'IRREFLEXIVE-RESTRICT (CONV-PATH (CADR PATH))))
    ((EQ (CAR PATH) 'EXCEPTION)
     (LIST 'EXCEPTION (CONV-PATH (CADR PATH)) (CONV-PATH (RAC PATH))))
    ((EQ (CAR PATH) 'DOMAIN-RESTRICT)
     (LIST 'RANGE-RESTRICT
           (CONV-PATH (RAC PATH))
           (LIST (CAADR PATH) (CADR (CADR PATH)))))
    ((EQ (CAR PATH) 'RANGE-RESTRICT)
     (LIST 'DOMAIN-RESTRICT
           (LIST (CAR (RAC PATH)) (CADR (RAC PATH)))
           (CONV-PATH (CADR PATH))))))

PLIST
NIL
```

INTERSCTN

THIS FUNCTION COMPUTES THE INTERSECTION OF TWO SETS- THAT IS, IT RETURNS L1 ∩ L2.

```
     INTERSCTN
VALUE
(LAMBDA (L1 L2) (MAPCONC L1 (LAMBDA (X) (IF (MEMEQL X L2) (LIST X)))))

PLIST
NIL
```

COMPLEMENT

THIS FUNCTION COMPUTES SET DIFFERENCE. IT RETURNS THE SET OF NODES WHICH REPRESENTS L1 - L2.

```
     COMPLEMENT
VALUE
(LAMBDA (L1 L2)
   (MAPCONC L1 (LAMBDA (X) (IF (NOT (MEMEQL X L2)) (LIST X)))))

PLIST
NIL
```

SHORTER-LENGTH

GIVEN TWO LISTS OF ELEMENTS, EACH ELEMENT CONSISTING OF (NODE-IDENTIFIER PATH-LENGTH), THIS FUNCTION RETURNS ANOTHER LIST OF ELEMENTS SAY L3, SUCH THAT EVERY NODE-IDENTIFIER IN L3 IS ALSO FOUND IN L1 AND L2 AND THAT IT IS ASSOCIATED WITH A SHORTER OR EQUAL PATH-LENGTH IN THE LIST L2 THAN THE PATH-LENGTH WITH WHICH IT IS ASSOCIATED IN THE LIST L1. THIS IS USED TO IMPLEMENT THE EXCEPTION FEATURE OF THE SYNTAX. THIS FUNCTION IS CALLED BY THE FUNCTION LENGTH-PATH.

```
     SHORTER-LENGTH
VALUE
(LAMBDA (L1 L2)
   (MAPCONC
    L1
    (LAMBDA (X)
       (MAPCONC L2
                (LAMBDA (Y)
                   (IF (AND (EQ (CAR X) (CAR Y))
                            (NOT (LESSP (CADR X) (CADR Y))))
                       (LIST X)))))))

PLIST
NIL
```

APPENDIX

LONGER-PATH-NODES

This returns a list of elements each of the form (node identifier path-length) such that every node-identifier in this list can be found in either the list s1 or the list s2 with a shorter path-length- i.e. can be reached from the original node by a shorter path. These nodes will eventually be deleted from the union of the lists s1 and s2.

```
     LONGER-PATH-NODES
VALUE
(LAMBDA (S1 S2)
   (MAPCONC
    S1
    (LAMBDA (X)
       (MAPCONC
        S2
        (LAMBDA (Y)
           (IF (EQ (CAR X) (CAR Y))
               (COND ((LESSP (CADR X) (CADR Y)) (LIST Y))
                     ((LESSP (CADR Y) (CADR X)) (LIST X))
                     (T NIL))))))))

PLIST
NIL
```

COMPL-WITH-PLENGTH

RETURNS THE SET OF NODES WHICH IS FOUND BY TAKING THE SET DIFFERENCE OF S1 AND S2 I.E.- (S1 - S2). THE ONLY DIFFERENCE BETWEEN THIS AND THE FUNCTION COMPLEMENT IS THAT THE ELEMENTS HERE COMPRISE OF A NODE-IDENTIFIER AND A PATH-LENGTH. THIS FUNCTION IS CALLED BY THE FUNCTION LENGTH-PATH.
TWO ELEMENTS ARE CONSIDERED TO BE THE SAME IF THEIR NODE-IDENTIFIER IS THE SAME (I.E. THE PATH-LENGTH IS IGNORED).

