Answering. *Journal of Computers and Information Science 3*, 1974, 225-243.
22 Robinson, J.A. A machine oriented logic based on the resolution principle. *Journal of the ACM 12*, 1965, 25-41.
23 Roussel, P. *PROLOG manuel de reference et d'utilisation.* Groupe D'Intelligence Artificielle, Universite d'Aux-Marseille II, 1975.
24 Tarnlund, S.A. Logic information processing. The Royal Institute of Technology and the University of Sweden, Department of Information Processing Computer Science, Report TRITA-IBADB-1034, 1975.
25 Wilson, G. A description and analysis of the PAR technique - an approach for parallel inference and parallel search to problem solving systems. Ph.D. Thesis, Department of Computer Science, University of Maryland, 1976. Also, Computer Science TR-464, 1976.
26 Wilson, G.A. and Minker, J. Resolution, refinements, and search strategies: A comparative study. *IEEE Trans. on Computers, C-25 9*, 1976, 782-800.

Deductive Inference: Knowledge Representation

Representing and Locating Deduction Rules in a Semantic Network

Stuart C. Shapiro[1]
Department of Computer Science
State University of New York at Buffalo Amherst, NY 14226

Abstract
A semantic network is defined with its arcs and nodes separated into various sets. Arcs are partitioned into descending, ascending, and auxiliary arcs. Nodes are partitioned into base, variable, assertion, pattern and auxiliary nodes. Nodes can be temporary or permanent.

Some pattern and assertion nodes, called rule nodes, represent propositional functions of the nodes they dominate. Rule nodes may bind the variables they dominate with any one of a set of binding relations representing quantifiers. A rule node which dominates variables all of which are bound is a constant deduction rule.

Deduction rules may be viewed as pattern-invoked procedures. The type of propositional function determines the procedure, the variables bound by the rule are the local variables, and the quantifier determines the type of binding.

A binding is defined as a list of variables associated with the nodes they are bound to. A binding can be used like a substitution, except it is seldom actually applied. Instead, a pattern node and a binding for it are used as a pair.

A match routine is defined which is given a source node and a binding and finds target nodes, target bindings and more fully specified source bindings. Target nodes that are patterns provide entrees into relevant rules.

1. Introduction

The first logically adequate proposal for representing quantified deduction rules in semantic networks was made by the author in 1971 [11, 12]. A somewhat different representation, though derived from the same project, was presented by Kay in 1973 [7]. These, and a close variant, were discussed by Woods in 1975 [15]. In 1974-1976, Schubert [8, 9, 10] presented another close variant, apparently developed independently, since he did not compare his work with the others. In 1975, Hendrix [4, 5] presented a representation that adds the notion of network

1. The work reported herein was done, for the most part, while the author was at the Computer Science Department, Indiana University, Bloomington, Indiana 47401.

partitions in a fundamental way and compared his representation with the previous ones [5, pp. 222-4, 266-72].

Sections 2 and 3 of this paper present a more recent syntax for semantic networks, discussed less completely but with more examples elsewhere [13], and another representation for deduction rules, adapted from the earlier one, but designed to accommodate non-standard logics [1, 2, 14] and some of the criticisms of Hendrix. Section 4 discusses how deduction rules may be viewed as pattern-invoked procedures. Sections 5 and 6 present for the first time the matching algorithms used to retrieve information from the semantic network and for identifying relevant rules.

This paper is not concerned with the particular arc relations used in any particular domain of information nor those to be used in a general model of understanding natural language, except as they relate to the representation of deduction rules.

2. Basic Representation

A semantic network is a directed graph with labeled nodes and arcs in which nodes represent concepts and arcs represent non-conceptual binary relations between concepts. The same concept is always represented by the same node and whenever an arc representing a relation, R, points from node n to node m, there is an arc representing the converse relation of R, R^c, from m to n. The labels of nodes and arcs are meaningless symbols, but may be chosen to be mnemonics suggesting the concepts or relations represented.

