THE USE OF A SEMANTIC NETWORK IN A DEDUCTIVE QUESTION-ANSWERING SYSTEM

James R. McSkimin
Bell Telephone Laboratories
Columbus, Ohio
43209

.Jack Minker
Department of Computer Science
University of Maryland
College Park, Maryland 20742

Abstract

The use of a semantic network to aid in the deductive search process of a Question-Answering System is described. The semantic network is based on an adaptation of the predicate calculus. It makes available user-supplied, domain-dependent info mat ion so as to permit semantic data to be used during the search process.

Three ways are discussed in which semantic information may be used. These are:
(a) To apply semantic information during the pattern-matching process.
(b) To apply semantic well-formedness tests to query and data inputs.
(c) To determine when subproblems are fully-solved (i.e., they have no solutions other than a fixed, finite number).

An example is provided which illustrates the use of a semantic network to perform each of the above functions.

1. Introduction

Semantic Networks have been used primarily in natural language applications to help disambiguate sentences and to understand natural language text. In this paper we consider the use of a semantic network to aid in the deductive search process of a Question-Answering (QA System. The semantic network is based on an adaptation of the predicate calculus and is described only briefly in this paper and more extensively by McSkimin and Minker (McSkimin [19761, and McSkimin and Minker [1977]*)*. Terminology from the predicate calculus will be used throughout the paper.

Three ways will be discussed in which semantic information may be applied to help restrict deductive searches. These are:
(1) To apply semantic information during the pattern-matching process (unification algorithin). Most current pat tern-matching systems are based solely on syntactic tests. Using the semantic network, semantic constraints may be applied during the pattern-matching process to inhibit data base assertions and general axioms that are semantical!}' irrelevant to the search from entering into the deductive search space.
(2J To apply semantic well-formedness tests to query and data base assertions input to the system so as to reject queries that have no answer because they violate semantic restrictions. Thus, it should be possible to determine that a query such as "Who is the person who is both father and the mother of a given individual?[n] is not answerable.
(3) To identify those queries which have a known maximum number of solutions so as to terminate searches for additional answers once the known fixed number is found.

The semantic network discussed in this paper is described briefly in Section 2. In Section 3, we describe how the semantic network may be used to solve some of the problems associated with the above items. An example which illustrates the use of the semantic network is presented in Section 4. A summary of the work and future directions is given in Section 5.

2. Seniantjc Network

Although the term 'semantic network' has been used extensively in the literature, there is no universal agreement as to what constitutes such a network. Hence, we shall define the term in the context of this paper.

The semantic network to be described arose out of the need to provide meaning to objects in a domain and to statements made about these objects so as to make deductive searches more efficient. Although the semantic network developed is used in deductive searches, it nevertheless bears considerable relationship to those developed through the need to understand natural language by computers. The semantic network developed by Schubert [1976], for example, bears many similarities to the one used here.

The semantic network described here is an adaptation of the predicate calculus and is able to express quantification, functions, terms and logical connectives. The adaptation is based upon the notation of Fisimian and Minker (Fishman [1973], Fishman and Minker [1975]), who modified predicate calculus clause notation to handle sets of objects that have the same template structure.

Thje article by Schubert [1976] discusses many semantic network representations used for natural language processing, and surveys the literature so that we neither refer to nor compare our work on semantic networks with that achieved by others.

In order to implement the techniques described in the introduction, domain-dependent information must be stored in the computer in a form convenient for use. Coasequently, a major

part of this research has concerned the identification of the types of semantic information to be stored, and the development of structures in which to store the information. To this end, a collection of structures termed the "semantic network" has been developed which contains all information available to the question-answering system.

The semantic, network consists of four components: (1) the semantic graph which specifies the set-theoretic relation between named subsets of the domain; (2) the data base of assertions and inference rules; (37 the semantic form space which defines the semantic constraints placed on arguments of relational n-tuples; and (4) the dictionary which defines the set membership for each element of the domain. All four components of the semantic network are used by the techniques described above for making the QA process more efficient. Illustrations of how this information is used will be given in the next section.

(a} The Semantic Graph

The major emphasis of this effort is the investigation of techniques by which user-supplied semantic information may be stored in a computer and used to make the deductive inference process more efficient. The approach taken is to define explicitly the' contents of the domain of discourse D as well as the relationships in which various subsets of the domain may occur.

To this end, much of the work has involved the investigation of how one might subdivide the domain D into a finite number of named subsets Sj such that all elements of each S^ have some set of properties in common. These sets are expressed as Boolean Category Expressions (BCE). Examples are: *senatord male -liberal, state, judge f] lawyer.* The names "senator", "state" and "judge" are examples of what are defined to be the simplest type of BCE possible and are called semantic categories. A BCE is any arbitrary combination of categories using the set operations of union (II), intersection (n) and complement (-). It is necessary to subdivide the domain D into subsets since certain relational statements may only be made about specified subsets of D, and one would like to make these subsets explicit rather than implicit. This subdivision is specified by a semantic graph G_s which defines how each category C is subdivided into subsets $C_1, C2, \ldots, C_n$ and how each of the Cj is similarly defined. Figure 1 shows an example of such a graph. Note that both *animate* and *living* are the superset of *animal;* however *animate* is the superset of *robot* which is disjoint from *Living,* and *living* is the superset of *plant* which is disjoint from *animate.* Thus, *animate* and *living* overlap.

