Question-Answering by a Semantic Network of Parallel Automata

JOSEPH R. FIKSEL AND GORDON H. BOWER

Stanford University, Stanford, California

Human semantic memory is modeled as a network with a finite automation embedded at each node. The nodes represent concepts in the memory, and every arc bears a label denoting the binary relation between the two concepts that it joins. The process of question-answering is formulated as a mathematical problem: Given a finite sequence of labels, find a path in memory between two given nodes whose arcs bear that sequence of labels. It is shown that the network of automata can determine the existence of such a path using only local computation, meaning that each automaton communicates only with its immediate neighbors in the network. Furthermore, any node-concept along the solution path can be retrieved. The question-answering algorithm is then extended to incorporate simple inferences based on the equivalence of certain sequences of relational labels. In this case, it is shown that the network of automata will find the shortest inferable solution path, if one exists. Application of these results to a semantic corpus is illustrated.

The two semantic topics of negation and quantification receive special treatment. Careful study is made of the network structure required to encode information relating to those topics and of the question-answering procedures required to extract this information. The notions of a negated relation and a negated question are introduced, and a negation-sensitive path-searching algorithm is developed that provides for strong denials of queries. For sentences involving universal and existential quantifiers, it is shown how the terminal can translate a first-order language question into a sequence of network queries. In both areas, the network model makes reaction-time predictions that are supported by several experimental findings.

Extensions of the model that would permit the encoding and retrieval of propositional information are mentioned.

1. INTRODUCTION

This paper proposes a new theory of how people answer questions on the basis of information available in memory. This is an important but currently unsolved puzzle for cognitive psychology. A "question-answering system" is normally unde-
stood to mean a computer program capable of: (i) accepting and interpreting propositional information and questions; (ii) storing the information according to some internal representation scheme; and (iii) answering questions by retrieving the relevant information and employing deductive reasoning. The underlying goal in question-answering research is to develop a computational theory that simulates some aspects of the way humans answer questions. However, in artificial intelligence work of this type, the actual psychological processes underlying human performance are usually considered in only a casual, intuitive fashion. Instead, emphasis has typically been placed on achieving efficient programs and data structures that enable computations to proceed within the limitations of present-day serial computers.

The goals of this paper, however, are to be distinguished from these traditional goals of artificial intelligence work. We try to model (at least roughly) the actual psychological mechanisms involved in question-answering, as well as the representational design of human semantic memory. Thus, these efforts would more properly fall under the domain of theoretical psychology. Indeed, the criteria used for testing the validity of the question-answering model are drawn from the growing literature in experimental psychology dealing with human reaction times in question-answering. At the same time, computational efficiency is given no weight at all in the construction of our model. In fact, the parallel-processing search algorithm used here lends itself very poorly to realization on a serial computer. But then, there are no necessary reasons to suppose that memory-searching methods of the brain are analogous to serial rather than parallel processes.

Nevertheless, there are important similarities between this model and previous question-answering systems in terms of the problematics of language representation, memory organization, and inference. Simmons (1969) provides an excellent review of the progress in the field of question-answering. The principal obstacles to a workable system seem to have been (i) correctly analyzing the semantics of a question, and (ii) developing efficient techniques or heuristics for searching through a large database (memory). Following the example of most others, we assume that problem (i) is solvable in principle, and that questions (and statements) are to be transformed (by some unanalyzed semantic processor) into an unambiguous formal language. Coles (1968) has already dealt with this problem with some measure of success.

One of the more successful question-answering systems was created by Green and Raphael (1968), who employed formal mathematical theorem-proving methods (so-called "Robinson resolution" methods) to deduce the answer to a question. Another well-known system was developed by Winograd (1972), who conversed with a hypothetical "robot" operating in a small world of colored blocks on a table. Both of these systems are unconcerned with providing an accurate psychological description of question-answering. Of more direct relevance to our work are the graph-structure belief systems of Colby (1968) and the semantic networks of Quillian (1966), which necessarily sacrifice efficient realizability in order to suggest a plausible
structure for semantic memory. By formalizing the notion of a network of automata with parallel search techniques, this paper builds on the foundations provided by these latter papers.

1.2. The Question-Answering System

The general components and information flow of a proposed question-answering system are diagrammed in Fig. 1. A question formulated in natural language is input to the semantic processor, which encodes it into a precise formal language. The control terminal accepts this formal language query and initiates a search in the memory store for the required information. When the control terminal receives a reply from memory, it outputs the information in the formal language. Finally the semantic processor transforms this output into an answer in natural language.

The scope of this paper includes only the control terminal and the memory, not the semantic processor. Hence, we will not be concerned with all of the thorny issues of synonymity, syntactic ambiguity, and other aspects of natural language that hinder mechanical translation. The focus is rather on the organization of the memory store and the search techniques employed to extract information. We assume that the questions received by the terminal have already been rephrased in terms of the network structure of memory. The elements of the network represent semantic concepts, not words, and several words (e.g., rabbit, bunny, hare) may all map into a single concept.

In simplest terms, information may be roughly divided into items (e.g., any concept that can act as the subject of a proposition) and relations between these items. Each item corresponds to a node of the network, and to every relation between two items there corresponds a directed arc joining the appropriate nodes. The set of arcs (relations) in the network is partitioned into a finite set of types, each type represented by a label on its arcs. For example, the phrase "rabbits have stomachs" could be represented as two nodes, RABBIT and STOMACH, with an arc labeled HAS-AS-
PARTS directed from RABBIT to STOMACH, as illustrated in Fig. 2(a). It is assumed that a certain finite set of concepts and relations is used as "primitives," from which complex meanings can be constructed. Schank (1969, 1972) is developing a system of semantic primitives along with a parser (semantic processor) that will encode natural-language phrases into this system.

Our theory supposes that each node of the network contains a finite-state automaton whose input is the state of each of its "neighbors" in the network, and whose state-changes are a function of these inputs. Thus, we postulate a network of identical automata, each interacting only locally with its neighbors as it undergoes state transitions from moment to moment.

We also postulate a control mechanism, external to the network, called the terminal, that may transmit and receive simple pulses to and from the network automata. An automaton can be "excited" by the terminal and can notify the terminal if it achieves a particular, designated state. Otherwise, however, the operation of the automata proceeds without any global control.

In general, the state-transition rules for the automata could be probabilistic, as exemplified by simple word-association norms. However, we will assume that question-answering is basically a deterministic search procedure. For example, suppose that the terminal receives the question, "Does a rabbit have a stomach?" The query causes the terminal to excite the two node-automata RABBIT and STOMACH. Since it is seeking a path in the network of the form shown in Fig. 2(a) or Fig. 2(b), the state-transition function is specified roughly as follows: RABBIT sends a signal along all outward-directed arcs that are labeled HAS or SUBSET OF; similarly, STOMACH sends a signal along all inward-directed arcs that are labeled HAS. (The exact manner in which these signals propagate through the network is explained in Section 2.2). When some intermediate node receives the proper signals from two directions, it notifies the terminal that an "answer" path has been found, and the terminal composes an appropriate answer.

If a notification is received, the answer returned is YES. If no notification is received after a certain time, the answer is NO. In this case, the NO answer merely indicates the absence of a path in the network of the desired type. Later we will discuss stronger forms of negation and the return of factual answers rather than simply YES or NO.
2.1. Mathematical Formulation

A directed network may be formally defined as a doublet \((N, A)\), where \(N\) is a (possibly infinite) set of nodes, and \(A \subseteq N \times N\) is a set of ordered pairs corresponding to arcs of the network. Thus, if \((x, y) \in A\) there is a directed arc running from node \(x\) to node \(y\), where \(x, y \in N\), and \(x\) and \(y\) are said to be adjacent.

A labeled directed network may be defined as a triplet \((N, A, \mathcal{T})\) where \(N\) is a finite set of nodes, \(\mathcal{T}\) is a finite set of relational types, and \(A \subseteq N \times \mathcal{T}\). That is, if \((x, y; u) \in A\), then there is an arc from \(x\) to \(y\), with the arc labeled as type \(u\). Now, the degree of a node is the number of arcs incident to that node, either incoming or outgoing. Let the maximum degree of \((N, A, \mathcal{T})\) be \(d\).

The type of automata used in this paper are quite simple mathematically, and may be imagined as little demons with exactly \(d\) “limbs” or tentacles radiating from a central cell. For example, \(d = 5\) in Fig. 3. Let \([d] = \{1, 2, 3 \ldots d\}\); then an arbitrary numbering of the limbs can be established by associating each limb with an element of \([d]\). We will embed these automata in the network by placing one at each node. All of the automata in the network are assumed to be identical. Henceforth, we refer to the automaton at node \(x\) as automaton \(x\).

![Fig. 3. Two adjacent node-automata with \(d = 5\), \(\rho(x, 3) = (y, 5)\), and \(\rho(y, 3) = (y, 3)\).](image)

The limbs of each automaton are connected to the limbs of other automata in such a way that every arc \((x, y; u) \in A\) corresponds to a connection between one limb of \(x\) and one limb of \(y\). More precisely, we define a mapping \(\rho: N \times [d] \rightarrow N \times [d]\) that maps every limb into itself or some other limb. \(\rho(x, r) = (y, s)\) means that the \(r\)th limb of automaton \(x\) is connected to the \(s\)th limb of automaton \(y\), implying an arc between \(x\) and \(y\). Note that \(\rho\) composed with itself is the identity mapping, since \(\rho(\rho(x, r)) = \rho(y, s) = (x, r)\).

Unless node \(x\) has degree \(d\), some of the limbs of \(x\) will remain unconnected. If \(\rho(x, r) = (x, r)\), we say that limb \((x, r)\) is not used, or “dead.” In effect, it is connected to itself; by this device, \(\rho\) is defined over the entire set of limbs in the network. The purpose of the dead limbs is to make all the automata identical, regardless of the effective degree of their nodes.

Let \(\mathcal{T}\) be a set with the property that for every \(u \in \mathcal{T}\), there exists an inverse relation of type \(u^{-1} \in \mathcal{T}\); that is, for every \(x\) and \(y\) the arc \((x, y; u)\) is equivalent to the arc \((y, x; u^{-1})\). Thus, the direction of an arc may be arbitrarily chosen. If \(u = u^{-1}\), the relation \(u\) is symmetric, and the arc may be considered undirected.
Let \( g_x = (g_x^1, \ldots, g_x^u) \) be the relational vector associated with automaton \( x \). \( g_x^r \in \mathcal{F} \) is the relation on the arc corresponding to the limb \((x, r)\), assuming this arc is directed away from \( x \). This means that if \( \rho(x, r) = (y, s) \) and \( g_x^r = u \), then \( g_x^y = u^{-1} \). Of course, dead limbs are assigned the null relation, \( \emptyset \in \mathcal{F} \).

Now the structure of a network of automata has been completely described from a topological point of view. Next, we discuss the dynamics of this network under the condition that an automaton may undergo a "change of state" at successive discrete instants of time, \( t = 0, 1, 2, 3, \ldots \). Formally, an automaton is defined as a triple \((\mathcal{S}, \mathcal{L}, \phi)\), where:

\( \mathcal{S} \) is the set of possible states of the automaton at any given instant \( t \). The state of automaton \( x \) at time \( t \) is written \( e_{x,t} \), and is actually a \( d \)-component vector \((e_{x,t}^1, \ldots, e_{x,t}^d)\), where \( d \) is the number of limbs of the automaton. Thus a state \( e_{x,t} \in \mathcal{S} \) is only a composite of \( d \) elementary states, one corresponding to each limb. We assume \( \mathcal{S} = \mathcal{E}^d \), where \( \mathcal{E} \) is the finite set of elementary limb states.

\( \mathcal{L} \) is the set of possible input vectors to the automaton. The input to automaton \( x \) at instant \( t \) is written \( \lambda_{x,t} \). For each automaton \( x \), the time-invariant relational vector \( g_x \) forms one part of the input vector. The other part is the vector \( f_{x,t} = (f_{x,t}^1, \ldots, f_{x,t}^d) \), which lists the states of all limbs connected to those of \( x \). In particular, if \( \rho(x, r) = (y, s) \), then \( f_{x,t}^r = e_{y,t}^r \). Verbally, the time-varying input to \( x \) consists of a portion of the states of all node-automata adjacent to \( x \). Hence, \( \lambda_{x,t} = (f_{x,t}^1, g_x) \), and \( \mathcal{L} = \mathcal{E}^d \times \mathcal{F}^d \).

\( \phi \) is a transition function that determines the state of the automaton at time \( t + 1 \), given its state and its input vector at time \( t \). Thus, \( \phi: \mathcal{S} \times \mathcal{L} \rightarrow \mathcal{S} \), and in our notation \( e_{x,t+1} = \phi(e_{x,t}; \lambda_{x,t}) \).

The sequence of states of each automaton in the network is determined solely by the states of its immediate neighbors. This is what is meant by the phrase "local interaction." Thinking again in terms of the metaphor of the little demons, none of them realizes that it is part of a large network. All that each knows is that its tentacles are contacting those of a number of neighboring, identical demons, and that they are in a particular state at time \( t \).