In SNePS semantic networks [13], we distinguish three kinds of arcs: *descending*, *ascending* and *auxiliary*. For each relation represented by ascending 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 non-nodal information on nodes and for typing the nodes as discussed below. If a descending arc goes from node n to node m, we say that n *immediately dominates* m. If there is a path of descending arcs from node n to node m, we say that n *dominates* m. If R is an arc label and n is a node, we will use the notation R(n) for the set of nodes into which arcs labeled R go from n. In what follows, we will often use the phrase "the relation R" when we mean "an arc labeled R".

There are three kinds of nodes: *constant*, *non-constant*, and *auxiliary*. Auxiliary nodes are connected to each other and to other nodes only by auxiliary arcs. Constant nodes represent unique semantic concepts. Nodes which dominate no other node are called *atomic* nodes. Atomic constants are called *base* nodes and atomic non-constants are called *variable* nodes or *variables*. Variables are distinguished by being in the auxiliary relation :VAR to the auxiliary node T. Non-atomic nodes are called *molecular* nodes. There is a set of descending relations called *binding* relations. A molecular node that immediately dominates one or more variable may have at most one binding relation to an arbitrary number of those 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 arc labeled :SVAR to each of them. If a node m immediately dominates a set of variable nodes $\{v_1, \ldots, v_l\}$ and a set of non-variable nodes $\{n_1, \ldots, n_k\}$ and $V = \{v_1, \ldots, v_l \cup$:SVAR(n_1) $\cup \ldots \cup$:SVAR(n_k) is non-empty, m may have at most one binding relation, say Q, to one or more variables in V. These are referred to as bound by m. The remainder, V - Q(m), are free in m and have the arc :SVAR to each of them from m. It should be the case that no variable bound by a node m is free in any node not dominated by m, and we will assume that this restriction holds. A node n such that :SVAR(n) is non-empty is a non-constant molecular node and is called a *pattern* node. 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 referred to. They are invisible to all the semantic network retrieval operations. We will refer to non-temporary nodes as *permanent* nodes.

In figures in this paper, we show auxiliary arcs as labeled dashed arrows and descending arcs as labeled solid arrows. We do not show ascending arcs. Auxiliary and temporary nodes are shown as labels only, other nodes as labeled circles. In Figure 1, MEMBER and CLASS are descending relations, :VAR and :SVAR are auxiliary relations, MOBY-DICK and WHALE are base nodes, M_1 is an assertion node, T2 is a temporary variable, T3 is a temporary pattern node and T is an auxiliary node.

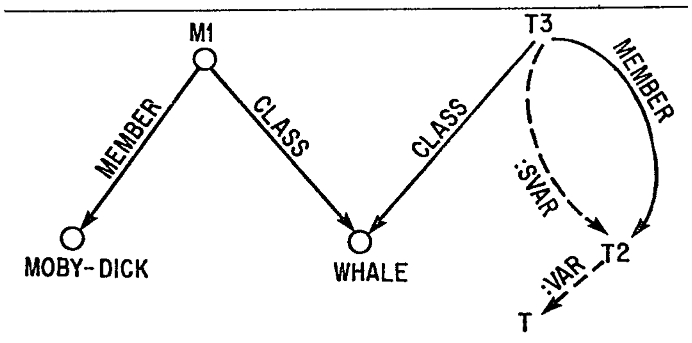

Figure 1. An example of various kinds of nodes and edges

3. Representation of Deduction Rules

To represent deduction rules in SNePS, we distinguish a set of molecular nodes called *rule* nodes. Each rule node represents a propositional formula of molecular nodes that are arguments to a particular propositional connective. Three kinds of connectives are currently used: entailment, AND-OR, and THRESH. An entailment rule has the descending relation ANT (antecedent) to a set of molecular nodes and the descending relation CQ (consequent) to a set of molecular nodes. The interpretaion of an entailment rule n is that each molecule in ANT(n) entails each molecule in CQ(n). Figure 2 shows the network representation of the entailment rule written linearly as $(A_1, \ldots, A_k) \rightarrow (C_1, \ldots, C_l)$.

