Procedural semantics for a question-answering machine

by W. A. WOODS

Harvard University
Cambridge, Massachusetts

INTRODUCTION

Structure of a question-answering system

Simmons has presented a survey of some fifteen experimental question-answering and related systems which have been constructed since 1959. These systems take input questions in natural English (subject to varying constraints) and attempt to answer the questions on the basis of a body of information, called the data base, which is stored inside the computer. This process can be conceptually divided into three phases—syntactic analysis, semantic analysis, and retrieval, as illustrated schematically in Figure 1. The first phase consists of parsing the input sentence into a structure which explicitly represents the grammatical relationships among the words of the sentence. Using this information the second component constructs a representation of the semantic content or “meaning” of the sentence. The remaining phase consists of procedures for either retrieving the answer directly from the data base, or else deducing the answer from information contained in the data base. The dotted lines in the figure represent the possible use of feedback from the later stages to aid in parsing and semantic interpretation.

The objective of the research described here has been to develop a uniform framework for performing the semantic interpretation of English sentences. It was motivated by the fact that, although there exists a variety of formal parsing algorithms for computing the syntactic structure of sentences, the problem of using this information to compute their semantic content remains obscure. A question-answering system provides an excellent vehicle for such a study, because it forces consideration of semantics from the point of view of setting up a correspondence between the structures of a sentence and objects in some model of the world (i.e., the contents of the data base).

The initial phases of this research consisted of an evaluation of the semantics of existing question-answering systems and the design of the semantic interpreter described here. A more detailed discussion of these topics may be found in the author’s Ph.D. thesis. Subsequently, the semantic interpreter has been implemented in LISP on the Harvard Time-Sharing System and tested on a variety of sentences.

Limited deductive systems

The systems reviewed by Simmons comprise a wide variety of approaches to the question-answering problem. The structures used for the data base range from natural language text (bolstered by indices, concordances, and synonym dictionaries) to various sorts of hierarchies and networks of lists and sets of attribute-value pairs. Moreover, the methods which are used in the
does take place and that it results in a significant reduction in the number of ambiguous parsings and semantic interpretations which are generated. On the other hand, it is possible to consider the operation of an experimental question-answering system without this feedback by systematically generating all ambiguous parsings of the sentence, and applying the semantic interpreter to each of them. This latter approach has been taken in this study.
three phases of processing are strongly influenced by the data structure chosen. In the extreme cases the syntactic analysis of English is done with a grammar whose constituents correspond not to grammatical categories of English but to structures in the data base. Consequently, the techniques developed in these systems would not be expected to carry over to new types of data bases, and this indeed seems to be the case.

One common feature of existing question-answering systems is that they all deal with limited areas of discourse and limited subsets of English. An argument in favor of this approach is that by studying restricted models one discovers principles and techniques with wider applicability. In the area of fact retrieval and automatic deduction useful techniques do seem to be emerging—notably the use of list-structured data bases for rapidly locating data and performing simple types of deduction. However, the efficiency of these techniques seems to be directly related to the restricted nature of the problem area. Those systems which enjoy reasonable efficiency gain it by applying special purpose solutions that capitalize on special features of the restricted problem area. Examples of such systems are Bobrow’s STUDENT, Raphael’s SIR, and Lindsay’s kinship relations program. Most of these systems gain their efficiency from special types of data structures and database organization as well as special retrieval techniques. On the other hand, attempts at completely general deductive systems, such as Black’s Specific Question-Answerer have encountered excessively long search times as the size of the data base increases. It seems that for efficient processing, different sorts of data require different sorts of data structures.

Expanding the universe of discourse

A promising method for achieving reasonable efficiency in larger, less restricted universe of discourse is to provide the system with a variety of different types of data structures and special purpose deduction routines for different subdomains of the universe of discourse. Integrating a variety of special purpose routines into a single system, however, requires a uniform syntactic and semantic framework. In general it is only after parsing and semantic interpretation have been carried out that such a system would be able to tell whether a sentence pertained to a given subdomain or not. Therefore, if the syntactic and semantic analyses were different for each subdomain, then the system would have to parse and interpret each sentence several times by different procedures in order to determine the appropriate subdomain. Moreover, there can be sentences that simultaneously deal with two or more subdomains, requiring a semantic framework in which phrases dealing with different subdomains can be combined.

Subsequent sections of this paper will describe a semantic interpretation procedure that can be used in an encyclopedic question-answering system hosting a multiplicity of special purpose deductive routines. This procedure will be illustrated by a semantic interpreter for a hypothetical question-answering system that answers questions about airline flight schedules using the Official Airline Guide as its data base. In this prototype system, a standard interface is imposed between the semantic interpreter and the retrieval component in order to achieve independence between the semantic interpretation procedure and the particular data structures and retrieval techniques employed in the data base. This interface takes the form of a formal query language that reflects the logical structure of English sentences rather than the structure of the data base. Unlike previous question-answering systems, the semantic interpretations of sentences in this system are procedures to be carried out rather than structures to be matched in the data base.

Semantics of question-answering systems

Among the question-answering systems reviewed by Simmons, Bobrow’s STUDENT provides the most insight into the semantic structure of a question-answering system. His “speaker’s model of the world” contains the following components:

1. a set of objects \( O \)
2. a set of functions \( F \)
3. a set of relations \( R \)
4. a set of propositions \( P \)
5. a set of semantic deductive rules.

The semantic interpretation of an English statement is expressed in terms of these components as a set of relations.

Bobrow explicitly points out the advantages of using “limited deductive models” that provide special facility for performing deductions in a restricted universe of discourse.

§This is true of the DEACON system for example.

**The notion of semantics presented here is quite different from the notion of semantics presented in Katz and Fodor’s “Structure of a Semantic Theory.” See the thesis for a comparison of the Katz and Fodor theory and that described here.
tions which are asserted to hold among specified objects. Functions carry ordered n-tuples of objects into objects (they need not be single valued). They are represented by linguistic forms such as "the number of ---," "the father of ---," "the sum of --- and ---.

