Artificial Intelligence/ Language Processing
C. Montgomery Editor

Logic and Semantic Networks

Amaryllis Deliyanni
University of Athens

Robert A. Kowalski
University of London

An extended form of semantic network is defined, which can be regarded as a syntactic variant of the clausal form of logic. By virtue of its relationship with logic, the extended semantic network is provided with a precise semantics, inference rules, and a procedural interpretation. On the other hand, by regarding semantic networks as an abstract data structure for the representation of clauses, we provide a theorem-prover with a potentially useful indexing scheme and path-following strategy for guiding the search for a proof.

Key Words and Phrases: logic, semantic networks, theorem-proving, indexing, resolution, deduction, logic programming

CR Categories: 3.42, 3.64, 5.21

Introduction

Logic and semantic networks are different formalisms for representing information. Several authors have shown that simple semantic networks can be extended so that they have the same expressive power of predicate logic [5, 15, 16]. Still more recently, deductive inference systems for semantic networks have been investigated [2, 3, 17, 18].

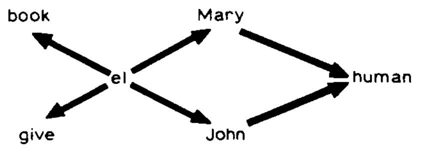
Authors' Addresses: R.A. Kowalski, Department of Computing and Control, Imperial College of Science and Technology, London SW7 2BZ, England; A. Deliyanni, Division of Electronics, University of Athens, Athens, Greece.

In this paper we shall define an extended form of semantic network which can be regarded as a syntactic variant of the clausal form of logic. We define top-down and bottom-up inference and resolution more generally for the extended semantic network. In particular, top-down inference gives us a procedural interpretation of reasoning in the network.

Not only can the extended semantic network be regarded as a syntactic variant of logic, but it can also be used as an abstract data structure for the representation of clauses in an implementation of a predicate logic proof procedure. The semantic network data structure provides an indexing scheme and helps to guide the search for a solution. In particular, as a data structure for an interpreter of predicate logic programs, it guides the execution of procedure calls. The strategy suggested by the network gives the proof procedure a path-finding flavor.

1. Simple Semantic Networks

A *semantic network* is a directed graph whose nodes represent individuals and whose arcs represent relationships between individuals. An arc is labeled by the name of the relationship it represents. Several arcs can have the same label. However, each individual is represented by only a single node. The English sentences "John gives the book to Mary" and "John and Mary are human" are represented in the following semantic network:



Here "el" names an event which is an act of giving, whose actor is John, object is book, and recipient is Mary.

Arcs are not to be confused with access pointers. However, given a node in the network, it is assumed that the network provides direct access to all the relationships in which the node participates, independent of the direction of the arc.

2. The Clausal Form of Logic

The relationships in the network above can be expressed in the clausal form of logic as follows:

Obj (el, book) ←
Act (el, give) ←
Actor (el, John) ←
Rec (el, Mary) ←
Isa (John, human) ←
Isa (Mary, human) ←

Relations are named by predicate symbols (in this case, "Obj," "Act," "Actor," "Rec," and "Isa") and individuals are named by constants ("e1," "book," "give," "John," "Mary," and "human"). Predicate symbols corresponding to labels on the arcs of semantic networks always have two arguments. But in general a predicate symbol can have n arguments, $n \geq 1$. For example the sentence "John gives the book to Mary" might equally well be represented in logic by using a three argument predicate symbol:

Give (John, book, Mary) $\leftarrow$

In addition to simple assertions, logic can also express general propositions. The sentence "John gives the book to everyone he likes" is expressed by the clause

Give (John, book, x) $\leftarrow$ Likes (John, x)

Here the symbol "x" is a variable representing any individual. The arrow represents the logical connective "if." In general a clause can have several conditions, all of which must hold for the conclusion to hold. For example the clause

Likes (John, x) $\leftarrow$ Give (John, y, x), Likes (John, y)

expresses that if John gives away something he likes, then he must like the person he gives it to. (Variables in different clauses are unrelated even if they look the same.) A clause can have several alternative conclusions, at least one of which must hold if all the conditions hold. That every animate being is either an animal or a vegetable is expressed by the clause

Animal (x), Vegetable (x) $\leftarrow$ Animate (x)

Thus, the conclusions of a single clause are a *disjunction* of alternatives, whereas the conditions are a *conjunction.* A clause without conditions is an unconditional assertion. However, a clause without conclusions is a denial. The clause

$\leftarrow$ Give (x, y, John), Likes (John, y)

denies that anyone gives John anything which John likes.

The existence of individuals is expressed by using constant symbols or function symbols. "Someone likes John" requires a constant to name the anonymous individual who likes John, for example:

Likes (A, John) $\leftarrow$

However, "everyone is liked by someone" is ambiguous. It requires a constant symbol if one individual likes everyone

Likes (B, x) $\leftarrow$ Human (x)

and a function symbol if different individuals like different people

Likes ($f(x)$, x) $\leftarrow$ Human (x)

Here, for any individual x, the expression $f(x)$ names the individual who likes x.

