A LOGIC FOR SEMANTIC NETWORKS

by

Robert Bechtel

Stuart C. Shapiro

Computer Science Department

Indiana University

Bloomington, Indiana 47401

TECHNICAL REPORT NO. 47

A LOGIC FOR SEMANTIC NETWORKS

ROBERT BECHTEL

STUART C. SHAPIRO

MARCH, 1976

Presented at the 1976 Computer Science Conference

Anaheim, California, February 10-12, 1976.

A LOGIC FOR SEMANTIC NETWORKS

Robert Bechtel

Stuart C. Shapiro

Computer Science Department

Indiana University

Bloomington, Indiana 47401

Abstract

Attempts to use classical logic in natural language understanding systems have met with varying degrees of success, none of them total. This paper describes a non-standard logic developed for use in a semantic network based system, and the associated network representations of logical constructs. The starting point is a four-valued logic proposed by Belnap as a method of dealing with incomplete and contradictory information. The primary changes are the introduction of non-standard connectives and quantifiers. The connectives introduced, ⋎ (AND-OR) and Θ (THRESH), are generalizations of the familiar symmetric binary connectives. Negation and a form of implication are also defined. Finally, representations of constructions of the logic are developed for a semantic network.

Introduction

Question-answering systems, if at all sophisticated, require a method of obtaining information that is not explicitly stored in their data base. Generally, such methods are based on the application of rules which govern the extraction of implicit information from explicit. Such rules may be termed deduction rules. A desirable feature of such rules is an ability to express them independent of the representation in which they may be written (e.g. LISP code). One such possible independent representation is as a logic system. If such an independent, uniform representation is possible, comparisons and contrasts between various sets of deduction rules are easier to make and formulate. Such an approach is followed here.

In practice, deduction rules may be represented separately from the information they apply to, or uniformly with it. The second choice, which is comparable to the uniform representation of data and instructions at the machine language level, seems more desirable, as the rules may then be treated as information by other rules. Again, this approach is followed here.

Network Representations and Connectives

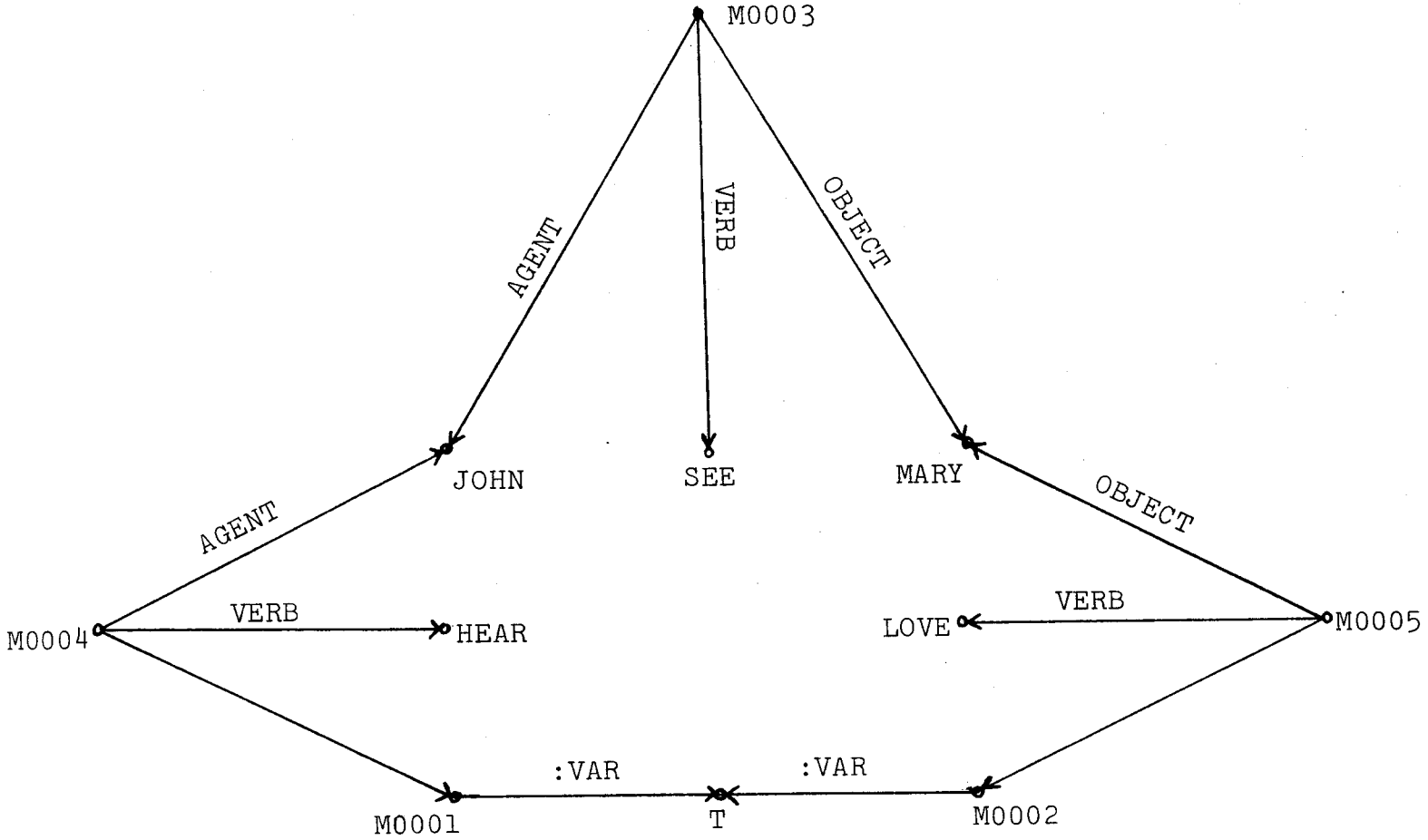
The internal representation of interest here is a semantic network (Shapiro, 1975). The network consists of nodes connected by edges called relations. The nodes may be constants or variables, with constants representing anything about which information may be given - in short, "conceptual entities". Variables act as "empty slots" that range, in general, over any node. Nodes are

generally labelled, solely for convenience in referring to them.

Relations may not represent variable data. When relations are defined, their converses are also defined, and represented in the network whenever their corresponding relation is. Usually (again for convenience), converse relations are not shown in graphic representations of network structures. Conventionally, a relation is an arc between a node and another node which represents less information. Such relations are termed descending, while their converses are ascending. If this convention is followed, it may be said that one node dominates a second if there is a path made up exclusively of descending relations between the first and second nodes.

Nodes may be classified by their dominance over other nodes. Nodes which dominate no others (but which may be dominated) are termed atomic constants or variables, depending on their nature. Variable nodes are indicated by the special relation :VAR, which has no converse, and which relates variable nodes to the special node T. Nodes which dominate other nodes are termed molecular. Molecular nodes which dominate only constants (atomic or molecular) are termed assertions, while those which dominate variables are termed patterns. Examples of network structures are given in figure 1.

Assertions represent constant propositions, while patterns represent propositions with free variables. Unfortunately, not all propositions are simple, but rather are compounded out of a number



Constants - M0003
JOHN, MARY, SEE, HEAR, LOVE
Variables - M0001, M0002
Patterns - M0004, M0005
Assertion - M0003

Figure 1. Network Structures

of other propositions. Traditionally, such compounding has been accomplished by use of standard connectives from classical two-valued (TV) logic, & (AND) and v (OR). Such compounds can be represented in the network by introducing more relations, as shown in figure 2.