```
     COMPL-WITH-PLENGTH
VALUE
(LAMBDA (S1 S2)
   (MAPCONC S1
            (LAMBDA (X)
               (IF (NOT (MEMEQL (CAR X)
                                (MAPCAR S2 (LAMBDA (Y) (CAR Y)))))
                   (LIST X)))))

PLIST
NIL
```

APPENDIX

INT-NODES-WITH-PLENGTH

RETURNS THE SET OF NODES WHICH REPRESENTS THE INTERSECTION OF THE SETS S1 AND S2. IF THE SAME NODE-IDENTIFIER IS FOUND IN BOTH SETS S1 AND S2, THE ONE WITH THE SHORTER PATH-LENGTH ASSOCIATED WITH IT IS INCLUDED.

```
INT-NODES-WITH-PLENGTH
VALUE
(LAMBDA (S1 S2)
   (PROG (TEMP1 TEMP2)
         (SETQ TEMP1 (LONGER-PATH-NODES S1 S2))
         (SETQ TEMP2 (MAPCONC
                      S1
                      (LAMBDA (X)
                         (IF (MEMEQL (CAR X)
                                     (MAPCAR S2 (LAMBDA (Y) (CAR Y))))
                             (LIST X)))))
         (RETURN (COMPLEMENT TEMP2 TEMP1))))

PLIST
NIL

NIL
```

GET-NODES

GET-NODES TAKES A NODE AND A RELATION AS ARGUMENTS. IT CHECKS TO SEE WHETHER THERE IS A PATH-DEFINITION FOR THAT RELATION. IF THERE IS NOT, GET-NODES IS THE SAME AS GET. OTHERWISE, GET-NODES CALLS ON THE FUNCTION PATH-GET TO RETRIEVE ALL NODES THAT CAN BE REACHED BY FOLLOWING THE PATH ASSOCIATED WITH THE GIVEN RELATION STARTING AT THE GIVEN NODE.

```
GET-NODES
VALUE
(LAMBDA (START-NODE RELN)
   (COND
    ((AUXREL RELN) (GET START-NODE RELN))
    ((NOT (ISPATH RELN)) (GET START-NODE RELN))
    (T (PATHGET START-NODE (GET RELN 'PATHDEF)))))

PLIST
NIL
```

APPENDIX

PATHGET

FINDS AND RETURNS A SET OF NODES WHICH CAN BE REACHED BY FOLLOWING A PATH "PATH" STARTING AT ANY NODE IN THE SET OF NODES "NODE". "PATH" MUST CONFORM TO THE SYNTAX OF PATHS AS DESCRIBED IN THE SNEPS USER MANUAL AND "NODE" MUST CONSIST OF A LIST OF NODE-IDENTIFIERS. ONLY TOP-LEVEL NODES ARE RETURNED.