A more formal definition of the clausal form of logic is the following:

A *sentence* is a collection of clauses.
A *clause* is an expression of the form

$A_1, \ldots, A_n \leftarrow B_1, \ldots, B_m$

where $A_1, \ldots, A_n$ are called the *conclusions* of the clause and $B_1, \ldots, B_m$ are called the *conditions.* Both conclusions and conditions are expressions of the form

$P(t_1, \ldots, t_k)$

called *atoms,* where P is a k-argument predicate symbol and $t_1, \ldots, t_k$ are *terms.* Terms are either constants, variables, or *functional terms* which are expressions of the form

$f(t_1, \ldots, t_l)$

where f is an l-argument function symbol and $t_1, \ldots, t_l$ are terms.

Convention: Throughout this paper variables begin with one of the letters u, v, w, x, y, z, in order to distinguish them from constants.

Deductive inference rules have been developed for the clausal form of logic. Bottom-up inference derives new assertions from old ones. For example by matching the assertions

Give (John, book, Mary) $\leftarrow$
Likes (John, book) $\leftarrow$

with the conditions of the general clause

Likes (John, x) $\leftarrow$ Give (John, y, x), Likes (John, y)

we obtain the new assertion

Likes (John, Mary) $\leftarrow$

Top-down inference derives new denials from old ones. By matching the denial

$\leftarrow$ Likes (John, Mary)

with the conclusion of the same general clause, we obtain the new denial

$\leftarrow$ Give (John, y, Mary), Likes (John, y)

Both top-down and bottom-up reasoning are special cases of resolution. Resolution involves matching a condition of one clause $\mathscr{A}$ with a conclusion of another clause $\mathscr{B}$. The derived clause, called the *resolvent,* consists of the unmatched conditions and conclusions of $\mathscr{A}$ and $\mathscr{B}$, instantiated by the matching substitution. The clauses $\mathscr{A}$ and $\mathscr{B}$ are called the *parents* of the resolvent. *Matching* two atoms amounts to finding a substitution of terms for variables which if applied to the atoms would make them identical. A more formal definition of resolution can be found in the original paper [12].

Bottom-up inference is a form of hyper-resolution [13], and top-down inference is a form of model elimination [8] and linear resolution [8, 9]. More detailed references can be found in [7].

Clauses which contain at most one conclusion (*Horn clauses*) can be given a procedural interpretation [6].

A clause of the form A $\leftarrow$ $B_1, \ldots, B_n$ is interpreted as a *procedure* with *head* A and *body* $B_1, \ldots, B_n$. Each atom in the body of the procedure is interpreted as a *procedure call.* A denial is interpreted as a set of procedure calls and is called a *goal clause.* Top-down inference, which matches a selected procedure call in a goal clause with the head of a procedure, is interpreted as *procedure invocation.* The new goal consists of the *old procedure calls*, which are the instantiated copies of the unmatched procedure calls in the old goal clause, and the *new procedure calls*, which are the instantiated copies of the procedure calls in the body of the invoked procedure. It is useful to divide the matching substitution associated with procedure invocation into two parts. One part affects variables in the procedure head and transmits input into the body of the procedure. This is the *input component* of the substitution. The other part affects variables in the procedure call and transmits output to the remaining procedure calls of the goal clause. This is the *output component.*

The restriction of semantic networks to the representation of two-argument relationships is not a significant limitation. On the contrary, it has several advantages.

Every n-argument (n-ary) relationship can be reexpressed as a conjunction of two-argument (binary) relationships. If $n > 2$, $n + 1$ binary relationships are needed. If $n = 1$, then only one is necessary. For example the 3-ary relationship

Give (John, book, Mary) $\leftarrow$

can be reexpressed (as in the beginning of Section 2) as a set of four assertions using binary predicate symbols. In general, it is necessary to introduce a name for the original n-ary relationship, "e1" in this example. The n-ary predicate symbol becomes a constant symbol. For each argument of the n-ary relationship, as well as for its predicate symbol, we express how it is related to the original n-ary relationship by means of a binary relationship.

If $n = 1$, the predicate symbol also becomes a constant symbol, and it is only necessary to introduce a single binary relationship which expresses how the original predicate symbol is related to its argument. For example,

Human (x)

becomes

Isa (x, human)

The transformation from n-ary to binary replaces predicate symbols with constant symbols. In the new formulation the original predicate symbols can also be replaced by variables; this gives some of the expressive power of higher order logic.

The use of binary predicate symbols also has the advantage that unknown arguments of the original n-ary relationship can be ignored. To express that "John was given the book," we only need the assertions

Act (e2, give) $\leftarrow$
Obj (e2, book) $\leftarrow$
Rec (e2, John) $\leftarrow$

whereas in the n-ary representation, it is necessary to name the unknown donor

Give (C, book, John) $\leftarrow$

In database applications the binary formulation corresponds to the use of variable-length records. Space is saved, and the unnecessary introduction of constant symbols is avoided.