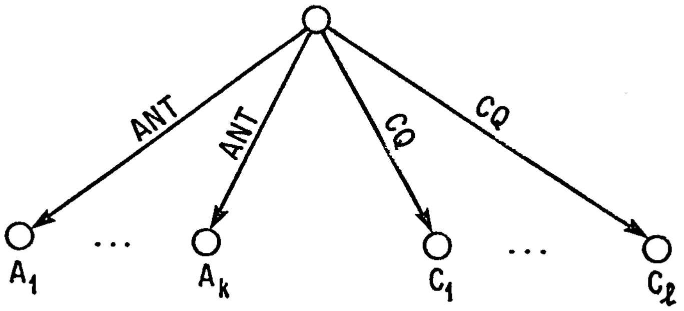

Figure 2. Network representation of $(A_1, \ldots, A_k) \rightarrow (C_1, \ldots, C_\ell)$

An AND-OR rule has auxiliary relations MIN, MAX and TOT to integers i, j and n respectively and the descending relation ARG to a set of n molecular nodes. The interpretation of an AND-OR rule m is that at least i and at most j of the n molecules in ARG(m) are true. Figure 3 shows the network representation of the AND-OR rule written linearly as ${}_n\otimes_i^j(P_1, \ldots, P_n)$.

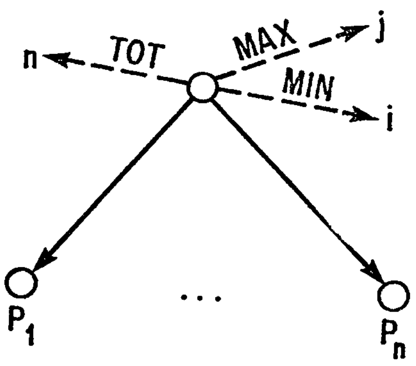

Figure 3. Network representation of ${}_n\otimes_i^j(P_1, \ldots, P_n)$

A THRESH has auxiliary relations THRESH and TOT to integers i and n respectively and the descending relation ARG to a set of n molecular nodes. The interpretation of a THRESH rule is that if at least i of the molecules are true, then all n are true. Figure 4 shows the network representation of the THRESH rule written linearly as ${}_n\theta_i(P_1, \ldots, P_n)$.

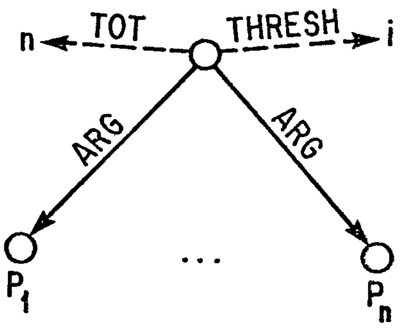

Figure 4: Network representation of ${}_n\theta_i(P_1, \ldots, P_n)$

One could consider AND-OR rules with different <MIN, MAX, TOT> triples to be different types of propositional formulas, and likewise THRESH rules with different <THRESH, TOT> tuples. For example, AND-ORs of the type ${}_n\otimes^n_1$ represent the disjunction of the n molecules, AND-ORs of the type ${}_1\otimes^0_0$ represent negation and THRESHs of the type ${}_n\theta_1$ represent the mutual equivalence of the n molecules. More complete discussions of the logic of these rules may be found elsewhere [1, 2, 14]. As an example, Figure 5 shows a possible representation of "John is either at home, at the airport, or at the office".

Rule nodes are the only molecular nodes allowed to have binding relations. At the current time, AVB is used for universal quantification, EVB for existential quantification, OVB for unique existential, LVB for "almost all" (almost-all(x) (P(x)→Q(x)) means that if a is such that P(a) holds and such that Q(a) cannot be shown, deduce that Q(a) holds), and NVB for "none" (None(x)(P(x)→Q(x)) means that if P(a) holds, Q(a) doesn't). A rule node, R, for which :SVAR(R) is empty is a constant deduction rule. Figure 6 shows a representation for "Whales live in water and each has a blowhole".