The objects of the system are denoted by noun phrases—simple objects by simple nouns (e.g., "flights," "John," "Chicago," "3"), and functionally determined objects by composite nouns (e.g., "the number of flights," "the father of the boy," the sum of 3 and 4"). The relations are represented by verbs and their associated prepositional modifiers (e.g., "---flies from --- to ---", "--- hits ---", "--- equals ---"). The relations in the system correspond to the "concepts" which the system can "understand." The set of propositions {P_i} are instances of the relations with particular objects filled in for the arguments. They correspond to the "beliefs" of the system, i.e., the relations which it knows to hold among particular objects (e.g., "AA-57 flies from Boston to Chicago," "the sum of 3 and 4 equals 7"). The semantic deductive rules are procedures for deducing new propositions or beliefs from the ones which are already held by the system. These components are essentially those of a formal system for the first-order functional calculus without the quantifiers and logical connectives.

Although Bobrow's system does not recognize quantifiers and recognizes the logical connectives only as separators that mark the boundaries of kernel sentences, question-answering systems in general will require all of these components as part of their semantic structure. Both quantifiers and logical connectives, however, can be viewed as predicates which take propositions as arguments and can be included in the above framework by simply relaxing the restriction that arguments of relations must be objects in the data base. Thus, we will permit arguments of predicates and functions to be filled by propositions as well as data base objects.† For example, "AND" can be represented as a predicate which takes two propositions as arguments and is true if both of its arguments are true. Likewise, the quantifier, "ALL," can be thought of as a predicate which takes one designator (a variable name) and one proposition (containing a free occurrence of the variable) as arguments. This quantifier ALL is true if the quantified proposition is true for every assignment to the variable.

There is one remaining component of the semantic structure which has been implicit in previous question-answering systems but has not been explicitly recognized. This component is the set of commands that the system will carry out. In most systems there has been only one or at most two commands—"record X as data" and/or "answer X," and they have been left implicit rather than explicitly represented in the semantic interpretations of these systems. In our discussion, however, the semantic interpretations of sentences will explicitly represent commands to the system.

Semantic primitives

We will call the predicates, functions, and commands which a question-answering system "understands" the semantic primitives of the system, and we will say that a collection of semantic primitives is adequate for a given data base if, using them to ask questions, it is possible to reconstruct the data in the data base (i.e., everything in the data base is retrievable). This is the minimum requirement which a set of primitives must satisfy. Beyond this minimum requirement additional primitives may be added to improve efficiency or usefulness.

For unrestricted English, an adequate set of primitives would be large, but for a restricted area of discourse an adequate set of primitives may be quite small. An adequate set of primitives for the prototype system to interrogate the Airline Guide is listed in Table I.

It is assumed that the retrieval component contains a programmed subroutine or other operational procedure for each semantic primitive which defines the meaning of that primitive. Thus the predicate CONNECT, for example, will be defined by a programmed subroutine named CONNECT, which given three arguments X1, X2, and X3, determines the truth or falsity of the proposition CONNECT (X1, X2, X3). For the Airline Guide Flight Schedules Table, the operation of this subroutine would be to scan the table for the section which deals with flights to X3. It would then scan this section for the subsection which contains flights from X2 to X3, and finally it would scan this subsection to see if flight X1 is listed there. For a different organization of the data base, the program for CONNECT might be defined differently. For example, in another table of the Airline Guide, the Flight Itineraries Table, the computation would proceed by looking in the table under flight X1 to find the flight itinerary for flight X1 and then looking down that list for the place X2 and, if X2 is found, looking further to see if X3 occurs later in the list. Thus the definition of primitive concepts by programmed subroutines allows complete freedom from the particular organization of the data base. It even allows one to extend the system to concepts which are not explicitly contained in the data base.

†Although such constructions are not handled by the present system, the decision to allow propositions as arguments of predicates paves the way for handling embedded sentences as in "Columbus thought that the world was round."
**Primitive Commands.**

- **TEST (D1)**: Test whether D1 is a true proposition or not.
- **LIST (X1)**: Print the name of the object X1.

**Primitive Predicates.**

- **CONNECT (X1, X2, X3)**: Flight X1 goes from place X2 to place X3.
- **DEPART (X1, X2)**: Flight X1 leaves place X2.
- **ARRIVE (X1, X2)**: Flight X1 goes to place X2.
- **DAY (X1, X2, X3)**: Flight X1 leaves place X2 on day X3.
- **IN (X1, X2)**: Airport X1 is in city X2.
- **SERVCLASS (X1, X2)**: Flight X1 has service of class X2.
- **MEALSERV (X1, X2)**: Flight X1 has type X2 meal service.
- **JET (X1)**: Flight X1 is a jet.
- **DAY (X1)**: X1 is a day of the week (e.g. Monday).
- **TIME (X1)**: X1 is a time (e.g. 4:00 p.m.).
- **FLIGHT (X1)**: X1 is a flight (e.g. AA-57).
- **AIRLINE (X1)**: X1 is an airline (e.g. American).
- **AIRPORT (X1)**: X1 is an airport (e.g. JFK).
- **CITY (X1)**: X1 is a city (e.g. Boston).
- **PLACE (X1)**: X1 is an airport or a city.
- **PLANE (X1)**: X1 is a type of plane (e.g. DC-3).
- **MEAL (X1)**: X1 is a type of meal service (e.g. breakfast).
- **CLASS (X1)**: X1 is a class of service (e.g. first-class).
- **EQUAL (X1, X2)**: X1 is the same as X2.
- **GREATER (X1, X2)**: X1 > X2 (where X1 and X2 are times or numbers).
- **AND (S1, S2)**: S1 and S2.
- **OR (S1, S2)**: S1 or S2 (where S1 and S2 are propositions).
- **NOT (S1)**: S1 is false.
- **IFTHEN (S1, S2)**: If S1 then S2.

**Primitive Functions.**

- **TZ (X1)**: The time zone of place X1.
- **DTIME (X1, X2)**: The departure time of flight X1 from place X2.
- **ATIME (X1, X2)**: The arrival time of flight X1 in place X2.
- **NUMSTOPS (X1, X2, X3)**: The number of stops of flight X1 between place X2 and place X3.
- **OWNER (X1)**: The airline which operates flight X1.
- **EQUIP (X1)**: The type of plane of flight X1.
- **FARE (X1, X2, X3, X4)**: The cost of an X3 type ticket from place X1 to place X2 with service of class X4 (e.g. the cost of a one-way ticket from Boston to Chicago with first-class service).