The binary representation is also at an advantage when additional information needs to be associated with the n-ary relationship. To express that "John was given the book in the park," it is only necessary to add the assertion

Location (e2, park) $\leftarrow$

In the n-ary representation the original 3-ary relationship would have to be replaced by a new 4-ary one

Give* (C, book, John, park) $\leftarrow$

The binary representation suggests a way of dealing with simple aspects of time. For example the addition of the assertion

After (e1, e2) $\leftarrow$

can be used to express that "John gave the book to Mary after it was given to him in the park."

The translation from the n-ary to binary representation does not always result in an improvement. The three-place Plus-relation, for example,

Plus (x, y, z)	x plus y is z

can be reexpressed by means of binary relations

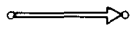
Isa (w, Plus)	w is a Plus-fact
Add1 (w, x)	x is the first number added in w
Add2 (w, y)	y is the second number added in w
Sum (w, z)	z is the sum of the numbers added in w

The binary formulation is more obscure and no more useful than the original three-place relation.

3. Extended Semantic Networks

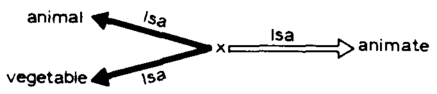
Simple semantic networks can only express collections of variable-free assertions. We define an extended form of semantic network which can be interpreted as a variant syntax for the clausal form of logic. It is an immediate consequence of this interpretation that the semantics of the extended semantic network is identical to that of the clausal form of logic.

In the extended semantic network, terms are represented by nodes. Constant, variable, and functional terms are represented by *constant*, *variable*, and *functional nodes* respectively. As in the simple semantic network, every term is represented by a single node. The same conventions for distinguishing variables from constants are employed as in the clausal form of logic. Binary predicate symbols are represented by labels on arcs. An *atom* is a labeled arc together with its two end nodes. The direction of the arc (*link*) indicates the order of the arguments of the predicate symbol which labels the arc. *Conclusions* and *conditions* are represented by different kinds of arcs: conditions are drawn with two lines

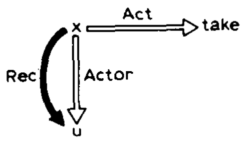


and conclusions are drawn with one heavy line as before. A *clause* is represented by the network representation of its conditions and conclusions. As in the clausal form of logic, different clauses have different variables. Consequently, variables in different clauses are represented by different nodes.

That every animate is a vegetable or an animal is represented by the following network:

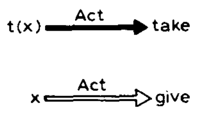


That the actor of any act of taking is also the recipient of the action is expressed by the network:



Rec (x, u) ← Act (x, take), Actor (x, u)

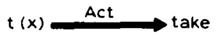
The network



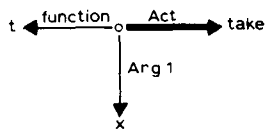
Act ($t(x)$, take) ← Act (x, give)

expresses that for every act of giving, there is an act of taking.

In the extended semantic network, functional terms are represented by single nodes. This contrasts with many semantic network schemes, which require that everything be represented either by a node or an arc. For example the atom



would be represented by the more complicated network

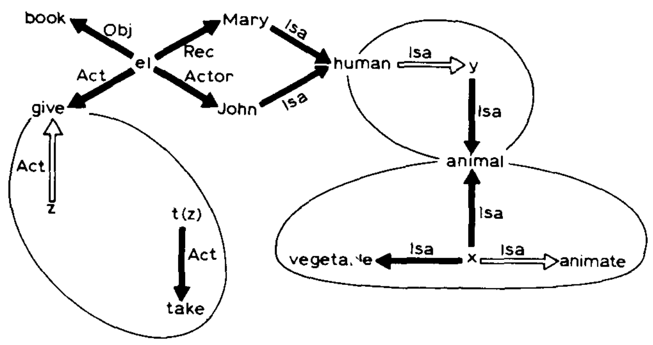


It is worth noting that although such a representation might be useful for indexing purposes, it interferes with the semantics of the network. The new arcs no longer represent semantic relationships between individuals but syntactic relationships between the constituents of names of individuals.

Given a set of clauses, if all occurrences of the same term are represented by a single node, then all the clauses should be incorporated in a single network. This gives rise to the problem of distinguishing which arcs belong to which clauses. The delimitation of clauses can be effected in several ways:

(i) Clause numbers can be associated with arcs.
(ii) The network can be "partitioned" into subnetworks, each of which represents a single clause. Partitions overlap if the clauses they represent contain the same fully instantiated atom.

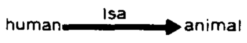
In this paper we delimit clauses pictorially, as in the following example, by partitions drawn around clauses which contain more than one atom.



