THE SNePS SEMANTIC NETWORK PROCESSING SYSTEM

Stuart C. Shapiro

1. INTRODUCTION

Semantic networks have been used as representations of knowledge since the mid 1960s. Quillian's 1966 semantic memory, described in Quillian [1968], is considered the first semantic network, but it had roots in Raphael's 1964 SIR [Raphael, 1968] and in the work of Reitman [1965]. An excellent historical review is in the chapter by Brachman in this volume.

One can define a semantic network as a labeled directed graph in which nodes represent concepts, arc labels represent binary relations, and an arc labeled R going from node n to node m represents that the concept represented by n bears the relation represented by R to the concept represented by m. Each concept represented in the network is represented by a unique node.

The notion of "concept" is vague, but one can think of it as including anything about which information can be stored and/or transmitted. Various semantic networks have included as concepts concrete and abstract individuals, prototypical and generalized individuals, actions, sets, propositions, facts, beliefs, roles, relations, hypothetical worlds, and others. Since Woods [1975] first pointed it out, it has been generally recognized that each node represents an intensional rather than an extensional concept. Furthermore, two concepts that are extensionally equivalent but intensionally distinct may be represented by different nodes.

Since each concept represented in the network is represented by a node, it follows that the relations represented solely by arc labels are not conceptual. I refer to such relations as *structural relations* since they are used to form the basic structure of the semantic network, and distinguish them from *conceptual relations*, which are represented by nodes. This distinction is discussed further in Shapiro [1971a,b], where structural relations are also called *item relations* and conceptual relations are also called *system relations*.

There are four levels at which semantic networks can be discussed: an

ISBN 0-12-256380-8

abstract graph level, a two-dimensional pictorial level, a one-dimensional symbolic level, and a computer implementation level. These levels, though related, are independent in the sense that two semantic networks can differ on one or more levels and be the same on the other levels. Often the levels are not clearly distinguished, and one must be careful not to conclude from a discussion at one level what another level must be like. Similarly, when comparing different semantic networks, one must be careful to note on which levels they differ and on which, if any, they are the same.

This chapter describes the SNePS semantic network processing system, which is a direct descendent of MENTAL [Shapiro, 1971a,b]. SNePS is currently implemented in ALISP [Konolige, 1975] and runs interactively on the CDC CYBER 173 at the State University of New York at Buffalo. Three levels of SNePS are described: the abstract graph level, the pictorial level, and the linear symbolic level. For the latter, the SNePS User Language, SNePSUL, is used.

2. BASIC REPRESENTATION

2.1. Abstract Graph Level

A SNePS semantic network is a labeled directed graph in which nodes represent concepts and arcs represent nonconceptual binary relations between concepts. Each concept is represented by a unique node. Whenever an arc representing a relation **R** goes from node **n** to node **m**, there is an arc representing the converse relation of **R**, $\mathbf{R}^c$, going from **m** to **n**. An arc is labeled with a symbol intended to be mnemonically suggestive of the relation the arc represents.

I distinguish three kinds of arcs: *descending*, *ascending*, and *auxiliary*. For each relation represented by descending arcs, there is a converse relation represented by ascending arcs and vice versa. Together, descending and ascending arcs are the regular semantic network arcs referred to above. Auxiliary arcs are used for hanging nonnodal information on nodes and for typing the nodes as discussed below. If a descending arc goes from node **n** to node **m**, I say that **n** *immediately dominates* **m**. If there is a path of descending arcs from node n to node m, I say that **n** *dominates* **m**. If **R** is an arc label and **n** is a node, I shall use the notation **R(n)** for the set of nodes into which arcs labeled **R** go from **n**. In what follows, we shall often use the phrase "the relation **R**" when we mean "an arc labeled **R**."

There are three kinds of nodes: *constant*, *nonconstant*, and *auxiliary*. Auxiliary nodes are connected to each other and to other nodes only by auxiliary arcs. Auxiliary nodes do not represent concepts but are used by the SNePS system or the SNePS user to type nonauxiliary nodes or to maintain a reference to one or more nonauxiliary nodes. Constant nodes

represent unique semantic concepts. Nodes that dominate no other node are called *atomic* nodes. Atomic constants are called *base* nodes and atomic nonconstants are called *variable* nodes or *variables*. Variables are distinguished by being in the auxiliary relation :VAR to the auxiliary node T. Variable nodes are used in SNePS like variables are used in normal predicate logic notations. Nonatomic nodes are called *molecular* nodes. They are often used for representing propositions. There is a set of descending relations called *binding* relations that act like quantifiers in normal symbolic logic formalisms. A molecular node that immediately dominates one or more variables and no other variable may have at most one binding relation to an arbitrary number of its dominated variables, which are referred to as *bound* by that molecular node. The remaining dominated variables are referred to as *free* in the molecular node, which has an auxiliary :SVAR to each of them. If a node **m** immediately dominates a set of variable nodes $\{\mathbf{v}_1, \ldots, \mathbf{v}_l\}$ and a set of molecular nodes $\{\mathbf{n}_1, \ldots, \mathbf{n}_k\}$ and $\mathbf{V} = \{\mathbf{v}_1, \ldots, \mathbf{v}_l\} \cup$:SVAR$(\mathbf{n}_1) \cup \cdots \cup$:SVAR$(\mathbf{n}_k)$ is nonempty, **m** may have at most one binding relation, say Q, to one or more variables in **V**. These variables are referred to as bound by **m**. The remainder, $\mathbf{V} - \mathbf{Q}(\mathbf{m})$, are free in **m** and have the arc :SVAR to each of them from **n**. A node **n** such that :SVAR(**n**) is nonempty is a nonconstant molecular node and is called a *pattern* node. Pattern nodes are comparable to well-formed formulas with free variables. A molecular node **n** for which :SVAR(**n**) is empty is a molecular constant or *assertion* node.

Temporary molecular and variable nodes can be created. Temporary molecular nodes have no ascending arcs coming into them from the nodes they dominate. Temporary nodes are not placed on any permanent system list and are garbage-collected when no longer referenced. They are invisible to all the semantic network retrieval operations. I shall refer to nontemporary nodes as *permanent* nodes. Temporary nodes are used to build patterns of network structures, which can be matched against the network but do not match themselves. Occasionally, a structure of temporary nodes is used as a template for building a permanent structure resembling it.

2.2. Pictorial Level

The discussion so far has treated SNePS as an abstract graph. I can also discuss it diagrammatically or pictorially. In the diagrams, a base node is drawn as an oval inside of which is an identifier meant to be suggestive of the concept the node represents. A permanent assertion node is drawn as a circle inside of which is an arbitrary identifier of the form Mn; a permanent variable node as a circle inside of which is an arbitrary identifier of the form Vn; a permanent pattern node as a circle inside of which is an arbitrary identifier of the form Pn. A temporary variable node is shown as

an identifier of the form Qn; a temporary molecular node as an identifier of the form Tn; an auxiliary node as a mnemonically suggestive identifier. Temporary and auxiliary nodes are drawn without enclosing circles. Figure 1 shows a network with various kinds of nodes and arcs. In future figures, the :VAR and :SVAR arcs will be omitted since they can be reconstructed from the information shown. For the same reason, the ascending arcs will be omitted.

2.3. Linear Symbolic Level—The SNePS User Language

At the linear symbolic level, I shall use the SNePS User Language, SNePSUL. Presentation of SNePSUL will give the reader an idea of what can be done in SNePS. It will also show the relationship between the linear symbolic form and the two-dimensional pictorial form, and will allow the former to be used in some examples, when the latter would be too complicated for easy understanding.