**TABLE I—A set of semantic primitives for the flight schedule table.**

The query language

Once a set of semantic primitives has been selected and an adequate query language has been defined, one can proceed to the design of the semantic interpreter without concern for the data structures that will be used in the data base or the retrieval techniques that will be used. It suffices to assume that the retrieval component will contain procedures for determining the truth of any given predicate (given its arguments), for determining the value of a function (for given arguments), and for carrying out any of the primitive commands. Communication between the two components will take place in terms of the names of the primitive predicates, functions, and commands which the system understands.

The query language contains three basic types of construction—commands, propositions, and designators. At the top level, every expression in the query language will be a command that will in general contain propositions and/or designators as arguments. The prototype system contains two basic commands—TEST and LIST, where TEST(P1) is a command to determine...
the truth-value of a proposition P1 and LIST(X1) is a command to print out the name of the object designated by a designator X1. A proposition consists of a predicate name followed by a list of its arguments enclosed in parentheses (e.g., DEPART(AA-57, BOSTON)), and a designator consists of a proper name (e.g., Boston), a variable name (e.g., X1), or a function name followed by a list of arguments (e.g., OWNER(AA-57)). Moreover, both propositions and commands may be quantified by quantifiers of the form:

\[(\text{FOR QUANT } X / \text{CLASS } : R(X) ; P(X) )\]

where QUANT is a quantifier (EVERY, SOME, THE, etc.), X is the variable being quantified, CLASS is the name of a set over which quantification is to range, R(X) is a (possibly vacuous) further restriction on the range of quantification, and P(X) is the proposition or command being quantified. For example, (FOR EVERY X1/FLIGHT: DEPART(X1, BOSTON) ; LIST(X1)) is a quantified command which directs the retrieval component to print the name of every flight which leaves Boston. It is assumed that there are a relatively small number of sets that can take the place of CLASS in these quantifiers and that for each such set, the retrieval component contains a successor function which enumerates the members of that set. More restricted ranges of quantification are obtained by imposing additional restrictions R(X). The reader is referred to the thesis for a discussion of the advantages of this formulation of the quantifiers.

In addition to the ordinary quantifiers SOME and EVERY and the quantifier THE, the prototype system contains two types of numerical quantifiers. The first type,

\[(\text{FOR n MANY } X / \text{CLASS } : R(X) ; P(X) )\]

is true if there are at least n members X in CLASS such that R(X) and P(X) are true. The second type allows the insertion of an arbitrary proposition to specify a condition on the number of objects—e.g.,

\[(\text{FOR GREATER } (N, 30) \text{ MANY } X / \text{CLASS } : R(X) ; P(X) )\]

Also there is a counting function, NUMBER (X/CLASS : R(X)), which returns as value the number of objects X in CLASS for which R(X) is true. For a detailed description of the syntax of these expressions and their effects when carried out in a retrieval component, the reader is again referred to the thesis. Examples of query language expressions and their meanings are listed in Table II. The language is roughly equivalent to the first-order functional calculus, with somewhat more elaborate quantifiers and with the addition of commands, a concept which is not interpretable in systems of formal logic.

### Semantic interpretation

#### Semantic rules

The task of the Semantic Interpreter is to “look at” an input phrase marker and to translate the phrase marker into a formal representation of its meaning in terms of the semantic primitives of the system. Both in human beings and in the machine this process must be finitely specifiable, even though the class of phrase markers to be interpreted and the class of interpretations to be produced are both infinite. Thus, the Interpreter must decompose the phrase marker into substructures which it can interpret directly, and it must compute the interpretation of larger structures as some
sort of composite of the interpretations of its substructures.

In the prototype described here, the manner in which the semantic interpretation of a construction is built up from the semantic interpretations of its constituents is specified by a finite set of semantic rules. These rules are represented in a "pattern \(\Rightarrow\) action" format, where, the "pattern" is a description of a fragment of a phrase marker together with some conditions on the constituents of that fragment, and the "action," specifies the semantic interpretation of such a fragment in terms of the interpretations of its constituents. These rules have a kinship to the "pattern \(\Rightarrow\) substitution" rules of Post production systems and the rewrite rules of phrase-structure grammars, but the effect is somewhat different. The action in this case is not a substitution but an operation that attaches a semantic interpretation to the given fragment.

In the syntax tree of a sentence, there are basically two types of nodes that receive semantic interpretations—S nodes, corresponding to sentences and clauses, and NP nodes, corresponding to noun phrases. The semantic interpretations of the former are either propositions or commands, while the semantic interpretations of noun phrases are designators. Corresponding to these two types of nodes, there are two processors—the S-processor, for interpreting S nodes, and the NP-processor for interpreting NP nodes. Each of these processors attaches a semantic interpretation to the node of the syntax tree that it is interpreting. In addition, the NP-processor may also generate quantifiers that are to govern the sentence of which the NP is a constituent.

**The S-processor**

We may illustrate the operation of the S-processor independent of the NP-processor by considering sentences whose noun phrases are all proper nouns (which we assume to be directly interpretable). For example, consider the sentence:

**AA-57 flies from Boston to Chicago.**

A phrase marker for this sentence might look something like that of Figure 2.

Since verbs in English correspond roughly to predicates, and noun phrases are used to denote the arguments of the predicate, the verb in the structure diagram will be the primary factor in determining the primitive predicate (or composite of primitive predicates and functions) which represents the meaning of a sentence. For example, in Figure 2, the predicate will CONNECT as defined in Table 1. In addition to determining the predicate, however, the S-processor must verify that all of the required arguments for the predicate are present in the structure diagram, that they meet appropriate conditions for the predicate to take them as arguments, and that they have the correct grammatical relationships with the verb. For example, for the predicate CONNECT it is necessary that the subject be a flight, and it is necessary that there be prepositional phrases whose objects are places representing origin (from) and destination (to). These tests are performed in order to rule out semantically bad interpretations and to choose the correct senses from among the various senses which a given verb may have. For example, the sentence:

**Boston flies from AA-57 to Chicago**

is semantically bad, while the verb "fly" in:

**Can I fly from Boston to Chicago?**

has a different sense from that of the sentence in Figure 2. Thus the S-processor must be able to ask questions about the constituents of a sentence (e.g., whether they are flights or places, etc.) and about the grammatical relationships among them (e.g., what is the subject, is there a prepositional phrase, what is the object of the prepositional phrase, etc.).

