STATE UNIVERSITY OF NEW YORK AT BUFFALO

Department of Computer Science

Bi-Directional Inference

João P. Martins, Donald P. McKay & Stuart C. Shapiro

March 1981

Technical Report Number 174

This work was supported in part by the National Science Foundation under Grants MCS78-02274 and MCS80-06314 and by the Instituto Nacional de Investigação Cientifica (Portugal) under Grant No. 20536.

Abstract

In this paper we present a brief overview of the SNePS deduction system and show through an example the interaction between forward and backward inference. This interaction -- resulting in a class of inference termed bi-directional inference -- enables an easy and elegant way of performing bi-directional searches and the possibility of performing resource-limited deduction in a natural and simple way. Furthermore, bi-directional inference focus a system's attention towards the interests of the user and can cut down the fan out of pure forward or pure backward chaining.

1. Introduction

During the last decade Artificial Intelligence (AI) researchers tried to create tools to enable inference to be done by a computer program. Some of the languages which were designed with this goal are MICRO-PLANNER [Sussman et al. 71], CONNIVER [McDermott and Sussman 72], FUZZY [LeFaivre 77] and PROLOG [Coelho et al. 80; Colmerauer 79]. These languages allow both forward and backward inference to be performed but require that one set of rules be written for forward and another set for backward inference (eg, the assertion theorems and consequent theorems of MICRO-PLANNER). This means that if one rule is written to be used in forward inference it can not be used in backward inference and vice-versa. Also, in these systems, once an inference is completed all intermediate results are lost (unless explicitly stored by the user).

Our inference system relies on a declarative representation of inference rules (SNePS semantic network) and every rule may be used both in forward and backward inference. Also, all intermediate results gathered by an inference are remembered (unless explicitly erased by the user) so that if in a future deduction the system needs some of the results generated during a previous deduction it can make use of them directly instead of rederiving them. These two features enable a smooth interaction between forward and backward inference. Such interaction, resulting in a form of inference termed bi-directional inference, is described in this paper. Bi-directional inference corresponds to a search procedure which is more general than "classical" bi-directional search [Pohl 71; Shubin and Shapiro 81]. In bi-directional search, the system has fixed start and goal states and tries to find a path connecting them by working forward from the start state(s) and backward from the goal state in "parallel". The search terminates with success whenever the two search frontiers meet. In bi-directional inference, there are no fixed a priori start or goal states. A start state can be any node which has been added by forward inference and a goal state can be any node that has been querried by backward inference. The search is successful whenever the frontier growing from any start state meets the frontier growing from a goal state. In this way we have a search procedure which adapts itself dynamically to the past history of deductions.

2. Basic SNePS network notions

A SNePS semantic network [Shapiro 79a] is a labeled directed graph in which nodes represent concepts and arcs represent non-conceptual binary relations between concepts. The labels on the arcs represent binary semantic relations which are used to structure the network and about which no information can be stored in the network. Each concept is represented by a unique node [Maida and Shapiro].

There are three kinds of arcs in the network: descending, ascending and auxiliary. For each relation represented by a descending arc there is a converse relation represented by an ascending arc and vice-versa. If a descending arc is labeled R then its converse ascending arc is labeled by R^C. Together descending and ascending arcs are the regular semantic network arcs referred to above. Auxiliary arcs are used for storing non-nodal information on nodes. If a path of descending arcs goes from node n to node m we say that node n dominates node m.

Since in SNePS semantic networks, nodes are used to represent all information that can be discussed, nodes are used to represent specific assertions and general deduction rules. We will refer to nodes which represent deduction rules as rule nodes. A rule node represents a propositional formula of molecular nodes using one of the non-standard connectives available in SNePS [Shapiro 77; 79a]. In this paper our discussion will be limited to four of them:

1. v-entailment (v->): The formula $(A_1 \ldots A_m)$ v-> $(C_1 \ldots C_n)$ means that the disjunction of the antecedents implies the

conjunction of the consequents;

2. &-entailment (&->): The formula $(A_1 \ldots A_m)$ &-> $(C_1 \ldots C_n)$ means that the conjunction of the antecedents implies the conjunction of the consequents;
3. AND-OR $({}_n\mathcal{X}_i^j)$: The formula ${}_n\mathcal{X}_i^j(A_1 \ldots A_n)$ is true just in case at least i or at most j of the arguments are true;
4. THRESH $({}_n\Theta_i)$: The formula ${}_n\Theta_i(A_1 \ldots A_n)$ is true if either fewer than i arguments are true or all n are true.

The network representation of these connectives is shown in Figure 1. In this and in following Figures, descending arcs will

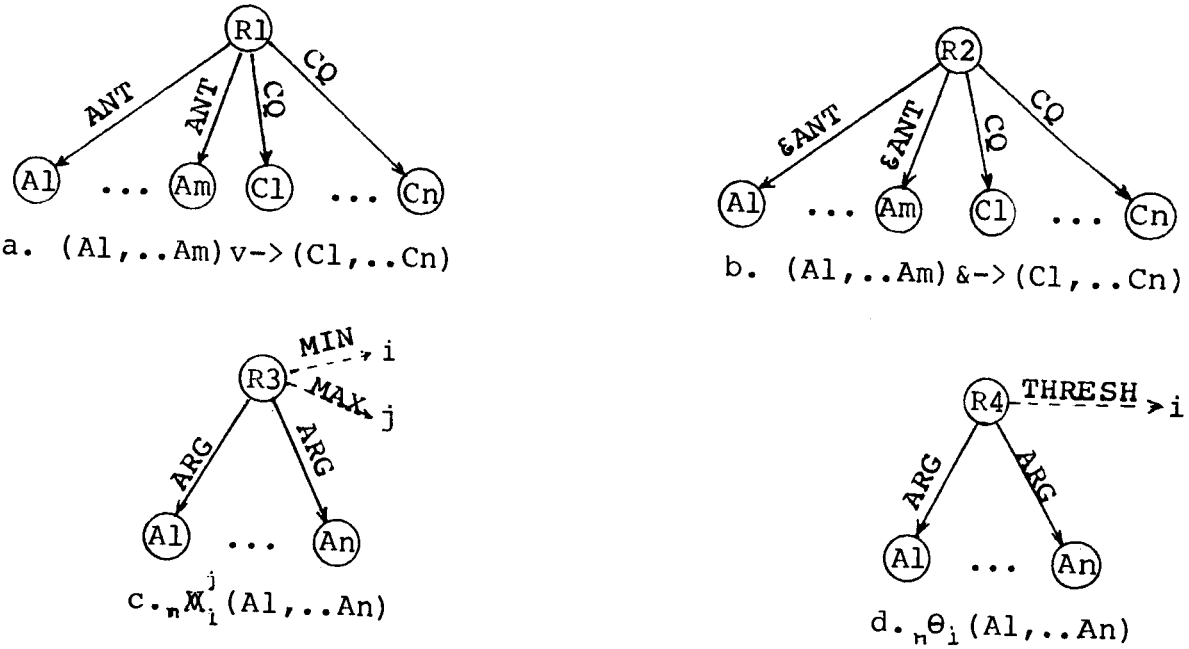

Figure 1
Representation of connectives

be represented by solid arrows, ascending arcs will not be shown, and auxiliary arcs will be represented by dashed arrows.