SNePSUL is embedded in LISP and consists of a set of functions for which the unquote convention (see Bobrow and Raphael [1974]) holds. An atom refers to itself unless it is unquoted. A list is either a reference to a SNePSUL function or a list of elements, which can be atoms, unquoted atoms, or SNePSUL function references.

SNePS itself is a read–evaluate–print loop, which assumes that each expression it reads is in SNePSUL. However, LISP may be accessed from SNePSUL and vice versa. Unless otherwise noted, all examples will show interaction with SNePSUL. The SNePSUL prompt is ** for the first line of an expression and * for subsequent lines.

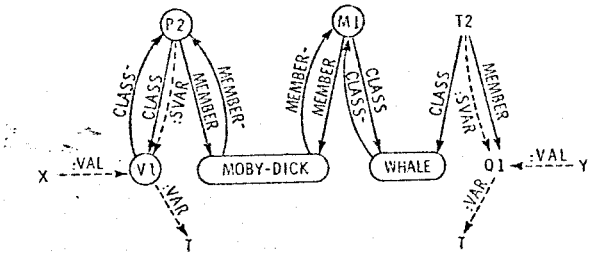


Fig. 1. An example of various kinds of nodes and arcs.

Descending arcs	MEMBER, CLASS
Ascending arcs	MEMBER-, CLASS-
Auxiliary arcs	:VAL, :VAR, :SVAR
Base nodes	MOBY-DICK, WHALE
Assertion node	M1
Variable node	V1
Pattern node	P2
Temporary variable	Q1
Temporary pattern	T2
Auxiliary nodes	X,Y,T

2.3.1. *Defining Arc Labels*

Since SNePS is not a particular semantic network but a system for building, operating on, and experimenting with semantic networks, the user is responsible for choosing arc labels. The function DEFINE declares the labels of descending and ascending arcs. The syntax is

$$(\text{DEFINE } R_1\ R_1^c\ R_2\ R_2^c\ \ldots)$$

Here, R_1, R_2, etc., are declared to be labels of descending arcs, with R_1^c, R_2^c, etc., the corresponding ascending arc labels. An error message is given if any label is already in use. The system remembers the relationship between an arc label and the label of the converse arc by making each an auxiliary node with the auxiliary arc :CONV from each to the other. That is, for each i, :CONV(R_i) = $\{R_i^c\}$ and :CONV(R_i^c) = $\{R_i\}$.

When defining a set of arcs, one should keep in mind that it is a basic precept of semantic networks that all concepts, including assertions, are represented by nodes. Figure 2 shows three possible ways of representing "Socrates is a man," using ISA as an example of any binary relation. One can analyze the differences in these representations as follows. In Fig. 2a, there is a node representing Socrates and a node representing the set of men, but no node representing the assertion that Socrates is a man. SNePS retrieval functions will allow the retrieval of all members of the set of men and all the sets of which Socrates is a member, but not the specific assertion that Socrates is a man. In Fig. 2b, there are nodes representing Socrates, the set of men, and the assertion that Socrates is a man, but no node representing set membership as a conceptual relation. SNePS retrieval functions will allow the same retrievals as in Fig. 2a, plus the assertion that Socrates is a man, but not the relationship between Socrates and the set of men. In Fig. 2c, there are nodes representing Socrates, the set of

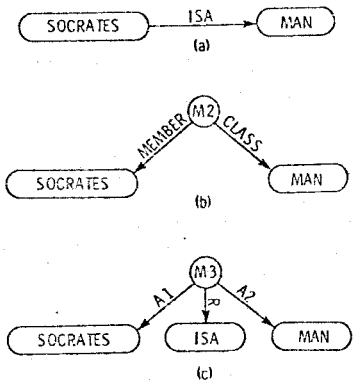


Fig. 2. Three ways of representing "Socrates is a man."

men, the ISA relation, and the assertion that Socrates is a man. SNePS retrieval functions will allow the same retrievals as in Fig. 2b, plus retrieval of the relationship between Socrates and the set of men. In this chapter, set membership will be represented as in Fig. 2b.

What SNePS retrieval functions can retrieve is coextensive with what the understander being modeled by SNePS can express (see the section on generating, below). Thus, although Fig. 2a represents the knowledge that Socrates is a man, it is inexpressible ("unconscious") knowledge. If it cannot be expressed by the understander, it cannot be expressed to the understander, and, indeed, the network of Fig. 2a cannot be built in SNePS, as will be explained in the discussion of BUILD.

The user chooses auxiliary arc labels by defining them with the function DEFINE-AUX, which takes a set of labels to be defined. Several auxiliary arcs are used by the system itself and are predefined as if

(DEFINE-AUX :CONV :VAL :VAR :SVAR)

had already been executed.

2.3.2. *Adding Information*

The main function for adding information to the network is BUILD, whose syntax is

(BUILD $\mathbf{R_1}$ **nodeset**$_1$ $\mathbf{R_2}$ **nodeset**$_2$ . . .)

Each $\mathbf{R_i}$ is the label of an ascending, descending, or auxiliary arc that has already been defined. Each **nodeset** is a node or set of nodes. BUILD creates a new node that has arcs $\mathbf{R_1}$ to the nodes of **nodeset**$_1$, $\mathbf{R_2}$ to the nodes of **nodeset**$_2$, etc. If the new node dominates any variable nodes, :SVAR arcs will be added automatically as appropriate. The identifier of the new node is created by the system, and a list of that identifier is returned as the value of BUILD. Table I shows the effect of the different possible forms for the **nodesets.**

Temporary nodes are created with the TBUILD function. The syntax is the same as that for BUILD, the only difference being that the newly created node is temporary rather than permanent. Figure 3 shows the SNePS session for building the network of Fig. 1.

Notice that when a new node is created by the #**atom**, $**atom**, or %**atom** form, **atom** becomes an auxiliary node with the auxiliary arc :VAL to the new node. This auxiliary node is called a *SNePSUL variable*, and is quite different from a variable node. Variable nodes are like variables in predicate logic. SNePSUL variables are like variables of a programming language. Another way to set the value of a SNePSUL variable is by appending to any **nodeset** an equal sign followed by the variable. At the top level, an enclosing set of parentheses is required. This technique is demonstrated in Fig. 4. (From now on figures will be cumulative and diagrams of the network will show new material enclosed by dashed lines.)

TABLE I

Effect of the Different Possible Forms of nodeset$_i$ in the SNePSUL Function Call (BUILD . . . R_i nodeset$_i$ . . .)

Form of nodeset$_i$	New node gets arc(s) labeled R_i to
atom	the node whose identifier is atom
#atom	a new base node, which is also made the value of :VAL(atom)
$atom	a new variable node, which is also made the value of :VAL(atom)
%atom	a new temporary variable node, which is also made the value of :VAL(atom)
*atom	all the nodes in :VAL(atom)
list whose first element is the name of a SNePSUL function	the set of nodes returned as the value of the function call
list whose first element is not the name of a SNePSUL function	the set of nodes obtained by treating each element of the list as a form given in this table
(↑ LISP S-expression)	the set of nodes obtained by evaluating the LISP S-expression and treating the value as a form given in this table.