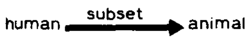
Act ($t(z)$, take) ← Act (z, give)
Isa (x, vegetable), Isa (x, animal) ← Isa (x, animate)
Isa (y, animal) ← Isa (y, human)
Obj (el, book) ←
Rec (el, Mary) ←
Actor (el, John) ←
Act (el, give) ←
Isa (Mary, human) ←
Isa (John, human) ←

The extended semantic network, being equivalent to the clausal form of logic, is a uniform, general-purpose formalism for the representation of information. The Isa and Part-of hierarchies, which are characteristic of many semantic networks, are special cases of the extended semantic network.

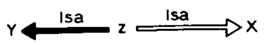
Isa hierarchies, which in simple semantic networks are represented by sets of Isa or Subset assertions, e.g.



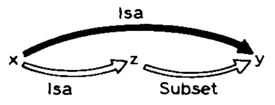
or



can be represented in the extended semantic network in two different ways. One representation expresses statements of the form "X is a Y" where both X and Y name sets of objects by means of clauses of the form

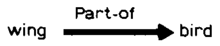


The other representation retains the simple variable-free assertions but adds a general law which expresses the semantics of the Isa and Subset predicate symbols.

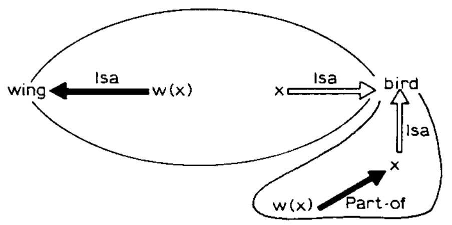


The clauses in the first representation are logically implied by those in the second representation. They can be derived by means of the bottom-up inference rule defined in the next section.

Part-of hierarchies can also be represented in two different ways, with the complication that Part-of assertions conceal a hidden existential quantifier. For example the assertion



means that for every bird, there is a wing which is part of the bird. In clauses:



Isa ($w(x)$, wing) ← Isa (x, bird)
Part-of ($w(x)$, x) ← Isa (x, bird)

Several authors [5, 15, 16] have defined extensions of simple semantic networks. These extensions are all based on the standard form of logic, rather than on the clausal form. Some of them, however, avoid existential quantifiers by using functional terms. A common characteristic of these extensions is that they use explicit nodes and arcs to represent the logical connectives and quantification. Like the use of arcs for representing the structure of functional terms, this introduces the complication of distinguishing arcs which represent semantic relationships from those which represent syntactic ones. Some authors [11, 18] also include in their networks explicit "result," "causes," or "enables" relations which, like Isa- and Part-of arcs, conceal implicit implication or quantification. Hendrix [5], in addition, also uses multilevel partitions to indicate logical structure.

McSkimin and Minker [10] describe a theorem-proving system which uses semantic networks. They regard arbitrary clauses as part of their semantic network, but treat the Isa-hierarchy in a special manner which is essentially compiled.

Woods in his analysis of the semantics of semantic network links [19] is more concerned with the semantics of natural language and the representation of natural language meanings than with the semantics of the semantic network formalism itself.

Recently Fikes and Hendrix [3], Shapiro [17], and Chester and Simmons [18] have described deductive inference procedures for their extended semantic networks. We share their interest in extending the deductive capabilities of semantic networks. However, we are also concerned whether semantic networks have anything to contribute to improving the efficiency of proof procedures.

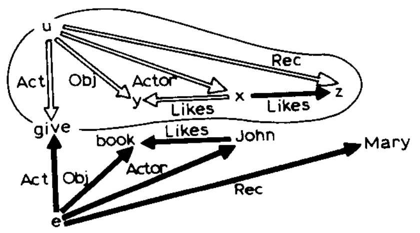
4. Deduction in Extended Semantic Networks

The correspondence between logic and the extended semantic network provides the network with not only semantics but also inference rules. Moreover, top-down inference provides the network with a procedural interpretation.

In order to demonstrate that a set of clauses in a network implies a conclusion, we add the denial of the conclusion to the network and show that the resulting set of clauses is inconsistent. This is done by performing successive steps of resolution until an explicit contradiction is generated.

Resolution and the special cases of top-down and bottom-up inference have already been defined for the clausal form. By virtue of the correspondence between extended semantic networks and clausal form, the definition of resolution also applies to semantic networks.

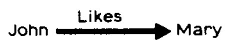
For example applied to the network



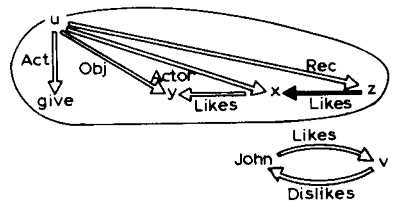
Likes (x, z) ← Act (u, give), Obj (u, y), Actor (u, x),
Rec (u, z), Likes (x, y)