Subdividing D in this manner and defining where in the hierarchy each domain element lives (the function of the dictionary), has several advantages over expressing set memberships by unary relations. In particular, it should be computationally more efficient to perform trivial set membership inference using such a structure rather than by using unary relations. Thus, Sirica £ *judge* might be stored in the dictionary rather than storing the unit clause JUDGE(Sirica) in the data base. The rationale for this choice is given in McSkimin [1976].

(b) Data Base

Assertions are facts, whereas general axioms are used to infer assertions about domain elements that are otherwise stored implicitly in the data base. Both types are stored in a "parallel clause" notation, termed n-o notation, an extension of the n-representation of Fishman and Minker. An example of an assertion in n-o notation is: ((a,x,y), {{ [PARENT]/a,[Ruth,Herb]/x, lAnne,Carol,Jim]/y}}). The assertion states that Ruth and Herb are the parents of Anne, Carol and Jim. An example of a general axiom is: *(^(a*,x, y.) v (3, x, y), {{ [RESIDE]*/a*, congressperson/x,state/y,[REPRESENT]/B)3). The axiom states that for all «, 3, x, and y, if the object x is in the set congressperson, and the object a is the predicate" RESIDE, and x resides in

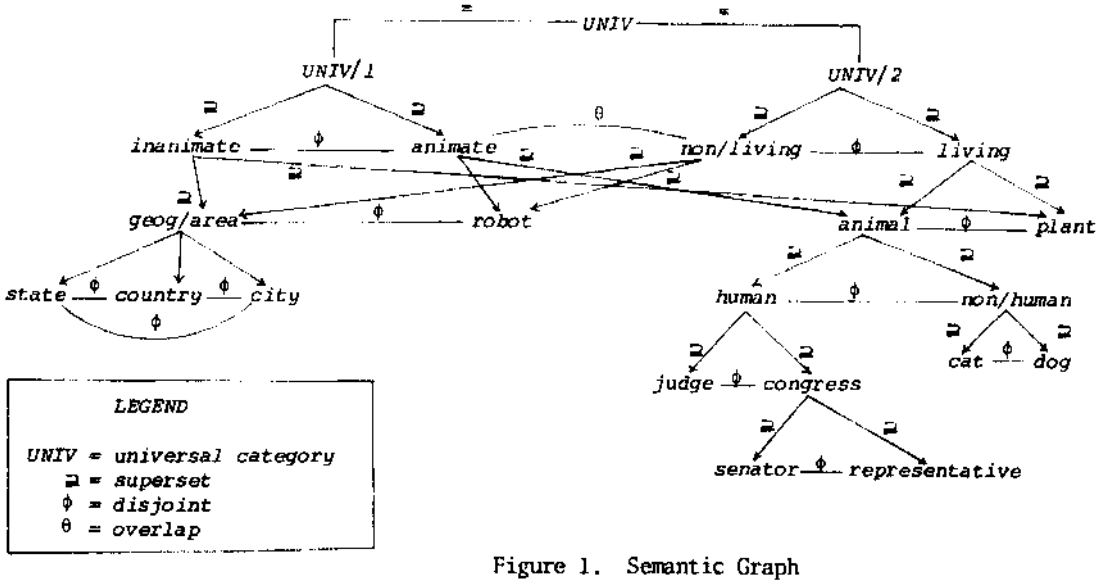

Figure 1. Semantic Graph

the set given by the state y, then the object x represents the state y. The first form represents 6 clauses in first-order predicate calculus. The second clause restricts the variable x to the set congressperson, while y is restricted to the set state.

A Π-σ clause, in more specific terms, is an ordered pair, $\mathbb{T} = (T,\Phi)$ where T is a template which is like a first order predicate calculus clause, where T contains terms which are free of constants and predicates. Literals of the template T are (n+1) ary tuples where the first tuple entry denotes the predicate name. Φ is a finite set of extended Π-substitution sets, $\Phi = \{\varphi_1,\ldots,\varphi_n\}$, where $\varphi \in \Phi$ is the form $\varphi = [S_1/v_1, \ldots, S_m/v_m]$, and where a Π-substitution component S/v is an expression of the form $S/v = ([D^E,D^X], D^I/v)$. D^E is a possibly empty finite set of constants explicitly included in S; D^X is a possibly empty finite set of constants explicitly excluded from S; D^I is a possibly empty set of constants represented as a BCE and implicitly included in S; v is termed a placeholder (or a variable). The following conditions hold: (1) $D^E \cap D^I = \phi$; (2) $D^X \subseteq D^I$.

A Π-substitution component indicates that v is universally quantified over the set $D^E \cup (D^I \cap -D^X)$. A Π-σ literal is an ordered pair, (L,Φ), where L is a literal template (i.e., free of constants and predicates), and Φ is represented as an (n+1) ary tuple.

(c) Dictionary

For each predicate, constant and function name in the system, the dictionary lists the semantic category, of which it is a member. In addition, it defines the set of categories that are subsets of each category in the semantic graph G_s.

(d) Semantic Form Space

The semantic form space defines semantic constraints that are to be imposed on each argument of a predicate. Each semantic form defines what predicates, constants, or functions are legal to substitute into arguments of some subset S_i of the possible set of n-tuples in the system. Not only are possible argument types stored, but so are possible combinations of types. An example of a semantic form is: $((\alpha,x,y), \{[\text{REPRESENT}]/\alpha, \underline{\text{senator}}/x, \underline{\text{state}}/y\}, \{[\text{REPRESENT}]/\alpha, \underline{\text{representative}}/x, \underline{\text{state}}/y\})$. That is, senators and representatives can represent a state - no other BCE combinations are permitted. Additional information is retained with a semantic form to count solutions possible for a given predicate. This is described in Section 3.3.1. The above constitutes the semantic network.