The SNePS semantic network system allows the representation of several kinds of quantifiers in deduction rules [Shapiro 77;

79a; 79b]. These quantifiers are represented by arcs connecting the rule and the variable nodes quantified by the rule.

3. Overview of the SNePS deduction system

If deduction rules exist in the network, inference is triggered using the functions ADD and DEDUCE: (ADD R_1 NS_1...R_n NS_n) instructs the deduction system to add to the network the node with arcs labled R_i to the nodes in the set NS_i ($1 \leq i \leq n$) and to do forward inference with that node; (DEDUCE numb R_1 NS_1...R_n NS_n) triggers the backward inference mechanism to try to deduce instances of the node with arcs labeled R_i to nodes in NS_i ($1 \leq i \leq n$), numb controls how many answers are desired [Shapiro 79a].

The SNePS inference system is implemented in MULTI [McKay and Shapiro 80; Shapiro and McKay 80]. MULTI is a LISP-based multiprocessing system which consists of a simple evaluator, system primitives, a scheduler and debugging facilities. The evaluator continuously executes processes from a process queue until the queue becomes empty. System primitives include functions for creating processes, for scheduling processes to be executed and for manipulating local variables or registers. The scheduler inserts a process in the process queue.

Depending on the actions they perform, the MULTI processes used in the SNePS deduction system which will be discussed in this paper can be divided into four groups:

1. Pattern matchers (INFER and F-INFER): these processes match

a given node (or network structure) against the current network and do further processing if the match is successful, i.e. if the process can find in the network assertion or pattern nodes which match the desired structure;

2. Data-collectors (ANS-CATCH, TOPINF and TOPMOST-TOPINF): these processes receive answers and remember each answer received. Everytime they receive an answer not previously received the answer is sent to all the processes to which the data-collector reports;
3. Active connectives (CH-processes and IMPLY): each of these processes is responsible for a given rule node. It corresponds to the active or procedural representation of a rule instance. Each of them works by creating and scheduling processes to try to establish a sufficient number of the rule's antecedents and then waits for answers, instances of antecedents. When answers are received the process decides whether it has received sufficient answers to deduce instances of its consequents.
4. Mixed purpose (USE, GO-UP and SWITCH): These processes are used for several different purposes: setting up the structure of processes necessary to use some rule (USE); checking if a rule is asserted (i.e., not dominated by any other node) and, in some cases, traversing arcs (GO-UP); acting as the variable translators between processes working in two different rules which share some common node structure (SWITCH).

The same basic deduction procedure is used by the SNePS

deduction system in both forward and backward inference: to use some rule the CH-process tries to find instances of its antecedents, by doing a network pattern match (see [Shapiro 77]) of the antecedents of the rule. Note that the network match finds all network nodes which are unifiable with a particular antecedent. If such instances are found (or derived), messages reporting this are sent to the CH-process via data-collectors. The CH-process considers the answers received so far and decides whether they are sufficient to deduce the consequents of the rule. If they are, it reports the instances of the consequents to all other processes interested in such results.

The network match is avoided if appropriate "producers" already exist. This avoids redundant work and allows recursive rules to be used [Shapiro and McKay 80; McKay and Shapiro 81]. After a deduction is completed, the set of processes used during the deduction is retained by the system unless explicitly overriden. If later on, during another deduction, the system needs some process which was already created by a previous deduction, it is able to use it instead of creating a new one. The main advantage of this is not merely avoiding the work of setting up a single new process but rather the system can thereby use all the processes which report to the process being reused and all the information which was gathered by the previous deduction. The set of processes left behind after forward inference is similar to the set of processes that would have been left behind by backward inference through the same rules. The processes built in either type of inference are able to intersect smoothly with each other. The benefits of this are discussed in

section 6.

4.Forward inference

This section addresses the question of when to do (and when not to do) forward inference. Forward inference should only be triggered when the newly ADDed node matches a node which is in antecedent position of some rule, but should it be triggered every time this situation happens? Perhaps the best way to analyse this problem is by looking at some examples.

Suppose that the network contains the rule represented in Figure 2. This rule represents an v-entailment and can be read

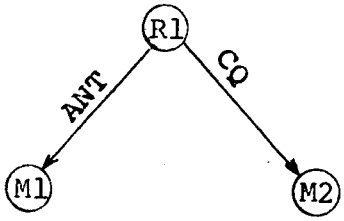

Figure 2
Simple rule

as "if M1 then M2". In the typical case both M1 and M2 contain variables and the rule should be interpreted as: "If there is an assertion in the network, say A, which is a fully ground instance of the node M1, with b being the substitution that when applied to M1 produces A, then we can infer the node resulting from applying the substitution b to M2". We will henceforth refer to this by saying that if there is a node, say A, that matches M1b, then we can assert M2b.

With rule R1 and the assertion in the network of some node

which matches M1b, by "modus ponens", forward inference should deduce M2b.

But now, suppose that rule R1 is not asserted in the network but rather embeded within some other rule. There are two possible cases for this rule embeding:

1. R1 is in antecedent position of some other rule (Fig.3a).
2. R1 is in consequent position of some other rule (Fig.3b).