The grammatical relations among elements of a phrase marker are defined by partial tree structures which match subtrees of the phrase marker. A set of such partial structures is given in Figure 3. The numbers in parentheses in these partial trees are labels used to refer to the nodes to which they are connected. Thus, two nodes A and B in a phrase marker are in the subject-verb relation if the partial tree G1 matches a subtree of the phrase marker in such a way that the noun phrase (1) matches node A and the verb (2) matches node B. Notice that this grammatical relation is defined with respect to the particular S node that matches the topmost node (or root) of the partial tree structure G1.
Structure of the semantic rules

The "pattern" part of the semantic rules is specified in terms of a template consisting of the name of a partial tree structure (such as G1, G2, G3) together with a collection of conditions on the numbered nodes of that tree structure. The conditions are expressed in terms of the semantic predicates of the system and the additional relation "=" which is used in the format "(n) = w" to mean that the terminal string of the subtree rooted at node n is identical to the string w. An example of a template is:

\[(G1: \text{Flight } ((1) \text{ and } (2) = \text{fly})].\]

This template matches a subtree of a phrase marker if the tree structure G1 matches that subtree and the node numbered (1) (i.e., the subject of the sentence) denotes a flight and the node numbered (2) directly dominates the word "fly" (i.e., the verb is "fly"). The "pattern" part of a semantic rule consists of a set of such templates, with each template identified by a preceding integer so that the nodes which are numbered within it may be referred to in the right-hand side (or "action" part) of the rule. The node which matches the node (m) in the template numbered n is referred to on the right-hand side of the semantic rule as "n-m."

The pattern of a semantic rule matches a given node A in a phrase marker if each template of the pattern matches a subtree of the phrase marker rooted at node A. The semantic interpretation of a node is obtained by successively applying semantic rules to the node, with each matching rule specifying a part of the semantic interpretation. The total semantic interpretation is the conjunction of these parts. For example, the topmost S node of the phrase marker of Figure 2 is matched by the semantic rule:

\[
1- (G1: \text{FLIGHT } ((1) \text{ and } (2) = \text{fly}) \text{ and } \\
2- (G3: (1) = \text{from and } \text{PLACE } ((2)) \text{ and } \\
3- (G3: (1) = \text{to and } \text{PLACE } ((2)) \text{ and } \\
=> \text{CONNECT } (1-1, 2-2, 3-2)
\]

which is to be interpreted as follows:

If the node under consideration is the root of a subtree matching G1 and the node matching (1) denotes a flight and the node matching (2) directly dominates the word "fly", and if the given node is also the root of a subtree matching G3 with the node matching (1) directly dominating the word "from" and the node matching (2) denoting a place, and finally if the node is the root of another subtree matching G3 with the node matching (1) directly dominating the word "to" and the node matching (2) denoting a place, then part of the semantic interpretation of the node is CONNECT (1-1, 2-2, 3-2), where 1-1 denotes the interpretation of the node which matches node (1) of template 1 (i.e., the template (G1: FLIGHT((1) and (2) = fly)), 2-2 denotes the interpretation of the node which matches node (2) of template 2, and 3-2 denotes the interpretation of the node which matches node (2) of template 3.

Thus, part of the interpretation of the sentence in Figure 2 is:

\[\text{CONNECT } (AA-57 \text{ Boston, Chicago}).\]

Notice that this rule treats the prepositional phrases "from Boston" and "to Chicago" as commutative, since they are matched by different templates and no relative order is specified. That is, this same rule interprets both sentences, "AA-57 flies from Boston to Chicago" and "AA-57 flies to Chicago from Boston." In a situation in which the order of the prepositional phrases was semantically relevant, one would use a single template to specify both prepositional phrases and therefore also their relative order.

S-rules

The semantic rules for the S-processor are called "S-rules" to distinguish them from the rules for the NP-processor which will be described in a later section. A representative set of S-rules for the prototype S-processor is given in Table III. Notice that in addition to the rule S1 that we have already discussed, the rules S4 and S6 will also match the phrase marker of Figure 2, resulting in the redundant information DEPART (AA-57, Boston) and ARRIVE (AA-57, Chicago), so that the total semantic interpretation of the phrase marker would be:
TABLE III — A representative set of S-rules


This redundant information may simply be left alone since it does no harm other than to cut down on efficiency. In the implementation of the prototype, however, this redundancy is eliminated by an appropriate ordering of the semantic rules, and a procedure for
determining what rule to try next depending on the success or failure of the previous rule.

Notice that these rules are able to describe the close connections between verbs and the prepositions which they can take, as embodied in constructions such as "fly from X to Y", "leave from X for Y," "arrive in X." They are also capable of distinguishing the meanings of the preposition "at" in "arrive at Chicago" and "arrive at 4:00 p.m." Notice furthermore that although the number of semantic rules in a system may be quite large, only a small number of rules can apply for any given verb, so that for interpreting a given phrase marker we need actually consider only a small number of semantic rules. In the implementation of the prototype the dictionary contains entries for each verb and noun specifying the set of possible semantic rules that could apply to a construction with that word as its head. This entry also specifies the order in which the rules are to be tried, and this order may in general be conditional on the success or failure of previous rules, allowing considerable flexibility for optimizing the interpretation procedure.

Semantic features

Notice that in the semantic rules of Table III, the semantic conditions, FLIGHT((1)), Place ((2)), etc., are all single place predicates corresponding to membership in some class of objects—i.e., flights, airports, etc. Furthermore, the truth of such conditions can be deduced from the noun of the noun phrase, and the necessary information can be recorded in the dictionary entries for the nouns. That is, in the dictionary entry for a proper noun, we can record semantic features corresponding to the sets of which the named object is a member, e.g.:

AA-57/FLIGHT
Boston/CITY, PLACE
JFK/AIRPORT

In the dictionary entry for a common noun, we can record features corresponding to the sets of which the class denoted by the noun is a subset, e.g.:

flight/FLIGHT
plane/FLIGHT
city/CITY,PLACE
town/CITY, PLACE
place/PLACE
airport/AIRPORT, PLACE.