3. Use of the Semantic Network in Deductive QA Searches

The development of the semantic network is intended to restrict the search for solutions to questions when a deductive search is required. There are three ways in which the search can be restricted. These are: (a) to semantically unify clauses; (b) to perform semantic well-formedness tests; and (c) to detect fully solved problems (i.e., problems which, because of the knowledge of the domain of application, have a known maximum number of solutions). Each of these three topics is discussed below.

3.1 Semantic Unification

Insisting that variables be restricted to specific domains, limits the domain of a variable more than is normally possible in the predicate calculus. In the first-order predicate calculus, all variables are universally quantified; because no restriction on the variables is possible, in order to unify two clauses, one must therefore assume that when one variable is to be substituted for another, that they belong to consistent domains. However, if variables could be restricted explicitly to disjoint domains, then no possible substitution of one variable for the other can be made, and unification must therefore fail because of semantic considerations.

The details of a Π-σ semantic unification are too cumbersome to describe in this paper; they are described by McSkimin. We shall provide a brief summary of the tests needed to effect Π-σ semantic unification. Two Π-σ literals must unify syntactically to be considered for semantic unification. When two variables are to be substituted for one another, an algorithm, called the CONFLICT algorithm, determines if the BCEs of the two variables overlap. If they do, a new variable whose BCE is the intersection of the two BCEs replaces each variable. If the two BCEs are disjoint then the two variables are semantically inconsistent since there exists no domain element that can be substituted for both. If a constant is to replace a variable with BCE B_1, the constant must belong to some semantic category B_2 such that $B_2 \subseteq B_1$. If a function is to be substituted for a variable, the BCE of the range of the function must be a subset of the BCE of the variable. In all of these areas the semantic graph is used to determine the relation between the two BCEs.

For example, consider the following Π-σ clauses:

(1) $(\sim(\alpha,x,y), \{\{[\text{Jack}]/x, [P]/\alpha, \underline{\text{human}}/y\}\})$

(2) $(\sim(\beta_1,u_1,v_1) \vee (\alpha_1,u_1,v_1), \{\{\underline{\text{male}}/u_1, \underline{\text{human}}/v_1, [F]/\beta_1, [P]/\alpha_1\}\})$

(3) $(\sim(\beta_2,u_2,v_2) \vee (\alpha_2,u_2,v_2), \{\{\underline{\text{female}}/u_2, \underline{\text{human}}/v_2, [M]/\beta_2, [P]/\alpha_2\}\})$

where male, female, and human are semantic categories, Jack $\in$ male, and male $\cap$ female $= \phi$. In a first-order predicate calculus system, there is only one universal semantic category. Thus, human, female, and male do not appear and the variables are universally quantified over the entire domain. In this case, two resolvents would result when resolving clause (1), with clause (2) and clause (3). However, with Π-σ semantic unification, there is only one resolvent that is obtained from clause (1) and clause (2). The Π-σ literal, in clause (3), $((\alpha_2,u_2,v_2), \{\{[P]/\alpha_2, \underline{\text{female}}/u_2, \underline{\text{human}}/v_2\}\})$ cannot unify with $((\alpha,x,y), \{\{[P]/\alpha, [\text{Jack}]/x, \underline{\text{human}}/y\}\})$. This is so since a conflict exists as Jack $\notin$ female. The CONFLICT routine would find that Jack $\in$ male by accessing the dictionary entry for Jack. The semantic graph would denote that male $\cap$ female is empty,

and the two ll-a literals would be prevented from unifying. Thus, clause (3) would never be entered into the search space, so that it would not lead to a deductive search path, thereby decreasing the time and space used over that of a purely syntactic pattern match.

Semantic unification is applied during the deductive search process. It is also applied when one is entering new facts or general rules into the system, and when a query is entered. These are described in the following sections.

5.2 Semantic Well-Formedness of n-o Clauses

One way in which the n-o unification algorithm may be used is to perform semantic well-formedness tests on n-o clauses input to a question-answering system, n-o clauses are used in two different ways: both as assertions and general axioms to be stored in the data base, and as questions posed to the system. As noted previously, the data base comprises one part of the semantic network. The data base stores facts about members of the domain (i.e., assertions) and provides general axioms that are used to deduce implicit assertions from those already known. Both of these are stored as IT-a clauses.

The semantic form space defines how domain elements may interact; that is, in what combinations they may co-occur in n-o clauses of the system. In particular, semantic forms define the subsets of the domain *UNIV* from which arguments of relations must be chosen. It is necessary that all data base clauses conform to these rules. If data base clauses were stored that did not conform with these rules, incorrect deductions could be generated.

In addition, when a query is posed to a QA system, it is useful to know if it is potentially unanswerable. A query may be unanswerable for several reasons. If a query makes reference to domain elements that, are unknown to the system, then it is very unlikely that the questions about that object could be answered.

A query could also be unanswerable because it violates the constraints specified by the semantic form space. Thus, if one asked whether President Carter voted for a certain congressional bill, the system should answer that the question was not-well-formed since only members of Congress may vote for bills and Carter is not a member of Congress. Naturally, doing so requires the availability of a complete semantic network including dictionary, semantic graph, and semantic form space. By rejecting such a query, it is possible to avoid a futile search for an answer which, assuming the data base is consistent, will never be found.