Several SNePSUL variables are maintained by the system:

NODES	the set of permanent nodes
VARBL	the set of permanent variable nodes
DRELST	the set of descending arc labels
ARELST	the set of ascending arc labels
AUXRELST	the set of auxiliary arc labels

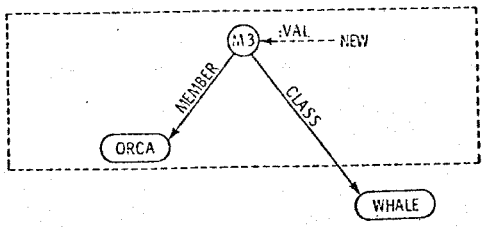
There is no way in SNePSUL to add a nonauxiliary arc between two already existing nodes. This enforces the notion that any information given to the system is a concept and so must have a node representing it. The new node created by BUILD serves this purpose. Specifically, it is impossible to have a nonauxiliary arc between two nodes neither of whose identifiers were created by the system. For this reason the network of Fig. 2a could not exist in SNePS.

```
**(DEFINE MEMBER MEMBER- CLASS CLASS-)
(MEMBER MEMBER-)
(CLASS CLASS-)
(DEFINED)

**(BUILD MEMBER MOBY-DICK CLASS WHALE)
(M1)

**(BUILD MEMBER MOBY-DICK CLASS $X)
(P2)

**(TBUILD MEMBER %Y CLASS WHALE)
(T2)
```

Fig. 3. SNePS session building the network of Fig. 1.

```
**((BUILD MEMBER ORCA CLASS WHALE) = NEW)
(M3)
```

(a)



(b)

Fig. 4. Use of the equal sign for setting SNePSUL variables. (a) SNePSUL interaction; (b) modified network with the new material enclosed by dashed lines.

2.3.3. *DUMP and DESCRIBE*

Pieces of the network may be examined via the functions DUMP and DESCRIBE, which have the same syntax:

$$\left(\left\{\begin{matrix}\text{DUMP}\\ \text{DESCRIBE}\end{matrix}\right\} \textbf{nodeset}_1 \ldots \textbf{nodeset}_k\right)$$

For each node, DUMP prints the node's identifier, all arcs emanating from it, and the set of nodes to which each arc goes. DESCRIBE only prints descending and auxiliary arcs, but prints the information for all molecular nodes dominated by the nodes given. Figure 5 demonstrates the use of DUMP and DESCRIBE.

2.3.4. *Deleting Information*

There are three functions for removing information from the network.

(ERASE $\textbf{node}_1 \ldots \textbf{node}_k$) removes each $\textbf{node}_i$ from the network along with any other nodes that thereby become isolated.

(REMVAR $\textbf{variable}_1 \ldots \textbf{variable}_k$) unassigns each of the listed SNePSUL variables.

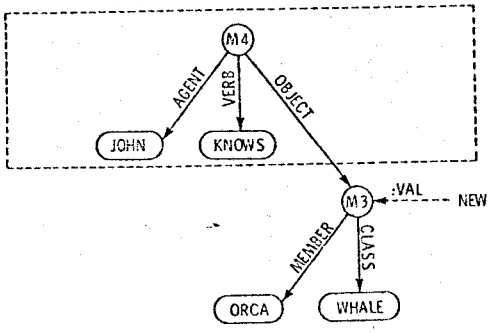
(DELREL $\textbf{label}_1 \ldots \textbf{label}_k$) undefines each of the arc labels and their converses by removing them from the values of DRELST, ARELST, and AUXRELST. However, if any arcs with these labels already exist in the network, the arcs are not removed.

2.3.5. *Finding Nodes*

The function FIND performs a pattern match on the network and returns a list of the matched nodes. The syntax is the same as for BUILD, but specifies a node to be found rather than a node to be built. Figure 6 shows some simple uses of FIND.

```
**(DEFINE AGENT AGENT- VERB VERB- OBJECT OBJECT-)
(AGENT AGENT-)
(VERB VERB-)
(OBJECT OBJECT-)
(DEFINED)

**(DUMP(BUILD AGENT JOHN VERB KNOWS OBJECT *NEW) *NEW)
(M4(AGENT(JOHN))(VERB(KNOWS))(OBJECT(M3)))
(M3(MEMBER(ORCA))(CLASS(WHALE))(OBJECT-(M4)))
(DUMPED)

**(DESCRIBE M4)
(M4(AGENT(JOHN))(VERB(KNOWS))(OBJECT(M3)))
(M3(MEMBER(ORCA))(CLASS(WHALE)))
(DUMPED)
```

(a)



(b)

Fig. 5. Demonstration of DUMP and DESCRIBE. The material enclosed by dashed lines in (b) was added by the instructions of (a).

It should be noticed that (FIND R_1 **nodeset**$_1$ R_2 **nodeset**$_2$) finds all nodes with an R_1 arc to any node in **nodeset**$_1$ and an R_2 arc to any node in **nodeset**$_2$. So (FIND R (N_1 N_2)) returns $R^c(N_1) \cup R^c(N_2)$, whereas (FIND R N_1 R N_2) returns $R^c(N_1) \cap R^c(N_2)$.

Besides the forms listed in Table I, a **nodeset** in a FIND may be of the form **?atom**. This represents a pattern variable that can match any node.

```
**(DESCRIBE (FIND MEMBER MOBY-DICK CLASS WHALE))
(M1(MEMBER(MOBY-DICK))(CLASS(WHALE)))

**(FIND MEMBER- (FIND CLASS WHALE))
(MOBY-DICK ORCA)

**(DESCRIBE (FIND VERB KNOWS OBJECT (FIND CLASS WHALE)))
(M4(AGENT(JOHN))(VERB(KNOWS))(OBJECT(M3)))
(M3(MEMBER(ORCA))(CLASS(WHALE)))
```

Fig. 6. Some examples of FIND.

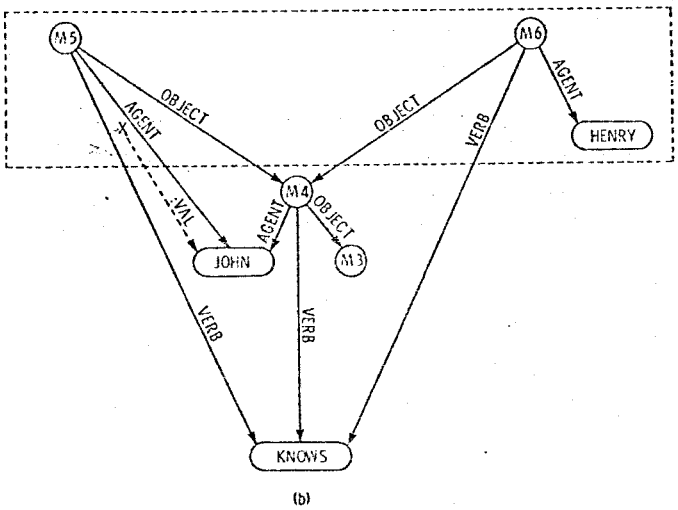
```
**(FIND VERB KNOWS OBJECT(FIND MEMBER ?KNOWN-WHALE CLASS WHALE))
(M4)

** *KNOWN-WHALE
(ORCA)
```

Fig. 7. Simple use of a variable pattern in a FIND.

When SNePS has finished evaluating the top level FIND, the atom is given the list of nodes that it matched as its SNePSUL value. Figure 7 shows a simple use of this feature. Figure 8 shows a more complicated use of the pattern variable, where it is used to specify a node at the intersection of two paths.

A node may be found that satisfies one specification, but does not satisfy another with the aid of the infix set difference operator "-."

```
**(BUILD AGENT JOHN VERB KNOWS OBJECT M4)
(M5)

**(BUILD AGENT HENRY VERB KNOWS OBJECT M4)
(M6)

**(DESCRIBE(FIND AGENT ?X VERB KNOWS
*                  OBJECT(FIND AGENT ?X VERB KNOWS)))
(M5(AGENT(JOHN))(VERB(KNOWS))(OBJECT(M4)))
(M4(AGENT(JOHN))(VERB(KNOWS))(OBJECT(M3)))
(M3(MEMBER(ORCA))(CLASS(WHALE)))

** *X
(JOHN)
```



(b)

Fig. 8. Using a variable pattern to specify a node at the end of two paths. (a) The SNePSUL dialogue, (b) the network with material resulting from (a) enclosed by dashed lines.

```
**(FIND MEMBER- (FIND CLASS WHALE) - (FIND OBJECT- (FIND VERB KNOWS)))
(MOBY-DICK)
```

Fig. 9. A use of the set difference operator.