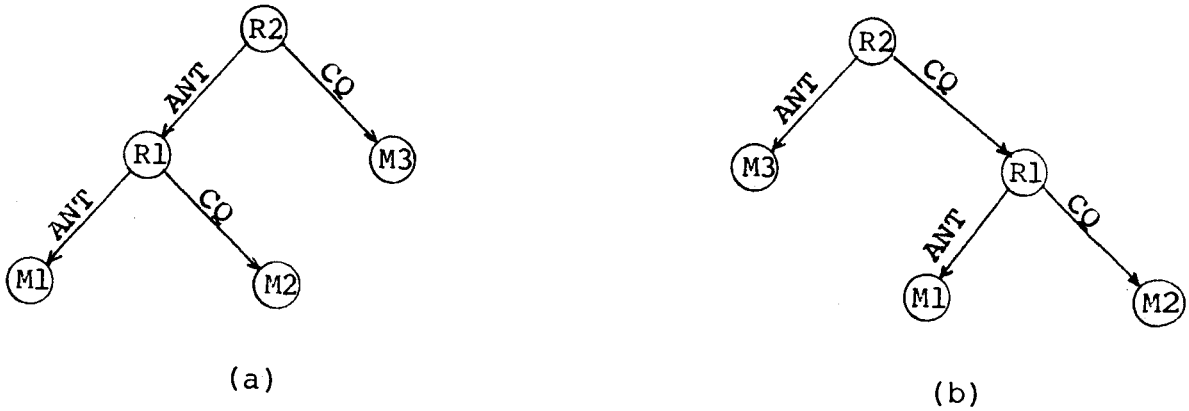

Figure 3
Simple cases of rule embeding

Suppose that, in the first case, we assert a node that matches M1b. Since R1 is not asserted in the network we can not infer M2b since we don't know if the rule R1 "holds". The first step that we would have to do is to prove that rule R1 holds. This would have to be done in the following way: assume an arbitrary instance of M1, and then using only sound rules of inference deduce a corresponding instance of M2. The SNePS inference system is not yet able to do this (but see [Martins and Shapiro 81]), and therefore when some newly asserted node matches a node which is in antecedent position of some rule which is in turn in antecedent position of another rule no forward inference

is done.

Suppose now that in the case represented in Figure 3b we assert a node matching M1b. Again, we can not use rule R1 unless we prove that it holds. But in this case, "modus ponens" enables us to infer R1b provided that M3b holds. What the forward inference system then does is start working on rule R2b, trying to prove its antecedent, i.e. trying to show that an instance of M3b can be deduced with the existing facts and rules. In the case of such a proof's being successful, M2b can be asserted.

This line of reasoning can be applied directly to the SNePS connectives v-> and &->, the only difference between them being that for v-> all the consequents of the rule are deduced as soon as one antecedent is shown, whereas for &-> all the antecedents of the rule need to be deduced (in a consistent binding) before the consequents can be deduced. The representation of AND-OR and THRESH made no distinction among their arguments (see Section 2). This means that one particular argument may be considered an antecedent or a consequent depending on the situation. If we ADD to the network a node which is an instance of an ARGument of one of these connectives, the node matched will be looked at as being one antecedent and, depending on the parameters of the connective, we might or might not need to find more antecedents. In any case all the ARGuments that are not derived are considered to be consequents. Thus, the path of arcs to be followed during forward inference is represented in Figure 4 where "top?" means that a node is not dominated by any other node, i.e., is a top level assertion in the network.

$$\left\{ \begin{array}{c} ANT^{c} \\ \&ANT^{c} \\ ARG^{c} \end{array} \right\} \cdot \left[\left[CQ^{c} \right]^{*} \cdot \left[ARG^{c} \right]^{*} \right]^{*} \cdot \text{top?}$$

Figure 4
Path of arcs to be followed during forward inference

5. Backward Inference

In this section the backward inference mechanism is discussed. When the user asks a question using the function DEDUCE, the inference system does a network pattern match. Among the set of matched nodes are constant nodes which are answers to the original question. In addition, the system looks for rules which may enable an answer to be deduced. Consider again the rule represented in Figure 2. If the user asks for an instance of M2b, i.e. some question node matches M2 with a binding b, the system will try to find an instance of M1b and if such an instance is found M2b can be deduced. The rule in Figure 3a will not be used to deduce M2b because of the reasons pointed out in the last section. Now consider the case represented in Figure 3b. If the user asks for an instance of the node M2b the system looks at rule R1 and realizes that this rule may enable the deduction of such a node provided that it can prove that rule R1b holds. To do that it follows the CQ^{c} arc leading to rule R2 and tries to prove M3b. If such an instance is found modus ponens enables the deduction of R1b and now the inference system may start looking for an instance of M1b which in turn enables the deduction of M2b.

As before the analysis presented above only directly applies to the connectives v-> and &->. The discussion for the other connectives is very similar to the one presented in the last section and will not be presented here. The path of arcs to be followed during backward inference is shown in Figure 5. Note

$$\left[\left[\overset{c}{[CQ]}\right]^{*} . \left[\overset{c}{[ARG]}\right]^{*}\right]^{*} . \text{ top?}$$

Figure 5
Path of nodes to be followed during backward inference

that this path of arcs being only "top?" corresponds to the case of finding an instance of a node explicitly asserted in the network.

6. Interaction between forward and backward inference

In this section an example of how forward and backward inference interact is fully developed.

Suppose that the assertions and rules represented in Figure 6 exist in the network. Suppose also that we use the function (ADD MEMB JO CLASS HARD WORKER) to build and do forward inference on node N1. The function ADD creates an F-INFER process which matches N1 against the network tofind rules to be used in forward inference. This process finds M1, which is in antecedent position of rule R1 (see section 4 for definition of antecedent position). F-INFER builds two processes: an IMPLY process which will be responsible for rule R1 and a GO-UP process that checks