4. Deduction Rules as Pattern-Invoked Procedures

The general form of a deduction rule is $Q(x_1, \ldots, x_n)F(P_1, \ldots, P_k)$ where Q is a quantifier, $x_1, \ldots, x_n$ are variables bound by the rule node, F is the propositional function represented by the rule node, and $P_1, \ldots, P_k$ are the molecular node

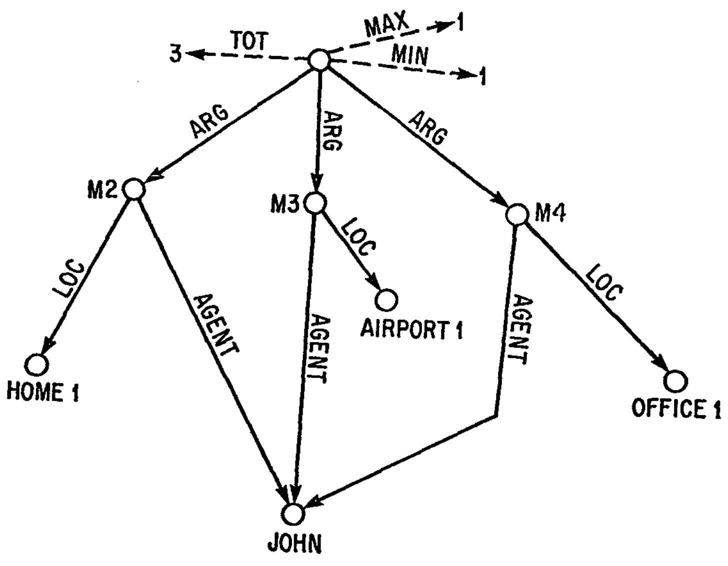

Figure 5. "John is either at home, at the airport, or at the office"

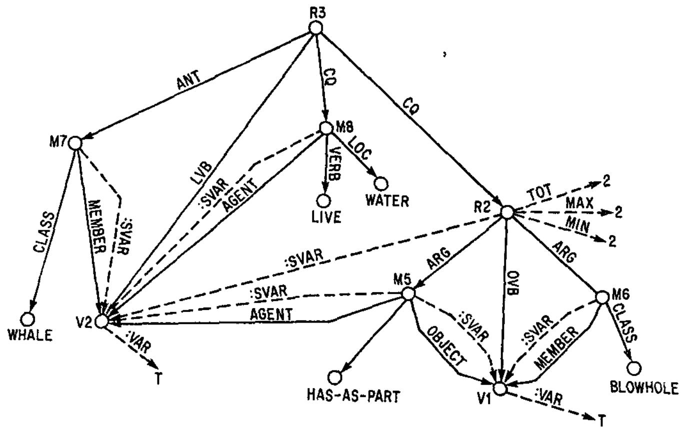

Figure 6. "Whales live in water and each has a blowhole"

propositions immediately dominated by the rule node. The function F may be viewed as a procedure type that determines how the P_j are to be processed to derive new information. For example, if F is →, the P_j are partitioned into antecedents and consequents and any consequent may be deduced if any antecedent is shown to hold; if F is ${}_n\partial_i$ then any i of the P_j that are found to hold are considered to be antecedents and are adequate grounds for deducing any of the remaining m-i as consequents. Actually, when we say, "if the antecedents are shown then a consequent may be deduced", we should say, "if the antecedents are shown in a given binding of the variables x_1, . . ., x_n, then a consequent may be deduced in that binding", for the variables x_1, . . ., x_n act as local variables in the rule-procedure. The quantifier determines how the variable binding is managed. For example, if Q is AVB any binding is allowed. If Q is LVB, once the antecedent(s) is shown to hold in a given binding, that binding is tested in the negation of the consequent. If that is not found to hold, the consequent is deduced in that binding. If Q is EVB, the variables must be bound to new base nodes (Skolem functions). A binding of the variables is similar to, but not quite the same as a substitution to be applied to F(P_1, . . ., P_n) in resolution terminology (see [3]). This point will be clarified in the following section.