Abandoning the metaphor and returning to the mathematical formulation, such networks of finite automata have a remarkable property, which we will exploit throughout our work. If the set \( \mathcal{S} \) and the function \( \phi \) are appropriately chosen, the automata can perform complex tasks involving properties of the network as a whole, in the absence of any global guidance. Rosenstiehl, Fiksel, and Holliger (1972) deal extensively with the problem-solving capabilities of automata in non-labeled graphs. Below we demonstrate that automata are capable of performing question-answering in the special case of a labeled directed network. The only communications of the automata with the terminal are at time 0 and at the end of
their operation. This is tantamount to switching the network on and allowing it to change state "automatically" until it switches itself off. The set of automata compute locally and in parallel until they reach a stationary terminating state.

The advantage of this type of computation is that the automata behave much like neural packages, reacting only to immediate stimuli and ignoring their previous history of state transitions. In contrast, a global intelligence would have to store tree-like lists of information to conduct a question-answering search.

2.2. THE FUNDAMENTAL ALGORITHM

Now we will show in detail how the network of automata deals with the most simple class of questions. Extensions to more complex cases will occur in due course. Given a sequence \( q \) of relations and two nodes \( x \) and \( y \), the terminal is asked to investigate the existence of a path between \( x \) and \( y \) whose labeled arcs match the sequence of relations in \( q \). When an arc labeled \( A \) is traversed in the backward direction, it appears in the sequence \( q \) as \( A^{-1} \). Let \( Q \) be the set of all relational sequences of the form \( A_1A_2 \cdots A_n \), where \( A_i \in \mathcal{I} \), \( i = 1, \ldots, n \), and \( n \leq \alpha \). Note that the \( A_i \) need not be different from one another. For convenience, let the upper bound \( \alpha \) be an even number. Because the length \( n \) of a sequence is bounded by \( \alpha \), it is sufficient to take \( \mathcal{P} = \mathcal{E}^d \) where the set of elementary limb states \( \mathcal{E} \) has only \( 1.5\alpha + 2 \) members, as shown below.

A question is presented to the network of automata in the following way: The terminal receives a triple \((x, q, y)\), where \( x, y \in \mathcal{N} \) and \( q \in Q \). The terminal then "excites," or activates, the two automata \( x \) and \( y \), causing them to initiate an algorithmic search procedure. After a finite time interval, either a path matching \( q \) has been found or no such path has been found. If several such paths exist, they are all found simultaneously. The following theorem asserts the existence of an algorithm whereby the network is capable of answering any question \((x, q, y)\), for \( q \in Q \). Later we will consider more complex questions involving inference.

**Theorem 1.** Given the network of automata \((\mathcal{N}, \mathcal{A}, \mathcal{I})\), there exists a finite set of states \( \mathcal{P} \) such that to each question \((x, q, y)\), with \( x, y \in \mathcal{N} \) and \( q = A_1 \cdots A_n \in \mathcal{Q} \), there corresponds a transition function \( \phi_q \) on \( \mathcal{P} \) having the following property: When \( x \) and \( y \) are excited by the terminal, the network will ascertain the existence of a path \( q \) between \( x \) and \( y \), within a finite interval of time.

A constructive proof of this theorem is given in Fiksel (1973); here we will simply exhibit the required set of states and the state-transition function. Let us define the set of limb states \( \mathcal{E} = \{0, 1, 1, 1_2, 1_2, 2, 2, 2_2, 2_2, \ldots, (\alpha/2)_1, (\alpha/2)_2, (\alpha/2)_12\} \), and let \( \mathcal{P} = \mathcal{E}^d \). The denotation of each of these states is defined below for any time \( t \).
$e_{x,t}^r = \omega$ means that the limb $(z, r)$ is dead; the state of a dead limb can never change. $e_{x,t}^r = I$ means that limb $(z, r)$ is idle, i.e., it has not yet received any signals for the question being processed. $e_{x,t}^r = k_1$ means that a path of limbs labeled $A_1 \cdots A_k$ has been found starting from node $x$ where $A_1 \cdots A_k$ are the first $k$ relations in $q$; the limb $(z, r)$ corresponds to $A_k$. $e_{x,t}^r = k_2$ means that a path of limbs labeled $A_{n-k+1} \cdots A^{-1}_n$ has been found starting from node $y$, where $A_{n-k+1} \cdots A_n$ are the last $k$ relations in $q$; the limb $(z, r)$ corresponds to $A_{n-k+1}^{-1}$. $e_{x,t}^r = k_12$ means that $(z, r)$ is the $k$th limb on two paths, one starting from $x$ and one from $y$. This event can occur only if $A_k = A_{n-k+1}^{-1}$.

Assume that before every question $(x, q, y)$, the network is initialized into an idle state, so that all dead limbs are in state $\omega$ and all other limbs are in state $I$. We will say that “$z$ receives a signal $k_1$” if some idle limb of $z$ receives as input the state $k_1$. Receiving a $k_12$ signal is equivalent to receiving both a $k_1$ and $k_2$ signal.

Let $q = A_1 \cdots A_n$ be a specific sequence of $Q$. We define a function $\phi_q$ that will answer $(x, q, y)$ for any $x, y \in \mathbb{N}$. One way to define $\phi_q$ would be to present a table showing $e_{x,t+1}$ for each possible combination of state $e_{x,t}$ and input $\lambda_{x,t}$. However, it would be tedious to compile and impractical to read. Instead a set of verbal transition rules will be given that indicate all the state-input configurations for which a change of state occurs. Unless automaton $x$ satisfies one of the transition conditions stated below at time $t$, its state remains unchanged. The search for a path $q$ is initiated at time $t = 0$, when the terminal excites $x$ with a type-1 excitation and $y$ with a type-2 excitation. Pragmatically speaking, the terminal would also have to supply to each automaton the sequence $q$ at $t = 0$, in order for $\phi_q$ to be uniquely defined.

**Transition Rules for $\phi_q$**

1. (a) If automaton $x$ receives a type-1 excitation, it places all limbs labeled $A_1$ in state $1_1$.
   
   (b) If automaton $y$ receives a type-2 excitation, it places all limbs labeled $A_n^{-1}$ in state $1_2$. (In other words, starting from either end of the desired path, the endpoint automaton finds all limbs that bear the first label in the sequence $q$.)

2. Rule 2 below is operative only when condition 5 is not satisfied.
   
   (a) If node $x$ receives a $k_1$ signal, it places all limbs labeled $A_{k+1}$ into state $(k + 1)_1$.
   
   (b) If $x$ receives a $k_2$ signal, it places all limbs labeled $A_{n-k}$ into state $(k + 1)_2$.

* The connection between the terminal and each automaton could be formalized as a $(d + 1)$th limb (see Fiksel, 1973). However, because this connection is active only at the beginning and end of the network computation and because the signals transmitted along it are trivial in nature, it is sufficient to describe it informally.
(In other words, when informed that it lies on a possible path \( k \) arcs from either end, automaton \( z \) signals all limbs bearing the next label in the sequence \( q \).)

(c) In the special case where \( z \) simultaneously receives a \( k_1 \) and a \( k_2 \) signal, and if \( A_{k+1} = A_{n-k} \), then \( z \) places all limbs with that label in the state \((k + 1)_{12}\). (The need for this complication is shown in Fig. 4a where two paths intersect.)

3. If \( e_{z,r}^i = k_i \) and \( f_{z,r}^i = k_j \), where \( i, j \in \{1, 2, 12\} \) (that is, the limb \((z, r)\) transmits one signal and receives another simultaneously) and Condition 5(b) is not satisfied, then \( z \) behaves just as if \((z, r)\) were idle, and receives the signal \( k_j \) as described in Rule 2 (see Fig. 4b).

4. If \( e_{z,r}^i = k_i \) where \( i \in \{1, 2, 12\} \), then \( e_{z,r}^{i+1} = I \). (After transmitting a signal, the limb \((z, r)\) lapses into the idle state.) Rule 2 may supersede this rule, in the event that automaton \( z \) must transmit two signals along limb \((z, r)\) at two consecutive instants of time.

5. Conditions for success. If either

(a) \( z \) simultaneously receives a \( k_1 \) and a \((n - k)_2 \) signal, or

(b) \( z \) simultaneously receives a \( k_1 \) signal and transmits a \((n - k + 1)_2 \) signal along the same limb, then \( z \) informs the terminal that a path \( q \) has been found. We shall call this event \( \$ \). Condition 5(a) will occur if \( n \), the path-length of \( q \), is even, so that the type-1 and type-2 signals intersect at a single automaton. Condition 5(b) covers the case where \( n \) is odd and two automata simultaneously receive intersecting signals.
The terminal, upon receiving the $ message, instantly orders all automata to cease their state transitions, effectively “freezing” the state of the network for a while. On the other hand, if $ does not occur within \((n/2) + 1\) time units (where \(n\) is the length of the question path \(q\)), then the terminal turns off the computational network.

Event $ is interpreted as a YES answer to the question “Does there exist a path \(q\) between \(x\) and \(y\)?” The absence of event $ is interpreted as a NO answer, meaning that the network lacks sufficient information to answer YES. In later sections we shall consider more specific forms of “NO” and “DON’T KNOW” answers.

Although the algorithm performed by \(\phi_q\) seems complicated when expressed as transition rules, it is really quite simple in principle. The path-searching signals fan out from both endpoints along all those paths labeled in the desired manner. When an automaton receives a signal, the signal carries a digit telling the automaton how far along in the \(q\) sequence it lies. The automaton then selects those of its limbs labeled with the next relation in the question sequence and sends a signal to the corresponding adjacent nodes. The remarkable point here is not that the algorithm works (which is proven in Fiksel, 1973), but that all the automata can execute the algorithm without global supervision.

Retrieving the Solution Paths

It is very simple to strengthen the above theorem so that the algorithm not only determines whether the path \((x, q, y)\) exists, but it also traces out any and all solution paths. By marking each automaton that transmits a signal, a record of all these paths is preserved, so that after event $ the terminal can read off a sequence of nodes \(z_0, z_1, \ldots, z_n\) with \(z_0 = x, z_n = y\), along which the relations \(A_1 \cdots A_n\) hold. To permit such a marking, a secondary automaton is “stacked” on the primary automaton defined in Theorem 1.

**DEFINITION.** Let \(\mathcal{C}\) be an automaton with transition function,

\[
\phi_C: \mathcal{C}^d \times (\mathcal{C}^d \times \mathcal{T}^d) \to \mathcal{C}^d,
\]

and let \(\mathcal{D}\) be an automaton with transition function \(\phi_D\) and elementary limb states \(\mathcal{E}_D\). We say that \(\mathcal{D}\) is stacked on \(\mathcal{C}\) if

\[
\phi_D: \mathcal{E}_D^d \times (\mathcal{E}_D^d \times \mathcal{T}^d) \times \mathcal{E}_C^d \to \mathcal{E}_D^d.
\]

Thus, for automaton \(\mathcal{C}\), the state vector at node \(x\) is \(\epsilon_{x,t+1} = \phi_C(\epsilon_{x,t}, \lambda_{z,t})\), whereas for \(\mathcal{D}\) the state vector at node \(x\) is \(\epsilon_{x,t+1} = \phi_D(\epsilon_{x,t}, \lambda_{z,t}, \epsilon_{x,t+1})\). In other words, for its \((t + 1)\)th transition, \(\mathcal{D}\) takes into account the state that \(\mathcal{C}\) will be in at time \(t + 1\), as well as its own state and input at time \(t\). Notice that \(\mathcal{D}\) is dependent on \(\mathcal{C}\), but \(\mathcal{C}\) is independent of \(\mathcal{D}\). The stacked automata may be considered as a single
automaton, written as the composition of $\mathcal{C}$ and $\mathcal{D}$, $(\mathcal{C} \circ \mathcal{D})$. Stacking is useful when each node-automaton is called upon to perform several clearly separable functions.3

Let $\mathcal{C}_0$ be the automaton defined in Theorem 1, which uses transition function $\phi_q$ and elementary limb states $\mathcal{E}_0$. We define the marking automaton, $\mathcal{M}$, having elementary limb states $\{I, R_1, R_2, R_{12}\}$ and the following rules for its transition function (dependent on $\mathcal{E}_0$):

1. If $z$ transmits a $R_i$ signal along limb $(z, r)$ and if $\mathcal{E}_t^z = I$, then $\mathcal{E}_{t+1}^z = R_i$.
2. If $z$ transmits a $k_i$ signal along limb $(z, r)$ and if $\mathcal{E}_t^z = R_j, i \neq j$, then $\mathcal{E}_{t+1}^z = R_{12}$.
3. If $z$ transmits a $k_{13}$ signal along limb $(z, r)$, then $\mathcal{E}_{t+1}^z = R_{13}$. The $R_i$ states are passive states that serve only to mark the limbs along which $k_i$ signals were transmitted. The fact that these markers may be employed to retrace all solution paths is expressed in the following corollary to Theorem 1:

**Corollary 1.** Given the network $(\mathcal{N}, \mathcal{A}, \mathcal{T})$ and the question $(x, q, y)$, with $q = A_1 \cdots A_n \in Q$ and $x, y \in \mathcal{N}$, let the automaton $(\mathcal{C}_q \circ \mathcal{M})$ be embedded at each node of $\mathcal{N}$. When $x$ and $y$ are excited by the terminal, if a solution path exists, then event $\$ will occur within $(n/2) + 1$ time units. Furthermore, after event $\$, the entire set of solution paths may be retrieved within $m$ iterations, where $(n/2) - 1 \leq m \leq (n/2)$.