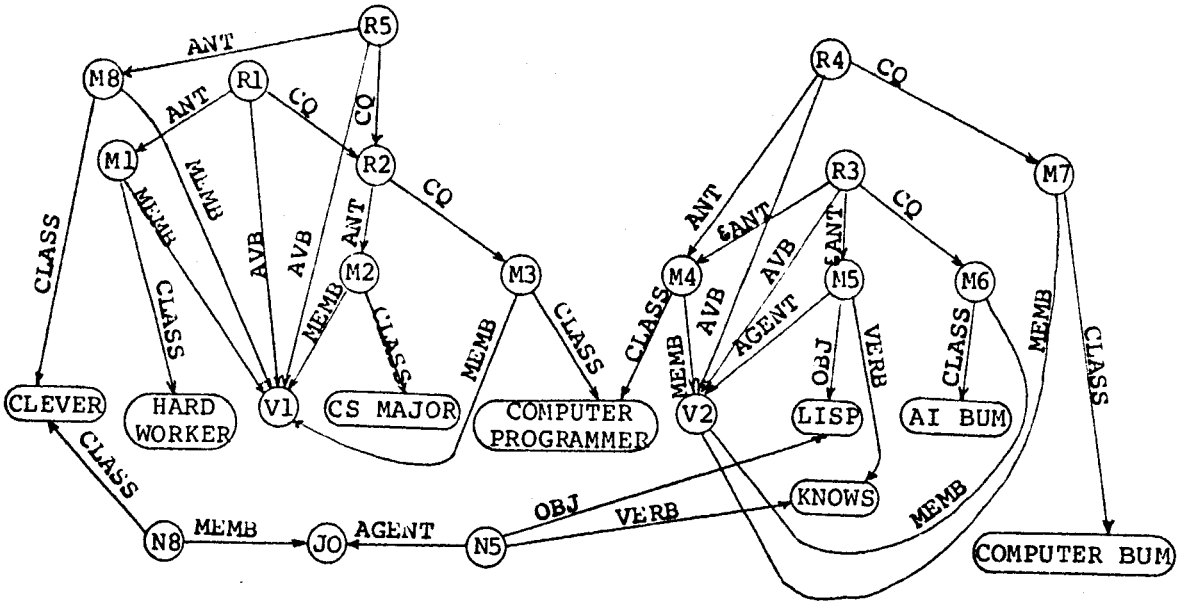

Figure 6
Current network

whether the rule R1 is asserted in the network and if not follows CQ^C or ARG^C arcs (if such arcs exist) to examine the rules which contain R1. The process GO-UP is scheduled. Figure 7 shows the processes built so far -- later in the text we will explain why P1 also created processes P7 and P8. In this Figure and in the following Figures, each process is represented by a rectangle, on top of which is the name of the process and its unique identifier (of the form Pn); inside the rectangle there is the set of registers used by the process and their values; a solid arrow going from process A to process B means that A can report (send messages) to process B (sometimes said that B is the "boss" of A); a dashed arrow going from process A to process B means that process A creates process B. If process A also schedules process B, the arrow head will be solid. The job of the F-INFER process is now completed, it stops execution and the scheduler initiates the process P3.

P3 looks at rule R1 and since it is asserted in the network

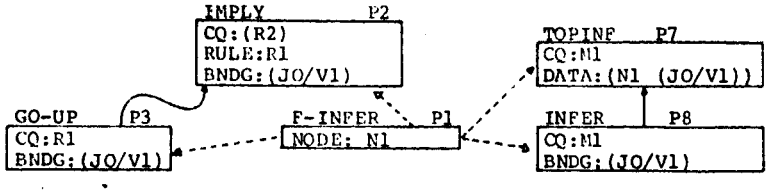

Figure 7
Complete set of processes - phase 1

it creates processes to use the rule, i.e., an ANS-CATCH (P4) and a USE (P5), and schedules process P5 (see Figure 8). The scheduler initiates process P5 which determines the connective for rule R1 and accordingly creates a CH-process to work on the rule, in this case, a CHENT (P6), which interprets v-entailments.

P6 creates and schedules processes to establish instances of

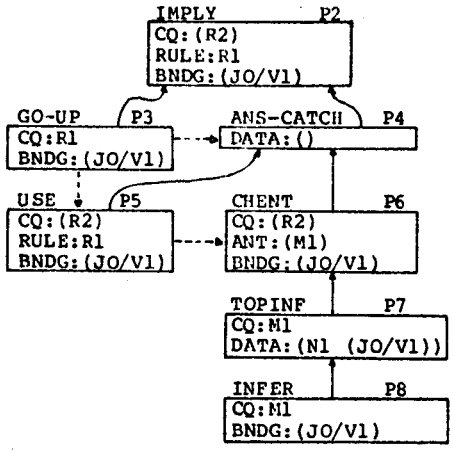

Figure 8
Complete set of processes - phase 2

the antecedent of rule R1. P6 attempts to create an INFER process to match the antecedent of R1 against the network and a TOPINF, a data-collector to which the INFER reports. The TOPINF, in turn, would report to the CH-process. In this case, however, since forward inference was triggered by the ADDition of N1 which matched M1, these two processes have already been created by the

F-INFER (P1) which anticipated that these processes would be needed. The set of processes (P7 and P8) is linked to the process P6 (see Fig.8). The advantage of having F-INFER create these processes is that the system does not need to perform a network match to find if there is a node which matches M1{JO/V1}, because N1 is already known to be such a node. After linking P7 and P8 to P6, P6 receives a message from P7 stating that an instance M1{JO/V1} was found. P6 deduces R2{JO/V1} and sends a message to the ANS-CATCH (P4).

P4 compares the answer received with the data stored in its DATA: register (containing a record of all the messages received so far) and if a similar message has not been received before, as is the present case, it reports the message to its bosses, in this case P2.

When P2 gets this message, it looks at the consequent of R1 and, since it is a rule (R2), P2 sets up the structure of processes necessary to use R2{JO/V1}, which, as we have already seen, is a USE process (P11) which reports to an ANS-CATCH (P1Ø). Furthermore, P2 also creates another IMPLY process (P12) which is responsible for rule R2 and creates a path going from P2 to P12 via an ANS-CATCH (P9). The job of the IMPLY process is now done, it schedules the newly created P11 and stops execution (see Figure 9).

The scheduler now initiates process P11 which, as before, determines what kind of rule R2 is, creates a CHENT process (P13) to work on this rule, which, in turn, creates and schedules an INFER process (P15) to work on the antecedent of R2 (M2) and a

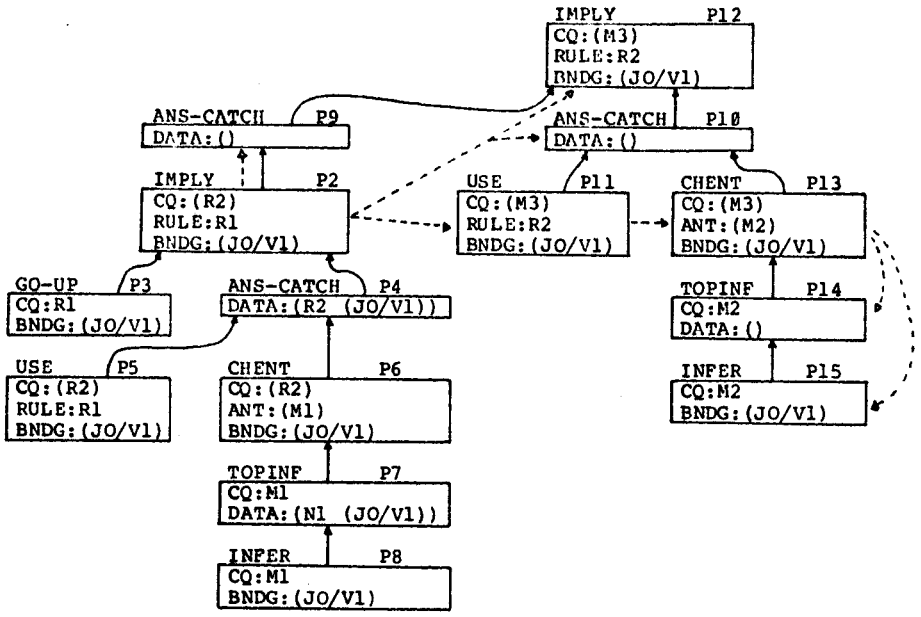

Figure 9
Complete set of processes - phase 3

data-collector TOPINF (P14). When process P15 is initiated it tries to find nodes in the network which match M2{JO/V1}, but since there are none, it stops execution without initiating any other process. The process queue now becomes empty and the inference process stops (see Figure 9).