The purpose of the n-o unification algorithm is to detect such discrepancies in both data base clauses and queries. This is done by unifying each ll-a literal *L* of an input clause against all semantic forms (which are stored in ll-o clause representation themselves). If some instance of the ll-a literal *L* fails to unify with any n-o semantic form in the semantic space, then that instance of the n-o literal *L* does not conform with semantics because at least one of its arguments conflicts with the corresponding argument of all semantic forms for that n-tuple size.

Although some instances of the n-o literal *L* might fail to unify with any semantic form, others may succeed. What is desirable therefore, is to transform a ll-o clause input to the system into one (or perhaps several) clauses that are entirely well-formed. These clauses may then be entered into the data base or input to the deductive mechanism as appropriate. Those instances failing to unify should be isolated and the user informed of the error. The semantic well-formedness algorithm which does all of these things is given by McSkimin.

An important part of the well-formedness algorithm is the unification of input literals against the semantic form space. Each semantic form P = (T,<J>) consists of a template T of the form: (vo,Vj,...,v_n) v S and a n-o set <K The tuple (v_0,...,v_n) is called the distinguished literal of F, and the literals in S are called the semantic literals of P.

Unifying substitutions are also applied to the semantic literals which are carried along during the process, "appended to the query, during the deductive search to perform further semantic checks on a literal by retrieving or inferring information from the data base (rather than the semantic space).

The tuple V = (VQ,VI,...,v_n) along with the substitutions for the placeholder variables v-j occurring in * determine what elements of the domain may co-occur in arguments of an (n+1) tuple. If a portion of an input literal *h* unifies with some semantic form F on literal V, then that combination of predicate and domain elements is considered well-formed. If a literal *L* does not unify with any semantic form, then it is not well-formed since it conflicts with all possible ways that domain elements may be combined.

An example of semantic well-formedness, and how the semantic form S is used will now be given.

Example: Let F - ((u,x,y) v(B,x,y) {{[M]/a, *female/x, animal/y_y* [EQUAL)/£}}) be a semantic form that determines the well-formedness of the "mother" predicate. (a,x,y) is the distinguished 1 iteral in the semantic form. The semantic literal (3,x,y) is used to indicate that if x and y are equal then a semantic conflict has been detected. For example, the following query, asking if Mary is her own mother, when resolved with the semantic form F, yields the new clause, ((6,x,y), {{IMj/a, [Mary]/x, [Mary]/y, [EQUALl/p,]}) .

The inferred n-o literal ((B,x,y),{{ [F(JIAL]/3, [Mary]/x,[Mary]/y}}), is not unified against the semantic forms since it descended from a semantic literal. Thus, all literals that came from the query have been successfully unified against the semantic form. The literal that remains is then appended to the query to form the new query: %(a,x,y) v (|3,x,y) {{[M]/cx, [Mary]/x, [Mary]/y, [EQUAL]/3}K

Before the proof mechanism starts to answer

the query, a test is made to determine if there are any literals eligible for predicate evaliiation. If there are, then they are evaluated. Thus, in this case, the proof mechanism evaluates the literal (3,x,y) to true and the entire clause is therefore removed from the search space, which indicates that the question is unanswerable. If the literals appended to the query are not eligible for predicate evaluation, they are treated as any other literal and must be resolved away in order for the query to be answered.

In this section it lias been shown how semantic constraints may be used to reject semantical!y inconsistent queries and data base assertions. The third use of the semantic network is given in the following section.

3.3 Semantic Actions in the Search Space

A major problem facing any problem solving system is the growth of the search space. when the problem to be solved is complicated, the search space grows and usually, when a solution is not found, one runs out *of* machine work-space rather than time. One can use knowledge about, the problem domain to help decrease the workspace needed.

3.5.1 Representing Counting Restrictions

A natural candidate for decreasing the workspace is to have knowledge concerning predicates. In particular, *one* might refer to counting predicates as ones in which there can be either a fixed or an upper bound to the number of solutions to the problem. For example, when referring to the U.S. Senate, there are two senators who represent one state. Thus, if one is searching for an answer to a subproblem which concerns senators who represent Maryland, only a maximum of two may be found in the system.

To take advantage of counting predicates, counting information must be represented in the semantic network, and a bookkeeping mechanism must exist during the search process to keep track of when a solution has been found, and when all possible solutions have been found. We sketch how this is accomplished below. (See McSkimin for details.)

To motivate the discussion, consider the following query, "Who represents the states of New York or New Jersey?". The query, in n-o notation may be given in negated form as: f^(a,x,y), {{[REPRESENT^*, [NY,NJ]/y}}). Information of a general nature concerning the predicate REPRESENT is contained in the semantic form space. Thus, it is desired to denote that two senators represent every state, and that there are 15 representatives for NJ and 39 from NY, and that each senator or congressman can represent at most one state.

The above semantic information can be represented in the semantic form space as,

F: C(a,x,y),{{[REPRESENT]/q>senator#l/x,state#2/y}
{[REPRESENT]/a, rep#l/x, [NJ]#15/y]
{[REPRESENT]/a,repfa/x,[NY]#39/y}}).

By the notation B #m/v, is meant, that each element of a Boolean Category Expression (BCE), B, can occur in at most m distinct n-tuplcs which unify with that particular substitution set.

