DOCUMENT RESUME

ED 053 585 FL 002 343

AUTHOR Alexander, Bill
TITLE A Questioning-Answering Program for Simple Kernel Sentences (QUE2).
INSTITUTION Texas Univ., Austin.
SPONS AGENCY National Science Foundation, Washington, D.C.
REPORT NO NSF-GJ-509-X; TR-NL-5
PUB DATE Mar 71
NOTE 29p.

EDRS PRICE EDRS Price MF-$0.65 HC-$3.29
DESCRIPTORS Applied Linguistics, *Computational Linguistics, *Computer Assisted Instruction, Computer Graphics, Computer Programs, Computer Science, Deep Structure, *Educational Technology, Flow Charts, *Kernel Sentences, *Language Research, Logic, Programing Languages, Semantics, Sentence Diagraming, Structural Analysis, Structural Linguistics, Syntax

ABSTRACT

QUE2 is a recently devised, natural language, questioning-answering program written in LISP1-5. It deals in simple, kernel sentences and employs the theory that the semantic content of a sentence is the set of relationships between conceptual objects (represented by the words in it), which the sentence and its structure imply. The data base of the program is an arbitrary list of simple kernel sentences. The lexicon is a list of pairs; the first element is the word itself, and the second element is its definition--the first element a relation and the second a list of all things which are in the given relation to the word being defined. The structure of the question is also a simple kernel sentence. Through a hierarchical set of functions, the program is capable of taking a kernel-sentence question and, based on its knowledge, providing a one-word answer (true, false, or don't know) accompanied by a copy of the internal semantic structure of the sentence, if any, from which the answer was deduced. The program's capabilities are not infinite, but further details could easily be added. Tables showing the data base, function truth tables, flow charts, and questions and answers are included along with a list of references. (VM)

A QUESTIONING-ANSWERING PROGRAM FOR SIMPLE KERNEL SENTENCES (QUE2)

Technical Report No. NL-5

Bill Alexander

March 1971

NATURAL LANGUAGE RESEARCH FOR COMPUTER-ASSISTED INSTRUCTION

Supported By:

THE NATIONAL SCIENCE FOUNDATION
Grant GJ 509 X

Department of Computer Sciences

and

Computer-Assisted Instruction Laboratory

The University of Texas
Austin, Texas

S E C T I O N 1 :

INTRODUCTION

The program called QUE2 is a natural language questioning-answering program written in LISP1.5. Its data base is an arbitrary list of simple kernel sentences, and it can answer many questions based on these sentences, as long as the structure of the question is also a simple kernel sentence.

The basic method employed is a template- or pattern-matching process with transformations on the questions (described fully in Section III). The purpose of this project was not to present a complete and immediately useful question-answering system, but rather it was to demonstrate the power and potential applicability of the methods used here to the problem of answering questions based on a natural language text when the text has been broken down into kernel sentences. Since kernel sentences tend to fall into a relatively small number of rather simple semantic and structural patterns, these patterns can be manipulated without too much difficulty, and questions can be rather thoroughly compared with sentences having words in common and similar patterns.

It seems natural to view any natural language processor as consisting of three main components:

(1) A component to translate the natural language input into the representation used internally by the system. In the case of question-answering systems, the natural language sentences would have to be translated into a structure representing their semantic content. The structures used by QUE2 are described in Section III.

(2) A component to manipulate, transform, compare, etc., these internal structures. In a question-answering system, this component tries to deduce the answers to the questions from the structures representing the semantic content of the data text.

(3) A component to generate natural language output from the representation used internally.

The program QUE2 is only the middle component of a questioning-answering system. It only accepts questions which are already in the form used internally to represent their semantic content; its output is a one-word answer accompanied by a copy of the internal semantic structure of the sentence, if any, from which the answer was deduced.

The program QUE2 itself consists of three main parts:

(1) *A list (list DATA) of list-structures representing the semantic content of certain kernel English sentences.* These sentences are the data-text from which QUE2 tries to answer questions given it. Sentences, in the proper list form, may be added or deleted from list DATA from run to run without affecting the performance of the program, provided only that pointers to these sentences are added or deleted appropriately to the lexical definitions of the words in the sentences. Thus, QUE2 can answer questions based on any set of text sentences.