Suppose now that the function (DEDUCE MEMB JO CLASS AI BUM) is used. The processes created for this question are shown in Figure 10. DEDUCE creates a TOPMOST-TOPINF data-collector (P16) to receive answers concerning the instances of M6{JO/V2} and an INFER process (P17), reporting to P16, which tries to find such instances in the network, and schedules P17. P17, which is looking for instances of a temporary node (T1) matching M6{JO/V2}, fails to find such instances explicitly stored in the network but notices that rule R3 may enable the derivation of

such instances. It therefore creates and schedules a GO-UP process (P18) which focuses the attention of the inference system on rule R3. When P18 is initiated it follows the CQ^C arc leading from M6 to R3, creates an IMPLY process (P2Ø) which is responsible for rule R3 and creates and schedules a GO-UP process (P21) which checks whether R3 is asserted. When P21 is scheduled it creates a USE (P23) which, in turn, creates a CH&ENT process (P24) to work on the &-entailment represented by node R3 which in turn creates INFER and TOPINF processes for each of the antecedents of R3 (P25, P26, P27 and P28) and schedules processes P26 and P28 for execution.

Let us assume that P28 is executed first. It looks for instances of M5 and finds N5. It sends a message to P27 which in turn reports to P24. When P24 receives this message it records it and does nothing because it is not enough to deduce the consequent of R3. Process P26 is initiated. It can not find an instance of M4 explicitly stored in the network but it finds a rule (R2) that may enable the deduction of such an instance. In order to get an instance of M4{JO/V2}, P26 will try to use rule R2. However, since M3 and M4 are in different rules, with different variables, rule R2 is not directly applicable to rule R3 and, for this reason, process P26 creates a SWITCH process (P29) which acts as a variable name translator between the nodes M3 and M4. P26 also creates and schedules a GO-UP process (P3Ø) to work on node M3. When P3Ø is scheduled it finds R2 at the end of the CQ^C arc leaving M3 and notices that there is already a process working on such a rule, namely the IMPLY process P12. In this case P3Ø creates a path going from P12 to P29 via an

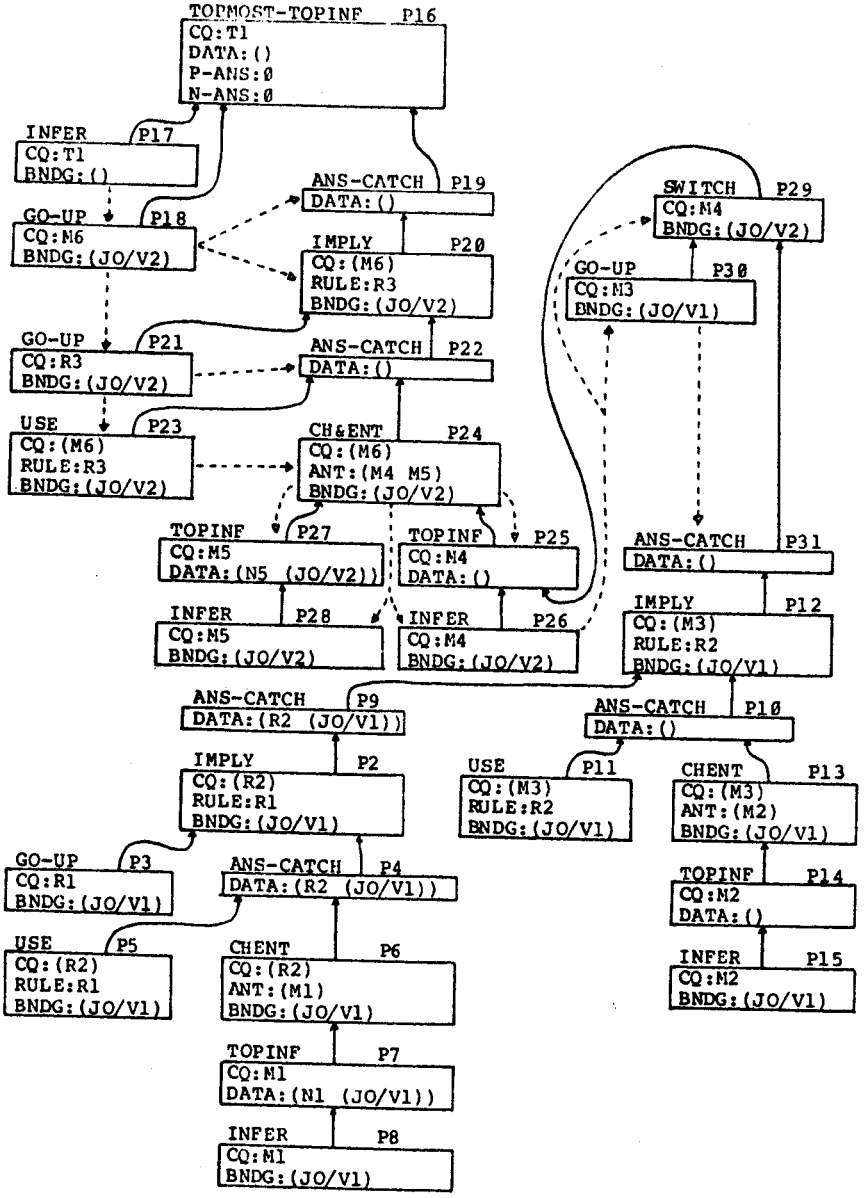

Figure 10
Complete set of processes - phase 4

ANS-CATCH (P31) and since the rule R2 has not been used yet P30 does not schedule any further process and the inference system stops, leaving behind the set of processes represented in Figure 10.