This approach has drawbacks, one of which is due to the binary nature of the connectives. Each connectives can take only two arguments at once. We may write A&B&C, but the expression is meaningful only through adopted conventions regarding association (to the left or to the right). To be strictly correct, the expression should be written either A&(B&C) or (A&B)&C. This seems like a minor problem, but does in fact cause trouble by expanding network structures enormously, as shown by the network structure required for the conjunction of five arguments in figure 3. However inconvenient though, this problem by itself does not seem to be sufficient grounds for replacing & and v with something else.

Another problem also concerns the network size required for a correct representation, and may best be demonstrated by use of an example. Imagine a species of creature which may be uniquely identified by checking for five physical characteristics. Every member of the species possesses at least two and no more than three of these five characteristics. Representing "at least two and no more than three of five" using TV connectives is not simple, as shown below.

$$((C_1\&C_2)\vee(C_1\&C_3)\vee(C_1\&C_4)\vee(C_1\&C_5)\vee(C_2\&C_3)\vee(C_2\&C_4)\vee$$
$$(C_2\&C_5)\vee(C_3\&C_4)\vee(C_3\&C_5)\vee(C_4\&C_5))\&\sim((C_1\&C_2\&C_3\&C_4)\vee$$
$$(C_1\&C_3\&C_4\&C_5)\vee(C_1\&C_2\&C_3\&C_5)\vee(C_1\&C_2\&C_4\&C_5)\vee(C_2\&C_3\&C_4\&C_5))$$

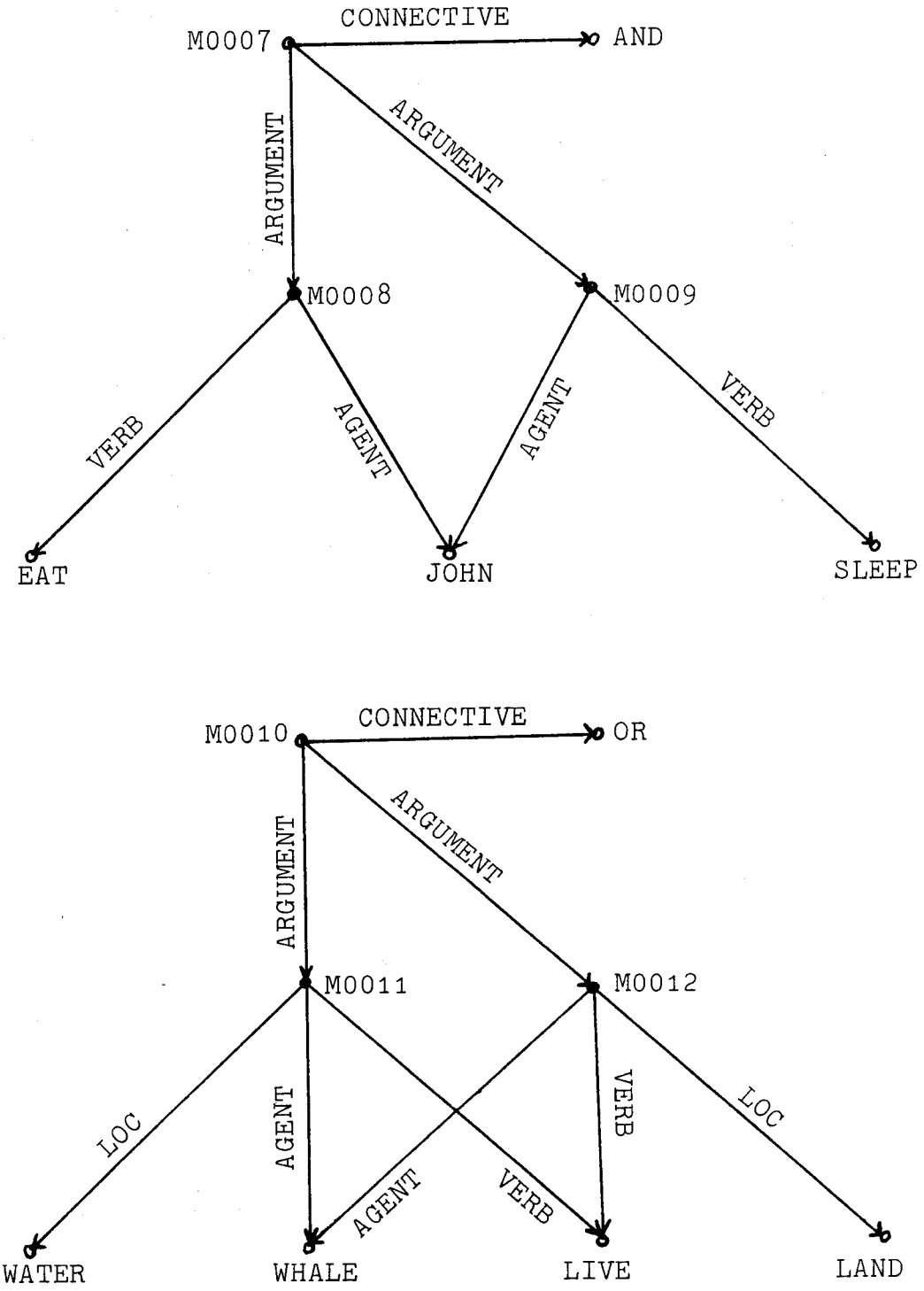


Figure 2. Compound Propositions using & and v.