Again, a rigorous proof will be found in Fiksel (1973). The presence of $\mathcal{M}$ does not hinder the operation of $\mathcal{C}_q$, so that the only difference from Theorem 1 is that certain limbs are left with markers on them. The retracing procedure is not difficult. Starting from each node $w$ where type-1 and type-2 signals intersected, we trace back along the paths marked by $R_i$ states (see Fig. 5). Since there were many stray signals transmitted that did not find a successful path, we must make sure that the arc labels are correct during retracing. As we fan out in the network from each $w$ toward $x$ and $y$, we build two trees, $\sigma_1$ and $\sigma_2$, recording the nodes that are traversed going in that direction. After approximately $n/2$ steps, the endpoints $x$ and $y$ should be among the tips of these trees.

Note that each node of $\sigma_i$ corresponds to a node of $\mathcal{N}$, and there may be many nodes of the arborescence corresponding to a single node of $\mathcal{N}$. For example, node 7 appears on three branches of $\sigma_2$ from $w_1$. After termination, there will be at least one unclosed tip of the completed tree $\sigma_1$ corresponding to $x$ for $i = 1$, or to $y$ for $i = 2$. This is due to the fact that there is a type-1 path of $n/2$ arcs terminating in $w$ and matching $q$ that originated at $x$, and a similar type-2 path originating from $y$. The closed tips (e.g., node 11 of $\sigma_1$ from $w_1$) correspond to signals that did not lie on an answer path; these will occur if the $A_i$ and $A_i^{-1}$ are not distinct.

3 See Rosenstiehl, Fiksel, and Holliger (1972) for examples of stacked automata used in network problem-solving.
FIG. 5. (a) A network configuration for which event $\$ has just occurred at two nodes, denoted $w_1$ and $w_2$. (b) The arborescences developed from $w_1$ and $w_2$ by the retracing procedure. Since $n = 8$, four iterations were required; three solution paths meet at $w_1$ and four meet at $w_2$.

The set of all solution paths is found by taking all possible combinations of type-1 and type-2 path segments, for each $w$.

The above corollary has an important interpretation in the context of question-answering by a semantic network. Given a question $(x, q, y)$, not only can the network find a path $q$ between $x$ and $y$, but the terminal also can read off the concept-name at any intermediate point along the path. Although the retracing procedure seems complex, in the case of a single answer path event $\$ occurs at only one node $w$, and the trees $\sigma_1$ and $\sigma_2$ are just subpaths of that single answer path. Retrieval of several nonintersecting solution paths is also relatively easy.

Applications and Extensions

To facilitate the representation of questions that require informative (as opposed to YES or NO) answers, the notation $(x, q, y)$ can be extended. Suppose that the network is asked to find a node $v$ such that there is a path of the form $q_1 = A_1 \cdots A_k$ between $x$ and $v$, and a path of the form $q_2 = A_{k+1} \cdots A_n$ between $v$ and $y$, where $q_1q_2 \in Q$. This question will be written as $(x, q_1q_2, y)$. The procedure for answering
it consists of three steps. The network employs the usual algorithm to answer \((x, q_1 q_2, y)\). Then the retracing procedure is applied, yielding the set of solution paths. The answer is obtained by selecting a node \(v\) that is the \(k\)th member of any solution path. Some examples of this type of question-answering are given in Section 2.5.

Another class of questions that lends itself readily to a network-of-automata solution has the following form: "Is there any node \(y\) such that a path exists from \(b\) to \(y\) having the sequence of relations \(q = A_1 \cdots A_n\)?" The fundamental algorithm will find any such \(y\) using only type-1 signals. The signals fan out from \(b\) in the usual way, and event $ occurs if some \(y\) receives a \(n_1\) signal. Extending the question notation, we can write this type of question as \((b, q, y)\), where \(y\) is a variable. Again, the network provides more than just a YES or NO answer; it can also supply the concept-names corresponding to each solution \(y\).

Finally, the network can be employed to simulate human performance on tasks requiring word-associations or semantic-relatedness of two concepts. Given any two concept-nodes, \(a\) and \(b\), we may ask the network to find any path joining them; this question could be represented as \((a, ?, b)\). A simple version of the fundamental algorithm will suffice here, in which arc labels are ignored and signals fan out along all available arcs. In fact, this is identical to the classic "Shortest Path through a Maze" problem of Moore (1959), which was one of the first problems to be formulated for a network of automata. Event $ occurs whenever an automaton receives intersecting type-1 and type-2 signals, and the solution path can be retrieved by the procedure given in Corollary 1. Clearly, the shortest path between \(a\) and \(b\) is the first one discovered; if it has length \(n\), then the solution requires approximately \(n/2\) time units. Since it may be that more paths are desired, the network need not be frozen after event $$. In the context of semantic memory, this type of path-search is a precise analogue to the semantic disambiguation procedure suggested by Quillian (1970) for a dictionary-like semantic network.

2.3. AN INFERENTIAL SEARCH ALGORITHM

To answer a question often requires more than simply a retrieval of known relations between concepts. It may require the application of certain rules of inference to create new facts from those stored in the memory. For example, "Does a robin have feathers?" asks whether the configuration shown in Fig. 6a exists in the semantic network.

\footnote{The question \((x, q_1 v q_2, y)\) could be answered using a modified transition function that marks every node that receives a signal along an \(A_k\) (or \(A_k^{-1}\)) arc, and then retraces all solution paths after event $ to determine which of these nodes is a solution for \(v\). The methods given in this paper are less efficient but more easily generalized.}
However, the relevant information may be encoded in the network in a different form, such as Fig. 6b. What is necessary is a rule of inference that says that the sequence of relations SUBSET-OF, HAS-AS-PARTS is equivalent to the single relation HAS-AS-PARTS. Below, we show how the search algorithm for the network of automata can be extended to incorporate exactly this sort of inference-making.

Returning to the formal theory, we define a production as an operator \( \theta \) on the set \( \mathcal{F}^* \) of all finite relational sequences such that for given sequences \( a, b \in \mathcal{F}^* \), \( \theta a = b \). If \( c \) and \( d \) are any sequences of relations and \( p \) is the sequence of relations \( cad \), then the production \( \theta \) may be applied to the subsequence \( a \), replacing it by \( b \). This operation is denoted by \( \theta p = cbd \). Applying \( \theta \) to \( p \) thus results in a single replacement of \( a \) by \( b \) at some unspecified point in the sequence \( p \). Let \( \Theta \) be a given set of "admissible" productions. Then the sequence of relations \( p \) is said to be reducible to \( q \in \mathcal{Q} \) with respect to \( \Theta \) if there exists a sequence \( \theta_1 \ldots \theta_m \) in \( \Theta \) such that \( \theta_1 \ldots \theta_m p = q \). This is abbreviated \( p \rightarrow^\Theta q \), or, if it is understood that \( \Theta \) is invariant, \( p \rightarrow q \).

The productions are to be interpreted as semantic rules of inference concerning relations in the network. The statement \( p \rightarrow q \) means that the relational sequence \( q \) may be inferred from the relational sequence \( p \) through the application of one or more of these rules of inference. Thus, to ask a question \( (x, q, y) \) is to ask whether any path \( p \) exists between \( x \) and \( y \) such that \( p \rightarrow q \). In the service of consistency, we can think of the question-answering algorithm presented in Theorem 1 as dealing with the case where \( \Theta \) is empty. We next consider the case where \( \Theta \) is nonempty.

For present purposes, \( \Theta \) can be restricted to contain only elementary productions, defined as having the form \( \theta(B_iB_j) = B_k \), where \( B_1, B_2 \in \mathcal{F}, i = 1 \) or 2. These delete one element of a pair of relations and may be written more concisely as \( B_1B_2 \rightarrow B_k \). The symmetry of labeled arcs in \( (\mathcal{N}, \mathcal{A}, \mathcal{F}) \) requires that if \( B_1B_2 \rightarrow B_k \) is a production in \( \Theta \), then \( B_2^{-1}B_1^{-1} \rightarrow B_k^{-1} \) also belongs to \( \Theta \). Thus, \( \Theta \) may be conceived of as a set of ordered triples of relations in \( \mathcal{F} \). A set of productions \( \Theta \subseteq \mathcal{F}^3 \) is called associative if it satisfies the following conditions for any relations \( A, B, C \):

(i) If \( AB \rightarrow A \) and \( BC \rightarrow B \) are in \( \Theta \), then \( AC \rightarrow A \) is also in \( \Theta \).

(ii) If \( BA \rightarrow A \) and \( BC \rightarrow B \) are in \( \Theta \), then \( CA \rightarrow A \) is also in \( \Theta \).

The associativity property means that if a sequence of relations \( p \) reduces to a
single relation $A$ then every pair of consecutive relations in $p$ must be the left-hand side of a production. That is, the outcome is $A$ regardless of the order of application of the productions.

Due to these inferential capabilities, the set of paths $p$ in the network that answers the question $(x, q, y)$ may now be extremely large, since the $\theta_i$ satisfying $\theta_1 \cdots \theta_m p = q$ need not be distinct. If looping phenomena occur, the set of solution paths may even be infinite. However, provided that this set is nonempty, the fundamental path-searching algorithm can be extended to find the shortest path reducible to $q$. This powerful theorem, expressed below, is the main theoretical result of our paper. On the other hand, if no solution path exists, the search process could continue indefinitely; to prevent this, the terminal fixes a time limit after which, if no path has been found, it returns a NO answer.

The new “inferential search” algorithm requires an expanded set of limb states. Define $G$ as the set of all combinations of distinct elements from the set

$$G = \{0_1, 0_2, 1_1, 1_2, \ldots, \alpha_1, \alpha_2\}.$$  

In other words, $G$ is the set of all unordered groups of signals $(S_1 S_2 \cdots S_\mu)$ where

(i) $S_i \in G$, $i = 1, \ldots, \mu$
(ii) $j \neq i \Rightarrow S_j \neq S_i$
(iii) $1 \leq \mu \leq 2(\alpha + 1)$.

From combinatorial theory, the cardinality (size) of $G$ is

$$\text{card } G = \sum_{\mu=1}^{2\alpha+2} \left(\binom{2\alpha+2}{\mu}\right) = 2^{2\alpha+2} - 1$$

which equals one less than the sum of the binomial coefficients of order $2\alpha + 2$. In practice, we have never found the length of a question $q$ to exceed 3. Hence, $\alpha = 3$ seems like a reasonable upper bound, in which case the cardinality of $G$ is 255.

Let $\mathcal{E}_1 = \{\omega, I\} \cup \hat{G}$ be the set of limb states for the network, where $\omega$ is the dead-limb state and $I$ is the idle state. Then $e_{r,t}^\alpha = (S_1 S_2 \cdots S_\mu) \in \hat{G}$ means that at time $t$ the limb $(z, r)$ simultaneously transmits the signals $S_1, S_2, \ldots, S_\mu$. Reception of a signal $k_t$ means that the $k$th relation of $q$ starting forwards from $y$ $(i = 1)$ or backwards from $y$ $(i = 2)$ has been found. Note that $\mathcal{E}_0 \subseteq \mathcal{E}_1$; the earlier non-inferential case occurs when $\Theta$ is empty. Thus, the following theorem is in a precise sense a generalization of Theorem 1. However, the complexity of the state transition function must increase enormously, as evidenced by the fact that $\text{card } \mathcal{E}_0 = 8$, whereas card $\mathcal{E}_1 = 257$, for $\alpha = 3$.

**Theorem 2.** Given the network of automata $(N', \mathcal{A}, \mathcal{T})$ and an associative set $\Theta \subseteq \mathcal{T}^3$ of elementary productions, let $(x, q, y)$ be a question such that a solution path
exists. Suppose that $v$ is the length of the shortest such path $\delta$. Then there exists a transition function $\psi_q$, operating on $\mathcal{S} = \mathcal{S}_1 \mathcal{A}$, having the following property: When $x$ and $y$ are excited by the terminal, the existence of $\delta$ will be verified after a time interval of $\xi$ where $(v - 1)/2 \leq \xi \leq v/2$.

The proof is reported in Fiksel (1973). The terminal initiates the search procedure at the time $t = 0$ by exciting node $x$ with a type-1 and node $y$ with a type-2 excitation. For any $A \in \mathcal{S}$, we define the two sets.

$$\text{Pre}(A) = \{ B \in \mathcal{S} : BA \rightarrow A \text{ is in } \Theta \}, \quad \text{and} \quad \text{Post}(A) = \{ B \in \mathcal{S} : AB \rightarrow A \text{ is in } \Theta \}.$$ 

Let $q = A_1 \cdots A_n$ be a specific sequence of $Q$. The function $\psi_q$ is similar to $\phi_q$, except that it permits the $q$ sequence to be interspersed with other relations according to the admissible productions.