Suppose now that the function (ADD MEMB JO CLASS CS MAJOR) is executed, creating node N2. The function ADD creates an

F-INFER process to do forward inference with node N2. When this process is initiated, it notices that there is a process (P15) which is waiting for the assertion of such a node. The F-INFER then sends P15 the node N2. This triggers a propagation of messages through the network: P13 deduces the consequent of R2. When P12 hears about this, it asserts an instance of M3{JO/V1}, and informs P31 which, in turn, sends M3{JO/V1} to P25 via the SWITCH process. The significance of the SWITCH process is described in [McKay and Shapiro 81]. P25 informs P24 which now can deduce the consequent of R3 with binding {JO/V2}. It sends a message to P20 which asserts the node M6{JO/V2} and informs P16, which tells the user that a positive answer to the question was found. And the inference mechanism stops.

We present next the output generated by the SNePS deduction system when running the example above. The program sends its output through a generating ATN [Shapiro 79c] for generating English sentences from the nodes of the network.

```
**(SURFACE
*  (BUILD AVB $X
*         ANT (BUILD MEMB *X CLASS HARD/ WORKER)
*         CQ  (BUILD ANT (BUILD MEMB *X CLASS CS/ MAJOR)
*                    CQ  (BUILD MEMB *X
*                                CLASS COMPUTER/ PROGRAMMER)
*             ) = R2 ))
FOR EVERY V1, IF V1 IS A HARD WORKER
   THEN IF V1 IS A CS MAJOR
          THEN V1 IS A COMPUTER PROGRAMMER
(DUMPED)

**(SURFACE
*  (BUILD AVB *X
*         ANT (BUILD MEMB *X CLASS CLEVER)
*         CQ  *R2))
FOR EVERY V1, IF V1 IS CLEVER
   THEN IF V1 IS A CS MAJOR
          THEN V1 IS A COMPUTER PROGRAMMER
(DUMPED)

**(SURFACE
```

```
*  (BUILD AVB $Y
*         &ANT (BUILD MEMB *Y CLASS COMPUTER/ PROGRAMMER) = M4
*         &ANT (BUILD AGENT *Y VERB KNOWS OBJ LISP)
*         CQ   (BUILD MEMB *Y CLASS AI/ BUM)))
FOR EVERY V2, IF V2 IS A COMPUTER PROGRAMMER AND
   V2 KNOWS LISP
      THEN V2 IS AI BUM
(DUMPED)

**(SURFACE (BUILD AVB *Y
*                 ANT *M4
*                 CQ  (BUILD MEMB *Y CLASS COMPUTER/ BUM)))
FOR EVERY V2, IF V2 IS A COMPUTER PROGRAMMER
   THEN V2 IS COMPUTER BUM
(DUMPED)

**(SURFACE (BUILD MEMB JO CLASS CLEVER))
JO IS CLEVER
(DUMPED)

**(SURFACE (BUILD AGENT JO VERB KNOWS OBJ LISP))
JO KNOWS LISP
(DUMPED)

**(SURFACE (ADD MEMB JO CLASS HARD/ WORKER))

SINCE
JO IS A HARD WORKER
WE INFER
IF JO IS A CS MAJOR
   THEN JO IS A COMPUTER PROGRAMMER

JO IS A HARD WORKER
(DUMPED)

**(SURFACE (DEDUCE MEMB JO CLASS AI/ BUM))

(DUMPED)

**(SURFACE (ADD MEMB JO CLASS CS/ MAJOR))

SINCE
JO IS A CS MAJOR
WE INFER
JO IS A COMPUTER PROGRAMMER

SINCE
JO IS A COMPUTER PROGRAMMER AND
JO KNOWS LISP
WE INFER
JO IS AI BUM

JO IS A CS MAJOR AND
JO IS A COMPUTER PROGRAMMER AND
JO IS AI BUM
(DUMPED)
```

7. Three different modes of inference

In the last section we completely developed an example of bi-directional inference, i.e., combined forward and backward inference. In this section we compare the results obtained using such inference with the results obtained using backward or forward inference only. Figure 11 shows an AND-OR graph representing the dependencies among the nodes in Figure 6. The

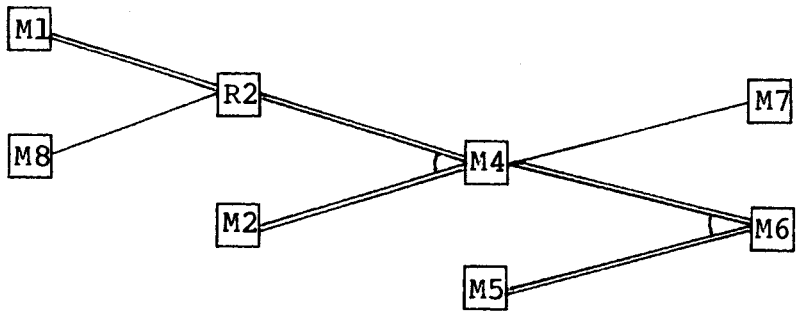

Figure 11
Nodes used in bi-directional inference

double lines represent the arcs followed in the example of Section 6 and therefore show the nodes actually used during bi-directional inference. Notice that although node M8 could have been used by the inference system to deduce R2, it was not used because we ADDed a node matching M1 which caused R2 to be proved. Also, an instance of M7 could have been derived, but since the user asked for an instance of M6 this fact caused the inference system to focus its attention on node M6, and when an instance of M2 was asserted the path leading to M7 was ignored. Let us further notice that if later on the user asks for an instance of M7 the system will find it much faster than if the question were asked prior to any inference since in this case

some of the processes that the inference system will use already exist with the appropriate data stored in their registers.

Now consider the case of pure backward inference. Suppose that there exist in the network instances of the nodes M1, M8, M2 and M5 (i.e., "JO is a hard worker", "JO is clever", "JO is a CS major" and "JO knows Lisp") and that no previous inference has been done. The AND-OR graph of Figure 12 shows the dependencies generated during backward inference from M6 and in Appendix 1 is presented the output generated by the program running in pure backward inference. If the user asks for an instance of M6 ("is JO AI bum?) the inference system sets up two goals: proving M4 and proving M5. The latter can be proved immediately and then to prove M4 the system tries to prove M3 which entails proving R2 and M2. R2 can be proved in two different ways either by finding an instance of M1 or by finding an instance of M8. In this case the system tries to prove M1 and M8 in "parallel" and as soon as one of them is proved (assume that M8 is proved first) the system concludes R2 and then, by proving M2, it can conclude M6. Notice

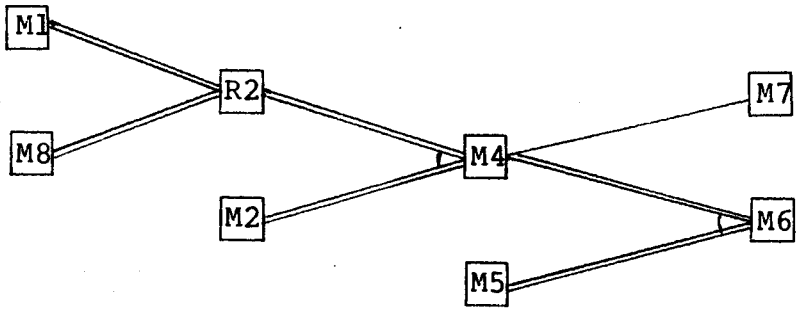

Figure 12
Nodes used in pure backward inference

that, again, in this case M7 was not proved since the user was not interested in such a node.