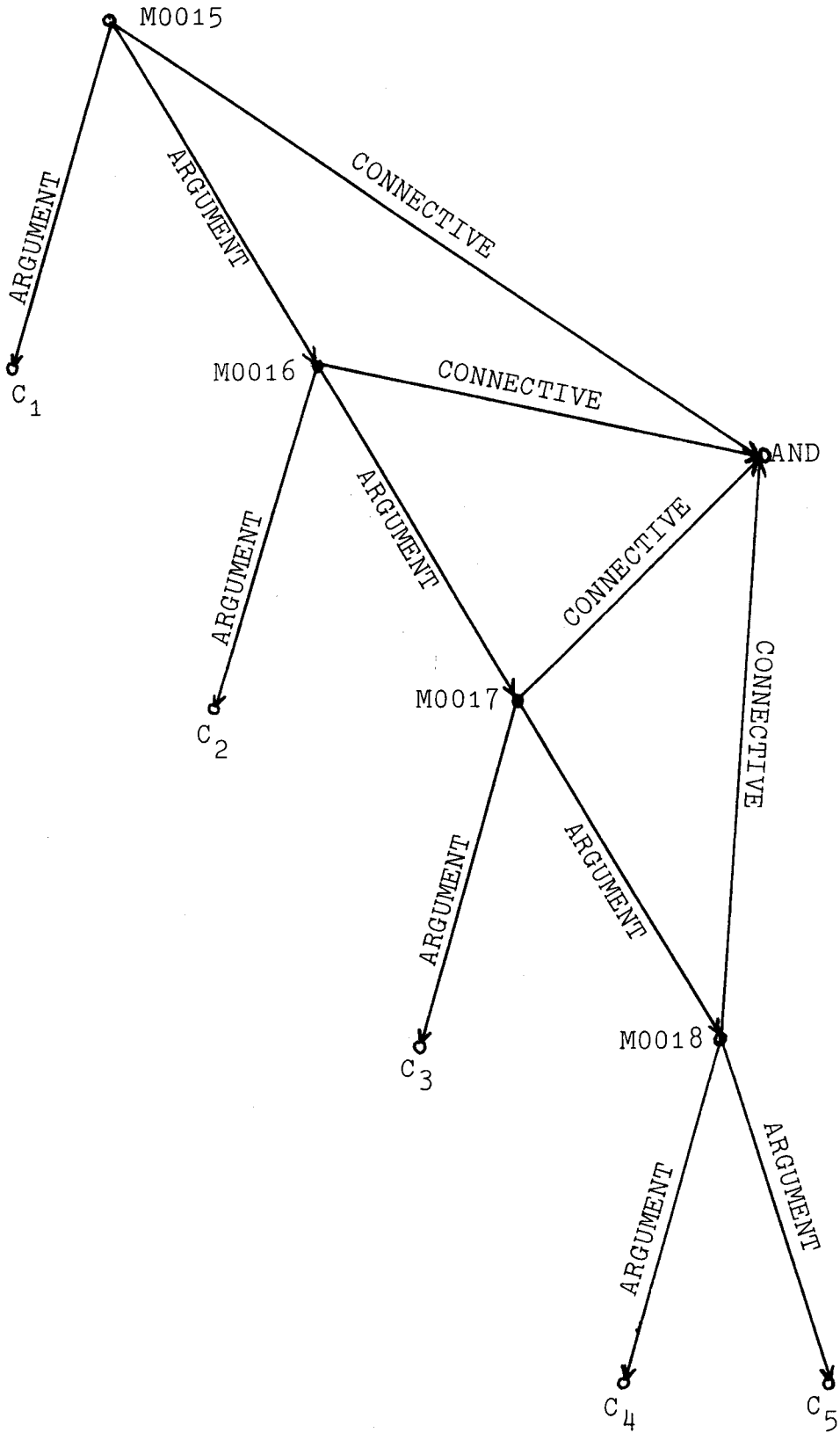


Figure 3. Conjunction of Five Arguments.

Not only is this long and involved in standard TV notation, but would require a large amount of network to reproduce. In addition, the standard TV formalism requires the use of negation (~ or NOT), a problem yet to be dealt with.

A third problem also arises from the binary nature of the primitive connectives. This problem appears with the use of "derived" connectives such as exclusive or (XOR or $\oplus$) and equivalence (EQUIV or $\equiv$). Not only can their use expand the network, either in the ways already discussed, or by requiring expansion to primitives, but they may also trip the unwary who expect them to follow the user's intuition. For the simple binary case, they are harmless enough, but an attempt to use them for more than two arguments can result in disaster even though they are associative. Consider XOR. Using it in the two argument case, one develops an intuition which runs something like "it's true if and only if only one of its arguments is". This appears to work for the three argument case, until all the arguments are true, when $A \oplus B \oplus C$ is true too. Intuition strikes out. A similar situation holds for $A \equiv B \equiv C$ as shown in the truth table below.

A	B	C	$A \oplus B \oplus C$	$A \equiv B \equiv C$
F	F	F	F	F
F	F	T	T	T
F	T	F	T	T
F	T	T	F	F
T	F	F	T	T
T	F	T	F	F
T	T	F	F	F
T	T	T	T	T

We expect A≡B≡C to be true when all its arguments are the same, and it just isn't so. Representing the intuitive understanding of A⊕B⊕C and A≡B≡C by properly associated binary connections is no easy task. The seemingly innocent statement heading many theorems, "the following are all equivalent" looks like this when seven things follow:

$$(A\equiv B)\&(B\equiv C)\&(C\equiv D)\&(D\equiv E)\&(E\equiv F)\&(F\equiv G),$$

when it seems that conjunction has nothing to do with it. This is a somewhat bigger problem, as we would like to represent as simply as possible intuitive relationships and rules used by people in making sense of natural language.

To resolve these problems, we introduce two non-standard connectives: ⋎ (AND-OR) and Θ (THRESH). These will replace AND, OR, EQUIV, and, to a certain extent, NOT.

⋎ is the connective that will do most of the work, and is the same as a connective which was independently developed by George Epstein for a different purpose (Epstein, 1958). ${}_n\curlyvee_i^j$ takes a set of n arguments (propositions) and is true if at least i and no more than j of its arguments are true. It is, by definition, an n-ary connective, and can readily replace AND and OR as shown:

$${}_2\curlyvee_1^2(A,B) = A\vee B$$

$${}_2\curlyvee_2^2(A,B) = A\&B$$

In addition, ⋎ enables an easy formulation of the species membership example discussed earlier. In this formalism, the conditions for membership may be stated

$$_5\text{⋇}^3_2(C_1,C_2,C_3,C_4,C_5).$$

The intuitive understanding of an n-ary XOR is also very simple:

$$_n\text{⋇}^1_1(A_1,\ldots,A_n).$$

Possibly the most unexpected benefit of ⋇ is its ability to replace negation in most, if not all, cases. By setting the minimum and maximum parameters both to 0, it functions as NOR, but without any associative difficulties that might occur, as with XOR. Also, since it accepts any number of arguments, direct negation is possible.

Of course, we could replace EQUIV with appropriately nested ⋇'s, e.g.

$$A \equiv B = {}_2\text{⋇}^1_1({}_2\text{⋇}^2_2(A,B),{}_2\text{⋇}^0_0(A,B)),$$

but to avoid reintroducting the problems of proliferation we wish to eliminate, we instead introduce ⊖. This connective grew out of discussions by the SNePS Users' Group during 1975 concerning network representation of natural language constructions. $_n\ominus_i$ takes n arguments and is true if less than i or all n are true. The theorem with seven equivalent statements becomes

$$_7\ominus_1(A,B,C,D,E,F,G).$$

Notice that EQUIV has been generalized by ⊖ so that i becomes a sort of threshold, which, when reached, requires the remaining arguments to be true. This is useful in cases where the truth of some subset of conditions guarantees the truth of the entire set.

The Failure of TV

Up to this point, the usefulness of classical logic has been assumed, with the connectives merely replaced with ones better suited to the purposes at hand. Now we will discuss some of the problems inherent in TV logic. The first has faced logicians since at least as far back as Aristotle, and has created problems for others working in natural language understanding. TV assumes that every proposition has a known truth value, that it is either true or false. Aristotle tried to assign a truth value to the proposition "There will be a sea-battle tomorrow at noon," and finally decided it was impossible. Even without the temporal element to add confusion, any natural language question answering system is faced with the possibility of a reference to something outside its carefully prepared domain of discourse. What truth value should be assigned to a proposition never before encountered? Of course, answering that question is part of the function of deduction rules, but even they may not be useful. The same problem is more apparent if the system is asked to reply "true" or "false" to some proposition, rather than only having to store some truth value for each proposition.