The query, when tested against a semantic form during a well-formedness test is modified to reflect the above possibilities. Thus, Q becomes

Q_1:(H"a>x>y)>{([REPRESEm·]/a>scnatorfl/x>[NY,NJ]#yy
{[REPRESENT]/a,rep#l7x, [NJl#15/yl
{[REPRESENT]/a,rej)#l/x, [NY|#39/y}}J.

For every substitution set cpi of a predicate there is associated a semantic set count (SSC), which represents the number of possible solutions that can be found relative to the predicate. Let the substitution set cp be given by, cp = {S_0//l/v_0, Siftiii/vj,...,S_n#m_n/v_nj. Then it is easy to see that

$$SSC = \min_{i-1}^{n} (Card(S.) \cdot m_i)>$$

where (.]ard(Sj_) is the cardinality of the set S., and mj_ is the element semantic count. Thus, for cp = {[REPRRSBNrr]/cx,senatorftl/x, [NY,NJ]fl2/y), SSO inin(iOO-l,2-2)=4, where Card (senator) =100. Tliat is, there are only 4 possible solutions - the two senators from each state for the particular substitution set. If one sums the SSC over all substitution sets relative to the subproblem (literal), one obtains the total possible solutions. If all solutions are found for all substitution sets, the literal is said to be fully solved. For the single subproblem associated with the sample query there are 4+15+39=58 possible semantic solutions.

The syntactic count (SC) for a substitution set is simply the number of possible entries in the relation, and is given by

$$SC = \prod_{i=1} Card(S_i).$$

This number is always greater than or equal to the semantic set count, and generally is considerably larger. In particular SC-200 for the above example.

Ciiven an n-o literal in the search space, a target syntactic count (TSC)is specified for each substitution set cp/ for each literal, and is defined as TSCj * (§Cj_-SSCj) . When a n-o literal is to be solved, it is removed from the clause and placed in a special list. The literal on the list is initially given the count, equal to SC. As unique solutions are found for the literal, they are subtracted out of the substitution set, and the count of the literal initially set to SC is decremented by the number of unique solutions found. When the count for the literal on the list equals TSC, then the literal is fully solved.

To take advantage of counting predicates, it is necessary to know, when, during the search process, a literal has been solved, and if so, whether the literal has been fully solved. In the environment of Question-Answering Systems, Fishman [1973,1974] has experimental evidence which indicates that linear resolution with selection function (SL resolution), developed by Kowalski and Kuehner [1971], and independently by Loveland [1969,1970] (who termed it model elimination), or a variant thereof, will be used for the inference

mechanism. SL resolution is very convenient to use since one knows exactly when a literal has been solved. This occurs when truncation takes place. The bookkeeping associated with SL resolution permits one to backup to the clause where one first started to search for a solution to the literal. It is at the clause where one first starts to search for a solution to the literal that one wants to initiate semantic actions.

Three types of semantic actions may be taken when fully solved substitution sets of a search space literal *L* =(L,) are found. Starting at the clause in which the fully solved substitution set appears, one must remove from the search space each substitution set that has been fully solved. This will inhibit any further clauses from being resolved against, the clause. If, all resolvents from the n-a literal have been found, some of the resolvents may not yet have been entered into the workspace. In this case, a pointer to the resolvent set would exist, and could be deleted. The reason for inhibiting any additional clauses from entering the search space is because all solutions have already been found, and no other solutions are possible. By bringing in additional clauses, one is merely trying to find another solution via a different path. The best that can result is that a duplicate solution will be found.

if all substitution sets cp; become fully solved, then the entire subproblem L=(L/i>) is fully solved, all further resolvents or potential resolvents of *I* should be deleted. Thus all further interactions between *L* and the data base are avoided. The semantic action of removing fully •solved sets and subproblems can thus potentially save a great deal in search effort.

Unfortunately, the interactions (i.e., resolution operations) between literal *L* and data base clauses are seldom serial in nature. Rather, it-is often the case that several interactions may proceed at the same time as cooperating processes (or coroutines) controlled by the search strategy. As a consequence, even though some literal *L* becomes fully solved, there may be deductions in progress for *L* that, if continued may duplicate solutions already found. Therefore, ideally, these processes should be terminated. Thus, the second type of semantic action taken is to prune possible redundant derivation trees from the search space. A convenient data structure to facilitate pruning is one in which the immediate resolvents of a clause are linked together in a list, where each clause in the list points to its parent clause, and the parent clause points to the first and last entries in the list (i.e., it is a binary tree). Pruning in such a search space data structure is straightforward, and is not discussed further here.

The third type of semantic action concerns the literals of generated clauses in redundant derivation trees. As each clause (T is generated by resolution or factoring, the search strategy may select a literal from (T to be resolved against the data base. The literal might be on a list of literals waiting to be resolved (for example, see Minker et al.[1973]), or in the process of being resolved. Leaving such a literal there will permit it to interact with the data base, thus producing more unnecessary deductions. Therefore, if such a list exists, the selected literals from all clauses of a redundant derivation must be removed from the list and resolvents in process or that liave been found must be terminated or deleted as the case may be.

The following section illustrates how the above techniques may be used together in answering a query.