Any rule may be used either in the forward direction, like a PLANNER [6] antecedent theorem, or in the backward direction, like a PLANNER consequent theorem. In the backward direction, if a substitution instance of a consequent is to be deduced, an attempt is made to deduce all the antecedents in the same substitution. As soon as the appropriate number of them are found (1 in the case of →, i in the case of ${}_n\partial_i$, n-i to be false in the case of ${}_n\otimes_i^j$), the consequent is deduced. In the forward direction, when an antecedent of an entailment is asserted, the consequents may be asserted. When an antecedent of AND-OR or THRESH is asserted, the appropriate number of other arguments must be deduced. Then the remainder of the arguments may be asserted as consequents.

A potentially useful rule is discovered when either a molecular node to be used to drive a forward inference or a temporary molecular node representing a proposition to be deduced matches a molecular node dominated by the rule. The path of arcs between the rule node and the matched node must be appropriate to the intended use, i.e. forward vs. backward inferencing. Matching is done first and only appropriate paths of ascending arcs are followed to find the potentially useful rules. Such a rule is only potentially useful since not enough antecedents may hold for it to be truly useful.

The appropriate path for backward inferencing is one of CQ^c or ARG^c arcs. The appropriate path for forward inferencing is one containing ANT^c, ARG^c or CQ^c, as long as at least one ANT^c or ARG^c occurs. To see this, note that in the rule (A→B) → (C→D), the assertion of B or of C might allow something new to be deduced. (The assertion of a node matching B might allow A→B to be proved, and if C were true, D would then be derivable). The proper bindings are found by the matching operation, and may be filled in as inferencing proceeds. Bindings and the matching operation are discussed more completely in the next two sections.

5. *Bindings*

We write a binding as a list of pairs in square brackets, $[v_1/t_1, \ldots, v_k/t_k]$, where the v_i are variable nodes and the t_i are nodes. This is the reverse order from the usual notation for substitutions, but the normal order for association or binding lists. We call each v_i/t_i a *pair* and say that v_i is the *variable of the pair* and t_i is the *term of the pair*. We will say that a variable, v, is *in a binding* β(or v∈β) if v is the variable of some pair in β. So v1 ∈ [v1/v2], but v2 ∉ [v1/v2]. If t is the term of some pair in a binding β, we will say that t is a *term of* β. Two differences between bindings and substitutions are that the pair v/v is allowed in a binding, and if p is a pattern node with :SVAR to v1 the binding [v/p, v1/t] is used to mean that v is bound to an instance of p in which v1 is bound to t. In fact, whenever a pattern, p, is a term of a binding, β, all variables in :SVAR(p) will also be in β. This allows for the use of the tuple <N,β> instead of actually creating a substitution instance of the network structure dominated by N. In what follows, we will write N/β instead of <N,β>.

To *apply* a binding β to a node N means the following:

1) If N is the variable of a pair N/v in β and v is a variable, return v.
2) Else if N is the variable of a pair N/t in β, return the application of β to t.
3) Else if N is an atomic node, return N.
4) Else return a node N' such that for every descending relation R and for every node M ∈ R(N), N' has R to the application of β to M. According to a parameter of the apply function, N' may be a temporary or permanent node. If permanent, and a node satisfying the description already exists, it will be used, otherwise a new node will be created.

It should be remembered that this operation is seldom carried out, and when it is, nodes created in step 4) may be temporary nodes rather than permanent nodes.

To apply a binding β to the terms of another binding α (written $\alpha \backslash \beta$) does not involve the above operation, but only means replacing each pair v1/v2 in α such that there is a pair v2/t in β by the pair v1/t.

We define the *union* of two bindings, $\beta \cup \alpha$, as the binding containing all the pairs in β plus every pair in α whose variable is not in β. For example, [v1/v2, v3/t1] $\cup$ [v2/t2, v3/t3] = [v1/v2, v2/t2, v3/t1]. Note that this union is not commutative.

The *composition* of two bindings can now be defined as $\beta \circ \alpha = (\beta \backslash \alpha) \cup \alpha$. It can be seen that binding composition is associative, and that if N is a node and β and α are bindings, $(N/\beta)\alpha$ is "equivalent" to $N(\beta \circ \alpha)$ but not necessarily identical, since if the two applications are carried out, different nodes may be created in step 4) of the "apply a binding to a node" routine.