Let us now consider the case of pure forward inference. Assume that instances of M5 and M8 exist in the network and that the user ADDs a node matching M1. The AND-OR graph in Figure 13 shows the dependencies for full forward inference and in Appendix 2 is presented the output generated by the program running in pure forward inference. The inference system will deduce R2 but it can not proceed to deduce M3 (and M4) since it can not prove M2. If the user now ADDs an instance of M2 the inference system deduces M3 and does forward inference with this node resulting in the derivation of M7 and M6. Notice that in this case both M6

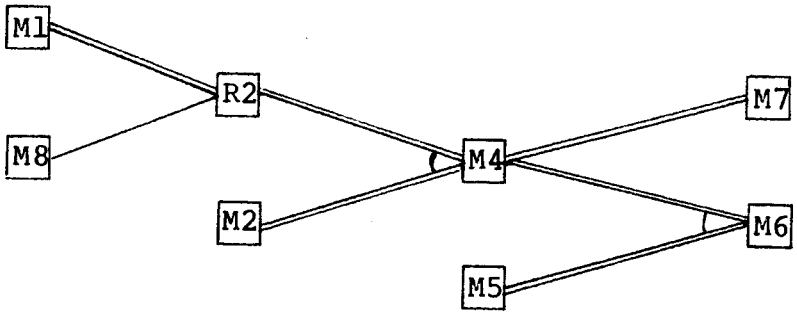

Figure 13
Nodes used in pure forward inference

and M7 are derived. Furthermore, M7 is derived first since its derivation only entails proving one node whereas to derive M6 the system has to prove two nodes and the system works in "parallel".

8. Bi-directional Inference and Resource-Limited Processing

In the example of section 6 we deliberately asserted M2{JO/V1} last to force a breakpoint in the growing of the processes: both the processes for forward and backward inference needed M2{JO/V1} to proceed. When these processes failed to find such node the whole inference was suspended. We stress the word

suspended since the inference was not aborted, like in most systems, but rather entered a latent state which was resumed as soon as relevant data was ADDed to the network.

We plan to augment the SNePS inference system with resource-limited processing. When a process creates another process it will allocate to this new process a certain amount of its resources. During computation a process expends its resources by creating and scheduling other processes and/or performing network pattern matches. When resources are exhausted, execution is suspended until allocated more resources.

Under this approach bi-directional inference is even more attractive: the functions ADD or DEDUCE will allocate some amount of resources to the processes scheduled for a particular inference. When the allocated resources are exhausted, the inference system enters a suspended state until new resources are allocated to the suspended processes. This will occur when an active process is interested in the results that are being generated by some suspended process (the details of how this is done are presented in [McKay and Shapiro 81]). If resources are allocated to some suspended process it resumes any inference it was doing as well as supplying any results it had already produced. In this case bi-directional inference corresponds to having two sets of processes growing towards each other and their growth can be interupted either by the lack of data in the network or by the exhaustion of available resources.

9. Conclusions

We have seen that combined backward and forward inference, which we term bi-directional inference, focuses a system's attention towards the interests of the user and can cut down the fan out of pure forward or backward chaining. This style of inference corresponds to bi-directional search since the system ends up building two different sets of processes growing toward each other, one of them growing backwards from the goal node (question asked) and the other growing forwards from the relevant data. As soon as enough data is found, these two sets of processes intersect smoothly and information flows through them, producing the desired answer.

References

1. Coelho H, Cotta J. and Pereira L., "How to solve it with PROLOG", Laboratório Nacional de Engenharia Civil, Lisbon, 1980.

2. Colmerauer A., "Sur les bases theoriques de PROLOG", Groupe de I.A., Université d'Aix-Marseille II, 1979.

3. LeFaivre R., "FUZZY reference manual", Computer Science Department, Rutgers University, 1977.

4. Maida T. and Shapiro S., "Intensional Concepts in Propositional Semantic Networks", Department of Computer Science, SUNY at Buffalo, 1980.

5. Martins J. and Shapiro S., "A Belief Revision System based on Relevance Logic and Heterarchical Contexts", Department of Computer Science, SUNY at Buffalo, 1981.

6. McDermott D. and Sussman G., "The CONNIVER reference manual", Technical Report 259, MIT, 1972.

7. McKay D. and Shapiro S., "MULTI - a LISP Based Multiprocessing System", Proc. 1980 LISP Conference, 29-37.

8. McKay D. and Shapiro S., "Using Active Connection Graphs for Reasoning with Recursive Rules", Department of Computer Science, SUNY at Buffalo, 1981.

9. Pohl I., "Bi-directional Search", in Machine Intelligence, Meltzer and Michie (eds.), American Elsevier, 1971, 127-140.

10. Shapiro S., "Representing and Locating Deduction Rules in a Semantic Network", in Proc. Workshop on Pattern-Directed Inference System, SIGART Newsletter, No. 63, (June 1977), 14-18.

11. Shapiro S., "The SNePS Semantic Network Processing System", in Associative Networks, N.V. Findler (ed.), Academic Press, 1979a, 179-203.

12. Shapiro S., "Numerical Quantifiers and their use in Reasoning with Negative Information", Proc. IJCAI-79, 1979b, 791-796.

13. Shapiro S., "Generalized Augmented Transition Network Grammars fo Generation from Semantic Networks", Proc. 17th Anual Meeting of the ACL, 1979c, 25-29.

14. Shapiro S. and McKay D., "Inference with Recursive Rules", Proc. First AAAI Conference, 1980, 151-153.

15. Shubin H. and Shapiro S., "Inference and Control in Multiprocessing Environments", Department of Computer Science, SUNY at Buffalo, 1981.

16. Sussman G., Winograd T. and McDermott D., "MICRO-PLANNER reference manual", Technical Report 203, MIT, 1971.

17. Wand M. "The Frame Model of Computation", Techical Report No.20, Computer Science Department, Indiana University, 1974.