This operator is demonstrated in Fig. 9, where we find whales that no one knows are whales. One can also specify a set of nodes except those with certain arcs emanating from them, with the infix $\setminus$ operator. The value of (**nodeset** $\setminus$ (R_1, . . . , R_k)) is (**nodeset** $-$ {n|R_i(n) $\neq \phi$, $1 \leq i \leq k$}). Figure 10 demonstrates this operator.

The function (FINDORBUILD R_1 **nodeset**$_1$ . . . R_n **nodeset**$_n$) first tries to find the specified node(s) but if none exist, it builds one. FINDORBUILD is used instead of BUILD when we wish to share nested molecular structures but are not sure if appropriate ones already exist.

3. INFERENCE

3.1. Representation of Deduction Rules

Automatic inference may be triggered by using the function DEDUCE, a generalization of FIND, or the function ADD, a generalization of BUILD. In order for these to accomplish anything, *deduction rules* must exist in the network. A deduction rule is a network structure dominated by a *rule node*. A rule node represents a propositional formula of molecular nodes, using one of the four connectives: $\vee$-entailment, $\wedge$-entailment, AND-OR, THRESH. A rule node r may also have either AVB(r) or EVB(r) nonempty, where AVB and EVB are the two binding relations representing universal and existential quantification, respectively.

This representation derives from that of Shapiro [1971a,b] and was further influenced by Kay [1973], Hendrix [1975a,b], and Schubert [1976]. It differs from the representation of Shapiro [1971a,b] by not using nodes to represent quantifiers, and from all the earlier work by the use of the new connectives, except for $\wedge$-entailment, which is the same as the generalized material implication of Schubert [1976, p. 173]. The match routine used by SNePS to locate relevant deduction rules is described in Shapiro [1977] and Shapiro and McKay [1979].

The four connectives all take sets of nodes as arguments and may be explicated as follows.

1. $\vee$-entailment: $\{A_1, \ldots, A_n\} \vee\!\!\rightarrow \{C_1, \ldots, C_m\}$ is true just in case each A_i, $1 \leq i \leq n$, entails each C_j, $1 \leq j \leq m$.

```
**(FIND MEMBER- (FIND CLASS WHALE) \ (OBJECT-))
(MOBY-DICK)
```

Fig. 10. A use of the arc restriction operator $\setminus$.

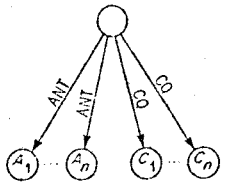


Fig. 11.

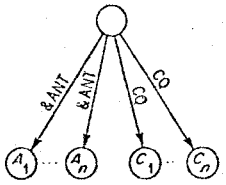


Fig. 12.

Fig. 11. The network representation of $\{A_1, \ldots, A_n\} \vee\!\!\rightarrow \{C_1, \ldots, C_n\}$.

Fig. 12. The network representation of $\{A_1, \ldots, A_n\} \wedge\!\!\rightarrow \{C_1, \ldots, C_n\}$.

2. $\wedge$-entailment: $\{A_1, \ldots, A_n\} \wedge\!\!\rightarrow \{C_1, \ldots, C_m\}$ is true just in case each C_j, $1 \leq j \leq m$, is entailed by the conjunction of the A_i, $1 \leq i \leq n$.

3. AND-OR: ${}_n\chi_i^j\{P_1, \ldots, P_n\}$ is true just in case at least i and at most j of the P are true.

4. THRESH: ${}_n\Theta_i\{P_1, \ldots, P_n\}$ is true just in case either fewer than i of the P are true or they all are.

Figures 11–14 show how these connectives are represented in the network. Each choice of parameters for AND-OR and THRESH gives, in effect, a different connective. Some familiar ones are shown in Table II. Specifically, we shall abbreviate ${}_1\chi_0^0(P)$ by $\sim P$.

Figure 15 illustrates how a negation can be stored in SNePS. In that figure, M7 represents "John loves Jane" and R1 represents "John does not love Jane." Since ARG^c (M7) is not empty, M7 is not considered "asserted in the network." Figure 16 shows a network for "Mary thinks that John loves Jane, but he doesn't." There, M8 and R1 are asserted in the network, while M7 is not. Figure 17 shows two alternative ways of representing "Mary thinks that John doesn't love Jane, but he does." In Fig. 17a, R2 acts as an assertion operator, asserting in the network that John loves Jane, while M9 asserts that Mary thinks that John doesn't love Jane. R1 is not asserted in Fig. 17. In Fig. 17b, M10 provides the assertion that John loves Jane. The technique of Fig. 17a has the benefit that it makes explicit that the very assertion (M7) that Mary thinks is not true is true. The technique

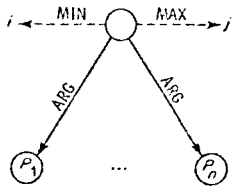


Fig. 13.

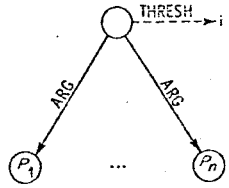


Fig. 14.

Fig. 13. The network representation of ${}_n\chi_i^j\{P_1, \ldots, P_n\}$.

Fig. 14. The network representation of ${}_n\Theta_i\{P_1, \ldots, P_n\}$.

TABLE II
Some Familiar Connectives Represented by AND-OR and THRESH

Formula	Meaning
${}_n\&_n\{P_1, \ldots, P_n\}$	P_1 and $\cdots$ and P_n
${}_n\vee_1^n\{P_1, \ldots, P_n\}$	P_1 and/or $\cdots$ and/or P_n
${}_n\&_0^0\{P_1, \ldots, P_n\}$	neither P_1 nor $\cdots$ nor P_n
${}_n\vee_1^1\{P_1, \ldots, P_n\}$	exactly one of P_1 or $\cdots$ or P_n
${}_n\&_0^{n-1}\{P_1, \ldots, P_n\}$	not (P_1 and $\cdots$ and P_n)
${}_n\Theta_1\{P_1, \ldots, P_n\}$	P_1 and $\cdots$ and P_n are all equivalent

of Fig. 17b, however, has the benefit that John loves Jane is asserted in the standard manner, namely, that there is a node in (FIND AGENT JOHN VERB LOVES OBJECT JANE) that is not dominated by any other node. In SNePS, R2 would be treated as a deduction rule capable of immediately deriving a node like M10. R1, if dominated by no other node, would also be treated as a deduction rule, but would derive itself, namely R1, the negation of M7. This discussion was motivated by the example of Scragg [1976, pp. 108–110], where "Peter said he went to the store, but he didn't" is handled as in Fig. 16, but "Peter said he didn't go to the store, although he did" is not discussed.

The LISP function (TOP? N) is provided, which returns T if the node N is asserted in the network, and NIL if it is not. The SNePSUL functions FORBTOP and FORBNOTOP are like FINDORBUILD but the former only finds nodes satisfying TOP?, while the latter only finds nodes not satisfying TOP? It is important to use FORBNOTOP when attempting to share dominated structures so that an asserted node does not become accidentally unasserted. Similarly, it is important to use FORBTOP when attempting to find-or-build a new asserted node.

A rule node that dominates one or more pattern nodes may have either AVB or EVB arcs to one or more of the variable nodes free in those pattern nodes. The restriction to either AVB or EVB arcs, but not both, is neces-

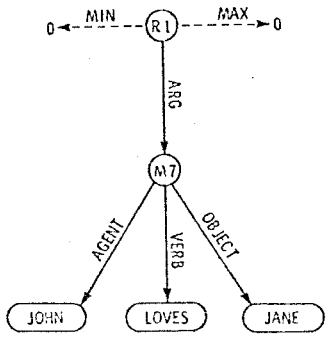


Fig. 15. "John does not love Jane."