4. An Example of the Use of Semantics in a QA System

The data base used in this example consists of assertions and general rules that might be useful in a political context. Assume there are many assertions that state where a member of Congress 1egally resides; i.e.,

A : (cx,x,y),l{ [RHSIDL]/a, tBeall,Mathias,Holt,...,
Hogan,Gude]/x,[MdJ/yj,
A.,: { [RESIDH/cx, [Buckley,Javits,Chisholm,
...,Abzugj/x,[NY]/y]}

These assertions mav be used to derive the state a member of Congress represents by the following general rule:

R, : ('^(a,x,y) v([i,x,y), {{ [RESTDEj/u, *congress/x,*
state/y, [REPRESENT/3}}) .

In addition, there might be several axioms that are used to deduce whether a member of Congress supports a special interest group (such as organized labor) by referencing numerical ratings established by different lobbying groups (e.g., COPE, the Committee on Political Education of the AFLC10). For instance if a senator or representative lias a COPE rating of 67-100 (on a 0-100 scale) then one may conclude that he (she) supports organized labor. Many of these rules may be stored as well as other general rules that deduce whether one supports some state, country, or interest group. These rules are given below.[1]

R.,: K(a,x,y) v (e,x,y), {{ [COPE J/a, *congress/x,*
R67100/y$_t$ [SUPPORT]/6,
labor/z,}

Py { [NSC]/a,congress/x,
R67100/y$_9$ [SUPPORTJ/3,
defense/Z*}*,

R$_4$: Ka,x,y) v (B,x,y),{{ [REPRESENT]/ex,*congress/x,*
state/y, [SUPPORT]/3)}).

The last rule states that if a member of Congress represents a state, then he or she would be expected to support that state.

To derive whether one would support a special-interest group, the ratings of each senator and representative could be stored as assertions:

A~:((a,x,y),{{[NSC]/a,[Mathias,Cranston,...,
Mitchell]/x,[11]/y},

[J]Note: NSC = Nat. Security Council, a defense lobby; NFU = Nat. Farm Union; and LCV = League of Conservation Voters; *R67loo* = A category containing all integers from 67 to 100.

A_4: {{[NSC/α,[Javits,Schweiker,...,
Podell]/x,[33]/y}}.

Besides the data base, semantic forms are also needed. The following semantic forms might be used:

F_1: (α,x,y){{[REPRESENT]/α,*senator#1*/x,*state#2*/y,
{[REPRESENT]/α,*rep#1*/x,[MD]#8/y},
{[REPRESENT]/α,*rep#1*/x,[NY]#39/y}},

F_2: (α,x,y) {[RESIDE]/α,*human#1*/x,*state#v*/y}

F_3: (α,x,y) {[SUPPORT]/α,*congress#v*/x,(*state* ∪ *country* ∪ *interest*)#v/y}.

The element semantic counts that apply to each form is provided.

It is assumed that dictionary entries for all predicates, constants, and BCEs have been created; because of their large number, these will not be shown, nor will the semantic graph be shown.

Suppose that the following query is asked: (∿(α,x,y),{{[SUPPORT]/α,[Sirica]/x,[Mining-interest/y}}), i.e., Does Sirica support mining interests?" This query fails to unify against any of the semantic forms; in particular, it fails to match F_3 since Sirica ∈ *judge* and *judge* is disjoint from *congress*. The query is rejected as meaningless.

Now suppose another query is entered: (∿(α,x,y) ∨ (β,x,z),{{[SUPPORT]/α,[REPRESENT]/β, *senator*/x,*defense*/y, [Md,NY]/z}});

i.e., "Are there any senators from New York or Maryland who support defense interests?" Each literal is resolved against the semantic form to test its well-formedness. The literal (β,x,y) unifies with form F_2 and the literal (α,x,y) unifies with the form F_3. These unifications are successful because *senator* ⊆ *congress* and *defense* ⊆ *defense*, as would be found in the semantic graph. The query is thus well-formed.

A scenario for the proof, shown in Figure 2, using SL resolution as the inference system of the above query, follows:

1) A heuristic function f is used to select which literal of the query to solve first. As discussed in McSkimin [1976], Minker et al. [1973], and Minker [1976], the number of potential solutions of literals is a good indication of the amount of work required to solve them. In this case, the element set counts of the semantic form relevant to each literal are referenced and the semantic set counts computed. Since at most two senators can represent a state, the literal with the REPRESENT predicate has 4 solutions as shown in Section 3.3.1.

No bounds can be placed on the SUPPORT literal since it is not a counting predicate as is REPRESENT. Solving the SUPPORT literal first could cause numerous inferences to be generated, assuming that there are many senators listed who support some defense interest. Thus, the REPRESENT literal is chosen to be solved first.

2) The target syntactic count for the REPRESENT literal, which equals the syntactic count minus the semantic set count is calculated. In this case, there are SOI00-2=200 syntactic solutions and SS04 semantic solutions. Thus, the value TSC =SC-SSC=200-4=196, is stored with the REPRESENT literal *L* and referenced whenever new solutions are found. *L* is then modified to remove those solutions found so far. When the syntactic count of L equals TSC=196, *L* will be fully solved. This process is illustrated by the proof tree of Figure 2 using SL resolution.

3) The general rule R1 is resolved with ∿(β,x,z) since no direct match is possible. Since *senator* c *congress* and [Md,NY] ⊆ *state,* resolvent 2 is formed.