APPENDIX 1: Run in Pure Backward Inference

In this Appendix we present the output generated by the SNePS deduction system when running the example presented in section 6, using backward inference only.

```
**(SURFACE (BUILD AVB $X
*                 ANT (BUILD MEMB *X CLASS HARD/ WORKER)
*                 CQ  (BUILD ANT (BUILD MEMB *X CLASS CS/ MAJOR)
*                            CQ  (BUILD MEMB *X
*                                       CLASS COMPUTER/
PROGRAMMER)
*                      ) = R2 ))
 FOR EVERY V1, IF V1 IS A HARD WORKER
    THEN IF V1 IS A CS MAJOR
            THEN V1 IS A COMPUTER PROGRAMMER
 (DUMPED)
 843 MSECS

**(SURFACE (BUILD AVB *X
*                 ANT (BUILD MEMB *X CLASS CLEVER)
*                 CQ *R2))
 FOR EVERY V1, IF V1 IS CLEVER
    THEN IF V1 IS A CS MAJOR
            THEN V1 IS A COMPUTER PROGRAMMER
 (DUMPED)
 393 MSECS

**(SURFACE (BUILD AVB $Y
*                 &ANT (BUILD MEMB *Y CLASS COMPUTER/ PROGRAMMER)
= M4
*                 &ANT (BUILD AGENT *Y VERB KNOWS OBJ LISP)
*                 CQ (BUILD MEMB *Y CLASS AI/ BUM)))
 FOR EVERY V2, IF V2 IS A COMPUTER PROGRAMMER AND
    V2 KNOWS LISP
    THEN V2 IS A AI BUM
 (DUMPED)
 821 MSECS

**(SURFACE (BUILD AVB *Y
*                 ANT *M4
*                 CQ (BUILD MEMB *Y CLASS COMPUTER/ BUM)))
 FOR EVERY V2, IF V2 IS A COMPUTER PROGRAMMER
    THEN V2 IS COMPUTER BUM
 (DUMPED)
 242 MSECS

**(SURFACE (BUILD MEMB JO CLASS CLEVER))
 JO IS CLEVER
 (DUMPED)
 77 MSECS

**(SURFACE (BUILD AGENT JO VERB KNOWS OBJ LISP))
```

```
 JO KNOWS LISP
 (DUMPED)
 87 MSECS

**(SURFACE (BUILD MEMB JO CLASS HARD/ WORKER))
 JO IS A HARD WORKER
 (DUMPED)
 81 MSECS

**(SURFACE (BUILD MEMB JO CLASS CS/ MAJOR))
 JO IS A CS MAJOR
 (DUMPED)
 78 MSECS

**(SURFACE (DEDUCE MEMB JO CLASS AI/ BUM))

 SINCE
 JO IS CLEVER

 WE INFER
 IF JO IS A CS MAJOR
    THEN JO IS A COMPUTER PROGRAMMER

 SINCE
 JO IS A HARD WORKER

 WE INFER
 IF JO IS A CS MAJOR
    THEN JO IS A COMPUTER PROGRAMMER

 SINCE
 JO IS A CS MAJOR

 WE INFER
 JO IS A COMPUTER PROGRAMMER

 SINCE
 JO IS A COMPUTER PROGRAMMER AND
 JO KNOWS LISP

 WE INFER
 JO IS AI BUM


 JO IS A COMPUTER PROGRAMMER AND
 JO IS AI BUM
 (DUMPED)
 3595 MSECS
```

APPENDIX 2: Run in Pure Forward Inference

In this Appendix we present the output generated by the SNePS deduction system when running the example presented in section 6, using only forward inference.

```
**(SURFACE (BUILD AVB $X
*                 ANT (BUILD MEMB *X CLASS HARD/ WORKER)
*                 CQ  (BUILD ANT (BUILD MEMB *X CLASS CS/ MAJOR)
*                            CQ  (BUILD MEMB *X
*                                       CLASS COMPUTER/
PROGRAMMER)
*                      ) = R2 ))
 FOR EVERY V1, IF V1 IS A HARD WORKER
    THEN IF V1 IS A CS MAJOR
            THEN V1 IS A COMPUTER PROGRAMMER
 (DUMPED)
 846 MSECS

**(SURFACE (BUILD AVB *X
*                 ANT (BUILD MEMB *X CLASS CLEVER)
*                 CQ *R2))
 FOR EVERY V1, IF V1 IS CLEVER
    THEN IF V1 IS A CS MAJOR
            THEN V1 IS A COMPUTER PROGRAMMER
 (DUMPED)
 382 MSECS

**(SURFACE (BUILD AVB $Y
*                 &ANT (BUILD MEMB *Y CLASS COMPUTER/ PROGRAMMER)
= M4
*                 &ANT (BUILD AGENT *Y VERB KNOWS OBJ LISP)
*                 CQ (BUILD MEMB *Y CLASS AI/ BUM)))
 FOR EVERY V2, IF V2 IS A COMPUTER PROGRAMMER AND
    V2 KNOWS LISP
    THEN V2 IS AI BUM
 (DUMPED)
 834 MSECS

**(SURFACE (BUILD AVB *Y
*                 ANT *M4
*                 CQ (BUILD MEMB *Y CLASS COMPUTER/ BUM)))
 FOR EVERY V2, IF V2 IS A COMPUTER PROGRAMMER
    THEN V2 IS A COMPUTER BUM
 (DUMPED)
 254 MSECS

**(SURFACE (BUILD MEMB JO CLASS CLEVER))
 JO IS CLEVER
 (DUMPED)
 78 MSECS

**(SURFACE (BUILD AGENT JO VERB KNOWS OBJ LISP))
```

```
 JO KNOWS LISP
 (DUMPED)
 92 MSECS

**(SURFACE (ADD MEMB JO CLASS HARD/ WORKER))

 SINCE
 JO IS A HARD WORKER

 WE INFER
 IF JO IS A CS MAJOR
    THEN JO IS A COMPUTER PROGRAMMER


 JO IS A HARD WORKER
 (DUMPED)
 1109 MSECS

**(SURFACE (ADD MEMB JO CLASS CS/ MAJOR))

 SINCE
 JO IS A CS MAJOR

 WE INFER
 JO IS A COMPUTER PROGRAMMER

 SINCE
 JO IS A COMPUTER PROGRAMMER

 WE INFER
 JO IS A COMPUTER BUM

 SINCE
 JO KNOWS LISP AND
 JO IS A COMPUTER PROGRAMMER

 WE INFER
 JO IS AI BUM


 JO IS A CS MAJOR AND
 JO IS A COMPUTER PROGRAMMER AND
 JO IS A COMPUTER BUM AND
 JO IS AI BUM
 (DUMPED)
 2624 MSECS
```