The obvious answer is that the proper reply when faced with insufficient information is "I don't know." "I don't know" is not, however, a truth value. It is the indication of a total lack of a truth value. In some sense, it is undetermined. Propositions which the system does not know about cannot be marked "true" or "false" but must remain unmarked, giving a third possible truth value to be dealt with, should it be introduced as such.

Another problem is probably just as old. This is the problem of contradiction. Most systems of logic exclude contradictions by careful selection of postulates, but a number of features of natural language systems make this a difficult method for use in that application. Again, TV logic assumes every proposition has one, and only one, of the two possible truth values, when in fact the "propositions" a natural language system has to deal with may have more than one value. This is particularly noticable when the system accepts without question any new information it is given as "fact". The system may then be told, by different informants, that a particular proposition is both true and false. Contradictions which arise in this way we will call explicit contradictions.

Contradictions may also arise by being introduced by the postulates or deduction rules, a manner which is avoided by most formal logical systems. There is no guarantee that deduction rules which are useful in understanding natural language are consistent. Furthermore, observations of human information processing and belief systems would indicate just the opposite. Contradictions arising from deduction rules may be termed implicit.

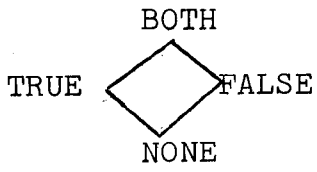
Finally, contradictions may arise out of the ambiguous nature of natural language itself. This ambiguity makes explicit contradiction more likely, if unnoticed, and may also create implicit contradiction, if deduction rules are taken from natural language.

We are not yet prepared to launch a full scale attack on contradiction, but rather wish to show that it can exist in a natural language question answering system without straining. At present,

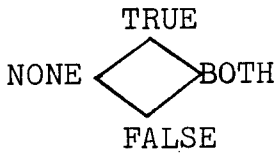
we only wish to argue that, since it can exist, we should have a way of representing it. Using the tag or label idea implicitly advanced in the discussion of unknown truth value, it would be possible to have two copies of every proposition, one labelled "true" and the other labelled "false". This is the method used in TV logic, with the prototypical statement of contradiction being A&~A. There are two objections to this scheme. One is our old friend, the problem of network size. Duplicating every contradiction might be alright if there are just a few, but we have no reason to believe that there will be just a few, or that they will be small, simple propositions. The second is a sort of self-protection against errors. We would like to make any data base check as simple as possible. If the contradictions are "stored" on two separate propositions, finding a single labelled proposition only means that the search is half complete. In the case that a proposition is true, the entire data base must be searched to be sure that an identical proposition is not false, signaling a contradiction. Having established the legitimacy of an empty label for the case of an unknown truth value, we now propose the label "both" or "true,false" to signal a contradiction. Now there are four truth values which a proposition may take, and we are faced with the problem of devising ways of dealing with them outside classical TV logic.

Is there some relation between these truth values, as there is in TV? For an answer, we turn to Belnap, 1975a,b and the idea of an approximation lattice. An approximation lattice is just a complete lattice which also satisfies the intuitive condition that

its partial ordering relation may be read "approximates" (e.g. $x \leq y$ is read "x approximates y"). Given our four truth values, an intuitive understanding of "approximates" might be "gives no more information than." With this understanding, the four values form the following lattice (called A4 by Belnap):



This approximation lattice may then be manipulated with a desire to preserve classical TV characteristics to produce L4,



in which & is meet and v is join, and negation is defined as follows:

A	NONE	FALSE	TRUE	BOTH
~A	NONE	TRUE	FALSE	BOTH.

For convenience in notation, we shall refer to these four truth values by their first letters, N, F, T, B. So now we have four truly glorious truth values, but are faced with the old connectives. Fortunately, the replacement of &, v, and ~ will be quicker this time. ${}_n\mathbb{W}^j_i(A_1,\ldots,A_n)$ is marked at least T just in case at least i and no more than j of its arguments are marked with at least T (in A4). ${}_n\mathbb{W}^j_i(A_1,\ldots A_n)$ is marked at least F just in case less than n-j or more than n-i of its arguments are marked with at least F (in A4). ${}_n\Theta_i(A_1,\ldots,A_n)$ is marked at least T just in case less than i or all n of its arguments are marked at least T, and is

marked at least F if more than 0 but less than or equal to n-i of its arguments are marked at least F.

Finally, we must have a way of representing these connectives in the network. Since the number of arguments may vary and order is unimportant, a single type of descending relation could serve to join arguments to a node representing a connective. We define such a relation, called ARG. The parameters on a connective provide the information necessary to its evaluation, and so must be included in the network. On ⊻, the parameters indicate the total number of arguments and both the minimum and maximum number that may be true to satisfy the connectives. We define the three relations TOT, MIN, and MAX to represent these. ⊖ may also use TOT, and additionally requires an indication of the threshold number of arguments, for which we will use THRESH. Examples of network structures may be seen in figure 4.

Having established a useful set of truth values and more convenient connectives, we still lack one of the most important parts of a logic, the ability to use known information to infer new information. In TV, this ability is vested in material implication. However, the suitability of material implication for such uses has been attacked by logicians, and one of their objections is particularly relevant to a question answering system. The objection, simply stated, is that a false proposition implies anything, and a true proposition is implied by anything. Do we wish to say that any implication whose antecedent is false holds? Again, consider the problem of contradiction. A contradiction (at least classically) is always false, and so implies anything. Anderson and Belnap