Act (e, give) ←	Rec (e, Mary) ←
Obj (e, book) ←	Likes (John, Mary) ←
Actor (e, John) ←	

bottom-up inference derives the new assertion

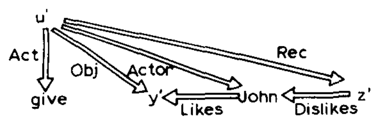


Applied to the network



Likes (x, z) ← Act (u, give), Obj (u, y), Actor (u, x),
Rec (u, z), Likes (x, y)
← Likes (John, v), Dislikes (v, John)

top-down inference derives the new denial

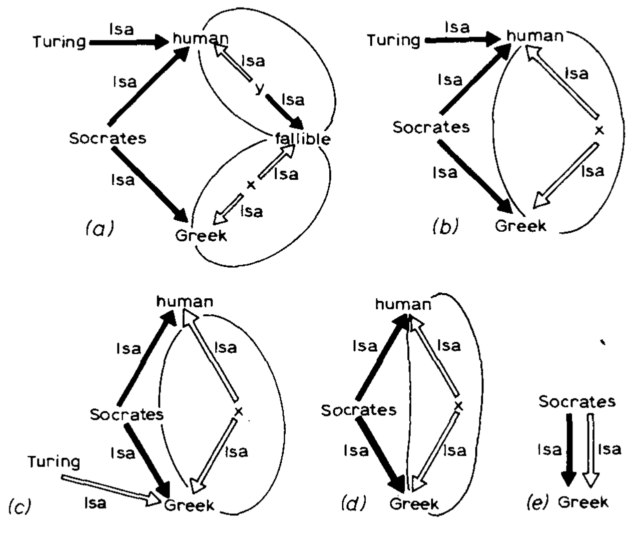


← Act (u', give), Obj (u', y'), Actor (u', John), Likes (John, y'), Rec (u', z'), Dislikes (z', John)

In a computer implementation, clauses can be represented explicitly by adding them to the network or implicitly by using the structure-sharing method of Boyer and Moore [1], in which resolvents are represented by pointers to their parents together with a record of the matching substitution. Anything explicit in the first representation can be computed in the second. Throughout the paper, without prejudicing the manner in which clauses are represented in a computer implementation, networks are drawn with resolvents added to them explicitly.

Independently, whether in an implementation resolvents are represented explicitly or implicitly because of the purity principle [12], a clause can be deleted if one of its atoms matches no other atom in the network. In particular, when a resolvent is created, a parent may be deleted if no other match exists for the atom being matched in that parent. Such a situation might arise if either there was only one match initially, or else all other matches have already taken place. In practice, when deletion is possible, it is generally more convenient to construct the resolvent from the constituents of the deleted parent.

Applied to the problem of finding a fallible Greek, top-down inference together with the deletion rule transforms the initial network until it eventually contains only the empty clause.

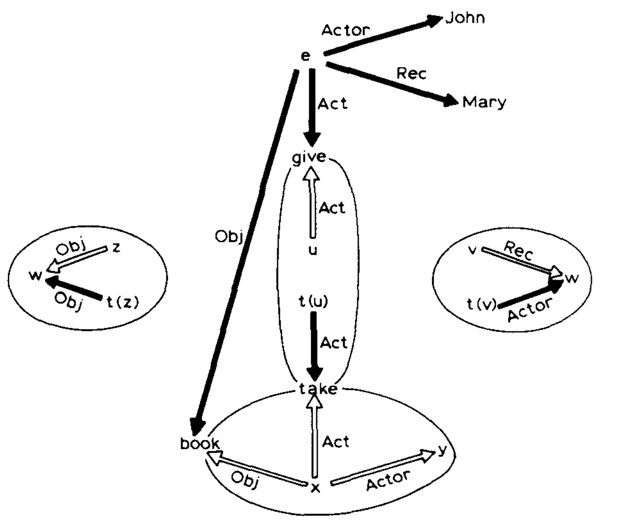


After matching the condition $x \Leftarrow$ Isa $\Rightarrow$ fallible of the initial goal clause in the only possible way, we add the resolvent to the network and delete both parents. The condition $x \Leftarrow$ Isa $\Rightarrow$ human matches two assertions. When the assertion Turing — Isa → human is chosen to match the condition, the assertion is deleted from the network, but the other parent remains. The new resolvent Turing $\Leftarrow$ Isa $\Rightarrow$ Greek matches no other atom in the network and is deleted. The remaining alternative match for the condition $x \Leftarrow$ Isa $\Rightarrow$ human is made, and both parents can now be deleted. The last two clauses in the network have the empty clause as their resolvent and are also deleted.

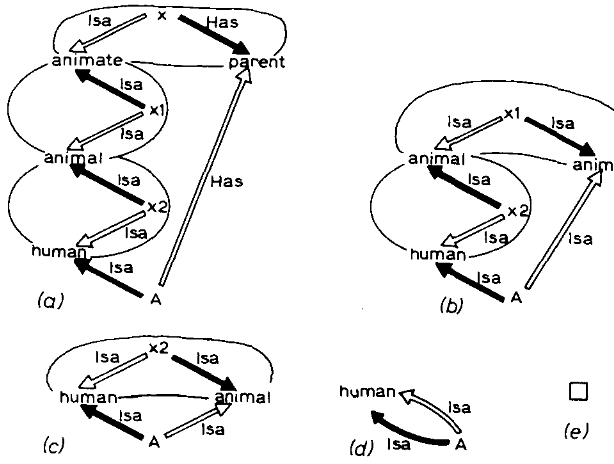
Deletion of clauses potentially destroys the original network. If it is desired to use the network for another purpose, then it is necessary to either save a copy of the original network or restore it to its original form.