I will stand for the identity binding over application and union (and therefore composition). If p is a pattern node, I(p) is defined as the binding consisting of pairs v/v for every $v \in$:SVAR(p). This is a right-identity over application. If n is a node with no :SVAR arcs, we define I(n) to be I.

6. *Match*

In this section, V stands for the set of permanent variable nodes, C for the set of permanent constant nodes, and P for the set of permanent pattern nodes.

The arguments of match(S, β) are a node S, which could be either temporary or permanent, and a binding β. Match returns a set of tuples, $\{<T,\tau,\sigma>\}$, such that for some bindings β_1, β_2, $\sigma = (\beta \backslash \beta_1) \circ \beta_2$, and such that for every descending relation R, every atomic node in $R(S\sigma)$ is also in $R(T\tau)$ and for every molecular node in $R(S\sigma)$ there is an equivalent node in $R(T\tau)$. T is referred to as the *target node*, τ as the *target binding* and σ as the *source binding*. Match (S, β) is defined recursively as follows:

1) If S is a base node,
   $\text{match}(S,\beta) = \{<S, I, \beta>\} \cup \{<v, [v/S], \beta> \mid v \in V\}$
2) If S is a variable node and $S\beta = S$,
   $\text{match}(S,\beta) = \{<c, I, \beta\circ[S/c]> \mid c \in C\}$
   $\cup \{<v, [v/S], \beta\circ[S/v]> \mid v \in V\}$
   $\cup \{<p, I(p), \beta\circ[S/p]> \mid p \in P\}$
3) If S is a variable node and $S\beta$ is a variable other than S,
   $\text{match}(S,\beta) =$
   $\{<T, \tau\backslash[S\beta/S], \beta\circ\sigma> \mid <T,\tau,\sigma> \in \text{match}(S\beta, I)\}$
4) If S is a variable node and $S\beta$ is not,
   $\text{match}(S,\beta) = \{<T,\tau, \beta\circ\sigma> \mid <T,\tau,\sigma> \in \text{match}(S\beta,\beta)\}$
5) If S is a molecular node, then let $S_1, \ldots, S_k$ be the set of nodes immediately dominated by S and R_i, $1 \le i \le k$, be the descending relation from S to S_i. The set of potential match sets, $M_1, \ldots, M_k$, is built as follows.
   a) For each $<T,\tau,\sigma>$ in match(S, β) and for each N in $R^c{}_1(T)$, put $<N,\tau,\sigma>$ in M_1.
   b) For each $<N,\tau,\sigma>$ in M_{i-1} and for each $<T, \tau_i, \sigma_i>$ in match(S_i, β) such that $N \in R^c{}_i(T)$ do set $\tau' \leftarrow \tau$ and $\sigma' \leftarrow \sigma$

```
      and FLAG to T
       for each v/t in $\tau_i$
         if v $\notin \tau'$, add v/t to $\tau'$
         else if v/t' $\in \tau'$ and t $\neq$ t' and t is a variable not
          in :SVAR(t')
           apply [t/t'] to the terms of $\tau'$ and $\tau_i$
           apply [v/t'] to the terms of $\sigma'$ and $\sigma_i$
         else if v/t' $\in \tau'$ and t $\neq$ t' and t' is a variable not
          in :SVAR(T)
           apply [t'/t] to the terms of $\tau'$ and $\tau_i$
           apply [v/t] to the terms of $\sigma'$ and $\sigma_i$
         else if t $\neq$ t' set FLAG to F
       for each v/t in $\sigma_i$
         if v $\notin \sigma'$, add v/t to $\sigma'$
         else if v/t' $\in \tau'$ and t $\neq$ t' and t is a variable not
          in :SVAR(T')
           apply [t/t'] to the terms of $\sigma'$ and $\sigma_i$
           apply [v/t'] to the terms of $\tau'$
         else if v/t' $\in \tau'$ and t $\neq$ t' and t' is a variable not
          in :SVAR(t)
           apply [t'/t] to the terms of $\sigma'$ and $\sigma_i$
           apply [v/t] to the terms of $\tau'$
         else if t $\neq$ t' set FLAG to F
      if FLAG = T add <N, $\tau'$, $\sigma'$> to $M_i$
   match(S, $\beta$) = $M_k$
```

When $S\beta$ is a question (a proposition to be deduced), each $T\tau$ such that $<T,\tau,\sigma> \in$ match(S, β) will either be an assertion which provides the answer or an entree to a potentially useful rule. In the latter case, if the rule works, the answer will be $S\sigma$, which is possibly more specific and detailed than the original question, $S\beta$.