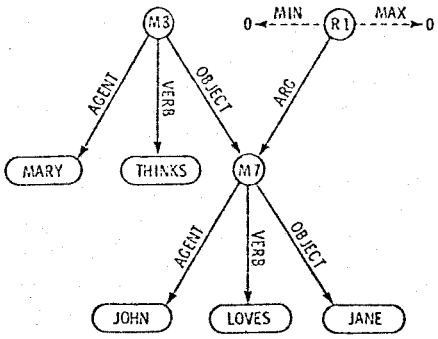


Fig. 16. "Mary thinks that John loves Jane, but he doesn't."

sary so that quantifier strings of the form $\forall\exists$ and $\exists\forall$ may be distinguished. Multiple AVB or EVB arcs are allowed since within quantifier strings of the form $\forall x_1 \cdots \forall x_n$ and $\exists x_1 \cdots \exists x_n$ order is irrelevant. Binding arcs may emanate only from rule nodes. This reflects the notion that in the formula $Q(x_1 \cdots x_n)A$, where Q is $\forall$ or $\exists$ and A is any formula, the quantifier clause $Q(x_1 \cdots x_n)$ may be associated with the main connective of A. If A has no main connective, either because it is atomic or because it is of the form $Q(y_1 \cdots y_m)B$, the formula can be represented as $Q(x_1 \cdots x_m)[\{\ \} \wedge\!\!\rightarrow \{A\}]$.

Figure 18 shows the SNePSUL instructions for building the deduction rules for "Every man loves some woman," and for the definition of transitive relations. A few comments are in order about the form chosen for these rules. If SNePS is asked to deduce instances of R(x) using the rule, $\forall x(\{P(x)\} \vee\!\!\rightarrow \{Q(x), R(x)\})$, and P(a) is true, it will build R(a) in the network, but not bother building Q(a). However, if the rule had been $\forall x(\{P(x)\} \vee\!\!\rightarrow {}_2\forall^2_2\{Q(x), R(x)\})$ it would build both Q(a) and R(a). That is the reason for the form of the rule in Fig. 18a. If, for example, one asks whom Bill loves, one would not want to introduce a new Skölem constant t and record "BILL LOVES t" without also recording "t $\in$ WOMAN". When SNePS uses a rule of the form {P, Q} $\wedge\!\!\rightarrow$ {R}, parallel processes are used to deduce P and Q. However, even though the rule $\{P\} \vee\!\!\rightarrow \{\{Q\} \vee\!\!\rightarrow \{R\}\}$ is formally equivalent, in this case SNePS establishes a process to deduce Q only after a deduction of P has been successful. The form of the rule in Fig. 18b forces SNePS to first check that a relationship is transitive before using the transitivity rule further. The way parallel processes are used to carry out deductions is discussed more fully in Shapiro and McKay [1979].

The rule in Fig. 18b looks like a second-order rule, but strictly speaking, it is not. Relations such as ON are represented by semantic nodes as are other individual concepts, and so variables can range over them. We can consider each *case frame*, i.e., each combination of descending arcs, to be a

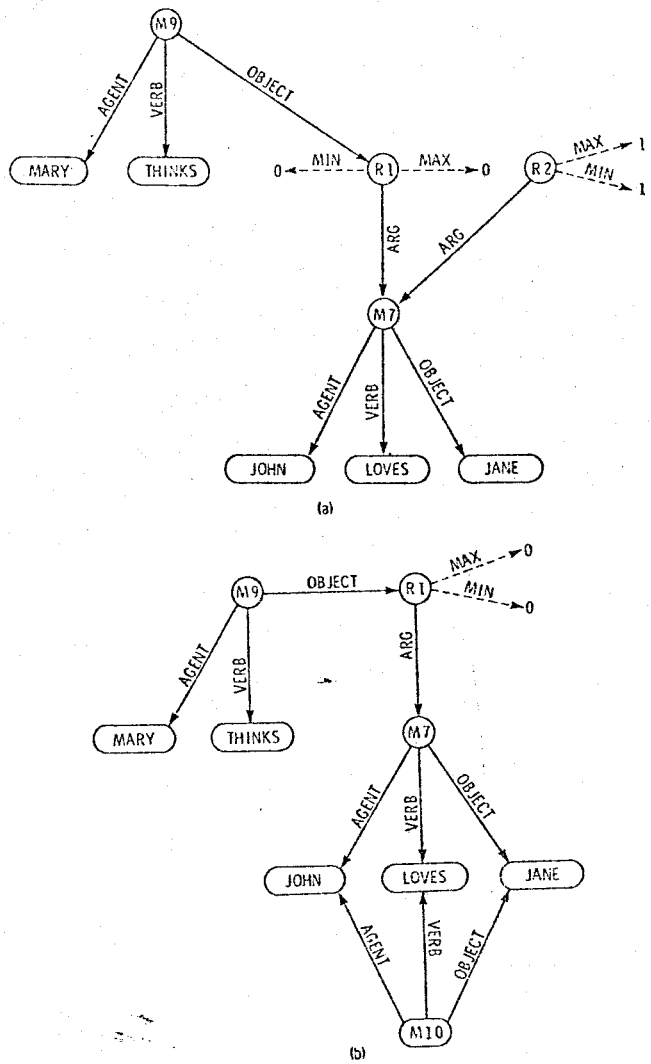


Fig. 17. Two ways of representing "Mary thinks that John doesn't love Jane, but he does." (a) with an assertion operator, (b) with a duplicated molecular node.

different predicate in the network. Thus, if we represent the proposition ON(BLOCK1,BLOCK2) by the node constructed by (BUILD AGENT BLOCK1 VERB ON OBJECT BLOCK2), it is accurate to say that we are asserting the proposition AGENT-VERB-OBJECT(BLOCK1,ON,BLOCK2). We never quantify predicates of the type of AGENT-VERB-OBJECT, even though we do quantify individuals like ON that are used as predicates in other notations.

```
**(BUILD AVB $X
        ANT(BUILD MEMBER *X CLASS MAN)
        CQ(BUILD EVB $Y MIN 2 MAX 2
                ARG((BUILD MEMBER *Y CLASS WOMAN)
                    (BUILD AGENT *X VERB LOVES
                            OBJECT *Y))))
(M11)
                                        (a)
**(BUILD AVB $R
        ANT(BUILD MEMBER *R CLASS TRANSITIVE)
        CQ(BUILD AVB($X $Y $Z)
                &ANT ((BUILD AGENT ,X VERB ,R OBJECT ,Y)
                      (BUILD AGENT *Y VERB *R OBJECT *Z))
                 CQ(BUILD AGENT *X VERB *R OBJECT *Z)))
(M17)
                                        (b)
```

Fig. 18. SNePSUL BUILDS for (a) Every man loves some woman: $\forall x(\{x \in MAN\} \vee\!\!\rightarrow \{\exists y_z(\{y \in WOMAN, xLOVESy\})\})$. (b) The definition of transitivity: $\forall R(\{R \in TRANSITIVE\} \vee\!\!\rightarrow \{\forall(x, y, z)(\{R(x, y), R(y, z)\} \wedge\!\!\rightarrow \{R(x, z)\})\})$.