The preceding example also illustrates the procedural interpretation of top-down reasoning in the semantic network. In a conventional semantic network, procedures would be written in the host programming language. In the extended semantic network, procedures are integrated with the rest of the database and are executed by the same general purpose mechanism which performs inference in the network.

The procedural interpretation of logic can be regarded as the thesis that computation is controlled deduction [4, 7]. Consider for example the following problem: "John gives the book to Mary. Who takes the book?" Its solution requires using the knowledge that every act of giving results in a corresponding act of taking. In a conventional procedure this knowledge would be mixed with information about how it is to be used. In the extended semantic network it is integrated, as it stands, with the rest of the network, and the information about its use is incorporated in the uniform, general-purpose inference system.



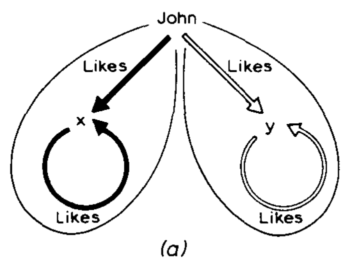
Similarly, special-purpose inference rules are unnecessary. Their effect is achieved by applying general-purpose inference rules to domain-specific information expressed in logic. In particular, it is common in semantic networks to employ special-purpose inference rules which are used to show that types lower in an Isa-hierarchy inherit properties of types higher in the hierarchy. In the extended semantic network, to show that humans inherit from animates the property of having a parent, we assert that someone is human, deny that he has a parent, and use general-purpose top-down inference.

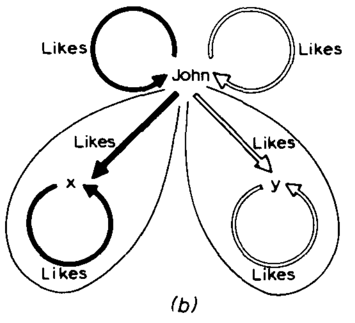


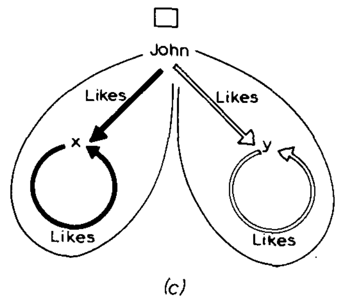
Note that resolution alone, as we have defined it, is inadequate for demonstrating the inconsistency of certain sets of clauses. Russell's "Barber paradox," for example, can only be demonstrated with the aid of the factoring rule.

The *factoring rule* applies to a single clause and derives an instance of the clause. The instantiating substitution matches two atoms of the same kind in the clause (either both conditions or both conclusions). The substitution makes the two atoms identical and only one of them is retained in the derived clause, which is called a *factor*. The figure below illustrates the use of resolution together with factoring to obtain a refutation for a variant of the Barber's paradox. Factoring is applied to each of the two initial clauses. The resolvent of the two factors is the empty clause.

John likes anyone who doesn't like himself.
John likes no one who likes himself.







5. The Use of Semantic Networks to Guide the Search for a Solution

Not only can the extended semantic network be regarded as a syntactic variant of the clausal form of logic, but it can also be regarded as an abstract data structure for an implementation of a proof procedure. Regarded as an abstract data structure, the semantic network provides an indexing scheme which can be used for guiding the search for a solution.

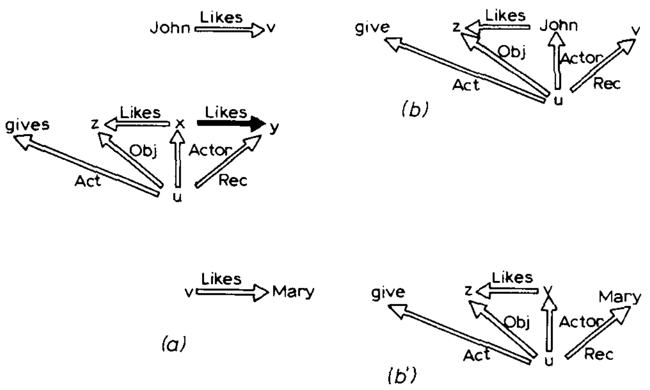
An indexing scheme is a method of organizing information for the purpose of accessing it efficiently. It is the characteristic feature of semantic networks that, given a term, direct access is provided to all atoms (both conditions and conclusions) containing that term. We call this *indexing on arguments.* It is also possible to *index on predicate symbols*: given a predicate symbol, direct access is provided to all atoms containing that predicate symbol. Indexing on predicate symbols is employed in almost all predicate logic implementations. It is interesting therefore, from a theorem-proving point of view, to investigate the consequences of using the semantic network indexing on arguments.