The way match is currently defined, the target node may immediately dominate more nodes than the source node. This seems appropriate when looking for rules to apply in a consequent fashion, but inappropriate when looking for rules to apply in an antecedent fashion. As we have not yet implemented the latter, we will not discuss this issue in detail. Step (5b) of match could be modified to allow target nodes with fewer immediately dominated nodes by allowing into M_i those $<N,\tau,\sigma>$ in M_{i-1} for which there is no $<T,\tau,\sigma>$ in match(S_i, β) for which $N \in R^c{}_i(T)$ and by putting into M_i triples of the form $<N,\tau,\sigma>$ whenever $<T,\tau,\sigma> \in$ match(S_i, β), $N \in R^c{}_i(T)$ and there is no triple $<N,\tau',\sigma'>$ already in M_{i-1}.

7. *Implementation Status*

The semantic network system, match routines, a multiprocessing system for executing the rules and processes for doing backward inferencing with entailment, THRESH and universal quantification have been implemented in LISP1.6 on a DEC System-10. Deductions have been carried out on several small domains. At the time of writing, further development is in progress and the system is also being brought up in U.T. LISP1.5 on a CYBER 173.

8. *Acknowledgements*

The author wishes to thank Bob Bechtel, Nick Eastridge, Steve Johnson, Don McKay, Jim McKew, and Ben Spigle for their work on the theory and implementation of the deduction system. Computer Services were provided by the IUPUI Computing Facilities as part of the I.U. Computing Network.

References

1 Bechtel, R.J. Logic for semantic networks. M.S. Thesis. Technical Report No. 53, Computer Science Department, Indiana University, 1976.

2 Bechtel, R.J. and Shapiro, S.C. A logic for semantic networks. Technical Report No. 47, Computer Science Department, Indiana University, 1976.

3 Chang, C.L. and Lee, R.C.T. *Symbolic Logic and Mechanical Theorem Proving*. Academic Press, New York, 1973, Section 5.3.

4 Hendrix, G.G. Expanding the utility of semantic networks through partitioning. *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, 1975, 115-121.

5 Hendrix, G.G. Partitioned networks for the mathematical modeling of natural language semantics. Ph.D. Thesis. Technical Report NL-28, Department of Computer Sciences, The University of Texas at Austin, 1975.

6 Hewitt, C. Description and theoretical analysis (using schemata) of PLANNER: a language for proving theorems and manipulating models in a robot. AI-TR-258. MIT Artificial Intelligence Laboratory, 1972.

7 Kay, M. The MIND System. In *Natural Language Processing*, Austin, R. (Ed.). Algorithmics Press, New York, 1973, 155-188.
8 Schubert, L.K. Extending the expressive power of semantic networks. TR 74-18, Department of Computer Science, University of Alberta, 1974.
9 Schubert, L.K. Extending the expressive power of semantic networks. *Advance Papers of the Fourth International Joint Conference on Artificial Intelligence*, 1975, 158-164.
10 Schubert, L.K. Extending the expressive power of semantic networks. *Artificial Intelligence 7*, 1976, 163-198.
11 Shapiro, S.C. The MIND system: a data structure for semantic information processing. R-837-PR, The Rand Corporation, Santa Monica, 1971.
12 Shapiro S.C. A net structure for semantic information storage, deduction and retrieval. *Proceedings Second International Joint Conference on Artificial Intelligence*, The British Computer Society, London, 1971, 512-523.
13 Shapiro, S.C. An introduction to SNePS (semantic net processing system). Technical Report No. 31, Computer Science Department, Indiana University, Bloomington, Revised December, 1976.
14 Shapiro, S.C. and Bechtel, R.J. Non-standard connectives and quantifiers for question-answering systems. In progress.
15 Woods, W.A. What's in a link: foundations for semantic networks. In *Representation and Understanding*, Bobrow, D.G. and Collins, A. (Eds.). Academic Press, New York, 1975, 35-82.