4) The RESIDE literal *L* is now the right-most (and only) literal in the SL resolvent, and is thus solved next. The element set counts from semantic form F_2 are read, and the number of solutions for *L* is calculated as, SSC=min(100-l,2-v) = 100, where "v" means an unknown number of solutions are possible. The target syntactic count is calculated as TSO(l00-2)-100=100, and stored with the RESIDE literal. Thus, 100 solutions must be found before the literal is considered fully solved. Since it is semantically impossible for 100 senators to reside in two states, this situation will never occur. Fortunately, the RESIDE literal will become indirectly fully solved because its ancestor the REPRESENT literal ∿(β,x,z) will be fully solved. This process is described below.

5) Many RESIDE assertions are stored in the data base; only those instances are allowed to enter the search space that involve senators from Md or NY - all representatives from Md and NY and any resident of some other state is excluded. Since {Beall, Mathias, Buckley, Javits} c *senator* they are the only ones that pass through the n-o unification algorithm.

6) Since the A-literal [∿(γ,x,z)] appears as the right-most literal of clause 3, it implies that a subproblem lias been solved. Up-links are followed until ∿(γ,x,z) appears without brackets in clause 2 (denoting the point at which the current deductive chain began as indicated by the dotted lines of Figure 5). The solutions found are:

∿(γ,x,z){{[RESIDE]/γ,[Beall,Mathias]/x,[Md]/z},
{[RESIDE]/γ,[Buckley,Javits]/x,[NY]/z}}.

By the procedure described in McSkimin, these solutions are removed from clause 2 (whose syntactic count is 200) leaving a new clause whose syntactic count equals 196. Since this does not equal the target syntactic count of 100, no semantic actions can be taken.

7) The A-literal [∿(γ,x,z)] is thus truncated from Clause 3, yielding clause 4.

8) Another subproblem has been solved since [∿(β,x,z)] is the right-most literal of clause 4. Up-links are followed until ∿(β,x,z) first appears without brackets (clause 1). Since four solutions have been found for the literal, its syntactic count becomes 196. Since this equals the target syntactic count, ∿(β,x,z) is therefore removed from the search space to prevent other inference rules from attempting to deduce the senators from Md or

NY. In addition, the literal HY,X,Z) from clause 2 is removed also since all residents of Md and NY of interest have been found. Thus, even though this literal had 96 solutions left to find, it could be pruned since it was a. *descendant* of a literal which was fully solved. These are examples of semantic actions.

9) The A-literal [M>,x,z)J is next truncated from clause 4 yielding clause S, which is next solved in a similar fashion.

This example lias illustrated how user-supplied semantic information may be incorporated within the framework of predicate calculus so as to make the deductive inference process more efficient. Three methods were shown to be effective in reducing the amount of effort involved in answering a query: semantic well-formedness tests, semantic operator selection, and semantic actions. Naturally, these are not beneficial for all data bases; conditions under which each is expected to prove most beneficial are given by McSkimin.

5. Summary

We have described three basic uses of a semantic network: (1) To semantic-ally unify two literals. (2) To perform semantic well-formedness tests. (3) To perform semantic actions. Semantic unification serves to decrease the number of deductions that one may have over syntactic methods. Semantic well-formedness tests serve to delete data from entering the data base if they are not soman - tically well-formed relative to the domain of application. Semantic well-formedness tests may also introduce new literals into the search space which mast be satisfied for a solution to be found. Semantic actions serve to use counting information to determine when a literal has been fully solved, and to take actions based upon this information. These actions serve to delimit the search space.