3.2. Backward Inference

Deduction rules are used for backward inference via the SNePSUL function (DEDUCE **numb** $\mathbf{R}_1$ **nodeset**$_1$ $\cdots$ $\mathbf{R}_n$ **nodeset**$_n$). This causes the temporary node (TBUILD $\mathbf{R}_1$ **nodeset**$_1$ $\cdots$ $\mathbf{R}_n$ **nodeset**$_n$) to be built and turned over to the deduction system, which uses a multiprocessing system to deduce both positive and negative instances of it. The parameter **numb** controls how many answers are desired. If **numb** is omitted, all answers are found. If **numb** is 0, the DEDUCE is equivalent to a FIND—only explicit answers are found. If **numb** is a positive integer, deduction ceases as soon as at least **numb** answers are found. Otherwise, **numb** is a list of two numbers (**npos nneg**), and deduction ceases as soon as at least **npos** positive instances and **nneg** negative instances are found. For example, to cease deduction as soon as three positive instances are found, **numb** should be (3 0); to stop as soon as one answer, either positive or negative, is found, **numb** should be 1. Since several instances may be found at the same time, the number of answers found may sometimes exceed the number requested. If fewer than the requested number of answers are deducible, all will be returned.

Table III summarizes the rules of inference used for each of the four connectives.

Figure 19 shows a deduction using the rule in Fig. 18a, and Fig. 20 shows a deduction using the rule in Fig. 18b to deduce four positive instances. When a deduction is interrupted, as in Fig. 20, sufficient information to resume the deduction is stored as the SNePS value of LASTINFER.

TABLE III
Rules of Inference for the Four Connectives

Deduction Rule	As soon as _____ are found	Deduce
$\forall x\{A_1(x), \ldots, A_n(x)\}$ $\vee\!\!\rightarrow\{C_1(x), \ldots, C_m(x)\}$	1 of $A_1(a), \ldots, A_n(a)$	$C_m(a)$
$\forall x\{A_1(x), \ldots, A_n(x)\}$ $\wedge\!\!\rightarrow\{C_1(x), \ldots, C_m(x)\}$	n of $A_1(a), \ldots, A_n(a)$	$C_m(a)$
$\forall x_n \chi^j_i\{P_1(x), \ldots, P_n(x)\}$	j of $P_1(a), \ldots, P_{n-1}(a)$	$\sim P_n(a)$
	$n - i$ of $\sim P_1(a), \ldots, \sim P_{n-1}(a)$	$P_n(a)$
$\forall x_n \Theta_i\{P_1(x), \ldots, P_n(x)\}$	i of $P_1(a), \ldots, P_{n-1}(a)$	$P_n(a)$
	$i - 1$ of $P_1(a), \ldots, P_{n-1}(a)$ and 1 of $\sim P_1(a), \ldots, \sim P_{n-1}(a)$	$\sim P_n(a)$

The deduction may be resumed by evaluating

(RESUME **numb** *LASTINFER),

as shown in Fig. 21.

3.3. Restricted Forward Inference

When the multiprocessing deduction system uses an inference rule, an INFER process is created for each antecedent proposition. For example, if the rule $\forall(x, y)\ {}_n\chi^{\max}_{\min}(P_1(x, y), \ldots, P_n(x, y))$ is to be used to deduce $P_n(a, z)$, an INFER process is created for each $P_i(a, z)$, $1 \leq i \leq n - 1$. Each INFER process is connected to a *data collector* process that stores all deduced instances of $P_i(a, z)$. While these processes are retained, if $P_i(a, y)$ becomes a subgoal of another deduction, the results stored in the data collector are used and repeated deduction of the same information is

```
**(BUILD MEMBER JOHN CLASS MAN)
(M18)
**(BUILD MEMBER HENRY CLASS MAN)
(M19)
**(DESCRIBE(DEDUCE AGENT %X VERB LOVES OBJECT %Y))
(M24(AGENT(HENRY))(VERB(LOVES))(OBJECT(M21)))
(M22(AGENT(JOHN))(VERB(LOVES))(OBJECT(M20)))
(DUMPED)
**(DESCRIBE (FIND CLASS WOMAN) \ (:SVAR))
(M23(MEMBER(M20))(CLASS WOMAN)))
(M25(MEMBER(M21))(CLASS WOMAN)))
```

Fig. 19. A use of rule Fig. 18(a), "Every man loves a woman." The data base includes the network of Fig. 16, but not of Fig. 17.

```
**(BUILD AGENT A VERB SUPPORTS OBJECT B)
(M26)
**(BUILD AGENT B VERB SUPPORTS OBJECT C)
(M27)
**(BUILD AGENT C VERB SUPPORTS OBJECT D)
(M28)
**(BUILD MIN 0 MAX 0
          ARG(BUILD AGENT B VERB SUPPORTS OBJECT E))
(M30)
**(BUILD MEMBER SUPPORTS CLASS TRANSITIVE)
(M31)
**(DESCRIBE(DEDUCE(4 0)AGENT %X VERB SUPPORTS OBJECT %Y))
(M32(AGENT(B))(VERB(SUPPORTS))(OBJECT(D)))
(M30(MIN(0))(MAX(0))(ARG(M29)))
(M29(AGENT(B)) (VERB(SUPPORTS))(OBJECT(E)))
(M28(AGENT(C))(VERB(SUPPORTS))(OBJECT(D)))
(M27(AGENT(B))(VERB(SUPPORTS))(OBJECT(C)))
(M26(AGENT(A))(VERB(SUPPORTS))(OBJECT(B)))
(DUMPED)
```

Fig. 20. A use of the transitivity rule Fig. 18b asking for four positive answers.

avoided. Processes are presently retained for the duration of the run in which they are created.

The SNePSUL function ADD is identical to FORBTOP, except that if a new node is built, it is matched against the network to find INFER processes for which the node is a new answer. If any are found, the new node is added to the appropriate data collectors from which it is passed to the deductions interested in it. In other words, structures built using ADD will cause forward inferences to be made just in case these structures are relevant to a question asked previously in the session. Figure 22 demonstrates this facility.

4. PARSING AND GENERATING

SNePSUL functions may be activated as actions on the arcs of an augmented transition network (ATN) grammar, so that natural language sentences can be parsed directly into SNePS networks. To facilitate this, the ATN interpreter was modified so that the nodes built while traversing an arc are deleted when back-tracking occurs on the arc.

The ATN interpreter was also modified so that when a SNePS node is

```
**(DESCRIBE (RESUME *LASTINFER))
(M34(AGENT(A))(VERB(SUPPORTS))(OBJECT(D)))
(M33(AGENT(A))(VERB(SUPPORTS))(OBJECT(C)))
(DUMPED)
```

Fig. 21. Resuming the deduction of Fig. 20.

```
**(DESCRIBE (ADD MEMBER SAM CLASS MAN))
(M37(AGENT(SAM))(VERB(LOVES))(OBJECT(M36)))
(M35(MEMBER(SAM))(CLASS(MAN)))
(DUMPED)

**(DESCRIBE (FIND MEMBER M36))
(M38(MEMBER(M36))(CLASS(WOMAN)))
(DUMPED)
```

Fig. 22. A demonstration of restricted forward inference.

given, ATN grammars can construct a natural language string that expresses the concept represented by the node. Space limitations preclude a further discussion of these facilities here. They are described in Shapiro [1975, 1979].

Tying the generator to the nodes is an important mechanism in enforcing the requirement that the set of intensional concepts represented in the network is coextensive with the set of constant nodes. If the system is to express a concept (an idea?), there must be a node representing it to give to the generator. Also, the generator must be able to create a string expressing the concept represented by each constant node.