Semantic Network Representations in Rule-Based Inference Systems

Richard O. Duda, Peter E. Hart, Nils J. Nilsson, and Georgia L. Sutherland
Stanford Research Institute Menlo Park, CA 94025

Rule-based inference systems allow judgmental knowledge about a specific problem domain to be represented as a collection of discrete rules. Each rule states that if certain premises are known, then certain conclusions can be inferred. An important design issue concerns the representational form for the premises and conclusions of the rules. We describe a rule-based system that uses a partitioned semantic network representation for the premises and conclusions.

Knowledge-Directed Inference in BELIEVER

N. S. Sridharan and C. F. Schmidt
Department of Computer Science
Rutgers University New Brunswick, NJ 08903

The BELIEVER theory is an attempt to specify an information processing system that constructs intentional interpretations of an observed sequence of human actions. A frame-based system, AIMDS, is used to define three domains: the physical world; the plan domain, where interpretations are constructed using plan structures composed from plan units; and the psychological description of the actor. The system achieves a shift of representation from propositions about physical events to statements about beliefs and intentions of the actor by hypothesizing and attributing a plan structure to the actor.

A paradigm for approaching a part of the interpretation problem is described in this report. Understanding is viewed as a process of assimilating incoming patterns with existing knowledge and expectations. The essential process of "expectation matching" is attended to in detail and a simple example is presented to illustrate the paradigm and its possible extensions.

A Knowledge Base Organization for Rules About Programming

David Barstow[1]
Stanford University Stanford, CA 94305

Abstract

PECOS is a knowledge-based system for automatic program synthesis. Programs are specified as abstract algorithms in a high-level language for symbolic computation. Through the successive application of programming rules, the specification is gradually refined into a concrete implementation in the target language. The existence of several rules for the same task permits the construction of a variety of distinct programs from a single initial specification. Internally, program descriptions are represented as collections of nodes, each labeled with a programming concept and with other properties related to that concept. The refinement process is guided by the selection and application of rules about programming. These rules are stated as condition-action pairs, but the identification of certain rule types permits the use of various techniques for efficient rule retrieval and testing, including the determination of retrieval patterns and the automatic separation of the condition into an applicability pattern and a binding pattern.

Introduction

PECOS is a knowledge-based system that constructs concrete implementations of abstract algorithms [1]. For current experiments the specification language centers around notions from symbolic programming, including information structures such as collections or correspondences, and operations such as testing whether an item is in a collection or computing the inverse of a correspondence. Programs are synthesized by gradually refining the original specification into a program in the target language. Currently the target language is LISP (in particular, a subset of INTERLISP [10]), but experimentation with SAIL (an ALGOL-like language) is underway [8]. From a given specification, PECOS is able to construct several different implementations, differing both in representations for data structures and in algorithms for abstract operations.

PECOS's abilities are derived from a large knowledge base of rules about programming. These rules have been carefully designed and constructed to deal explicitly with various aspects of the programming process, including intermediate-level constructs and certain design decisions. In previous experiments, such rules have been used to synthesize several simple sorting programs [5,6]. Detailed discussions of all of PECOS's rules may be found elsewhere [1]. The current discussion focuses on the organization of the knowledge base and the techniques used to retrieve and apply its rules.

Rules about Programming

The rules in PECOS's knowledge base constitute an explication of knowledge about writing programs in the domain of symbolic computation. While many of the rules are relatively specific to the task of writing simple symbolic programs, some are generally applicable to programming in other domains as well. Most are independent of any particular programming language, although some are quite specific to LISP. A representative sample is given below. (The rules are presented in English for ease of understanding; details of the internal representation are discussed later.)

1. This research was supported by the Advanced Research Projects Agency of the Department of Defense under Contract MDA 903-76-C-0206. The views and conclusions contained in this document are those of the author and should not be interpreted as necessarily representing the official policies, either expressed or implied, of Stanford University, ARPA, or the U. S. Government.