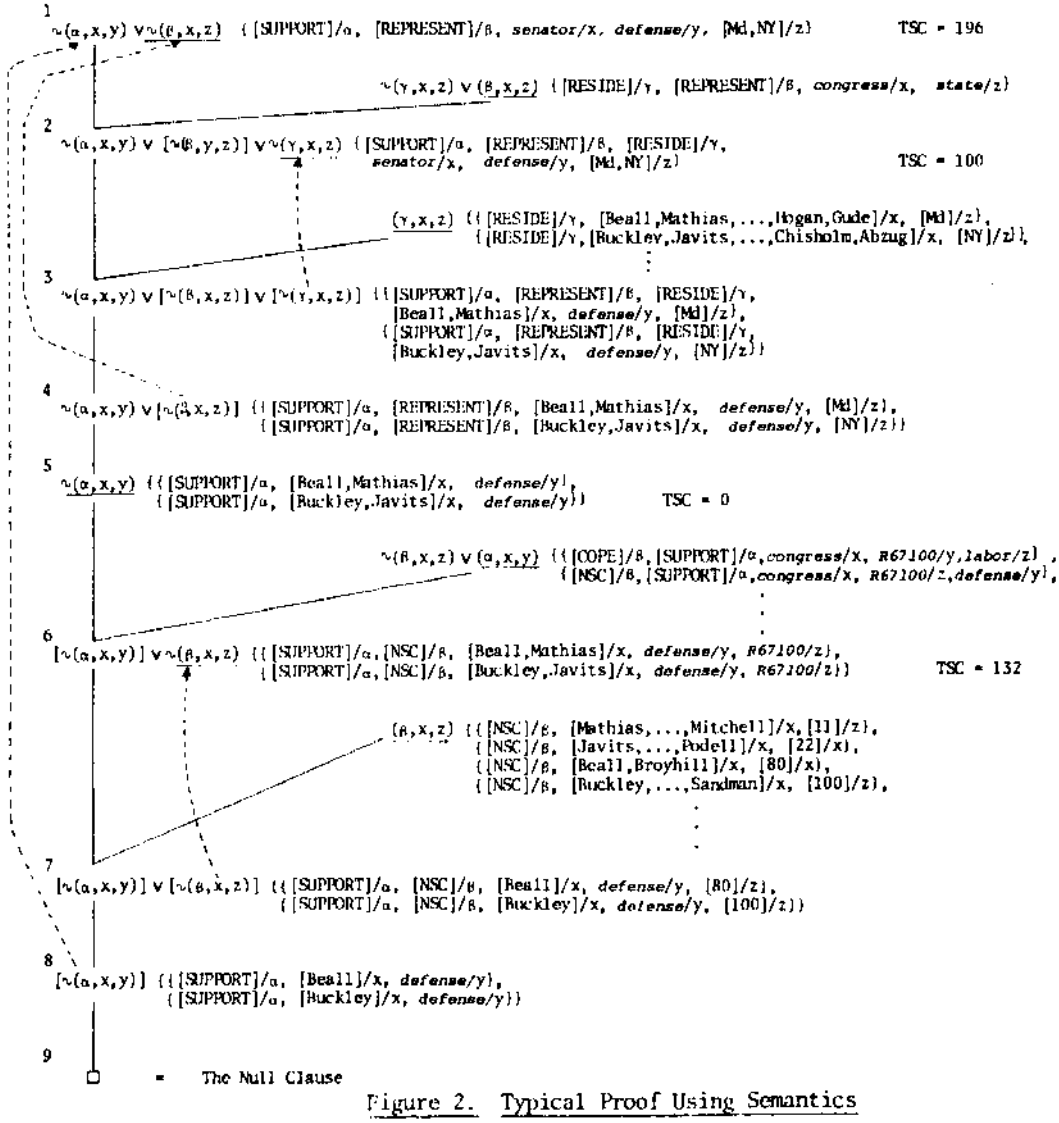

Figure 2. Typical Proof Using Semantics

An alternative approach to the one described here would be to build into the system unary predicates rather than set information as represented in the semantic graph. We do not believe that such an approach would be effective since the addition of axioms to the system to represent transitive superset relations and disjoint relations would be too cumbersome to deal with in practice and would lead to very long proofs.

Although the approach appears to be viable, we cannot yet provide experience which would demonstrate its effectiveness. We expect to determine its effectiveness. One factor is the data structure to be used to perform pattern-directed search for clauses which semantically and syntactically match literals in the data base. A second factor will be the amount of time required to perform the CONFLICT algorithm. This algorithm determines whetlicr or not two boolean category expressions are semantically consistent. If the conflict algorithm is too time consuming, it may defeat the whole approach and make it comparable to a strictly syntactic approach. Further analysis is needed to estimate whether the effort expended in executing the semantic routines will exceed the extra effort incurred if they are not performed at all.

We believe that an advantage will be shown for the techniques described here. We are currently implementing a system, MRPPS 3.0 which incorporates the techniques described here. We expect to experiment with the system to determine how well it will work on large data bases.

Acknowledgements

The authors wish to express their appreciation to the National Science Foundation for the supjx)rt that they liave provided for this effort under NSF CJ-43632. They would also like to express their appreciation to Mr. Guy Zanon for his careful reading of the paper and his many suggestions.

References

11] F i s hman, I). H. [1973] Experiments with a Re so - 1 ution-Based Deductive Question-Answering System and "aTProposed Clause Representat Ion for Parallel Search, ph.D. Thesis, Dept. of Computer Science, Univ. of Maryland, College Park, Md. Also Tech. Report TR-280, 1973.

[2] Fishman, 1). 11. [1974] "Experiments with a Deductive Question-Answering System," Computer and Inf. Sci. Dept., Univ. of Mass., COINS Tech. Report F4C-10, 1974.

(3J Fishman, I). H. and Minker, J. [1975] II-Representation: A Clause Representation for ParaJ - 1eJ Search,'' Artificial Intelligence 6 (2) (1975), 103-127.

[4] Kowalski, R. and Kuehner, D. [1971] "Linear Resolution with Selection Function," Artificial Intelligence 2 (3/4) (1971), 221-260.

[5] Loveland, D. W. [1969] "A Simplified Format for the Model-Elimination Theorem Proving Procedure," J.ACM 16 (July 1969), 349-363.

16] Loveland, D. W. [1970] "Some Linear Herbrand Proof Procedures: An Analysis," Dept. of Computer Science, Carnegie-Mellon Univ., Pittsburgh, Pennsylvania, 1970.

[7] McSkimin, J. R. [1976] The Use of Semantic Information in Deductive~~Question-Answering SysTems"^ Ph.D. Thesis, "Dept. of Computer Science, Univ. of MI., College Park, Md.(1976) Also Tech. Report TR-465, 1976.

18] McSkimin, J. R. and Minker, J. [1977] "A Predicate-Calculus Based Semantic Network for Question-Answering Systems," Dept. of Computer Science, Univ. of Maryland, College Park, Md., Tech. Report TR-509, 1977.

[9] Minker, J., Fishman, i). H. and McSkimin, J.R. [1973] 'The Q* Algorithm - A Search Strategy for a Deductive Question-Answering System," Artificial Intelligence 4 (1973.1, 225-243.

[10] Minker, J. [1976] "Search Strategy and Selection Function for An Inferential Relational System," Tech. Report TR-497, Univ. of Maryland, College Park, Md.,1976.

[11] Schubert, L. K. [1976] "Extending the Expressive Power of Semantic Networks," Artificial Intelligence 7 (1976), 103-198.