These set-membership conditions can then be tested by referring to the dictionary entry for the noun to see if the indicated set is listed there as a semantic feature. More complex semantic conditions could always be defined by programmed subroutines that the semantic interpreter may call (and this facility is available in the prototype), but there has not yet been any need for semantic conditions more complex than simple set-membership conditions.

The generation of quantifiers

When considering the semantic interpretation of sentences containing quantified nouns, one discovers that a recursive top-down method of analysis is preferable to a bottom to top analysis. This is due to the fact that the quantifier which results from the interpretation of a noun phrase must be attached to the S node which it governs. When processing from the bottom up, we encounter the quantified noun phrase before we know where the appropriate S node is. In the recursive process from the top down, however, we encounter the S node first, and while we are still "looking" at it, the S-processor calls the NP-processor for each NP that must be interpreted. The NP-processor, then, returns the appropriate quantifiers to the S-processor, which is still "looking" at the S node to which they are to be attached.

Specifically the S-processor begins a working string (which may actually be a tree instead of a string) that will ultimately contain the representation of the semantic meaning of the S node being processed. Initially this string consists only of a single symbol \( \Delta \). Each time the NP-processor is called to process a quantified noun, it tags the NP that it is processing with a variable name, and returns one of the quantifiers described earlier. This quantifier contains the symbol \( \Delta \) indicating the position in the quantifier where the remaining semantic interpretation of the S node is to be inserted (e.g., \( \text{FOR SOME X}/\text{FLIGHT}:-; \Delta \)). The S-processor then replaces the symbol \( \Delta \) in its working string by the quantifier which the NP-processor has returned (thus maintaining exactly one symbol \( \Delta \) in the working string). When all of the NP's have been processed, the S-processor uses the S-rules to interpret the sentence in terms of the variable names which the NP-processor has tagged onto the NP's. It then replaces the symbol \( \Delta \) in the working string with the interpretation specified by the S-rules, thus eliminating the symbol \( \Delta \) from the working string, which now contains the complete analysis of the S node. Finally, the S-processor tags the S node with the interpretation which it has built up in its working string and exits.

As an example to illustrate the process just described,
before explaining the mechanism in detail, consider the sentence "Every flight from Boston to Chicago leaves Boston at 8:00 a.m.,” which is diagrammed in Figure 4.* Looking at the topmost S, the S-processor starts a working string consisting only of the symbol Δ and then calls the NP-processor to process the NP “every flight from Boston to Chicago.” The NP-processor creates a new variable X1 which has not been used before (e.g. by concatenating an integer to the right of the symbol X) and tags the NP with it. The NP-processor then (by mechanism to be discussed in the next section) returns the quantifier (FOR EVERY X1/FLIGHT: CONNECT(X1, Boston, Chicago); Δ) to the S-processor. The S-processor now substitutes this quantifier for the symbol Δ in its working string (giving a new working string which consists of just the quantifier (FOR EVERY X1/FLIGHT: CONNECT(X1, Boston, Chicago); Δ)). It then uses the S-rules (in particular rule S9 from Table III) to obtain the semantic interpretation, EQUAL (DTIME (X1, Boston), 8:00 a.m.), which it finally substitutes for the symbol Δ in the working string to give the final semantic interpretation of the sentence as: (FOR EVERY X1/FLIGHT: CONNECT(X1, Boston, Chicago); EQUAL (DTIME (X1, Boston), 8:00 a.m.)). This mechanism allows for the nesting of an arbitrary number of quantifiers (depending on the number of quantified NP’s) before the S-processor finally interprets the sentence via the S-rules.

The NP-processor

Like the S-processor, the operation of the NP-processor is determined by a set of semantic rules in a “pattern ⇒ action” format. The “pattern” part of these rules is a set of templates just as for S-rules, except that the partial tree structures used in the templates are dominated by the node NP instead of S. Some partial tree structures for the NP-processor are given in Figure 5. The difference between G9 and G11 will become apparent later.

The process of interpreting a noun phrase consists of three parts: (1) determining the quantifier to be used, (2) determining the range of quantification, and (3) determining additional restrictive clauses on the range of quantification. The first part is governed primarily by the determiner of the noun phrase and is specified by a set of D-rules, a sample of which are shown in Table IV. The symbol Δ in these rules, as we mentioned before, marks the position where the semantic interpretation of the sentence will ultimately be inserted. Notice that this position is different in the quantifier that results from the interrogative determiners (rule D6) than it is for the others. The symbol V marks the position where the

*In this diagram, NU is a syntactic category for the number of a noun phrase (SG for singular and PL for plural). NPR is the category for proper nouns. The other node names should be self-explanatory.

**TABLE IV—D-rules for the NP-processor**
range of quantification and other restrictive clauses are to be inserted by the NP-processor. The symbol X stands for the variable name which the NP-processor will create. The NP-processor will tag the NP with this variable name and use it in place of X in the quantifier which it produces.

The second phase of the NP-processor is to determine the range of quantification. This is primarily determined by the noun of the noun phrase, and is specified by a set of N-rules. The right hand side of these rules specifies a successor function for the set over which quantification is to range and possibly some restrictions on the range. It indicates by the symbol "V" the position where any additional restrictions are to be inserted. The NP-processor will replace the symbol "V" in the quantifier (produced by the D-rules) with the right hand side of a matching N-rule (which will maintain exactly one symbol "V" in the working string). A small sample of relatively straightforward N-rules is given in Table V. Notice the ease of rule N2 where the noun specifies both a successor function and an additional restriction; it is this mechanism that allows us to have a small number of successor functions in the system and still use a considerably larger variety of nouns. Some more complicated N-rules will be described in the next section.