Given two atoms which have just been matched to create a resolvent, the extended semantic network provides direct access to their end nodes and consequently to all (adjacent) atoms which contain those nodes. The accessibility of adjacent atoms suggests the opportunistic strategy of selecting adjacent atoms for matching in the next resolution step. Repeatedly selecting adjacent atoms for resolution gives the proof procedure a path-following flavor.

Without intending to restrict the application of the path-following strategy, we shall discuss in detail only its application to top-down execution of Horn clauses. Selection of adjacent atoms guides the execution of both new procedure calls and old procedure calls, as well as the choice of procedures.

When the matching substitution has an input component which transmits input from the procedure call to the procedure body, the strategy of selecting adjacent atoms suggests selecting for execution an (adjacent) new procedure call which contains the input. Different patterns of input determine the selection of different procedure calls. In the example below, the initial procedure call John $\Leftarrow$ Likes $\Rightarrow$ v determines the selection of one (or both) of the new procedure calls John $\Leftarrow$ Likes $\Rightarrow$ z

and $u \Leftarrow$ Actor $\Rightarrow$ John (containing the input x = John). The initial procedure call $v \Leftarrow$ Likes $\Rightarrow$ Mary, on the other hand, determines the selection of $u \Leftarrow$ Rec $\Rightarrow$ Mary (containing the input y = Mary).



Matching a variable in a procedure call with a variable in a procedure head can be regarded either as transmitting input or as transmitting output. It is advantageous, however, to treat it as transmitting input. The input is a variable, for which it is desired eventually to obtain output by means of subsequent execution of new procedure calls containing the variable. It is a consequence of the semantic network storage scheme that when the output is eventually determined (by matching the variable with a nonvariable), it is transmitted directly to all old procedure calls which contain the same variable. The old procedure calls can use that output as input.

Applied to the execution of new procedure calls, selection of adjacent atoms has two characteristics. Output is sought without delay by executing procedure calls containing variable input. Input is used as soon as it is available by executing procedure calls containing nonvariable input. Thus selecting adjacent new procedure calls is bidirectional search, working both forwards from the input and backwards from the output.

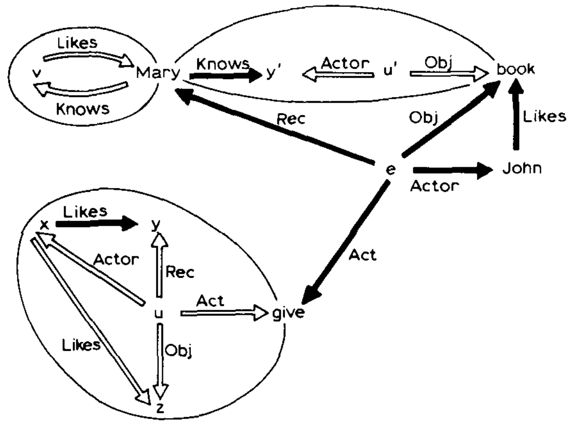
When the matching substitution has an output component, the strategy of selecting adjacent atoms suggests selecting for execution an (adjacent) old procedure call which contains the output. This can be interpreted as coroutining: The last executed procedure call has "produced" output to be "consumed" by old procedure calls waiting to be activated.

Suppose for example:

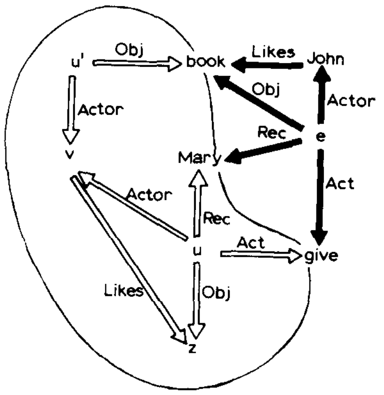
John gives the book to Mary.
John likes the book.
Anyone who gives away anything he likes
must like the person he gives it to.
Mary knows everyone who does
anything with books.

Consider the problem:

Does Mary know anyone
who likes her?



Suppose we first select the procedure call Mary $\Leftarrow$ Knows $\Rightarrow$ v and then decide to suspend the new procedure calls because the only adjacent one contains only variables. We activate the old procedure call v $\Leftarrow$ Likes $\Rightarrow$ Mary and match it with the procedure head $x \Leftarrow$ Likes $\Rightarrow$ y. Input is transmitted to the procedure body.



We activate the new procedure call $u \Leftarrow$ Rec $\Rightarrow$ Mary adjacent to the input constant "Mary." This transmits output $u = e$ to the three old procedure calls containing u. Activating the one $e \Leftarrow$ Actor $\Rightarrow$ v which contains the most recent variable v transmitted as input transmits output v = John to the previously suspended old procedure call which is now $u' \Leftarrow$ Actor $\Rightarrow$ John. Every procedure call which remains to be executed now matches an adjacent assertion.