5. AN EXAMPLE APPLICATION—CLUE

The development of SNePS has been carried out with concern for logical adequacy, for generality, and for the foundations of semantic networks as representations of knowledge. Nevertheless, tests of application domains are important. A SNePS-like collateral descendant of MENTAL [Shapiro, 1971a,b] is being used in SOPHIE, an AI-CAI system (see J. S. Brown *et al.* [1974]), and a version of SNePS is being used in a natural language graphics system [D.C. Brown *et al.*, 1977]. Example domains in medical information and art history are being planned. This section describes an application to the game Clue, implemented by Bill Neagle. This example illustrates the interaction of SNePSUL and LISP, and the use of AND-OR, FORBNOTOP, DEDUCE, and ADD.

Clue is a game of deductive reasoning marketed by Parker Brothers. The game equipment consists of a board representing a house of nine rooms (hall, study, billiard room, library, conservatory, ballroom, kitchen, dining room, lounge), six tokens representing suspects (Miss Scarlet, Professor Plum, Miss Peacock, Mrs. White, Mr. Green, Colonel Mustard), six pieces representing weapons (rope, revolver, candlestick, lead pipe, knife, wrench), one card for each of these 21 items, and an envelope. One room card, one weapon card, and one suspect card are placed in the envelope representing the location, weapon, and perpetrator of a murder. The remaining cards are shuffled and distributed to the players, each of whom uses one of the suspect tokens to move around the board (up to six people

can play). The object of the game is to deduce which cards are in the envelope. Each player knows that the cards he holds are not in the envelope. Players roll dice to move from room to room along corridors. When in a room, a player may "suggest" that the murder was committed in that room by any of the suspects using any of the weapons. Beginning on the suggestor's left, each player either states that he has none of the cards or that he has at least one of them until a player states that he has at least one card. This player shows one of these cards to the suggestor, refuting the suggestion. The other players know only that one of the cards has been shown, not which one. There is no rule preventing the suggestor from including one or more of his own cards in his suggestion, so if every other player denies having any of the three cards, they are either in the envelope or held by the suggestor. The suggestor does not say whether he holds any of the cards. On his turn, a player may make an "accusation," stating which three cards he believes to be in the envelope. He then looks in the envelope. If he is right, he openly displays the cards and wins the game. If he is wrong, he replaces the cards in the envelope and ceases to participate in the game except for replying to other players' suggestions.

We have written a LISP/SNePSUL program that can be used by a player to deduce which cards are in the envelope. The initial game information is established by a call to the LISP function CLUE, which takes as its one argument an ordered list of the players. The following LISP lists are established:

PLAYERS	the ordered list of players
HANDS	the players and the ENVELOPE
SUSPECTS	(SCARLET PLUM . . .)
WEAPONS	(ROPE REVOLVER . . .)
ROOMS	(HALL STUDY . . .)
CARDS	a list of the 21 cards

The set of cards is also established in the SNePS network by evaluating the LISP expression

```
(MAPC CARDS (LAMBDA (CARD)
                    (BUILD MEMBER (↑ CARD) CLASS CARD)))
```

Rules are built expressing such facts as that the envelope holds exactly one suspect, weapon, and room, that each card is held by exactly one hand, and that the kth player (of n players) holds exactly numdealt(player$_k$) cards, where

$$\text{numdealt}(\text{player}_k) = \lfloor 18/n \rfloor + \begin{cases} 1, & \text{if } k \leq 18 \bmod n \\ 0, & \text{otherwise} \end{cases}$$

These rules are built by the LISP function BUILD-BETWEEN, defined as follows:

```
(DE BUILD-BETWEEN (I J SET PRED)
    (BUILD MIN (↑ I) MAX (↑ J)
           ARG (↑ (MAPCAR SET
                          (LAMBDA(X)
                             (APPLY FORBNOTOP
                                    (SUBST X '+ PRED)))))))
```

This function builds a rule saying that between I and J elements, x, of SET are such that PRED(x). Rules of this sort suggest a parameterized existential quantifier, $\exists_j^i$. The rule $\exists_j^i(x)\ {}_2\!\not\!\vee\!\not\!\chi_2^2(A(x), C(x))$ would assert that at least i and at most j objects satisfy both A(x) and C(x). Inclusion of this quantifier in SNePS is planned.

The rules stated above are built as follows:

```
(BUILD-BETWEEN 1 1 SUSPECTS '(HOLDER ENVELOPE OBJECT +))
(BUILD-BETWEEN 1 1 WEAPONS '(HOLDER ENVELOPE OBJECT +))
(BUILD-BETWEEN 1 1 ROOMS '(HOLDER ENVELOPE OBJECT +))
(BUILD AVB $X
       ANT (BUILD MEMBER *X CLASS CARD)
       CQ (↑ (BUILD-BETWEEN 1 1 HANDS
                            '(HOLDER + OBJECT *X))))

(MAPC PLAYERS
      (LAMBDA (PLAYER)
         (BUILD-BETWEEN (NUMDEALT PLAYER)
                        (NUMDEALT PLAYER) CARDS
                        '(HOLDER (↑ PLAYER) OBJECT +))))
```

Note the use of the connectives ${}_n\!\not\!\vee\!\not\!\chi_1^1$ and ${}_n\!\not\!\vee\!\not\!\chi_k^k$. A great many clauses would be required to represent this information in a resolution-based theorem prover.

There are two ways of gaining information during the game. One is entered by the function

(SUGGEST **player suspect weapon room responder card**)

which asserts that **player** suggested that **suspect** committed the murder with the **weapon** in the **room**, that none of the players between **player** and **responder** had any of those cards, and that **responder** showed **player** the card **card**. If **player** or **responder** is the person using the program, **card** will be suspect, **weapon**, or **room**, otherwise it will be NIL. If no one responded, **responder** and **card** will both be NIL. The definition of SUGGEST is

```
(DF SUGGEST (PLAYER SUSPECT WEAPON
                    ROOM RESPONDER CARD)
    (MAPC (BETWEEN PLAYER RESPONDER)
          (LAMBDA (PASSED)
```

```
                (HAS-NONE PASSED SUSPECT WEAPON ROOM)))
    (IF RESPONDER
      (COND (CARD (ADD HOLDER (↑ RESPONDER)
                       OBJECT (↑ CARD)))
            (T (ADD MIN 1 MAX 3
                    ARG((FORBNOTOP HOLDER (↑ RESPONDER)
                                   OBJECT (↑ SUSPECT))
                        (FORBNOTOP HOLDER (↑ RESPONDER)
                                   OBJECT (↑ WEAPON))
                        (FORBNOTOP HOLDER (↑ RESPONDER)
                                   OBJECT (↑ ROOM))))))))
```

where (BETWEEN **player1 player2**) returns a list of the people sitting between (clockwise) **player1** and **player2**, unless **player2** is NIL in which case it returns a list of all players except **player1**. Furthermore, HAS-NONE is the function

```
(DE HAS-NONE (PLAYER SUSPECT WEAPON ROOM)
   (ADD MIN 0 MAX 0
        ARG ((FORBNOTOP HOLDER (↑ PLAYER)
                        OBJECT (↑ SUSPECT))
             (FORBNOTOP HOLDER (↑ PLAYER)
                        OBJECT (↑ WEAPON))
             (FORBNOTOP HOLDER (↑ PLAYER)
                        OBJECT (↑ ROOM)))))
```

The other way in which information may be gained during the game is if someone makes an accusation and is wrong. The accuser may be assumed not to have any of the cards in the accusation. This is handled by a call to HAS-NONE.