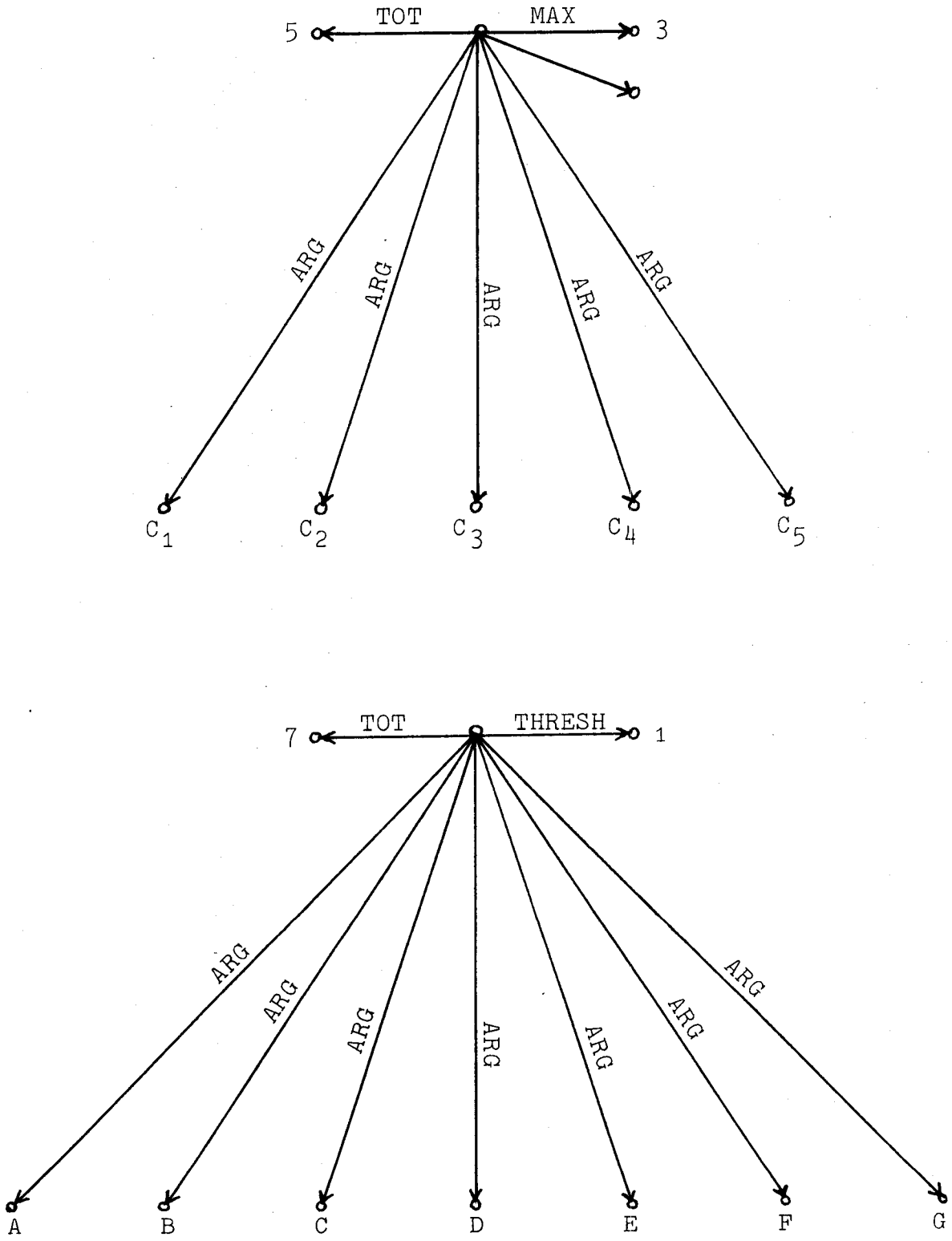


Figure 4. Network Representations of ⋀ and Θ.

(1975) have made perhaps the broadest and most complete criticism of material implication, in which they assert "Material implication is no more a 'kind' of implication than a blunderbuss is a 'kind' of buss." Along with the criticism, they propose alternative relations, which are intended to more closely approximate the intuitive understanding of implication. One of these surfaces again in Belnap, 1975a,b, and is simply defined on L4 as "going uphill". Moving from the antecedent to consequent (in the lattice) must not involve taking a downward path, or the implication (now an entailment, in Anderson and Belnap's terms) is false. Rather than being always (at least) T, an entailment whose antecedent is contradictory is now T only if its consequent contains T. Truth tables for ⊃ (defined as ~A∨B) and entailment (→) show other interesting differences.

⊃	N	F	T	B
N	N	N	T	T
F	T	T	T	T
T	N	F	T	B
B	T	B	T	B

→	N	F	T	B
N	T	F	T	F
F	T	T	T	T
T	F	F	T	F
B	F	F	T	T

Notice in particular that every entailment has a single, well-defined truth value (T or F) and is never undetermined or contradictory. Consider the expression A→A. If we have a well behaved logic, we would certainly hope that this implication would always be true, regardless of the truth value assigned to A. As can be seen, this is not the case for ⊃. As Belnap has noted, → also preserves truth and falsity, in the sense that it never leads from truth to the absence of truth or from the absence of falsity to its presence. ⊃ is guilty on both counts of non-preservation, as can be seen from the truth table.

Having selected a suitable implication connective, we must now find a way to represent it in the network. Entailment differs from the simpler connectives & and v in that, while it is also binary, it is <u>not</u> symmetric. B→A does not necessarily follow from A→B. Generalizing to any number of arguments, as was done in ⋀ and ⊖, we have $(A_1,\ldots,A_n)\rightarrow(B_1,\ldots,B_m)$ which is equivalent to the conjunction of the nm rules, $A_i\rightarrow B_j$, $1\leq i\leq n$, $1\leq j\leq m$. As in ⋀ and ⊖, a use of → will be represented as a node, with descending relations to the arguments. Since the two sets of arguments must be distinguished, we use two different descending relations, called ANT and CQ, intended to be suggestive of antecedent and consequent respectively. An example is shown in figure 5.

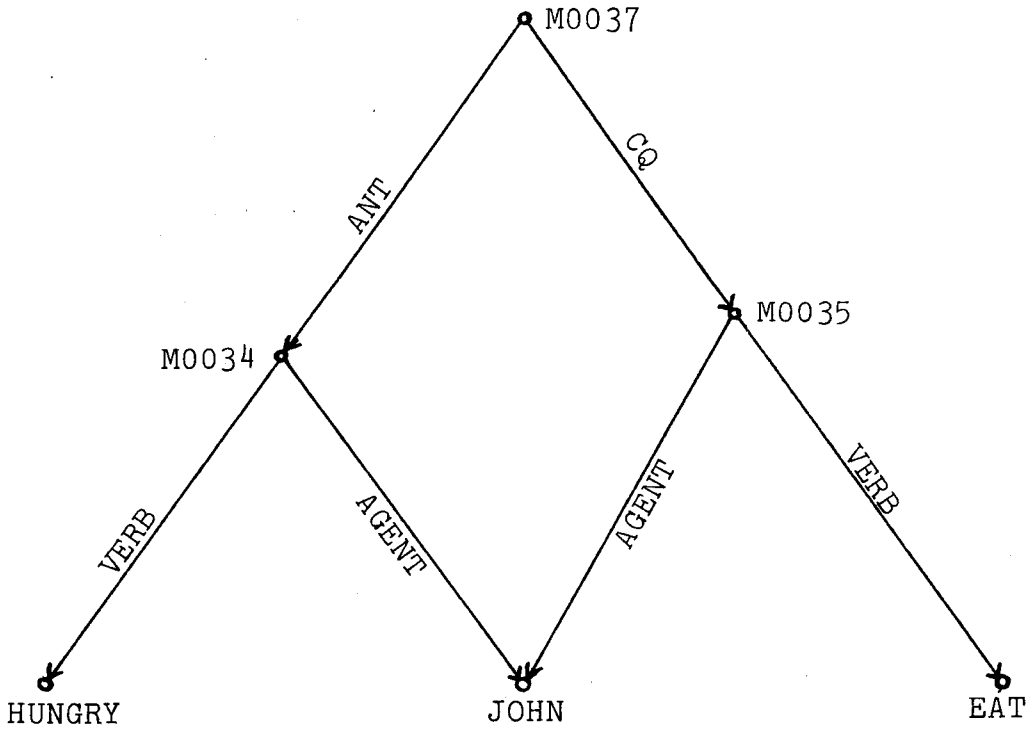


Figure 5. Network Representation of ANT-CQ.

Quantifiers

Now that connectives and truth values are settled, it would be desirable to include more than constants in the deduction rules. In classical logic, variables are included through the use of quantifiers and predicates; a similar path will be followed here. Assertions and patterns, which have been funny looking propositions (being made up of many parts, rather than a single entity) now may be viewed as predicates. The variables in the patterns range (generally) over the universe of nodes in the network, being validly substituted for by other nodes. Examples of valid substitutions are shown in figure 6.