In effect, the function $\psi_q$ employs the productions in reverse to expand the solution path wherever possible. If a node $v$ has just received a $k_1$ signal and is searching for the $(k + 1)$th arc, $A_{k+1}$, then it not only sends a $(k + 1)_1$ signal along any $A_{k+1}$ arcs out of $v$, but it also sends a $k_1$ signal along all arcs $B$ belonging to Post($A_k$) (i.e., those $B$ for which $A_kB \rightarrow A_k$) and a $k_1$ signal along all arcs $C$ belonging to Pre($A_{k+1}$) (i.e., those $C$ for which $CA_{k+1} \rightarrow A_{k+1}$). In other words, the $k_1$ signal is not augmented when sent along those arcs that have been inserted into $q$ by reversing the productions. This procedure effectively causes the search process to fan out along all possible inferential routes without at the same time “claiming” that another of the specified relations of the question sequence $q$ has been found.

**Transitional Rules of $\psi_q$**

The transition function will be specified in terms of the search fanning out from node $x$, moving forwards through the relational sequence $q$. By substituting the phrases in brackets, one treats the search signals originating from node $y$, moving backwards through the relational sequence $q$.

1. If automaton $x[y]$ receives a type-1 [type-2] excitation, then:

   (a) it places all limbs labeled $A_1[A_n^{-1}]$ in state $1_1 [1_2]$.

   (b) For each $B \in \text{Pre}(A_i)$ [$C \in \text{Post}(A_n)$], $x[y]$ places all limbs labeled $B[C^{-1}]$ in state $0_1 [0_2]$.

2. This is the main rule and it is operative only when Condition 5 is not satisfied. Rules 2(a) and 2(b) are illustrated in Fig. 7a, and 2(a) and 2(c) are shown in Fig. 7b. If limb $(z, r)$ receives a $k_1[k_2]$ signal, then:

   (a) $z$ places all limbs labeled $A_{k+1}[A_{n-k}^{-1}]$ in state $(k + 1)_1$ [state $(k + 1)_2]$.

   (b) If $g_z \in \{ A_k \} \cup \text{Post}(A_k) \{ (z)^{-1} \} \in \{ A_{n-k+1} \} \cup \text{Pre}(A_{n-k+1})$, then for each $B \in \text{Pre}(A_{k+1}) \cup \text{Post}(A_k)$, $[B \in \text{Pre}(A_{n-k+1}) \cup \text{Post}(A_{n-k})]$ $z$ places all limbs labeled $B[C^{-1}]$ in state $k_1[k_2]$. 


Fig. 7. (a) Illustration of Transition Rules 2a and 2b. Each sector of the circle represents a limb of an automaton, with its state written inside. At time $t$, a $k_1$ signal arrives along the path labeled $A_k$ or $\text{Post}(A_k)$. At time $t+1$, a $k_1$ signal is sent along all arcs with labels in the sets $\text{Post}(A_k)$ and $\text{Pre}(A_{k+1})$, while a $(k+1)_1$ signal is sent along all arcs labelled $A_{k+1}$. (b) Transition Rules 2a and 2c. If a $k_1$ signal arrives at time $t$ along an arc with a label in the set $\text{Pre}(A_{k+1})$, then the output at time $t+1$ is the same as in (a), except that $k_1$ signals are not sent along arcs with labels in the set $\text{Post}(A_k)$.

(c) If $g_z^{r} \in \text{Pre}(A_{k+1}) \cap (g_z^{r})^{-1} \in \text{Post}(A_{n-k})$, then for each $B \in \text{Pre}(A_{k+1}) \cap [C \in \text{Post}(A_{n-k})]$ places all limbs labeled $B[C^{-1}]$ in state $k_1 \{k_2\}$.

(Parts (b) and (c) allow the $k_1$ signal to propagate along any arc that can vanish from the sequence $p$ through the application of productions.)

3. Whenever automaton $z$ receives simultaneous signals, it applies Rule 2 to each signal, determines what signals must be sent along each of its limbs, and then transmits all signals simultaneously, using the limb states from $G$.

4. Unless Rule 2 dictates otherwise, a limb lapses into the idle state after transmission of signals.

5. Conditions for success. Event $\$ occurs if either:

(a) $z$ simultaneously receives a $k_1$ signal via limb $(z, r)$ and a $(n - k)_2$ signal via limb $(z, s)$, $r \neq s$, and the following condition is not satisfied:

$$ (g_z^{r})^{-1} \in \text{Pre}(A_{k+1}) - \text{Post}(A_k) \quad \text{and} \quad g_z^{s} \in \text{Post}(A_k) - \text{Pre}(A_{k+1}). $$
or (b) \( z \) simultaneously receives a \( k \) signal via limb \((z, r)\) and transmits a \((n - k + 1)z\) signal via limb \((z, r)\).

The expanded set of limb states allows an automaton to transmit up to \(2\alpha\) different signals simultaneously along any limb. Note that the absence of event $ is a weak negation of the question, and could be more accurately interpreted as a “don’t know” (DUNNO) answer.

To illustrate these transition rules, consider the simple question \((x, A_1A_2, y)\) in conjunction with the network shown in Fig. 8. Suppose \( \Theta \) contains the productions

\begin{align*}
A_1B & \rightarrow A_1, \quad DA_1 \rightarrow A_1, \quad \text{and} \quad CA_2 \rightarrow A_2. \\
\end{align*}

At time \( t = 0 \), node \( x \) sends a \( 1 \) signal along the arcs labeled \( A_1 \) and a \( 0 \) signal along the arc labeled \( D \) while node \( y \) sends a \( 1 \) signal along arcs labeled \( A_2 \). At time \( t = 1 \), nodes \( v_1 \) and \( v_2 \) apply Rule 2(a) while \( v_3, v_5, v_6, \) and \( v_7 \) apply Rule 2(b). Clearly, Condition 5(b) for event $ is now met at nodes \( v_2 \) and \( v_6 \), and at time \( t = 2 \), they will both inform the terminal that a solution path has been found. The path \( xv_2v_3y \) has used the production \( DA_1 \rightarrow A_1 \), while the path \( xv_3v_4v_5y \) has used the production \( CA_2 \rightarrow A_2 \). The path \( xv_3v_4v_5y \) is not found because it is not the shortest path. However, it is also reducible to the sequence \( q = A_1A_2 \) via the productions \( A_1B \rightarrow A_1 \) and \( CA_2 \rightarrow A_2 \).
Retrieving the Solution Path

As before, it may be desirable not only to know that some path is a solution, but also to recognize the node-sequence that corresponds to it. The marking procedure presented in Section 2.2 can be extended without difficulty to the inferential case. Again, let $\mathcal{M}$ be the node-marking automaton, with the following amendment to Rule 3:

3'. If $z$ simultaneously transmits a $k_i$ and $h_j$ signal along limb $(z, r)$, where $i \neq j$, then $e_{z,t+1}^{i} = R_{12}$.

Let $\mathcal{D}_q$ be the automaton defined in Theorem 2, which uses $\psi_q$ and $\mathcal{E}_1$. Notice that the transition rules for $\mathcal{M}$ depend on $\mathcal{E}_1^a$.

**Corollary 2.** Given the network $(\mathcal{N}, \mathcal{A}, \mathcal{T})$, an associative set of productions $\Theta \subseteq \mathcal{T}^3$, and a question $(x, q, y)$ for which a solution path exists, let $(\mathcal{D}_q \circ \mathcal{M})$ be embedded at each node of $\mathcal{N}$. Suppose that $\nu$ is the length of the shortest solution path. Then event $\%$ will occur within $\nu/2$ time units after $x$ and $y$ are excited. Furthermore, after event $\%$ the solution path can be retrieved in $m$ iterations, where $\nu/(2 - 1) \leq m < \nu/2$.

The iterative retracing procedure is analogous to the one illustrated in Section 2.2; a complete description may be found in Fiksel (1973).

The foregoing algorithms for finding and tracing inferential solution paths were constructed specifically to deal with elementary productions of the form $B_1B_2 \rightarrow B_1$ or $B_1B_2 \rightarrow B_2$. Such elementary productions seem adequate for most applications of the foregoing theory to an actual semantic network. However, it is entirely possible to extend the theory so that more complex productions are admissible. The number of limb states in $\mathcal{E}$ would have to increase, and the mechanics of the transition function $\psi_q$ would become more complex. Exploration of the full capabilities of finite automata embedded in a labeled directed network remains an open problem.

Variations in Associative Strength

The semantic memory model described above is intended to provide an approximate simulation of human performance in question-answering tasks. In particular, the model should make reaction-time predictions that are consistent with experimental findings in the psycholinguistic literature. However, to be more realistic, the state transition mechanism of each automaton must be modified to account for the associative strengths of the relations on all of its arcs. Wilkins (1971) argues that the associative strength of a relation between two concepts is proportional to the conjoint frequency of those concepts in everyday usage; for instance, the subset relation $(C)$ between OSTRICH and BIRD would have a much lower associative strength than the subset relation between CANARY and BIRD. It is assumed that a direct consequence of high associative strength is a rapid reaction time for a traversal of the
relational arc in question. Wilkins' data are quite consistent with this assumption, as are those of Rips, Shoben, and Smith (1973).

The modification that we propose will not alter any of the previous results concerning the deductive power of the automata; only the solution times will be affected. Suppose that the time required for an automaton $x$ to transmit a signal along its $r$th limb is an exponentially distributed random variable $V_{x,r}$ with mean $\mu_{x,r}$. The $k$th state transition of $x$ no longer consists of a single discrete instant of time; instead, the $k$th transition is not considered to be complete until all the required signals have been transmitted. Hence, the time interval for the $k$th transition is given by the maximum of all the $V_{x,r}$ for those limbs $r$ that are active. Furthermore, each automaton $x$ does not begin its $k$th state transition until all of its neighbors have completed their $(k+1)$th state transitions. Rosenstiehl, Fiksel, and Holliger (1972) have shown how to construct this type of a network, having arbitrary transition times, without hampering the computational abilities of the automata. Such a network is self-synchronizing, in the sense that the sequence of states of each automaton is unchanged from the discrete-time case.

In path-searching operations of the sort discussed in Theorems 1 and 2, the solution times should average out to be roughly proportional to the number of steps in the solution paths. However, in certain specialized tasks the reaction-time (RT) results will turn out to be quite different from the discrete-time case. For example, the experiments of Rosch (in press) indicate that strong "typicality" for an exemplar $S$ of a class $P$ will decrease the RT for verification of the sentence, "An $S$ is a $P$." While the original model would predict no typicality effect, the model with variable transition times accounts for the observed effect. Another example is an experiment by Anderson and Bower (1973), which is illustrated in Fig. 9. Subjects were taught a set of interrelated propositions, having the structure shown in the network diagram: "There is a hippie in the park," "There is a policeman in the park," etc. It was

![Network representation for the Anderson and Bower experiment. All arcs shown represent the relation $N("in")$. The parameter $\mu_{x,i}$ is the average time for a signal to traverse the $i$th arc from node $x$.](image-url)
found that the RT to verify a true sentence of the form, "There is an S in the P," increased with the number of arcs incident to both S and P. Anderson and Bower hypothesize that this effect is due to a rank ordering of the stored relations incident to any given node x. If there are n such arcs, then on the average (n + 1)/2 arcs will have to be searched before finding any particular arc. But similar effects depending on the degree of the S and P nodes are predicted by our variable transition-time model. Suppose that the terminal asks the question (x, N, y). The time for node x to change its state is the maximum of the times associated with its s incident arcs; the time for node y to change its state is the maximum of the times for its p incident arcs. Thus, the time for an overall decision is determined by the maximum of the s + p arc-times from the queried nodes. For instance, if each arc-time is identically exponentially distributed with mean \( \mu \), then the expected value of the slowest (maximum) of s + p such times is given by

\[
E(T_{\text{max}}) = \mu \sum_{i=1}^{s+p} \frac{1}{i}.
\]

This function, which applies equally to true or false test sentences, has the correct form to fit the observed RT's in the Anderson and Bower experiment.

2.4. SOME SIMPLE ILLUSTRATIONS

Figure 10 presents a typical portion of a semantic network of automata. For illustration, only a subset of the arcs incident to the selected nodes are shown. This subnetwork would be embedded in a more complex network, which would incorporate all of the semantic relations between concept nodes. The domain of information selected here is the ever-discussed one of animal taxonomy. There are three types

![Figure 10. Portion of a semantic network.](image-url)
of relations shown: \( E \) (element of, member of), \( C \) (subclass of), and \( H \) (has-as-parts). For simplicity Fig. 10 considers nodes labeled RABBIT, ROBIN, MOOSE, etc. to represent individual instances rather than classes of individuals. Thus, RABBIT is a member of the class of mammals, whereas MAMMAL denotes a subclass of the class of animals. In a more fine-grain semantic representation, RABBIT would also be a class-name, and the elements of that class would be specific rabbits, such as Bugs Bunny.