```
     PATHGET
VALUE
(LAMBDA (NODE PATH)
   (COND
    ((NULL NODE) NIL)
    ((ATOM NODE) (PATHGET (LIST NODE) PATH))
    ((NULL PATH) NODE)
    ((ATOM PATH)
     (U
      (COND
       ((VAR (CAR NODE)) (GET (CAR NODE) PATH))
       ((UPREL PATH) (MAPCONC (GET (CAR NODE) PATH)
                              (LAMBDA (X) (IF (TOP? X) (LIST X)))))
       (T (GET (CAR NODE) PATH)))
      (PATHGET (CDR NODE) PATH)))
    ((EQ (CAR PATH) 'COMPOSE)
     (PROG NIL
           (IF (CDDR PATH)
               (RETURN (PATHGET (PATHGET NODE (CADR PATH))
                                (CONS 'COMPOSE (CDDR PATH)))))
           (RETURN (PATHGET NODE (CADR PATH)))))
    ((EQ (CAR PATH) 'CONVERSE) (PATHGET NODE (CONV-PATH (CADR PATH))))
    ((EQ (CAR PATH) 'OR)
     (REPEAT (PATH-LIST U-LIST)
             (SETQ PATH-LIST (CDR PATH))
             (SETQ U-LIST NIL)
       BEGIN (SETQ U-LIST (U (PATHGET NODE (CAR PATH-LIST)) U-LIST))
             (SETQ PATH-LIST (CDR PATH-LIST))
             (IF (NULL PATH-LIST) (RETURN U-LIST))))
    ((EQ (CAR PATH) 'AND)
     (REPEAT (PATH-LIST INT-LIST)
             (SETQ PATH-LIST (CDDR PATH))
             (SETQ INT-LIST (PATHGET NODE (CADR PATH)))
       BEGIN (SETQ INT-LIST (INTERSCTN (PATHGET NODE (CAR PATH-LIST))
                                       INT-LIST))
             (SETQ PATH-LIST (CDR PATH-LIST))
             (IF (NULL PATH-LIST) (RETURN INT-LIST))))
    ((EQ (CAR PATH) 'NOT) (COMPLEMENT (GET 'NODES ':VAL)
                                      (PATHGET NODE (CADR PATH))))
```

APPENDIX

```
((EQ (CAR PATH) 'RELATIVE-COMPLEMENT)
 (COMPLEMENT (PATHGET NODE (CADR PATH)) (PATHGET NODE (RAC PATH))))
((EQ (CAR PATH) 'IRREFLEXIVE-RESTRICT)
 (MAPCONC (PATHGET NODE (CADR PATH))
          (LAMBDA (X) (IF (NOT (EQ X NODE)) (LIST X)))))
((EQ (CAR PATH) 'KSTAR)
 (REPEAT (FOUND)
         (SETQ FOUND NODE PATH (CADR PATH))
   BEGIN (SETQ NODE  (SETSUB (PATHGET NODE PATH) FOUND)
               FOUND (APPEND NODE FOUND))
         (IF (NULL NODE) (RETURN FOUND))))
((EQ (CAR PATH) 'KPLUS)
 (PATHGET (PATHGET NODE (CADR PATH)) (CONS 'KSTAR (CDR PATH))))
((EQ (CAR PATH) 'EXCEPTION)
 (PROG (TEMP1 TEMP2)
       (SETQ TEMP1 (LENGTH-PATH (MAPCAR NODE
                                        (LAMBDA (X) (LIST X 0)))
                                (CADR PATH)))
       (SETQ TEMP2 (LENGTH-PATH (MAPCAR NODE
                                        (LAMBDA (X) (LIST X 0)))
                                (RAC PATH)))
       (RETURN
        (COMPLEMENT
         (MAPCAR TEMP1 (LAMBDA (X) (CAR X)))
         (MAPCAR (SHORTER-LENGTH TEMP1 TEMP2) (LAMBDA (Y) (CAR Y))))
       )))
 ((EQ (CAR PATH) 'DOMAIN-RESTRICT)
  (PROG (TEMP)
        (SETQ TEMP (PATHGET (CADR (CADR PATH)) (CAADR PATH)))
        (RETURN (MAPCONC (PATHGET NODE (RAC PATH))
                         (LAMBDA (X) (IF (MEMEQL X TEMP) (LIST X)))))
  ))
 ((EQ (CAR PATH) 'RANGE-RESTRICT)
  (MAPCONC
   (PATHGET NODE (CADR PATH))
   (LAMBDA (X)
      (IF (MEMEQL (CADR (RAC PATH)) (PATHGET X (CAR (RAC PATH))))
          (LIST X)))))))