The classical quantifiers, $\forall$ (ALL) and $\exists$ (EXISTS), are retained with the following interpretations. $\forall$ states that it is impossible to make a substitution for the variables it binds which will invalidate the expression within its scope. $\exists$ states that there is at least one substitution (whether known or unknown) which will make the expression within its scope valid. It is possible, through the use of these quantifiers and connectives, to express a wide range of natural language constructions. However, certain of these constructions create network representation difficulties analagous to those of classical connectives; the size and complexity of the network expands out of all proportion to the intuitive concepts expressed. For this reason, three non-standard quantifiers are introduced.

The first of these non-standard quantifiers is NONE, which is a direct implementation of $\sim\exists$. NONE states that there is no substitution which can validate the expression in its scope. As negation is available so far only through an ✕ construction with appropriate parameters, the simplification this quantifier will allow in the network is obvious.

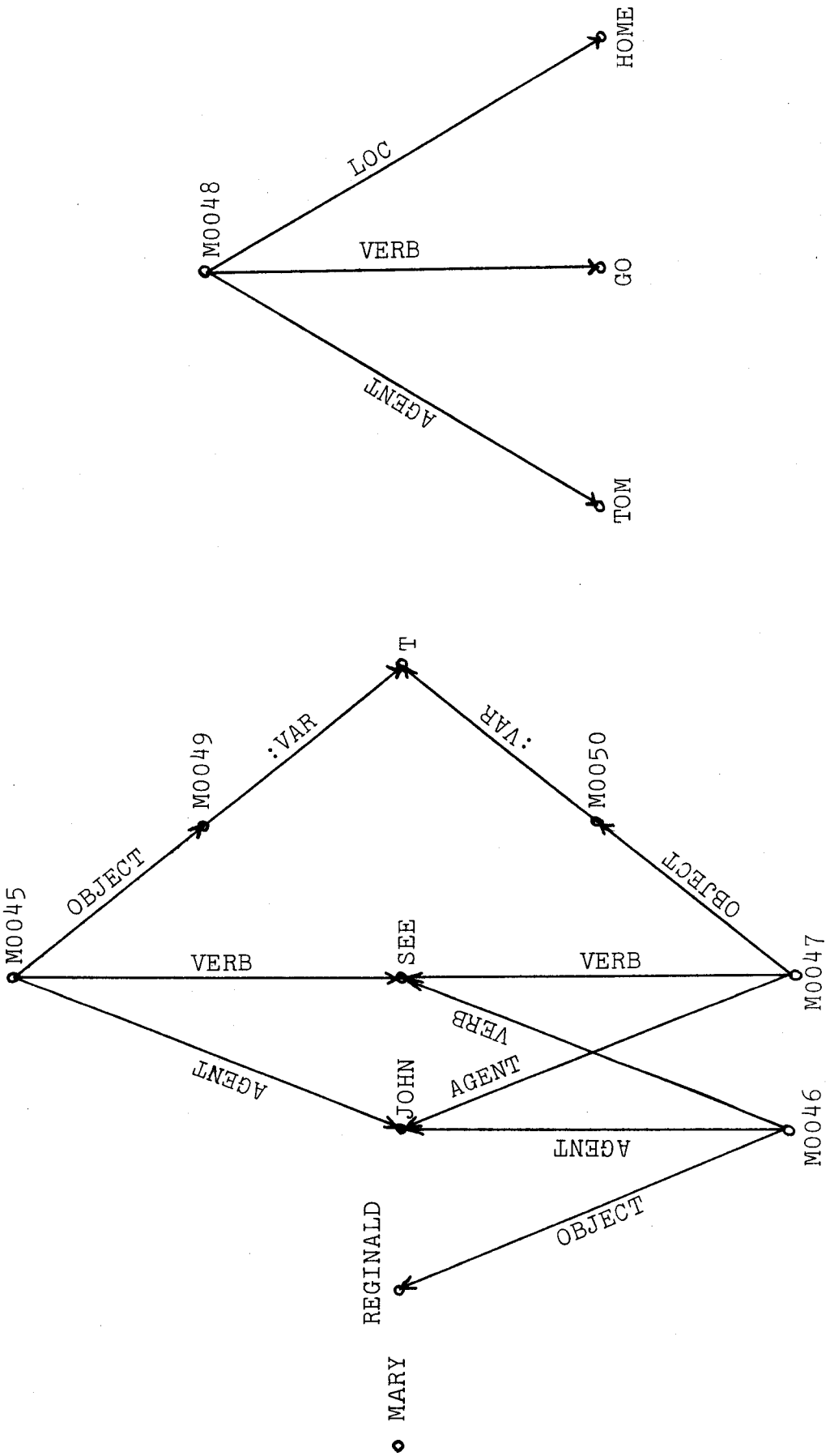


Figure 6. Valid Substitutions

MARY, REGINALD, JOHN, SEE, TOM, GO, HOME, M0050, M0049, M0048, M0047, M0046, and M0045 are all valid substitutions for M0049 (X)

Even greater simplification is made possible by use of the second non-standard quantifier. ONE states that there is one and only one substitution which will make the expression within its scope valid. ONE(x)(P(x)) will replace the complex $\exists(x)(P(x)\&\forall(y)(P(y)\rightarrow(x=y)))$, and additionally avoid the problem of representing equality.

The last quantifier introduced is ALMOST-ALL. In spirit, this quantifier replaces $\exists\sim$. ALMOST-ALL (ALA) states that there may be some substitution which invalidates the expression within its scope. Again the savings in the network is due primarily to the elimination of negation. By adopting a heuristic which states "contradictions to an ALA expression must be explicit," the ALA may be used as an $\forall$ unless such an explicit contradiction is found.

All of these non-standard quantifiers, in addition to the savings in network size they make possible, also ease the representation of natural language constructions. "No one in their right mind would try that", "there is no escape", and similar constructions are readily handled by use of NONE. ONE is useful in representing unique constructions, such as "the present king of France" and "automobiles have one engine". Perhaps the most useful new quantifier is ALA.

Many statements are made that are treated as true, unless specifically contradicted. Examples of this would be "mammals live on land" and "if someone used to own something, he still does, unless you know otherwise." ALA can be used to represent such cases. The possible existence of an instantiation which would invalidate the expression is guaranteed by the ALA. In the second example, "unless

you know otherwise" signals the possibility of a contradiction and forces the use of an ALA. The fact that "ALL" does not preface "mammals" in the first example signals possible contradiction forcing the use of an ALA. Once again notice that in the absence of an explicit contradiction, ALA and $\forall$ are treated identically.

The concept of likelihood of valid substitutions, as reflected in the preceding interpretations of quantifiers, may be used to reach a view of the applicability of rules. It would seem that the more likely the possibility (or impossibility) of valid substitution, the more generally applicable the rule is. We are concerned now only with quantifiers over an entire deduction rule. Quantification over portions of a deduction rule is covered elsewhere (Shapiro, et al, forthcoming).

$\exists$ and ONE need not even be considered in such an explication of applicability, as they may be replaced by constants which represent the variables they quantify, as in a Skolem function. This leaves only $\forall$, ALA, and NONE. ALA seems to state, by its distinction from $\forall$, that there may be cases in which it is not applicable. $\forall$ and NONE show no such restrictions. With the idea of applicability, quantifiers of deduction rules can be used to order proof trees. If, for example, two rules are available for use to reach a particular conclusion, it is possible that one will be more generally applicable than the other. In terms of anticipated effort required to check the applicability of a rule, it is possible to order the quantifiers from least to greatest effort (left to right) as shown below.