The third phase of the NP-processor attaches additional restrictive clauses onto the range of quantification. For relative clauses modifying the NP the NP-processor does this automatically by first taking the variable X which it has created and using it to tag the NP in the relative clause which designates the relative pronoun, "which," "who," or "that." It then calls the S-processor to interpret the relative clause. The interpretations of adjectives and prepositional modifiers, however, are specified by a set of R-rules. The right-hand side of these rules consists of a proposition which is to be added as a restriction to the range of quantification. When the third phase of the NP-processor has determined all of the restrictions implied by R-rules and by relative clauses, it conjoins them and replaces the symbol "V" in the working string with this conjunction. If there are no restrictions, it replaces the "V" with a vacuous restriction (represented by a dash). A sample set of R-rules is given in Table VI. Notice the use of a noun phrase denoting an airline as an adjective in rule R8. This is the reason for the partial tree structure GI1 for noun phrases used as adjectives as distinct from G9 for lexical adjectives.

R1 1-(G9: (1) = non-stop) and
    2-(G10: (1) = from and PLACE (2)) and
    3-(G10: (1) = to and PLACE (2))
⇒ EQUAL (NUMSTOPS (X, 2-2, 3-2), 0)
e.g. "a non-stop flight from Boston to Chicago"

R2 1-(G9: (1) = first-class)
⇒ SERVCLASS (X, first-class)
e.g. "a first-class flight"

R3 1-(G9: (1) = jet)
⇒ JET (X)
e.g. "a jet flight"

R4 1-(G9: (1) = propeller)
⇒ NOT (JET (X))
e.g. "a propeller flight"

R5 1-(G10: (1) = from and PLACE (2)) and
    2-(G10: (1) = to and PLACE (2))
⇒ CONNECT (X, 1-2, 2-2)
e.g. "a flight from Boston to Chicago"

R6 1-(G10: (1) = from and PLACE (2))
⇒ DEPART (X, 1-2)
e.g. "a flight from Boston"

R7 1-(G10: (1) = to and PLACE (2))
⇒ ARRIVE (X, 1-2)
e.g. "a flight to Chicago"

R8 1-(G11: AIRLINE (1)) and
    2-(G8: (1) = flight or (1) = plane or (1) = jet)
⇒ EQUAL (OWNER (X), 1-1)
e.g. "an American Airlines flight"

R9 1-(G9: (1) = morning or (1) = a.m.) and
    2-(G8: (1) = flight or (1) = plane or (1) = jet) and
    3-(G10: (1) = from and PLACE (2))
⇒ GREATER (1200, DTIME (X, 3-2))
e.g. "an A. morning flight from Boston"

<table>
<thead>
<tr>
<th>N1</th>
<th>1-(G8: (1) = flight or (1) = plane) ⇒ FLIGHT: V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. &quot;a flight&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N2</th>
<th>1-(G8: (1) = jet) ⇒ FLIGHT: JET (X) AND V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. &quot;a jet&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N3</th>
<th>1-(G8: (1) = airline or (1) = carrier) ⇒ AIRLINE: V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. &quot;an airline&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N4</th>
<th>1-(G8: (1) = city or (1) = town) ⇒ CITY: V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. &quot;a city&quot;</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>N5</th>
<th>1-(G8: (1) = airport or (1) = place) ⇒ AIRPORT: V</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>e.g. &quot;an airport&quot;</td>
</tr>
</tbody>
</table>

TABLE V—Some sample N-rules for the NP-processor

TABLE VI—Some R-rules for the NP-processor
Functionally determined objects

As we mentioned previously, linguistic forms such as "the departure time of ———- from ———-" correspond to functions which assign values (in this case a time) to each combination of arguments (in this case a flight and a place). That is, a noun phrase such as "the departure time of AA-57 from Boston" is a complex designator for an object that is functionally determined from the arguments "AA-57" and "Boston." Other such constructions are "owner of X1," "the father of the boy," "the captain of the ship," "the sum of x and y." In such constructions, the noun of the noun phrase is the name of the function, and the arguments are specified by prepositional phrases (usually with the preposition "of").

A few question-answering systems have recognized the functional nature of such constructions. Bobrow gives a general discussion of the correspondence between such linguistic forms and functions, and he uses such constructions for addition, subtraction, multiplication, etc., in his system. The DEACON system also, can handle a limited class of such functional noun phrases, namely those which denote the value of some attribute of an object. For example "the owner of the Enterprise" denotes "the Unites States," which is the value of the attribute "owner" associated with the object "Enterprise" in the data base. In both of the above systems, however, only single-valued functions are considered. In such cases the determiner of a functional noun phrase is always 'the,' and in both systems this determiner is either dropped or has no semantic effect.

Not all such functions are single-valued, however. Take for example, "a member of the crew," "a cousin of the duke," "an eigenvalue of the matrix," "an officer of the ship," "a root of the equation." In such cases the functional noun phrase may contain any of the determiners used to denote quantification over a set, e.g., "every officer of the ship," "no officer of the ship," "some officer of the ship," "which officer of the ship." We have already described a mechanism which handles these quantifiers given a successor function which enumerates the members of a set. We can handle them for functionally determined objects by defining a successor function which takes as additional arguments the arguments of the functional noun phrase. For example, a successor function OFFICER (x, y) could be defined to enumerate the officers of a ship x. Then, if the values of the pointers to the officers of x were a1, a2, ..., an, we would have:

\[
\begin{align*}
\text{OFFICER} (x; 0) &= a_1 \\
\text{OFFICER} (x, a_1) &= a_2 \\
&\vdots \\
\text{OFFICER} (x, a_n) &= \text{END}.
\end{align*}
\]

With this mechanism, we can represent the range of quantification in the quantifiers by naming a successor function together with the fixed arguments of the functional noun phrase (the argument for the index of enumeration is always assumed and need not be represented). Thus the sentence:

Every officer of the Enterprise is ashore

can be represented as:

\[
\text{(FOR EVERY X1/OFFICER(ENTERPRISE) :- ; ASHORE(X1))}
\]

where ASHORE(X) is a predicate meaning X is ashore.

Once the above mechanism has been constructed for multiple-valued functions, single-valued functions are seen to fall naturally into the same scheme. In this case, the successor function will enumerate exactly one object before returning the value "END" and the retrieval mechanism for the quantifier "THE" will automatically verify whether the use of the determiner "the" is correct (i.e., it will verify that exactly one object is generated). This mechanism will handle the usage of the determiner "the" in all cases where the object involved is uniquely determined, even in the case of a "very small" ship which has only one officer (or in the case where additional modifiers appear in the functional noun phrase and restrict the range of quantification, as in "the redheaded officer of the ship"). This approach to quantified noun phrases allows full generality in handling both functional and non-functional noun phrases with the full range of possible determiners and modifiers.