The network guides not only the selection of procedure calls but also the selection of procedures (including assertions). Given a procedure call, the strategy of selecting adjacent atoms suggests choosing a procedure head which shares a node (constant or functional term) with the procedure call. This strategy is useful in some cases but can be misleading in others. It is misleading for example when applied to the problem, given in the preceding section, of determining who took the book. If in the initial goal clause we select the procedure call $x \Leftarrow$ Obj $\Rightarrow$ book and match it with the adjacent assertion, then we are led to a failure. The procedure head $t(z)$ $\Leftarrow$ Obj $\Rightarrow$ $w1$ which needs to be matched with the procedure call contains no common node.

In order to access procedure heads, like the one needed in the last example, it is useful to supplement the semantic network indexing on arguments with indexing on predicate symbols. Indexing on predicate symbols is also useful when a procedure call shares a node with a large number of procedure heads. We can avoid investigating all of the procedure heads, searching for one which matches the call, by using the predicate symbol index to identify the ones which have the same predicate symbol as the procedure call.

6. Conclusions

Not only are logic and semantic networks compatible formalisms, but each has something to contribute to the other. Logic extends the expressive power of simple semantic networks, provides them with a semantics, inference rules, and a procedural interpretation. Semantic networks, on the other hand, draw attention to the advantages of using binary rather than *n*-ary relations. They provide a predicate logic inference system, with both an indexing scheme and a potentially useful path-following strategy for guiding the search for a solution.

Acknowledgments. This research was supported by assistance from the Science Research Council. We have benefited from useful discussions with K. Clark about his call-by-need design for a logic programming interpreter and B. Marsh about his indexing scheme for a logic database system. Helpful comments on the paper were given by D. Brough, F. Brown, and K. Clark.

Received August 1977

References

1. Boyer, R.S., and Moore JS. The sharing of structure in theorem-proving programs. In *Machine Intelligence* 7, B. Meltzer and D. Michie, Eds., Edinburgh University Press, 1972, pp. 101–16.
2. Deliyanni, A.J. A comparative study of semantic networks and predicate logic. M. Sc. Th., Dept. of Comptg. and Control, Imperial College, University of London, Sept. 1976.
3. Fikes, R.E., and Hendrix, G.G. A network-based knowledge representation and its natural deduction system. Proc. Fifth Int. Joint Conf. Artif. Intel., M.I.T., 1977, pp. 235–246.
4. Hayes, P.J. Computation and deduction. Proc. 2nd MFCS Symp., Czechoslovak Acad. Sciences, 1973, pp. 105–118.
5. Hendrix, G.G. Expanding the utility of semantic networks through partitioning. Proc. Fourth Int. Joint Conf. Artif. Intel., Tiblisi, Georgia, 1975, pp. 115–121.
6. Kowalski, R.A. Predicate logic as programming language. Information Processing 74, North Holland Pub. Co., Amsterdam, 1974, pp. 569–574.
7. Kowalski, R.A. Algorithm = logic + control. Res. Rep. 77/3, Dept. of Comptg. and Control, Imperial College, University of London, Nov. 1976; to appear in *Comm. ACM.*
8. Loveland, D.W. A unifying view of some linear Herbrand procedures. *J. ACM 19*, 2 (April 1972), 366–84.
9. Luckham, D. Refinement theorems in resolution theory. Proc. IRIA Symp. on Automatic Demonstration, Versailles, France, 1970, pp. 162–90 (available from Springer–Verlag).
10. McSkimin, J.R., and Minker, J. A predicate calculus based semantic network for question-answering systems. Tech. Rep. TR-509, Dept. Comptr. Sci., U. of Maryland, March 1977.
11. Mylopoulos, J., Cohen, P., Borgida, A., and Sugar, L. Semantic networks and the generation of context. Proc. Fourth Int. Joint Conf. Artif. Intel., Tiblisi, Georgia, 1975, pp. 134–142.
12. Robinson, J.A. A machine-oriented logic based on the resolution principle. *J. ACM 12*, 1 (Jan. 1965), 23–41.
13. Robinson, J.A. Automatic deduction with hyper-resolution. *Internat. J. Comput. Math. 1*, 1965, 227–34.
14. Robinson, J.A. Computational logic: The unification computation. In *Machine Intelligence 6*, B. Meltzer and D. Michie, Eds., Edinburgh University Press, 1971, pp. 63–72.
15. Schubert, L.K. Extending the expressive power of semantic networks. Proc. Fourth Int. Joint Conf. Artif. Intel., Tiblisi, Georgia, 1975, pp. 158–164.
16. Shapiro, S.C. A net structure for semantic information storage, deduction and retrieval. Proc. Sec. Int. Joint Conf. Artif. Intel., London, 1971, pp. 512–523.
17. Shapiro, S.C. Representing and locating deduction rules in a semantic network. Proc. of the Workshop on Pattern Directed Inference Systems, Sigart Newsletter (ACM), *63* (June 1977), pp. 14–18.
18. Simmons, R.F., and Chester, D. Inferences on quantified semantic networks. Proc. Fifth Int. Joint Conf. Artif. Intel., M.I.T., 1977, pp. 267–273.
19. Woods, W.A. What's in a link. In *Representation and Understanding*, D. Bobrow and A. Collins, Eds., Academic Press, New York, 1975.