PLIST
NIL
```

APPENDIX

LENGTH-PATH

THIS FUNCTION IS SIMILAR TO PATHGET IN THAT IT ALSO TAKES A SET OF NODES AND A PATH AS ARGUMENTS AND RETURNS THE SET OF NODES WHICH CAN BE FOUND BY FOLLOWING "PATH" STARTING FROM ANY NODE IN "NODE". HOWEVER, IT NOT ONLY RETURNS A SET OF NODES, BUT ALSO RETURNS THE PATH-LENGTH ASSOCIATED WITH REACHING EACH RESULTANT NODE. THE PATH-LENGTH RETURNED IS THE SHORTEST PATH-LENGTH AND IT IS ASSUMED THAT EVERY TIME AN ARC IS TRAVERSED, THE PATH-LENGTH IS INCREMENTED BY 1. ALL ELEMENTS OF THE ARGUMENT LIST "NODE" MUST HAVE SOME PATH-LENGTH ASSOCIATED WITH THEM WHEN THE FUNCTION IS CALLED (NORMALLY THIS WOULD BE 0). THIS FUNCTION IS USED TO IMPLEMENT THE EXCEPTION FEATURE OF THE SYNTAX WHERE PATH-LENGTHS ARE ESSENTIAL. IT IS CALLED BY THE FUNTION PATHGET (AND ITSELF).

```
     LENGTH-PATH
VALUE
(LAMBDA (NODE PATH)
   (COND
    ((NULL NODE) NIL)
    ((ATOM NODE) (LENGTH-PATH (LIST NODE) PATH))
    ((NULL PATH) NODE)
    ((ATOM PATH)
     (PROG (SET1 SET2)
           (SETQ
            SET1 (COND
                  ((VAR (CAAR NODE))
                   (MAPCAR (GET (CAAR NODE) PATH)
                           (LAMBDA (X) (LIST X (ADD1 (CADAR NODE))))))
                  ((UPREL PATH)
                   (MAPCONC
                    (GET (CAAR NODE) PATH)
                    (LAMBDA (X)
                       (IF (TOP? X)
                           (LIST (LIST X (ADD1 (CADAR NODE)))))))
                  (T (MAPCAR (GET (CAAR NODE) PATH)
                             (LAMBDA (X) (LIST X (ADD1 (CADAR NODE))))))
                 ))
           (SETQ SET2 (LENGTH-PATH (CDR NODE) PATH))
           (RETURN (COMPLEMENT (U SET1 SET2)
                               (LONGER-PATH-NODES SET1 SET2)))))
    ((EQ (CAR PATH) 'COMPOSE)
     (PROG NIL
           (IF (CDDR PATH)
               (RETURN (LENGTH-PATH
                        (LENGTH-PATH NODE (CADR PATH))
                        (CONS 'COMPOSE (CDDR PATH)))))
           (RETURN (LENGTH-PATH NODE (CADR PATH)))))
    ((EQ (CAR PATH) 'CONVERSE) (LENGTH-PATH NODE
                                            (CONV-PATH (CADR PATH))))
```