Below is the protocol of a reduced game in which the only cards were MUSTARD, PLUM, KNIFE, CANDLESTICK, HALL, LOUNGE, DINING-ROOM, and KITCHEN. The program's trace of its inferences is presented in English for ease of reading. Arrows point to the three statements that provide the solution. The program was being used by player "Don."

```
(CLUE '(DON CHUCK BILL STU))
(BUILD HOLDER DON OBJECT MUSTARD)
(BUILD HOLDER DON OBJECT KNIFE)
(DEDUCE (3 0) HOLDER ENVELOPE OBJECT %X)
Don holds Mustard.
Don holds knife.
Envelope doesn't hold Mustard.
```

→ Envelope holds Plum.
Stu doesn't hold Plum.
Bill doesn't hold Plum.
Chuck doesn't hold Plum.
Don doesn't hold Plum.
Stu doesn't hold Mustard.
Bill doesn't hold Mustard
Chuck doesn't hold Mustard.
Envelope doesn't hold knife.
→ Envelope holds candlestick.
Stu doesn't hold candlestick
Bill doesn't hold candlestick.
Chuck doesn't hold candlestick.
Don doesn't hold candlestick.
Stu doesn't hold knife.
Bill doesn't hold knife.
Chuck doesn't hold knife.
Don doesn't hold hall.
Don doesn't hold lounge.
Don doesn't hold dining room.
Don doesn't hold kitchen.
(SUGGEST STU MUSTARD KNIFE KITCHEN DON KNIFE)
(SUGGEST DON MUSTARD KNIFE DINING-ROOM STU
DINING-ROOM)
Bill doesn't hold dining room.
Chuck doesn't hold dining room.
Stu holds dining room.
Envelope doesn't hold dining room.
Stu doesn't hold hall.
Stu doesn't hold lounge.
Stu doesn't hold kitchen.
(SUGGEST CHUCK PLUM CANDLESTICK LOUNGE BILL NIL)
Bill holds lounge.
Envelope doesn't hold lounge.
Chuck doesn't hold lounge.
Bill doesn't hold hall.
Bill doesn't hold kitchen.
(SUGGEST BILL PLUM KNIFE HALL DON KNIFE)
(SUGGEST STU PLUM CANDLESTICK DINING-ROOM NIL NIL)
(SUGGEST DON MUSTARD KNIFE HALL CHUCK HALL)
Chuck holds hall.
Envelope doesn't hold hall.
→ Envelope holds kitchen.

6. SUMMARY

I have given a brief introduction to semantic networks as representations of knowledge and pointed out that they can be discussed on four levels: an abstract graph; a two-dimensional pictorial diagram; a linear symbolic string; a computer implementation. SNePS, the semantic network processing system, was then discussed on the first three of those levels.

SNePS is a general system for building and manipulating semantic networks but, because of our view of semantic networks, certain features are provided and some restrictions are imposed. Nodes and arcs are partitioned into several types. Auxiliary arcs and nodes are used to type other nodes (e.g., to distinguish variable and pattern nodes), to maintain references to nodes (SNePSUL variables are auxiliary nodes), and to hang nonnodal information onto nodes (AND-OR and THRESH parameters are integers hung on rule nodes by MIN, MAX, and THRESH auxiliary arcs). Pattern and variable nodes do not represent constant concepts but are used to construct deduction rules. Temporary nodes are used by the DEDUCE function to construct templates of desired structures. Universal and existential quantifiers are represented by binding relations between rule nodes and variable nodes. Four nonstandard logical connectives—$\wedge$-entailment, $\vee$-entailment, AND-OR, and THRESH—are provided for building deduction rules that can be used for backward inference or restricted forward inference. The major restriction imposed by SNePS is that nonauxiliary arcs cannot be added between two pre-existing nodes. Each addition of information to the network must be accompanied by a new node. The normal use of BUILD is to have this new node be a molecular node representing the new information.

An application of SNePS to the deductive game Clue was discussed. It illustrated the use of deduction rules, especially the use of AND-OR and restricted forward inference. Further applications are planned along with the continued development of the SNePS system and the SNePSUL language.

ACKNOWLEDGMENTS

SNePS is a descendant of MENTAL [Shapiro, 1971a,b] and is presently implemented in ALISP [Konolige, 1975] on the CDC Cyber 173 at the State University of New York at Buffalo. For aid in the development and implementation of successive versions of SNePS and its associated systems, I am grateful to Nick Vitulli, Nick Eastridge, Jim McKew, Stan Kwasny, Steve Johnson, Ben Spigle, John Lowrance, Darrel Joy, Bob Bechtel, Don McKay, Chuck Arnold, Bill Neagle, and Rich Fritzson.

For their comments on an earlier version of this chapter, I am grateful to Nick Findler, David G. Hays, Don McKay, Chuck Arnold, Rich Fritzson, and three anonymous reviewers, authors of other chapters in this book.

REFERENCES

[Bobrow, D. G., and Raphael, B., 1974]. New programming languages for artificial intelligence. *Computing Surveys* 6, No. 3, 153–174.

[Brown, D. C., Kwasny, S. C., Buttelman, H. W., Chandrasekaran, B., and Sondheimer, N. K., 1977]. NLG—natural language graphics. *Proceedings of The 5th International Joint Conference on Artificial Intelligence, 1977* p. 916.

[Brown, J. S., Burton, R. R., and Bell, A. G., 1974]. "SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting (An Example of AI in CAI)," AI Report No. 12. Bolt, Beranek & Newman, Cambridge, Massachusetts.

[Hendrix, G. G., 1975a]. Expanding the utility of semantic networks through partitioning. *Proceedings of the 4th International Joint Conference on Artificial Intelligence, 1975* pp. 115–121.

[Hendrix, G. G., 1975b]. "Partitioned Networks for the Mathematical Modeling of Natural Language Semantics," Technical Report No. NL-28. Department of Computer Sciences, Univ. of Texas, Austin, Texas.

[Kay, M., 1973]. The MIND system. In *Natural Language Processing*. R. Rustin (ed.). Algorithmics Press, New York, pp. 155–188.

[Konolige, K., 1975]. *ALISP User's Manual*. University of Massachusetts Computing Center, Amherst.

[Quillian, M. R., 1968]. Semantic memory. In *Semantic Information Processing*. M. Minsky (ed.). MIT Press, Cambridge, Massachusetts, pp. 227–270.

[Raphael, B., 1968]. SIR: A computer program for semantic information retrieval. In *Semantic Information Processing*. M. Minsky (ed.). MIT Press, Cambridge, Massachusetts, pp. 33–45.

[Reitman, W. R., 1965]. *Cognition and Thought*. Wiley, New York.

[Schubert, L. K., 1976.]. Extending the expressive power of semantics. *Artificial Intelligence* 7, No. 2, 163–198.

[Scragg, G., 1976]. Semantic nets as memory models. In *Computational Semantics*. E. Charniak and Y. Wilks (eds.). North-Holland Publ. Amsterdam, pp. 101–127.

[Shapiro, S. C., 1971a]. "The MIND System: A Data Structure for Semantic Information Processing," Report R-837-PR. The Rand Corporation, Santa Monica, California.

[Shapiro, S. C., 1971b]. A net structure for semantic information storage, deduction and retrieval. *Proceedings of The 2nd International Joint Conference on Artificial Intelligence, 1971* pp. 512–523.

[Shapiro, S. C., 1975]. Generation as parsing from a network into a linear string. *American Journal of Computational Linguistics* (Microfiche 33), 45–62.

[Shapiro, S. C., 1977]. Representing and locating deduction rules in a semantic network. *ACM SIGART Newsletter* 63, 14–18.

[Shapiro, S. C., 1979]. "A Generalization of Augmented Transition Networks That Allows for Parsing Graphs." Department of Computer Science, SUNY, Buffalo, New York.

[Shapiro, S. C., and McKay, D. P., 1979]. "The Representation and Use of Deduction Rules in a Semantic Network." Department of Computer Science, SUNY, Buffalo, New York.

[Woods, W. A., 1975]. What's in a link: Foundations for semantic networks. In *Representation and Understanding*. D. G. Bobrow and A. M. Collins (eds.). Academic Press, New York, pp. 35–82.