The \( H \) relation associates a class concept with the physical features possessed by members of that class. The locations at which \( H \) relations occur are far from obvious; Collins and Quillian (1969), among others, have attempted to determine experimentally the actual structure of such subnetworks. For the present, we are concerned purely with illustration of the network operation and not with claims about the detailed structure of memory.

Before a question is processed the entire network is re-initialized into the idle state \( I \). A simple example of a question is “Is a bird an animal?” This is coded as \( (\text{BIRD}, C, \text{ANIMAL}) \). The transition function \( \phi_q \) would send signals \( l_1 \) from BIRD and \( l_2 \) from ANIMAL, and event \$ \ would occur at the BIRD automaton due to crossing of signals (Rule 5(b)) after the first step.

A slightly more complex example is the question “Is there a mammal with horns?” The corresponding triple would be \( (\text{MAMMAL}, E^{-1}H, \text{HORNS}) \), which will search for a path \( E^{-1}H \) between the nodes MAMMAL and HORNS. More explicitly, the question is: “Does there exist \( z \) such that \( z \) is an element of MAMMAL and \( z \) has-as-parts HORNS?”

Figure 11 demonstrates the state transitions of the automata involved in answering this question. At time \( t = 0 \), the terminal excites the two nodes MAMMAL and HORNS. The search signals then fan out along admissible paths \( (E^{-1} \) out of MAMMAL and \( H^{-1} \) out of HORNS) until the node MOOSE receives the two simultaneous, complementary signals \( l_1 \) and \( l_2 \). The terminal is immediately informed of the discovery of a solution path (event \$), and it can return a YES answer, and identify the node MOOSE as a positive instance.

If BIRD were substituted for MAMMAL in the question, then no path satisfying \( E^{-1}H \) would have been found, since no bird has true horns (not even the owl). At time \( t = 2 \), the terminal would return a NO answer, indicating a failure to retrieve a positive instance. Later we deal with stronger types of negation.

We also can introduce productions into the scheme, allowing elementary inferences in the question-answering procedure. The sentence “\( p \rightarrow q \)” can be interpreted as “the string of relations \( p \) implies the string of relations \( q \) between any two nodes.” Two intuitively legitimate elementary productions would be:

\[
\theta_1: EC \rightarrow E \quad (X \text{ an element of } Y, \ Y \text{ a subclass of } Z \implies X \text{ an element of } Z) \\
\theta_2: EH \rightarrow H \quad (X \text{ an element of } Y, \ Y \text{ has-as-parts } Z \implies X \text{ has-as-parts } Z).
\]
FIG. 11. Transitions of the network at time $t = 0$ and $1$ in answering the question (MAMMAL, $E^1H$, HORNS).

For the set of admissible productions to be associative, $\theta_1$ and $\theta_2$ require a third production:

$$\theta_3: CH \rightarrow H \quad (X \text{ a subclass of } Y, Y \text{ has-as-parts } Z \text{ implies } X \text{ has-as-parts } Z).$$

Fortunately, $\theta_3$ also seems intuitively correct.

The inferential algorithm described in Section 2.3 permits the application of these productions. For example, consider the question: "Does a robin have wings?" Symbolically, this can be represented as the triple (ROBIN, $H$, WINGS). Using a production-sensitive transition function $\psi_q$ for the path $q = H$, the network of automata will display the path $EH$ as a solution using the production $\theta_2: EH \rightarrow H$. The shortest path reducible to $q$ is always the first one to be discovered, due to the parallel nature of the search. Even in an autosynchronous network, the length of time required to find a path is roughly proportional to the length of that path.

Another example of applying a production is with the question, "Does a rabbit have a stomach?" which transforms to (RABBIT, $H$, STOMACH), where $q = H$ again. In this case, the only solution path found in the network of automata is $p = ECH$. But $p \rightarrow q$, since $\theta_1 p = EII$, and $\theta_2(\theta_1 p) \rightarrow II$.

Several other types of questions can be handled by such automata. "What mammal has no horns?" requests an instance of a node $x$ for which there is a path ($x, E$, MAMMAL) but no path ($x, H$, HORNS). The terminal would symbolize
this question as a modified triple, with the variable node $x$ inserted at that point in the string where node identification is required, viz., $(\text{MAMMAL, } E^{-1}xH, \text{HORNS})$. The slash through the $H$ indicates which portion of the path must be absent. The search algorithm would be the same as for the question $q = E^{-1}H$, except that the rule for event $\$ $ becomes: Any node $x$ that receives a type-1 signal from MAMMAL, but no type-2 signal from HORNS after a finite time interval (say $t = 4$) notifies the terminal. For the subnetwork illustrated in Fig. 10, $x = \text{RABBIT}$ will be the only node satisfying this condition. Hence, the answer will be “rabbit,” rather than YES or NO.

The network can also handle questions involving several component solution paths. For instance, consider the question: “Is there an animal with both horns and ears?” The terminal would decompose this into two queries: $(\text{ANIMAL, } E^{-1}xH, \text{EARS})$ and $(x, H, \text{HORNS})$. Although the order of inquiry is arbitrary, the second component depends on the node identified as the answer to the first component. The initial answer path is $C^{-1}E^{-1}H \rightarrow E^{-1}H$, so that $x = \text{RABBIT}$ and $x = \text{MOOSE}$ are both possible answers. The second component is processed by exciting both RABBIT and MOOSE with type-1 signals, and the solution is found to be MOOSE. The sequential mechanism seems psychologically plausible, since it is difficult to imagine asking ourselves two questions simultaneously. Multiple queries will be treated more formally in Section 4.

3.1. Negations in Questions and Answers

For the theory presented thus far, the only case in which the network would answer NO to a question $(x, q, y)$ is when it fails to find a path $q$ between $x$ and $y$. However, the ability to store and retrieve information involving negations is indispensable to any question-answering system. Negations may appear explicitly in semantic memory, as in statements such as “An ostrich cannot fly.” The question “Can an ostrich fly?” should result in a strong denial, reflecting not merely the absence of a solution path but the presence of a contradictory path. Furthermore, the network should be able to infer a negation that is implicit in other facts it knows. For example, consider the question “Is ice warm?” A strong denial may be inferred from the stored fact that “Ice is cold” and the semantic relation “Cold is the contrary of warm.” In this section, we extend the network representations and algorithms to deal with such simple forms of negation. Section 4.2 will treat the more complex topic of propositional negation.

Explicit Negations

The first step in this extension is to augment the set of relational types $\mathcal{S}$ to include “negative relations.” This is done as follows:
Axiom 1. For every $Z \in \mathcal{F}$, there is another relation $\overline{Z} \in \mathcal{F}$ with the properties:

(a) $(\overline{Z}) = Z$.

(b) $(a, Z, b)$ is true $\Rightarrow (a, \overline{Z}, b)$ is false, for any $a, b \in \mathcal{N}$.

(c) $(a, b; \overline{Z}) \in \mathcal{A} = (a, Z, b)$ is false, for any $a, b \in \mathcal{N}$.

The relation $\overline{Z}$ is simply the negation of the relation $Z$. Axiom 1(b) means that the statement $aZb$ is interpreted as a denial of the statement $aZb$. Part (c) asserts that a $\overline{Z}$ arc between two nodes $a$ and $b$ implies that the answer to a question $(a, Z, b)$ will be NO. The converse is not necessarily true; in other words, $(a, Z, b)$ may be false without that information being stored as a $\overline{Z}$ arc. As an illustration of negated relations, let us define the relation $M$ to denote "is capable of the movement of" which holds between animate things and types or classes of movements. Within an animal taxonomy network some typical appearances of this relation would be of the form (BIRD, $M$, FLY) and (FISH, $M$, SWIM); whereas the datum "An ostrich cannot fly" would be encoded in the semantic network as (OSTRICH, $\overline{M}$, FLY). Clearly it would be absurd for us to store all possible negated relations, since this would mean that every pair of nodes would be joined by either a $Z$ or a $\overline{Z}$ arc for every $Z \in \mathcal{F}$. Negations are probably stored in semantic memory only in exceptional cases like the "ostrich" example, where it is necessary to take exception to a more general stored fact, i.e., that "birds can fly."

Figure 12 illustrates the network structure for the above situation. At first glance, the production $CM \rightarrow M$ would seem to lead the network into a logical inconsistency. However, the more direct $\overline{M}$ path takes precedence over the $CM$ path between OSTRICH and FLY simply because it is shorter and found sooner.

Surveying the entire semantic network, the majority of labeled arcs do not represent negations, since most of our stored world-knowledge appears to be affirmative in form. However, a few negated relations would have entered the memory as denials or negative propositions.
Implicit Negations

In analyses of negation, a distinction is often made between contradictory and contrary concepts. Two concepts are said to be contradictory if the negation of one implies the other; for example, "not present" implies "absent," "not open" implies "closed," and "not in" implies "out." Two antonymic concepts are said to be contrary if the negation of one does not imply the other; for example, "not large" does not imply "small," "not cold" does not imply "warm." Contraries always denote opposing ends of a one-dimensional continuum along which intermediate concepts such as "lukewarm" and "medium-sized" can be differentiated. These pairs can be represented in the semantic network in two fashions, depending on whether they are concepts or relations.

Contradictory concepts. Let the relation $P$ denote "has the property," so that "An elephant is large" would be written as $(\text{ELEPHANT}, P, \text{LARGE}).$ An example of contradictory concepts in the animal taxonomy would be the pair of properties MALE and FEMALE. To represent them, only one concept-node is really needed, say that of MALE. Any concept that is to be marked as female would then be connected to the MALE node by a $P$ relation. Thus we would have $(\text{RAM}, P, \text{MALE})$ but $(\text{EWE}, P, \text{MALE}).$ As another instance, the proposition "the door is closed" might be represented as $(\text{DOOR}, P, \text{OPEN}).$ Although "male" is a defining characteristic of "ram," "being open" is but a temporary state of a particular door: the latter is stored as a proposition in episodic memory (see Section 4).

Contradictory relations. There are some contradictory pairs that are most efficiently coded as semantic relations; for example, inside and outside, on and off. Clark (1970) cites experimental evidence that such positive-negative pairs may be encoded in terms of the dominant ("unmarked") member. In other words, outside would be coded as not inside, off as not on. If the relation $N$ denotes inside, then $N$ would denote outside in the semantic network. The statement, "The bird is out of the cage" would then be represented as $(\text{BIRD}, N, \text{CAGE}).$ No new concept-nodes need to be defined for these contradictories.

Contrary relations. Above and below are not contradictory relations, since "not above" is not equivalent to below. Another contrary pair is in front of and in back of. Such contrary relations still require only one new relational type. If we let $A$ denote "above," then $A^{-1}$ denotes "below," while $\bar{A}$ denotes "not above." Note that $aAb \sim bA^{-1}a,$ and $aAb \sim b\bar{A}a,$ but $a\bar{A}b \sim b\bar{A}a.$ Hence the proposition, "The box is below the bridge" would be represented as $(\text{BRIDGE}, A, \text{BOX})$ or $(\text{BOX}, A^{-1}, \text{BRIDGE}).$ Stylistically, English tends to place the anchor point at the less movable object and say "The box is under the bridge" rather than "The bridge is over the
Contrary concepts. The class of contrary concepts is much richer semantically than the class of contradictory concepts. To represent contraries, a special relation $X$ is required. It has the following properties, specified as Axiom 2.

**Axiom 2.**  
(a) $X \in \mathcal{T}$ and $X^{-1} = X$  
(b) If $(a, b; X) \in \mathcal{R}$ then for any $Z \neq E^{-1}$, $cZa \Rightarrow cZb$ for any $a, b, c \in \mathcal{N}$.

Part (a) says that the antonymic relation $X$ is symmetric ($X = X^{-1}$). Part (b) says that if $a$ and $b$ are antonyms, and if concept $c$ has relation $Z$ to $a$, then $c$ must have relation $Z$ to $b$. However, $cZa$ does not imply $cZb$. Part (b) is expressed by productions of the form $ZX \rightarrow Z$, although this is not an elementary production of the type considered in our inference theorem. The relation $Z$ may vary depending on the nature of $a$ and $b$. If $a$ and $b$ are properties, then $Z = P$; if they are directions, such as up and down, then $Z$ would be some prepositional relation. The case $Z = E^{-1}$ is excluded since the only feature that two contrary concepts share is that they are members of the same class. For example, COLD and WARM are both temperatures. "Ice is cold" would be represented as shown in Fig. 13.

![Fig. 13. Example of contrary concepts.](image)

Natural language has expressions for various gradations along the continuum between two contraries: "An elephant is very large," "The glass is half-empty," and "I feel slightly warm" are examples of quantified properties. Theoretically, such distinctions could be dealt with by assigning an integer coefficient $k$ (e.g., $0 \leq k \leq 10$) as a subscript to each $P$ relation. This assignment would lead to rules such as $P_k = P_{10-k}$, and $P_kX \rightarrow P_{10-k}$. Aside from enlarging the sets $\mathcal{T}$ and $\mathcal{S}$, such a modification would not alter the mathematical structure of the network. For the sake of simplicity, refinements of this sort will be ignored in the following.