```
((EQ (CAR PATH) 'OR)
 (REPEAT (PATH-LIST U-LIST)
         (SETQ PATH-LIST (CDR PATH))
         (SETQ U-LIST NIL)
   BEGIN (SETQ
          U-LIST (PROG (SET1)
                       (SETQ SET1 (LENGTH-PATH NODE
                                               (CAR PATH-LIST)))
                       (RETURN
                        (COMPLEMENT (U SET1 U-LIST)
                                    (LONGER-PATH-NODES SET1 U-LIST))
                       )))
         (SETQ PATH-LIST (CDR PATH-LIST))
         (IF (NULL PATH-LIST) (RETURN U-LIST))))
((EQ (CAR PATH) 'AND)
 (REPEAT (PATH-LIST INT-LIST)
         (SETQ PATH-LIST (CDDR PATH))
         (SETQ INT-LIST (LENGTH-PATH NODE (CADR PATH)))
   BEGIN (SETQ INT-LIST (INT-NODES-WITH-PLENGTH
                         (LENGTH-PATH NODE (CAR PATH-LIST))
                         INT-LIST))
         (SETQ PATH-LIST (CDR PATH-LIST))
         (IF (NULL PATH-LIST) (RETURN INT-LIST))))
((EQ (CAR PATH) 'NOT)
 (COMPL-WITH-PLENGTH
  (MAPCAR (GET 'NODES ':VAL) (LAMBDA (X) (LIST X 0)))
  (LENGTH-PATH NODE (CADR PATH))))
((EQ (CAR PATH) 'RELATIVE-COMPLEMENT)
 (COMPL-WITH-PLENGTH (LENGTH-PATH NODE (CADR PATH))
                     (LENGTH-PATH NODE (RAC PATH))))
((EQ (CAR PATH) 'IRREFLEXIVE-RESTRICT)
 (MAPCONC
  (LENGTH-PATH NODE (CADR PATH))
  (LAMBDA (X)
     (IF (NOT (MEMEQL (CAR X) (MAPCAR NODE (LAMBDA (Y) (CAR Y)))))
         (LIST X)))))
((EQ (CAR PATH) 'KSTAR)
 (REPEAT (FOUND TEMP1)
         (SETQ FOUND NODE PATH (CADR PATH))
   BEGIN (SETQ TEMP1 (LENGTH-PATH NODE PATH))
         (SETQ NODE (COMPL-WITH-PLENGTH TEMP1 FOUND))
         (SETQ FOUND (COMPLEMENT (U TEMP1 FOUND)
                                 (LONGER-PATH-NODES TEMP1 FOUND)))
         (IF (NULL NODE) (RETURN FOUND))))
((EQ (CAR PATH) 'KPLUS)
 (LENGTH-PATH (LENGTH-PATH NODE (CADR PATH))
              (CONS 'KSTAR (CDR PATH))))
((EQ (CAR PATH) 'EXCEPTION)
 (PROG (TEMP1 TEMP2)
       (SETQ TEMP1 (LENGTH-PATH (MAPCAR NODE
                                        (LAMBDA (X)
                                           (LIST (CAR X) 0)))
                                (CADR PATH)))
       (SETQ TEMP2 (LENGTH-PATH (MAPCAR NODE
                                        (LAMBDA (X)
                                           (LIST (CAR X) 0)))
                                (RAC PATH)))
       (RETURN (COMPL-WITH-PLENGTH TEMP1
                                   (SHORTER-LENGTH TEMP1 TEMP2)))))
```

APPENDIX

```
     ((EQ (CAR PATH) 'DOMAIN-RESTRICT)
      (PROG (TEMP)
            (SETQ TEMP (LENGTH-PATH (LIST (CADR (CADR PATH)) 0)
                                    (CAADR PATH)))
            (RETURN
             (MAPCONC
              (LENGTH-PATH (MAPCAR NODE (LAMBDA (Y) (LIST (CAR Y) 0)))
                           (RAC PATH))
              (LAMBDA (X)
                 (IF (MEMEQL (CAR X) (MAPCAR TEMP (LAMBDA (P) (CAR P))))
                     (LIST X)))))))
     ((EQ (CAR PATH) 'RANGE-RESTRICT)
      (MAPCONC
       (LENGTH-PATH (MAPCAR NODE (LAMBDA (Y) (LIST Y 0))) (CADR PATH))
       (LAMBDA (X)
          (IF (MEMEQL (CADR (RAC PATH))
                      (MAPCAR (LENGTH-PATH X (CAR (RAC PATH)))
                              (LAMBDA (P) (CAR P))))
              (LIST X)))))))

PLIST
NIL
```

9. BIBLIOGRAPHY

1) Shapiro, S.C. Path-based and node-based inference in semantic networks. Theoretical issues in Natural Language Processing-2. D.Waltz ed., ACM, 1978.

2) Shapiro, S.C. The SNePS semantic network processing system. In N.Findler, ed. Associative Networks- The Representation and use of knowledge in computers, Academic Press, New York.

3) Shapiro,S.C. Representing and locating deduction rules in a semantic network. Sigart Newsletter No. 63, June 1977.

4) Shapiro S.C. Techniques of Artificial Intelligence. D. Van Nostrand, New York, i979.