Since, in the formulation of functional noun phrases described above, the function of the determiner and additional modifiers in the noun phrase is the same as for non-functional noun phrases, the same D-rules and R-rules will work for both functional and non-functional noun phrases. Furthermore, the N-rules for functional noun phrases will operate exactly as do the N-rules for non-functional noun phrases. A sample set of N-rules for functional noun phrases is given in Table VII. Notice that in rule N12, some of the arguments of the function are specified by adjectives as well as by prepositional phrases. Notice also that many of the function names are compound forms. Note especially that while the construction "number of stops" has a meaning to the user which is based on counting stops, the system has no data base objects corresponding to stops and does no counting. Instead, it looks up the number of stops in a table. Thus the phrase "number of stops" is effectively an idiom to the system and serves merely to name a function.
Procedural Semantics for Question-Answering Machine

N6 1-(G8: (1) = departure time) and
2-(G10: (1) = of and FLIGHT (2)) and
3-(G10: (1) = from and PLACE (2))
⇒ DTIME (2-2, 3-2): v

e.g. "the departure time of AA-57 from Boston"

N7 1-(G8: (1) = arrival time) and
2-(G10: (1) = of and FLIGHT (2)) and
3-(G10: (1) = in or (1) = at) and PLACE (2))
⇒ ATIME (2-2, 3-2): v

e.g. "The arrival time of AA-57 in Chicago"

N8 1-(G8: (1) = owner or (1) = operator) and
2-(G10: (1) = of and FLIGHT (2))
⇒ OWNER (2-2): v

e.g. "the operator of AA-57"

N9 1-(G8: (1) = time zone) and
2-(G10: (1) = of and PLACE (2))
⇒ TZ (2-2): v

e.g. "the time zone of Boston"

N10 1-(G8: (1) = number of stops) and
2-(G10: (1) = of and FLIGHT (2)) and
3-(G12: (1) = between and PLACE (2) and
PLACE (3))
⇒ NUMSTOPS (2-2, 3-2, 3-3): v

e.g. "the number of stops of AA-57 between Boston and Chicago"

N11 1-(G8: (1) = type of plane or (1) = kind of plane) and
2-(G10: (1) = of and FLIGHT (2))
⇒ EQUIP (2-2): v

e.g. "the type of plane of AA-57"

N12 1-(G9: (1) = fare) and
2-(G9: (1) = one-way or (1) = round-trip) and
3-(G9: (1) = first-class or (1) = coach or (1) = stand by)
and
4-(G10: (1) = from and PLACE (2)) and
5-(G10: (1) = to and PLACE (2))
⇒ FARE (4-2, 5-2, 2-1, 3-1): v

e.g. "one-way first-class fare from Boston to Chicago"

TABLE VII—Some N-rules for functional noun phrases.

"What" questions

One frequently asks for the value of a function for specified arguments by means of a "what" question as:

What is the departure time of AA-57 from Boston?

If it can be avoided, we do not want such questions to be answered by ranging over the universe of possible answers and choosing the correct one, but instead we want to determine the correct answer directly from the arguments of the function by means of the operational procedure which defines the function. Thus the interpretation of this sentence should be:

(FOR THE X1/DTIME(AA-57, Boston) :-;
LIST(X1))

rather than:

(FOR THE X1/TIME: EQUAL (X1, DTIME(AA-57, Boston)); LIST(X1)).

The procedure that we have just described will generate the quantifier:

(FOR THE X1/DTIME(AA-57, Boston) :-; Δ)

which we require. All that remains is to interpret the sentence itself as the command LIST. This is specified by following semantic rule:

S23 1-(G1: (2) = be) and
2-(G2: (1) = be and (2) = what)
⇒ LIST (1-1).

Thus, assuming that the deep structure of the sentence is something like that of Figure 6, then rule S23 will apply to the topmost S, but will first call for the semantic interpretation of the subject noun phrase. Rules D5 and N6 will then generate the quantifier, and finally, rule S23 will insert the command LIST(X1) to give the final interpretation:

(FOR THE X1/DTIME(AA-57, Boston) :-; LIST(X1)).

CONCLUSION

Summary

In the preceding sections, we have presented a proposal for a uniform method of performing the semantic interpretation of English questions for a question-answering system. We have presented an operational definition of semantic meaning in terms of programmed subroutines and have outlined a formal query language for representing the semantic interpretation of questions and commands. We have concentrated on the prob-

FIGURE 6—Phrase marker for a "what" question
len of translating from a syntax tree that represents the syntactic structure of the input question into an expression in the query language which represents the "meaning" of the question, and we have developed a procedure for performing this translation by means of a set of semantic rules. While the techniques described are not completely general for unrestricted English usage, they are applicable in a wide variety of situations in which the computer is called upon to "understand" English input and take some appropriate action.

In addition to recognizing the predicate which corresponds to a given sentence, the proposed system handles universal and existential quantifiers, numerical quantifiers, interrogative determiners, and the determiner "the." It handles relative clauses, adjectival and prepositional modifiers in noun phrases, and functionally determined noun phrases. It also takes into proper consideration whether a noun phrase is singular or plural. This combination of features has not been attained in any existing question-answering system although some have appeared in isolation in various systems. To my knowledge, the handling of the determiner "the," the treatment of relative clauses, and the distinction between singular and plural nouns are new, as is the uniform framework in which these techniques are integrated. Moreover, the techniques presented here are general and easily extended to other situations in which there is a well-defined data base.

**Implementation**

A sample implementation of the prototype system for the flight schedules problem has been designed to interpret the deep structures assigned by a small transformational grammar containing 40 transformations. Although these phrase markers could in principle be mechanically assigned, they are currently produced by hand, since the central concern of this research is the design of the semantic interpreter. The semantic interpreter itself has been programmed in LISP and tested on a variety of sentences including those listed in Table VIII. The interpreter is driven by a set of 36 S-rules, *The grammar conforms sufficiently to the conditions imposed by Petrick* that it could be recognized by a slight modification of Petrick's transformational recognition procedure.