**Negated Questions**

In the notation of Section 2, consider a question $(x, q, y)$ where $x, y \in \mathcal{N}$ and $q = A_1 \cdots A_n$, and all $A_i \in \mathcal{T}$; then a negation of question $(x, q, y)$ will be defined
as any question \((x, \bar{q}, y)\), where \(\bar{q} = A_1 \cdots A_{k-1} A_k A_{k+1} \cdots A_n\) for some one \(h = 1, \ldots, n\). There are \(n\) possible negations for any \(n\)-step path \(q\). The interpretation of \(\bar{q}\) is simply “not \(q\),” although in practice ambiguity may arise since the meaning of \(\bar{q}\) depends on which \(A_k\) was negated. The definition permits no more than one \(A_k\) to be negated, since double negatives become confusing and intractable.

To illustrate the use of negated questions, consider the problem of answering: “Does an oyster belong to a vertebrate class?” This is coded as (OYSTER, EH, VERTEBRA). The relevant portion of the network might be (OYSTER, E, SHELLFISH) and (SHELLFISH, H, VERTEBRA). With \(q = EH\), the network algorithm will terminate without event $, which can be constructed as a weak negative answer. However, by searching for a negated question, the network can find the path $ = $, which provides a strong negative answer: “No, an oyster does not have a vertebra.” The capability for finding strong negations when they exist is an important one. We show below how the question-answering algorithm can be extended to search for negations. The general strategy is to search for \(q\), and if event $ does not occur, then to search for $\bar{q}\). This procedure is both intuitively satisfying and logically powerful.

A second illustration involves the antonymic production \(PX \rightarrow P\) mentioned earlier. “Is ice warm?” is coded as (ICE, P, WARM). In this case, \(q = P\) and \(\bar{q} = \bar{P}\). An inferential algorithm employing nonelementary productions would be able to reply with a strong negation.

Let us recall the “ostrich” example. In Section 2.4, a question was posed in the form (BIRD, \(E^{-1}xM\), FLY); that is, “Is there a bird which cannot fly?” The original procedure would not yield a correct response, since every node \(b\) for which \((b, E, BIRD)\) would satisfy the question. This is because \(EM \rightarrow M\) is a production; hence \(E^{-1}xEM \rightarrow E^{-1}xM\), for \(x = b\) (see Fig. 14 for the illustration). A correct response

![Fig. 14. Relevant portion of the network for answering the negated question: “Is there a bird that can’t fly?”](image)

would be obtained only by a subsequent search with the question (BIRD, \(E^{-1}xM\), FLY), which is one of the two negated questions corresponding to \(q = E^{-1}M\).

It is instructive to examine one further example, the question “Does a rabbit have wings?” coded as (RABBIT, H, WINGS). It is not likely that we ever would actually have stored the relation (RABBIT, H, WINGS), nor would we have stored
information that would permit a derivation of that relation through productions. Thus, the algorithmic procedures developed so far for our system fail to provide a decisive answer. Nevertheless, if asked such a question, people readily respond NO with confidence. A plausible explanation is that they employ visual imagery to answer the question, conjuring up a picture of a rabbit and then inspecting it for wings. (Kosslyn and Nelson, unpublished). This example shows that certain question-answering phenomena fall squarely outside the domain addressed by our network model of semantic memory.

The Negational Algorithm

The strategy for a negation-sensitive question-answering procedure can be formalized as follows:

A. Perform the usual search for \((x, q, y)\).
B. If event $ occurs, the answer is YES.
C. If event $ does not occur, perform the negational algorithm (below) for \((x, \bar{q}, y)\).
D. If event * occurs, the answer to \((x, q, y)\) is NO.
E. If event * does not occur, the search was inconclusive, and the answer is DUNNO.

For a negational search, event * is the analogue of event $. A description of the negational algorithm in step C will now be given. It is just a modification of the inferential algorithm of Section 2.4. Essentially, at each step in a possible solution path, the automata check for the existence of a negation in the next relation. Thus, if \(q = A_1 \cdots A_n\), and if \(A_1 \cdots A_{k-1}\) have already been found, then both \(A_k\) and \(\bar{A}_k\) are sought as the next relation in the path. Whenever the negation \(\bar{A}_k\) is found, the \(k_i\) signals are converted to \(\bar{k}_i\) signals, indicating that \(q\) has already been negated and that no more negated relations should be sought. Event * occurs if a \(k_i\) signal meets a \((n - k)_j\) signal, \(i \neq j\).

Analogous to \(G\), define \(\bar{G}\) to be the set of all combinations of distinct elements from the set \(\{k_1, k_2, k_1, k_2, k_1^0, k_2^0, k_1^0, k_2^0; 0 \leq k \leq \alpha\}\). Let \(\epsilon_2 = \{\omega, \bar{G}\} \cup \bar{G}\) be the set of limb states for the network. The interpretations of the \(k_i^0\) and \(\bar{k}_i^0\) signals are made clear in the following rules, which define \(\varphi_q\), the negational transition function. (The reader will find it helpful to compare this to the earlier transition rules for \(\psi_q\).)

Transition Rules for \(\varphi_q\)

1. Same as 1(a) and 1(b) for \(\psi_q\), plus
   - (c) \(\chi[y]\) places all limbs labeled \(A_i[A^{-1}]\) in state \(\bar{1}_0[\bar{1}_2^0]\).
   - (d) For each \(B \in \text{Pre}(\bar{A}_i)[C \in \text{Post}(\bar{A}_n)], \chi[y]\) places all limbs labeled \(B[C^{-1}]\) in state \(0_1^0[0_2^0]\).
2.1. Same as 2 for $\psi_0$, with the addition: If $g_{z^*} \notin \text{Pre}(A_{k+1})[(g_{z^*})^{-1} \notin \text{Post}(A_{n-k})]$ then

(d) $z$ places all limbs labeled $\bar{A}_{k+1}[\bar{A}_{n-k}^{-1}]$ in state $(k + 1)_{1}^{1}$ [state $(k + 1)_{2}^{1}$], and

(e) for each $B \in \text{Pre}(A_{k+1})[C \in \text{Post}(A_{n-k})]$, $z$ places all limbs labeled $B[C^{-1}]$ in state $k_{1}^{0}[k_{2}^{0}]$.

2.2. If limb $(z, r)$ receives a $k_{1}[k_{2}]$ signal

(a) $z$ places all limbs labeled $A_{k+1}[A_{n-k}^{-1}]$ in state $(k + 1)_{1}^{1}$ [state $(k + 1)_{2}^{1}$].

(b) Same as 2(b) for $\psi_0$ except that $z$ places all limbs labeled $B[C^1]$ in state $k_{1}[k_{2}]$.

(c) Same as 2(c) for $\psi_0$ except that $z$ places all limbs labeled $B[C^1]$ in state $k_{1}[k_{2}]$.

2.3. If limb $(z, r)$ receives a $k_{1}^{0}[k_{2}^{0}]$ signal

(a) $z$ places all limbs labeled $\bar{A}_{k+1}[\bar{A}_{n-k}^{-1}]$ in state $(k + 1)_{1}^{1}$ [state $(k + 1)_{2}^{1}$].

(b) For each $B \in \text{Pre}(A_{k+1})[C \in \text{Post}(A_{n-k})]$, $z$ places all limbs labeled $B[C^{-1}]$ in state $k_{1}^{0}[k_{2}^{0}]$.

2.4. If limb $(z, r)$ receives a $k_{1}^{0}[k_{2}^{0}]$ signal

(a) $z$ places all limbs labeled $A_{k+1}[A_{n-k}^{-1}]$ in state $(k + 1)_{1}^{1}$ [state $(k + 1)_{2}^{1}$].

(b) For each $B \in \text{Post}(A_{k})[C \in \text{Pre}(A_{n-k+1})]$, $z$ places all limbs labeled $B[C^{-1}]$ in state $k_{1}^{0}[k_{2}^{0}]$.

(c) For each $B \in \text{Pre}(A_{k+1})[C \in \text{Post}(A_{n-k})]$, $z$ places all limbs labeled $B[C^{-1}]$ in state $k_{1}[k_{2}]$.

---

**FIG. 15.** A portion of the solution path for $\bar{q} = A_{1} \ldots A_{k-1} \bar{A}_{k} A_{k+1} \ldots A_{n}$ demonstrating the application of Transition Rules 2.1-2.4. Each sector of a circle represents an automaton limb with the state written inside; the transitions shown span several instants of time.
(The $k_i^0$ signals are transmitted only along arcs that may combine with a negated relation via a production. Once a negated relation has been found, the $k_i^0$ signals are transmitted along the corresponding path until the next relation is found. Then they are converted to $k_i$ signals for the duration of the path. This is illustrated in Fig. 15.)

3. and 4. are the same as for $\psi_q$ with $G$ replaced by $G$ and Rule 2 replaced by Rules 2.1–2.4.

5. Conditions for event $\ast$.

(a) $z$ simultaneously receives a $k_1$ signal via limb $(z, r)$ and a $(n - k)_2$ signal via limb $(z, s)$, or a $k_1$ signal via limb $(z, r)$ and a $(n - k)_2$ signal via limb $(z, s)$, $r \neq s$, and the following condition is not satisfied: $(g_s, r)^{-1} \in \text{Pre}(A_{k+1}) - \text{Post}(A_k)$ and $g_z \in \text{Post}(A_k) - \text{Pre}(A_{k+1})$

or (b) $z$ simultaneously receives a $k_1^0$ signal via limb $(z, r)$ and a $(n - k)_2$ signal via limb $(z, s)$, $r \neq s$ and the following condition is not satisfied:

$$g_z \in \text{Post}(A_k) - \text{Pre}(A_{k+1}) - \text{Post}(A_k)$$

or (b') $z$ simultaneously receives a $k_1$ signal via limb $(z, r)$ and a $(n - k)_2$ signal via limb $(z, s)$, $r \neq s$ and the following condition is not satisfied:

$$(g_z, r)^{-1} \in \text{Pre}(A_{k+1}) - \text{Post}(A_k) - \text{Pre}(A_{k+1})$$

or (c) $z$ simultaneously receives a $k_1^0$ signal via limb $(z, r)$ and a $(n - k)_2$ signal via limb $(z, s)$, $r \neq s$.

or (c') $z$ simultaneously receives a $k_1^0$ signal via limb $(z, r)$ and a $(n - k)_2$ signal via limb $(z, s)$, $r \neq s$.

The above algorithm satisfies a theorem analogous to Theorem 2, the proof of which is straightforward though tedious.

**Theorem 3.** Given the network of automata $(N, A, \mathcal{T})$ and an associative set $\Theta \subseteq \mathcal{T}^a$ of elementary productions, let $(x, q, y)$ be a question such that a solution path exists for some negated question $\bar{q}$. Let $v$ be the length of the shortest solution path $\delta$. Then the transition function $\bar{\mathcal{P}}_q$, operating on $\mathcal{P} = \mathcal{E}_2^a$, has the following property: When $x$ and $y$ are excited by the terminal, the existence of $\delta$ will be verified after a time interval of $\tau$ where $(v - 1)/2 \leq \tau \leq v/2$.

**Experimental Justifications**

Clark (1970) has studied human reaction times associated with various types of negational question-answering extensively. He suggests a model that accounts
quite well for the data, but it is somewhat restricted in the scope of its application, since it is designed to deal specifically with single predicate assertions and negations. Each trial in Clark’s typical experiment consisted of input of a statement (or picture) regarding the spatial relation (above, below) of two small symbols (star, plus). This input was queried immediately by one of four statements to which the subject answered true or false, e.g., “The star isn’t above the plus.”

The network-of-automata model of question-answering makes predictions identical to those of Clark’s model for the simple cases considered in his experiments. As one example, Table I gives Clark’s prediction equations (which have been supported experimentally) for reaction times to verify the four different questions. The questions are expressed in the network notation. In each case, the question encounters the same relevant fact stored in the memory, $xAy$, or “$x$ is above $y$.” The reaction time components in Table I may be interpreted as follows: $t_0 =$ base time for setting up the question-answering procedure; $a =$ time required to search for a one-arc path using the ordinary algorithm (Step A); $b =$ time required to conclude that event $\$ has not occurred and terminate the search; $c =$ time required to search for a one-arc path using the negational algorithm (Step C); $d =$ time required to conclude that event * has not occurred and to terminate the search.

For each of the four questions, it can be shown that the network procedure gives the correct reaction-time predictions, as explained below:

**Question 1.** Step A terminates with event $\$, since a path $xAy$ is found. This requires a time of $t_0 + a$.

**Question 2.** Step A terminates without event $\$, which requires $t_0 + a + b$ time units. At this point it is useless to ask the question $(y, A, x)$, the negation of $(y, A, x)$, for the following reason. The only way that a positive statement $(y, A, x)$ can be true is if the information “$y$ is above $x$” is stored in affirmative form. Thus, if the question $(y, A, x)$ terminates without event $\$, the answer must be NO.