ALL
NONE ALA

Rules can then be selected on

the basis of effort anticipated to use them, thus (hopefully) reducing the amount of effort expended on a given proof or derivation.

Now it is necessary to define a network representation for these quantifiers. We define a relation, Q, which relates a pattern which represents the expression to be quantified to a node representing the quantifier. The node linked to the quantifier should be a pattern rather than an assertion, as there is no sense in quantifying over constants. To indicate which variables are bound by the quantifier, a relation named VB is inserted between the pattern node and those variables bound by the quantifier related to the pattern node. If a pattern node is related to a quantifier and all the variables it dominates are bound by VB relations from that pattern node or other pattern nodes it dominates, the pattern node represents a deduction rule with no free variables, and is thus an assertion. An example of a network representation of a quantified deduction rule is shown in figure 7.

Final Network Representations

The last construction needed for a useful logic system is a method if indicating the truth value of assertions and patterns. This is handled by introducing an edge, called TVAL, which relates an assertion or pattern node to another node which represents the truth value of the expression represented by the first node. This method grows out of the earlier idea of labelling, and examples are shown in figure 8. If a specific truth value is associated with an antecedent in a deduction rule, an explicit construction

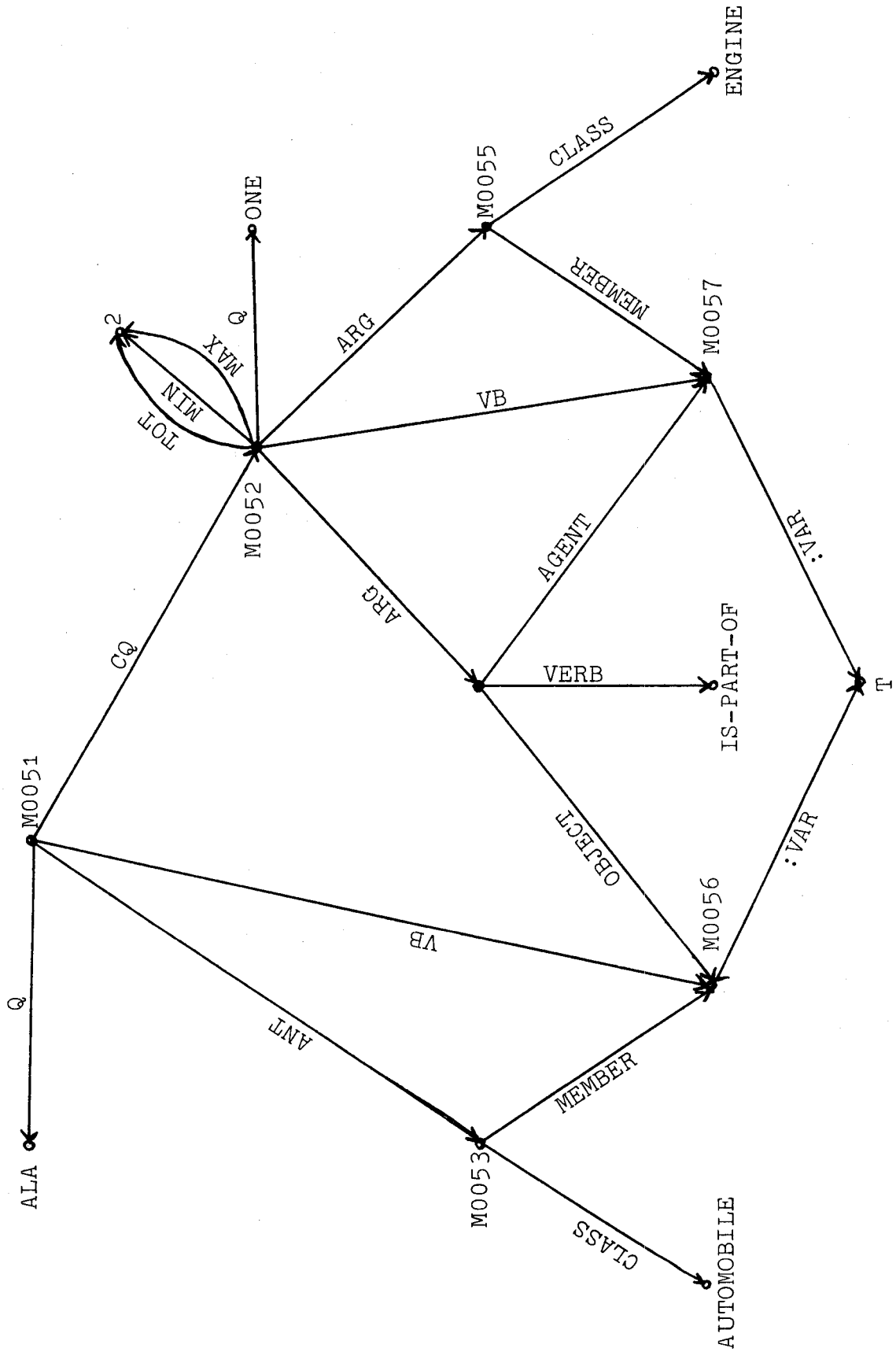


Figure 7. Network Representation of the Deduction Rule 'Almost all automobiles have an engine'.