---

1. a. AA–57 flies from Boston to Chicago.  
   b. CONNECT (AA–57, BOSTON, CHICAGO)
2. a. Doesn't American operate flight AA–57?  
   b. TEST (EQUAL (OWNER (AA–57), AMERICAN))
3. a. Isn't AA–57 an American Airlines flight?  
   b. TEST (FOR SOME X1/FLIGHT:  
      EQUAL (OWNER (X1), AMERICAN AIRLINES);
   
   EQUAL (AA–57, X1)))
4. a. Does American have a flight which goes from Boston to Chicago?  
   b. TEST (FOR SOME X1/FLIGHT:  
      CONNECT (X1, BOSTON, CHICAGO);  
      EQUAL (OWNER (X1), AMERICAN))
5. a. Does American have a flight which doesn't go from Boston to Chicago?  
   b. TEST (FOR SOME X1/FLIGHT:  
      NOT (CONNECT (X1, BOSTON, CHICAGO));  
      EQUAL (OWNER (X1), AMERICAN))
6. a. What American Airlines flight goes from Boston to Chicago?  
   b. (FOR THE X1/FLIGHT:  
      EQUAL (OWNER (X1), AMERICAN AIRLINES)  
      AND CONNECT (X1, BOSTON, CHICAGO);  
      LIST (X1))
7. a. What American Airlines flights go from Boston to Chicago?  
   b. (FOR EVERY X1/FLIGHT:  
      EQUAL (OWNER (X1), AMERICAN AIRLINES)  
      AND CONNECT (X1, BOSTON, CHICAGO);  
      LIST (X1))
8. a. What is the departure time of AA–57 from Boston?  
   b. (FOR THE X1/DTIME (AA–57, BOSTON):  
      LIST (X1))
9. a. What is the departure time from Boston of every American Airlines flight that goes from Boston to Chicago?  
   b. (FOR EVERY X2/FLIGHT:  
      EQUAL (OWNER (X2), AMERICAN AIRLINES)  
      AND CONNECT (X2, BOSTON, CHICAGO);  
      (FOR THE X1/DTIME (X2, BOSTON):  
      LIST (X1))
10. a. What American Airlines flights arrive in Chicago from Boston before 1:00 p.m.?  
   b. (FOR EVERY X1/FLIGHT:  
      EQUAL (OWNER (X1), AMERICAN AIRLINES)  
      AND ARRIVE (X1, CHICAGO) AND  
      GREATER (1:00 p.m., ATIME (X1, CHICAGO));  
      LIST (X1))
11. a. Are all flights from Boston to Chicago American Airlines flights?  
   b. TEST (FOR EVERY X1/FLIGHT:  
      CONNECT (X1, BOSTON, CHICAGO);  
      (FOR SOME X2/FLIGHT:  
      EQUAL (OWNER (X2), AMERICAN AIRLINES);  
      EQUAL (X1, X2)))
12. a. How many flights that go from Boston to Chicago does American Airlines operate?  
   b. LIST (NUMBER (X1/FLIGHT:  
      CONNECT (X1, BOSTON, CHICAGO) AND  
      EQUAL (OWNER (X1), AMERICAN AIRLINES)))
13. a. How many airlines have more than 3 flights that go from Boston to Chicago?  
   b. LIST (NUMBER (X1/AIRLINE:  
      (FOR GREATER (N,3) MANYX2/FLIGHT:  
      CONNECT (X2, BOSTON, CHICAGO);  
      EQUAL (OWNER (X2), X1)))))
14. a. What is the number of flights from Boston to Chicago?  
   b. (FOR THE X1/NUMBER (X2/FLIGHT:  
      CONNECT (X2, BOSTON, CHICAGO):  
      LIST (X1))

**TABLE VIII—Sample test sentences and their semantic interpretations.**
16 D-rules, 17 N-rules, and 11 R-rules. Again, since the primary concern was the design of the semantic interpreter, no retrieval component for the flight schedules problem has been implemented. To do so, however, would be a straightforward problem of writing LISP functions for each of the semantic primitives and for the quantifiers. The semantic interpretations produced by the interpreter are already in the form of LISP S-expressions** and would simply be evaluated in the retrieval component by the LISP function evaluator.

**Extendability**

The query language in the proposed scheme constitutes a standard interface between the semantic interpreter and the retrieval component, thus freeing the semantic interpretation algorithm from dependence on the particular structure of the data base. It also frees the interpreter from a distinction between answers that are actually stored in the data base and answers which must be computed from the data base. The query language is thus easily extended by allowing new predicates, functions, and commands to be added as actual coded subroutines which the retrieval component may call. In addition, the semantic interpreter is a general routine, driven by a set of semantic rules, thus completing the facility for extendability. Any concept that can be made explicit for the machine by means of a programmed subroutine can be added to the retrieval component, and new semantic rules can be added to the semantic interpreter to specify the ways in which the concept is expressed in English. If new kinds of syntactic structures in English are involved, then new rules may also have to be added to the parser in order to recognize the new constructions.

**The LISP S-expression notation is a trivial modification of the notation used in this paper, differing mainly in that the function name is included in the list of arguments rather than outside—e.g. (TEST X1) instead of TEST(X1).**

**BIBLIOGRAPHY**

1. R F SIMMONS
   *Answering English questions by computer: a survey*
   Communications of the ACM Vol 8 No 1 Jan 1965 pages 53–70

2. Deacon Project
   *Phrase-structure oriented targeting query language*
   RM65TMP-64 TEMPO General Electric Co Santa Barbara California Sept 1965

3. W A WOODS
   *Semantics for a question-answering system*
   Also Report NBF-19 Harvard Computation Laboratory

4. D G BOBROW
   *A question-answering system for high school algebra word problems*

5. B RAPHAEL
   *A computer program which understands*

6. R K LINDSAY
   *Inferential memory as a basis of machines which understand natural language*

7. F S BLACK
   *A deductive question-answering system*
   Unpublished PhD thesis Division of Engineering and Applied Physics Harvard University Cambridge Massachusetts June 1964

8. Official Airline Guide

   *The structure of a semantic theory*

10. S R PETRICK
    *A recognition procedure for transformational grammars*
    Unpublished PhD thesis Department of Modern Languages Massachusetts Institute of Technology Cambridge Massachusetts May 1965