**Question 3.** Step C is performed first, since $(y, \bar{A}, x)$ is a negated question,
and it terminates without event $. This requires $t_0 + e + d$ time units. One cannot conclude that the answer is NO, since the absence of a path $yAx$ will imply an affirmative answer to $(y, A, x)$. Hence, Step A is performed for $(y, A, x)$ and it terminates without event $. This requires an additional $a + b$ time units, and results in a YES answer to $(y, A, x)$.

**Question 4.** Step C is performed first for $(x, A, y)$, and terminates without event $\tau$, requiring $t_0 + e + d$ time units. Then Step A is performed for $(x, A, y)$, terminating in event $\$, and requiring an additional $a$ time units. This is interpreted as a NO answer to $(x, A, y)$.

Note that $yA^{-1}x \Rightarrow yAx$, so that if the production $A^{-1} \rightarrow A$ were used in Case 3, the reaction time should have been $t_0 + e$. This indicates that the inferences represented by this nonelementary production are not used in negational question-answering.

### 3.2. Quantification

The problem of dealing with universal and existential quantifiers is a thorny one for any natural language understanding system. Here, we will be concerned only with creating a viable representational structure in which to encode and retrieve information pertaining to quantifiers. In the introductory Section 2.1, we restricted our consideration to questions in a formal language that express the conceptual content of natural language questions. Thus far, it has been unnecessary to define this language explicitly since it is so directly matched to the network structure. In effect, the language consists of all triples $(x, q, y)$ such that $x, y \in V$ and $q \in Q$, i.e., the set of possible questions. However, the semantic content of these questions is still elementary, dealing only with the existence of a specified path between two nodes. The semantics of quantification requires the use of a symbolic formal language in which not every symbol maps into a component of the semantic network. Because of the mathematical formulation of the network representation, we will be able to adopt the symbolism of mathematical logic for this purpose.

**Universal Quantifiers**

English words such as “all,” “each,” and “every” are examples of universal quantifiers. When used in a sentence, they indicate that every instance of a proposition involving some variable is true. The statement “All soldiers are junkies” can be represented formally as

$$(\forall x)((x, E, \text{SOLDIER}) = (x, E, \text{JUNKIE}))$$
which is to be read: “For all nodes $x$, if $x$ is an element of the set of soldiers, then $x$ is an element of the set of junkies.” This illustrates the use of a first-order language in which the atomic sentences are of the form $(x, q, y)$, and complex sentences are constructed from these with the use of the logical connectives $\&$ (and), $\lor$ (or), $\neg$ (not), and $\Rightarrow$ (implication). If a sentence contains a variable $x$, then it can be preceded by either the existential quantifier $\exists x$ or the universal quantifier $\forall x$. Henceforth, any sentence in the first-order logic can be considered to be a question, in the sense that it is either true or false according to the information stored in the network. In particular, it is true if it can be verified through a finite search algorithm using productions and axioms, if required. The issue of what classes of sentences are “decidable” by such a procedure is taken up in Fiksel (1973), but here we are more concerned with the pragmatics of question-answering.

The conceptual design of the network has been based on the set-theoretic relations of $C$ (set inclusion) and $E$ (instantiation). These relations have certain special properties that are often used to expedite question-answering, and they play an important role in universal quantification. We state these properties as Axiom 3.

**Axiom 3.** For every $x, y, z \in \mathcal{N}$, the following are true:

(a) $xCx$ (reflexivity)
(b) $(xCy) \& (yCx) \Rightarrow (x = y)$ (antisymmetry)
(c) $(xCy) \& (yRz) \Rightarrow (xRz)$ for any $R \in \mathcal{F}$ (alternatively, $CR \rightarrow R$ is a production)
(d) $(xEy) \& (yRz) \Rightarrow (xRz)$ for any $R \neq C$ ($ER \rightarrow R$ is a production)
(e) $(xEy) \& (yCz) \Rightarrow (xEz)$ ($EC \rightarrow E$ is a production).

† Let us comment on the parts of this axiom. Part (a) says that every concept is a subset of itself, an obvious truth though one which is never coded in the network. Part (b) shows that synonymous concepts map into the same node. Parts (c) and (d) are crucial to the inference-making capability of the network, as discussed in previous sections. To illustrate, a belief such as “all soldiers are junkies,” would be encoded as $(\text{SOLDIER, C, JUNKIE})$. Now suppose the terminal receives the question, “Is it true that all soldiers are junkies?” Before initiating a search algorithm the terminal must do some preparatory work. The question has been translated by the semantic processor into the first-order language sentence.

$$(\forall x)((x, E, \text{SOLDIER}) \Rightarrow (x, E, \text{JUNKIE})).$$

† See Suppes (1957) for a complete discussion of the symbolic language of first-order logic and its application to natural language. In this paper we use the forms $xAy$ and $(x, A, y)$ interchangeably.
The terminal recognizes the form of a single universal affirmative, and applies Axiom 3(e) in the following form:

\[(x, E, \text{SOLDIER}) \& (\text{SOLDIER, C, JUNKIE}) \Rightarrow (x, E, \text{JUNKIE}).\]

Due to this axiom, asking the original question is logically equivalent to asking the simple question \((\text{SOLDIER, C, JUNKIE})\). If event $ occurs, the answer is YES.

Most statements involving universal quantifiers do not easily succumb to this kind of analysis, which is why a formal first-order language is required to express them. For instance, the question "Do all pets have claws?" has the same universal affirmative form.

\[(\forall x)((x, E, \text{PET}) \Rightarrow (x, H, \text{CLAW})),\]

and yet it cannot be answered so readily. The relation in the second atomic sentence has changed from \(E\) to \(H\), so that Axiom 3(d) can be applied. The terminal uses the same strategy as in the previous example, and attempts the question \((\text{PET, H, CLAW})\). However, suppose that no path of the form \(q = CH\) can be found between PET and CLAW (note that \(CH \rightarrow H\)). To obtain a definitive NO answer, the terminal must resort to a more complex procedure. By a logical identity, the formal expression becomes \((\forall x)(\sim(x, E, \text{PET}) \lor (x, H, \text{CLAW}))\). Making this transformation, the terminal handles the transformed question in the following manner:

(i) Ask the question \((x, E, \text{PET})\), thus finding the set

\[X = \{x \in \mathcal{X} : (x, E, \text{PET})\}.\]

(ii) For each \(v \in X\), ask the question \((v, H, \text{CLAW})\), until the set \(X\) is exhausted.

(iii) The statement is true only if every answer in (ii) is YES.

It is clear that this procedure (called "instance checking") is correct. In Fiskel (1973) the procedure is generalized to answer any question \((\forall x)(P(x))\), where \(P(x)\) is any logical expression containing no quantifiers and only one variable \(x\). The terminal plays a supervisory role to the network, first decomposing the question into its components and later combining the answers received during the computation.

**Existential Quantifiers**

English words such as "some," "few," and "many" are examples of existential quantifiers. In a sentence, they indicate that at least one instance of a proposition involving a variable is true. The semantic processor would formally represent the statement "Some soldiers are junkies" as

\[(\exists x)((x, E, \text{SOLDIER}) \& (x, E, \text{JUNKIE})),\]
which reads: “There exists a node $x$ such that $x$ is an element of both the set of soldiers and the set of junkies.” To encode world-knowledge that pertains to existential quantification, a new set-theoretic relation $D$ (set intersection) must be added to $\mathcal{T}$. The statement $xDy$ is read as “$x$ and $y$ have a nonempty intersection” or “$x$ and $y$ intersect.” The relation $D$ satisfies the following axiom:

**Axiom 4.** For every $x, y \in \mathcal{N}$,

- (a) $xCy \Rightarrow xDy$ (*$C \rightarrow D$ is a production*)
- (b) $xDy = yDx$ (*symmetry; $D^{-1} = D$*).

An immediate result of Axiom 4 is that $C^{-1} \rightarrow D$ is also a production,\(^6\) that is, if one set is included in another, then their intersection is nonempty.

The belief “Some soldiers are junkies” would exist in the semantic network as $(\text{SOLDIER}, D, \text{JUNKIE})$. The question “Are some soldiers junkies?” would first be translated by the semantic processor into the logical form shown above. The terminal recognizes the logical expression, and proceeds to apply Axiom 4(c), yielding the equivalent question $(\text{SOLDIER}, D, \text{JUNKIE})$, which is in the required format for network processing.

This substitution is possible only when both relations in the component sentences of the logical expression are $E$. For example, the substitution is not possible with simple statements like “Some fish can fly,” which must be represented as

$$(\exists x)((x, E, \text{FISH}) \land (x, M, \text{FLY})).$$

In this situation, an instance-checking procedure analogous to the one given for universal quantifiers must be employed by the terminal to answer whether some fish can fly. The effective procedure would be as follows:

(i) Ask the question $(x, E, \text{FISH})$ to find the set $X = \{x \in \mathcal{N}: (x, E, \text{FISH})\}$.
(ii) For each $v \in X$, ask the question $(v, M, \text{FLY})$ until the set $X$ is exhausted.
(iii) The statement is true only if at least one answer in (ii) was YES.

Again, this procedure generalizes to any question of the form $(\exists x) P(x)$ involving a complex logical expression in $x$.

An alternative procedure exists when the logical expression consists of only two component sentences. In this case, the terminal can follow a more direct procedure of searching for a designated path between two nodes. To illustrate, in the example above, the terminal would ask the question $(\text{FISH}, E^{-1}M, \text{FLY})$, that is “Is there

---

\(^6\) Although $C \rightarrow D$ is not an elementary production, there is no difficulty in including in $\Theta$ productions of this single-replacement type. No new limb states are required, and the only change in the transition function is that a $C$ label may be substituted for a $D$ label.
a fish that can fly?" This could be answered by the standard algorithm. This simple direct procedure is not applicable, however, when $P(x)$ contains more than two atomic sentences. In that case, the lengthier procedure that checks instances must be followed.

A brief digression is in order here regarding the many semantic gradations of existential quantifiers. "A few $a$ are $b$" and "many $a$ are $b$" express the fact that the sets $a$ and $b$ have a small or large intersection, respectively. As in the case of contrary concepts (Section 3.1), we could assign a numerical coefficient, say between 0 and 10, to each $D$ relation as a subscript, indicating the relative magnitude of the intersection. $D_0$ would be interpreted as $D$, or no intersection, whereas $D_{40}$ would be interpreted as near-identity of the two sets in question. Clark (1970) points out that "a few" is interpreted as a positive quantifier, while "few" is interpreted as the negation "not many." This is reflected in the coding of the statements listed below:

- "Some $a$ are $b$" $(a, D, b)$
- "No $a$ are $b$" $(a, \bar{D}, b)$
- "Many $a$ are $b$" $(a, D_8, b)$
- "Few $a$ are $b$" $(a, D_8, b)$
- "A few $a$ are $b$" $(a, D_2, b)$

For example, to answer "few $a$ are $b$" requires verification that $aDb$ and the subscript on the $D$ is not near 8. Further discussion of these matters is waived.

Finally, to make the information content of the network more precise, we will adopt certain conventions regarding the occurrence of subset (C) and intersection (D) arcs.

**Axiom 5.** For any $x, y, z \in A'$,

(a) \( (x, y; D) \in \mathcal{A} \Rightarrow (x, y; C) \notin \mathcal{A} \)

(b) \( (x, y; C) \in \mathcal{A} \Rightarrow (x, y; D) \notin \mathcal{A} \)

(c) \( (x, y; C) \in \mathcal{A} \Rightarrow (y, x; C) \notin \mathcal{A} \)

Parts (a) and (b) stipulate that two nodes cannot be joined by both a $C$ and a $D$ arc, even though $C$ is just a special case of $D$. Thus, if $xDy$ were coded in the network, and we later learned that $xCy$, the original $D$ arc would have to be removed before a $C$ arc could be created. These are simply conventions for setting up the network; relation $D$ is inferable if relation $C$ is known. Part (c) stipulates that two concepts cannot be subsets of each other; that would logically imply their identity, making one of the corresponding nodes redundant. This is actually a restatement of the antisymmetry convention of Axiom 3(b).
Quantiﬁers and Negation

The results on negation can now be combined with those developed for quantifiers to increase the question-answering power of the network. We illustrate this increased power with the following questions.

1. “Is it true that not all birds can fly?” is translated by the semantic processor into the logical form:

$$\neg(\forall x)((x, E, \text{BIRD}) \Rightarrow (x, M, \text{FLY}))$$

Through logical manipulation, the terminal transforms this statement into

$$\exists x((x, E, \text{BIRD}) \& \neg(x, M, \text{FLY}))$$

that is “Some birds cannot fly.” But it was shown in the previous section that this question can be treated as $$(\text{BIRD}, E^{-1}M, \text{FLY})$$.