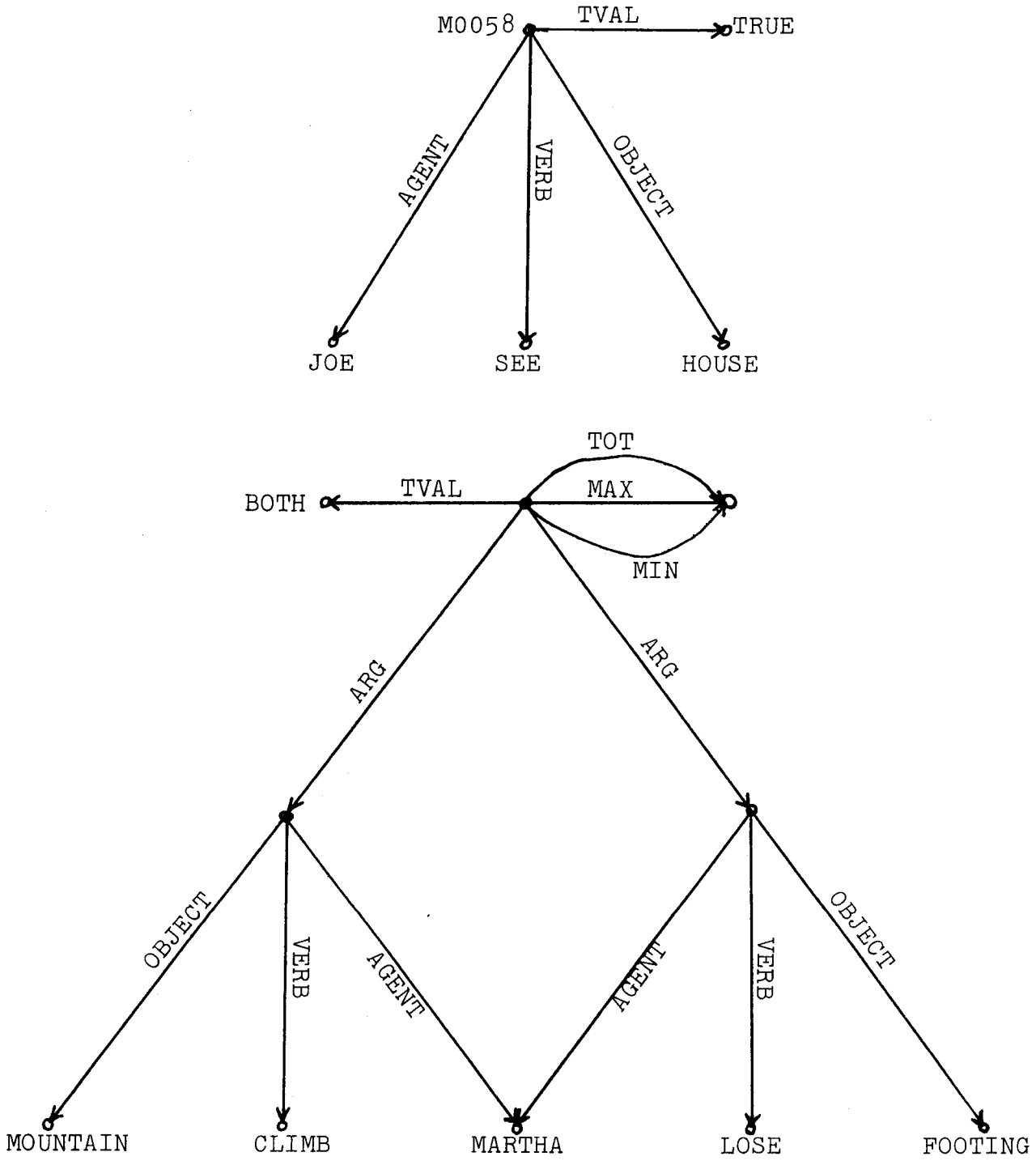


Figure 8. Netowrk Representation of TVAL.

with that truth value must be found or deduced in the network for the rule to be used. One unusual feature needs to be pointed out; expressions not found in the network either explicitly or implicitly (deduced) are assumed to have a truth value of NONE, that is, they are neither true nor false. This accords with the motivation for the establishment of the NONE truth value.

A Logic

The most familiar representation of logical systems is axiomatic, with a number of postulates given which allow derivation of all expressions valid in the logic, and no others. These postulates consist of axioms, or constructions which are defined as valid, and which may therefore be used at any point, and rules, which govern the insertion and deletion of connectives and expressions. Most axiomatic systems consist of axiom schema, which provide a general format for the axioms, which are formed by uniform substitution for the variables used in the scheme. As axiom schema are constructions just as any other, with the exception that they are guaranteed valid, they may be represented directly in the network as deduction rules. The rules of a system are a bit more difficult.

A common rule is modus ponens, or detachment. Application of this rule results in assertion of the consequent of an entailment alone, when the antecedent is true. This process cannot be represented directly in the network, and so must be handled by a procedure that creates (builds) a new assertion which is identical to the consequent of the entailment, provided the appropriate conditions for application of the rule hold. Similar procedures are required for other rules, such as adjunction.

The procedures which provide the power of rules are a part of an interpreter whose purpose is to use explicit information to derive implicit information. The interpreter is the facility which utilizes quantifiers as indicators of effort, and which needs an indication of truth value. It is essentially a theorem prover, with the rules (which may be changed) as a subpart. Since the axioms and rules are not fixed, but may be changed by providing different network constructions and procedures, it is possible to experiment with different logic systems.

The simple logic system presently being implemented was developed along with the network representation now in use. For the most part, it simply grew, with little or no thought given to a formal, representation independent expression of postulates. There are no axioms. The rules, imbedded in the interpreter, are variations on the theme of detachment, designed so that one may, given A→B, infer B directly from information sufficient to show A. The rules which fall strictly into this category are:

1. From ${}_i\mathbb{W}^k_j(A_1,\ldots,A_i)\rightarrow B$ and ${}_n\mathbb{W}^n_n(A_{t_1},\ldots,A_{t_n})$, where $j\leq n\leq k$, $1\leq t_x\leq i$ and $t_y\neq t_z$, to infer B.
2. From ${}_i\Theta_j(A_1,\ldots,A_i)\rightarrow B$ and ${}_i\mathbb{W}^i_i(A_1,\ldots,A_i)$ to infer B.
3. From ${}_i\Theta_j(A_1,\ldots,A_i)\rightarrow B$ and ${}_i\mathbb{W}^{j-1}_0(A_1,\ldots,A_i)$ to infer B.
4. From $(A\rightarrow B)\rightarrow C$ and A and B to infer C.

In addition, two rules permit $\mathbb{W}$ and Θ to be used as the main connective of a deduction rule, allowing a single ARG to be inferred when appropriate information is available about the other ARGs. These rules are very useful, as they, in combination with

the other four, permit a single predicate to be derived from an arbitrarily complex deduction rule. The special rules for $\curlywedge$ and $\ominus$ are:

5. From ${}_{i}\curlywedge^{k}_{j}(A_1,\ldots,A_i)$ and $j>0$ and ${}_{i-1}\curlywedge^{j-1}_{j-1}(A_1,\ldots,A_{t-1},A_{t+1},\ldots,A_i)$ to infer A_t.

6. From ${}_{i}\ominus_{j}(A_1,\ldots,A_i)$ and ${}_{i-1}\curlywedge^{i-1}_{j}(A_1,\ldots,A_{t-1},A_{t+1},\ldots,A_i)$ to infer A_t where $i>j$.

Experimentation with this logic continues. As the interpreter is developed, alternate rule sets will be implemented to permit investigation of other logics as well. In particular, we intend to work with the system R, as proposed by Anderson and Belnap (1975), as it seems particularly well suited to question answering systems (Shapiro and Wand, 1976).

Bibliography

Anderson, A.R., and Belnap, N.D. Jr. Entailment: The Logic of Relevance and Necessity (vol. I), Princeton University Press, 1975.

Belnap, N.D. Jr. How a Computer Should Think. Contemporary Aspects of Philosophy. Proceedings of the Oxford International Symposium, 1975a, forthcoming, 1976.

------ A Useful Four-valued Logic. Modern Uses of Multiple-valued Logic, ed. G. Epstein and J.M. Dunn. Proceedings of the 1975 International Symposium on Multiple-valued Logic. 1975b, Reidel, forthcoming, 1976.

Epstein, G. Synthesis of Electronic Circuits for Symmetric Functions. IRE Transactions on Electronic Computers. May, 1958.

Shapiro, S.C. An Introduction to SNePS. Technical Report No. 31. Computer Science Department, Indiana University, November, 1975.

Shapiro, S.C., Bechtel, R., McKew, J., and Eastridge, N. Implementation of Deduction Rules in a Semantic Network, forthcoming.

Shapiro, S.C., and Wand, M. The Relevance of Relevance. Technical Report No. 46. Computer Science Department, Indiana University, Bloomington, Indiana, 1976.