2. “Is it true that no fish can fly?” is translated into

$$\neg(\exists x)((x, E, \text{FISH}) \& (x, M, \text{FLY}))$$

This is just the negation of “Some fish can fly” and can be handled by reversing the truth-value of the answer to the latter statement.

The relation $D$, the negation of $D$, also plays an important role in negational question-answering. The interpretation of $aDb$ is “$a$ and $b$ have an empty intersection,” or more succinctly “no $a$’s are $b$’s.” The properties of $D$ are similar to those of $X$, as can be seen by comparing Axiom 2 with the following:

**Axiom 6.**

(a) $D \in \mathcal{F}$ and $D^{-1} = D$ (symmetry)

(b) If $(a, b; D) \in \mathcal{A}$, then for any $x, y \in \mathcal{N}$, $(xCa) \Rightarrow \neg(xCb)$, and $(yEa) \Rightarrow \neg(yEb)$.

Axiom 6(b) states that if $aDb$, no element or subset of $a$ can be contained in $b$. Analogous to the production $PX \rightarrow P$, this gives rise to the productions $ED \rightarrow E$ and $CD \rightarrow D$. Note also that $D \rightarrow C$.

Some additional examples using $D$ follow. These assume that the network contains the information that “no soldiers are priests,” “some soldiers are junkies,” and “John is a soldier.” These relations are shown in Fig. 16.

3. The query “Is John a priest?” would be coded as $(\text{JOHN}, E, \text{PRIEST})$. The negational algorithm applied to this question would terminate with event $\ast$, a NO answer, by using the production $ED \rightarrow E$. 


4. The query “Is John a junkie?” would be coded as (JOHN, E, JUNKIE). Here the negational algorithm fails to terminate in either event $ or event * . There is insufficient information in the network to answer the question, and we must settle for DUNNO, or “maybe.” Were one interested in probable beliefs, we would point out that the $ operator implies that there is a possibility that John is a junkie. The subscript on the $ operator would provide a rough measure of the probability that John is a junkie.

Some experimental work has been devoted to measuring human reaction times to verify statements involving quantifiers. An early conclusion by Meyer (1970) was that existential statements of the form “Some $ are $” would be answered faster than universals of the form “All $ are $.” However, Rips (1973) and Glass, Holyoak, and O’Dell (1973) have pointed out serious flaws in Meyer’s experimental design and conclusions. Specifically, in Meyer’s “Some $ are $” experiment, the only false statements used were ones which are anomalous (e.g., “Some typhoons are wheats”), so that the subject could judge truth or falsity of a test statement by merely checking whether there was any semantic relatedness of the $ and $ concepts. When this confounding is removed in the comparison, by using closely related but false “some” statements (e.g., “Some sparrows are robins”), then there is no consistent difference in times to answer universal or existential questions.

The most informative results on quantifier verification times are those of Glass et al. (1974). They first collected restricted “association norms” to a set of quantified concepts of the form “(QUANT)$ are _____,” where the blank was to be filled by nouns or adjectives. Five different quantifiers were used in the test: All, Many, Some, Few, and No. A first result of interest is that the predicates associated to $ varied widely depending on the quantifier. This elementary fact is consistent with our network model, of course, since $ or $ or $ or $ arcs out of a concept node are distinct and would lead to a distinct set of predicates. The Glass et al. results were clear in showing that the normative predicates provided for the negative quantifiers (Few and No) were often obtained by taking the contrary (negation) of the predicate associated to the corresponding positive quantifier (Many and All). Thus, while “hot” was a frequent response to “All fires are _____,” its contrary, “cold,” was a frequent response to “No fires are ______.” Similarly, “Many blossoms” often leads to the response “beautiful,” whereas “Few blossoms” often leads to the
opposite predicate, "ugly." It is as though the subject stored mainly positive information and produced predicates to Few and No by using a "contrary production" on the predicates attached to the arcs labeled with All ($D_{10}$) and Many ($D_{8}$).

In a later experiment, Glass et al. showed that the reaction time to verify true positively quantified statements (All, Many, Some) was faster for highly related predicates than for minimally associated (but true) predicates. Such results are consistent with our network model, which assumes differing strengths (and different state transition times) even for arcs with the same label (say, the $D_{8}$ arcs associating node S to its "Many S" predicates). Also, as is typically found, Glass et al. reported that time to reject false positive statements was faster when $S$ and $P$ were unrelated (e.g., "All buildings are roses") than when $S$ and $P$ were related (e.g., "All buildings are houses").

So far we have discussed the representation and retrieval of statements involving only one quantifier. However, in natural language there are many statements with two or more quantifiers, such as "All good soldiers have a few wounds." The first-order language representation for this statement is

$$(\forall x)((x, E, SOLDIER)(\exists y)((y, E, WOUND) \& (x, H, y))).$$

In Fiksel (1973), the instance-checking procedures presented above are generalized to cover a large class of multiply quantified questions. These results are interesting from a formal point of view, but are difficult to justify in psychological terms due to a lack of relevant experimental data.

4. Propositional Information

In the foregoing sections, we treated simple sentences that closely match the binary relational structure of the semantic network. However, in everyday discourse, we employ complex sentences that express equally complex thoughts. To denote the latter in a network representation, we must introduce "token" nodes. A token node of a concept node is defined to be a node connected to that concept node by a special $T$ (for "token") relation and representing the same concept. Essentially, a token node is synonymous with its "parent" concept node and can be thought of as a replica of the parent node.

This refinement allows us to represent propositions such as "Geese fly south in the autumn," which is depicted in Fig. 17. Here we have defined three new relations: $J$ (an "agent" relation connecting the subject and predicate of a proposition), $D$ (denoting the direction of the action), and $W$ (connecting an action to its "time" of occurrence). The nodes GEESE and FLY within the dotted line $\pi$ are joined to their parent nodes by $T$ relations. In a similar manner, any other predication about the concepts GEESE or FLY would necessitate the creation of another group of
Fig. 17. A representation of "Geese fly south in the autumn" where the relations $J$, $T$, $D$, and $W$ denote "agent," "token," "direction," and "when," respectively.

token nodes. Without this structure we could not differentiate between the different instances of those concepts.

Although we restrict ourselves here to a brief illustration, an analogous approach to the representation of semantic information has been explored thoroughly in the conceptual dependency theory of Schank (1972). His objective has been to develop a semantic processor of the type discussed earlier in reference to Fig. 1. Assuming that propositional information has been stored in network form, the authors (1973) have shown that a question-answering procedure that will search the network for a particular proposition can be defined. This procedure is a straightforward extension of the inferential algorithm of Section 2.3; instead of searching for a simple path, the automata search for a set of intersecting paths.

In the same paper, the authors examine an even more general class of propositions; namely, those containing predications of other propositions, e.g., "John knows that Mary baked a cake." To obtain a network representation of such information, a higher order network of automata must be defined, in which each node denotes some proposition in the primary semantic network, and labeling arcs represent logical or causal relationships between propositional nodes. This structure permits the representation of any expression in the propositional calculus. Finally, a "belief" state (true or false) is defined for each propositional node-automaton, and a question-answering procedure that allows the higher-order network to compute autonomously and to adjust its belief states so that they are consistent is presented.

The above model bears a striking resemblance to the belief systems of Abelson (1973), who constructs networks of nodes that represent simple Schank-type conceptualizations of purposes, actions and states. Complex subgroups composed of these nodes are represented by single nodes on a still higher level of generality. A natural refinement of Abelson's model would be to embed an automaton at each node, thus permitting parallel processing of belief-oriented questions or problems.
A network simulation of semantic memory has been constructed in which nodes represent individual concept names and arcs represent labeled relations between nodes. Most simple predications can be represented as a path joining two nodes in the network. Besides being able to receive and store information, this network also is endowed with autonomous computational powers due to the finite-state automata embedded at each node. By sending signals along labeled arcs, an automaton can communicate with its immediate neighbors in the network. Local interactions of this type cause the automata to change state according to a prespecified state transition function. A complex pattern of state changes ensues from moment to moment according to the transition function, and these local changes enable the network to perform problem-solving tasks on a global level.

The task of question-answering is interpreted as the problem of finding a path in the semantic network joining two specified nodes and bearing a specified sequence of arc labels. A question-answering algorithm was presented in the form of a set of local automaton states and a transition function, and it was proved that the simple network specified could answer any single-path question. The only external control on the computational process consists of a terminal that excites the two endpoint nodes initially and that receives a notification when intersecting search signals indicate the existence of a solution path. If a path corresponding to the question is found, the answer to the question is YES; if no path is found, the answer is NO. We then showed that by modifying the simple automata slightly, a trace could be left behind during its search, such that the sequence of nodes corresponding to every solution path can be retrieved later. This permits the terminal to supply factual replies (“recall” completions) rather than simply YES or NO answers.

To extend the power of the question-answering algorithm, elementary syllogistic inferences were introduced through the use of “productions” or equivalence rules operating on the set of arc labels. Using these inferential productions, the network would answer a question by searching for any path between two specified nodes that is reducible to (or derivable from) a question path. It was shown that, using this inferential algorithm, the network will find the shortest solution path, if one exists. Again, a retracing procedure was introduced to allow the terminal to retrieve factual information as well as answering YES or NO. This inferential scheme was then illustrated for questions regarding the restricted semantic domain of animal taxonomy, and several variants of questions were shown to be answerable by such a network. Having established the basic mathematical results, we then turned to pragmatic considerations of how to realize a competent semantic network.

Methods were developed for representing implicit and explicit negations in semantic memory such as contrary and contradictory concepts and relations. The idea of a “negated relation” as an arc label was introduced, and a negated question
was defined as a question path along which exactly one relation is negated. A negational algorithm was then described that will give a definite NO answer to any question if it finds an appropriate negated path. Finally, we showed that the reaction-time predictions of the network for negational question-answering are consistent with existing experimental data.

The subject of quantification was then discussed. First-order logical notation was used to express universal and existential quantification, and it was shown how the terminal would translate first-order expressions into queries suitable for computation by the network. Thus, the network of automata was proven competent to answer queries involving quantifiers. To allow consistent storage of information about the overlap and inclusion of sets relating to quantification, certain relational labels were introduced, satisfying strict axioms. Sentences involving quantifiers and negation were discussed, and the network model was shown to be consistent with reaction-time experiments.

Representation of propositions about concepts stored in the semantic network is essential for a complete question-answering system. In another paper, the authors have augmented the model with a set of propositional graphs, each connected to parent nodes in the network by token or instantiation relations. A question to the augmented network was now defined as a sequence of elementary questions, which were processed serially by the terminal with certain variable nodes being fixed during the computation. Thus, the network of automata could test for the existence of any given propositional graph. The last extension of the model was to a higher order network in which every node represented some propositional graph. This permitted the nesting of propositions within propositions and the storage of complex propositions through the use of logical relations in the higher order network. A set of states and transition functions were introduced to reflect the computation of truth functions in this higher order network. Interactions between automata on the two levels of the model—the propositional level and higher order networks—enabled propositional inferences to be made for difficult tasks of question-answering.

The ideas described here by no means constitute a full realization of an intelligent semantic memory, but they provide a reliable basis for constructing such a realization since they address many of the fundamental issues of storage and retrieval. Moreover, the structural assumptions that we have adopted are borne out to a limited extent by psychological experimentation, suggesting that the network-of-automata provides a roughly plausible simulation of human question-answering behavior. Dreyfus (1972) has pointed out that artificial intelligence research, after its initial successes, seems to be running aground because of the inability of serial digital computers to search through unwieldy masses of data. The network-of-automata model is the first question-answering system that does not employ serial search processes with major guidance by an executive monitor. Our approach, of parallel search and local interaction, may open a promising avenue for the development of models that emulate
the speed, flexibility, and multiplicity of functions that characterize the human brain.

Of course, a major problem area that we have skirted is the design of a semantic processor compatible with the network model. Relevant research, such as that on the translation of natural language into an unambiguous conceptual representation, has met with challenging obstacles. Heuristic efforts, e.g., that of Schank (1972), encounter an ever-expanding array of semantic and pragmatic nuances, exceptional syntactic cases, and contextual factors. Similar problems seem to block the program of Suppes (1971), which aims for a rigorous, grammatically based model-theoretic semantics that seems mathematically appealing. In either case, if the complexities of language translation can be overcome, the network model would serve as a viable self-processing data base. At the opposite boundary of our work lies the problem of simulating the higher reasoning processes. The propositional network, as explicated here, is only a primitive step in that direction. To conquer the innumerable sutleties of supposition, inference, generalization, and extrapolation (e.g., see Abelson, 1973) is a far cry from solving the simple information retrieval tasks that have been considered so far. However, it is important to perfect a sufficiently powerful representation and computation scheme before tackling those more complex topics. To this end, the network of automata eliminates the inefficiencies of serial list-processing, and offers a discrete approximation to a physical system (the brain) in which the elements are continuously interacting.

REFERENCES


RIPS, L. Quantification and semantic memory: why some can, but all must. Unpublished manuscript, Stanford University Psychology Department, 1973.


**Received: March 26, 1975**