(2) *A list (list LEX) of word-definition pairs which serves as the lexicon of words used in the text.* In addition to pointers to all sentences (if any), in which the word is used, the definitions include class, equivalence, and exclusion information.

(3) *A set of functions.* These functions are for (a) manipulating the structure representing the question, (b) comparing it to the text structures in list DATA using lexical information if necessary, and (c) deducing answers.

(Descriptions of the lists DATA and LEX, as well as of the main functions, are included in Section III.)

One unique aspect of QUE2 is that it is based on a three-valued logic rather than on the two traditional logical values TRUE and FALSE. The third value that QUE2 deals with is "don't know," which will subsequently be referred to as DK or Nil. A great deal of care has been taken to insure that QUE2 does not return a definite answer to a question when an answer cannot be rigorously deduced from the data text. For example, if one of the kernel sentences in the data text is:

Mary gave Tom John's letter. . . . *(s1)*

then QUE2 will answer TRUE to:

A woman gave Tom John's letter?

and FALSE to:

A man gave Tom John's letter?

but DK to:

Mary gave Tom John's letter Tuesday?

because the data text sentence does not strictly imply an answer to this last question. Another unique feature of QUE2 is that it can return answers to three distinct types of questions, with no special signal as to which type of question is being asked. The first type of question is the TRUE-FALSE-DK type illustrated in the examples above. The second type of question is Who-What-When.

Mary gave John's letter to whom?

can be entered as

Mary gave Q John's letter?

with the atom Q as the indirect object. QUE2 will return the indirect object of a sentence which would imply that the question was TRUE if

the indirect objects matched. In this example, QUE2 would answer "TOM" if sentence (s1) were in its data text. The third type of question which QUE2 can answer is a question based solely on its lexicon. If, for example, the definition of "dog" in the lexicon included the information that "dog" is a member of the class "animal," then the question

Is a dog an animal?

would be answered "TRUE BY DEFINITION." This answer will be returned by QUE2, whether or not there are any sentences in its data text containing the words "dog" or "animal."

S E C T I O N I I :

BACKGROUND

Natural language question-answering systems have been worked on and discussed by many people since about 1964. For a recent survey of the field and an extensive bibliography, see Simmons (5).

Some researchers are developing special-purpose systems which operate on a small subset of English and answer questions on only one or a few topics. An example of this approach is Charniak's (1) calculus problem solver.

Researchers attempting to solve the more general problem of answering questions based on general text are faced with two enormous problems. The first is to develop a formal data structure general enough to represent the wide range and complex structure of meanings typically found in general natural language text. The second major problem is to develop a sufficiently powerful logic upon which to base a general question-answering algorithm.

Semantic Structure. Several recent question-answering systems have represented their data in some form of directed graph in which the nodes are words, concepts, or phrases and the edges are relationships between the nodes. This type of structure is motivated by psychological considerations as well as by Chomsky's notion of deep structure. Possibly the first formal presentation of this kind of structure in relation to question-answering systems was in Quillian's early work, and he incorporates it in his recent Teachable Language Comprehender (3). A similar structure is used by Schwartz et al. in their Protosynthex III (4). Another isomorphic variant of this structure, developed by Simmons (6 and 7), is used in QUE2 (described in detail in Section III).

The other major approach to semantic structures is to translate the text into first order predicate calculus (*fopc*) statements. This approach is used by Green and Raphael in the QA2 (2). It has the tremendous advantage of greatly simplifying the question-answering algorithm. Unfortunately, not all English can be represented in the *fopc*. Either a higher order calculus will have to be mechanized or new operators will have to be added to the *fopc* to enable it to express a broader set of natural language.

In an attempt to make the semantic structures simpler and more manageable, many researchers have taken the approach of breaking the text down into simple kernel sentences, and algorithms have been developed for generating a meaning-preserving set of kernel sentences from text (8). This approach is also taken in QUE2. It is assumed by QUE2 that both the text and the questions are in relatively simple form.

Algorithms. The question-answering algorithms used by most of the recent question-answering systems fall into two major classes. The first type are *formal theorem-proving techniques*, typified by the Robinson resolution principle for proving theorems in the *fopc*. This algorithm is used by Green and Raphael in QA2 (2). It has several advantages:

(1) It is a relatively short and very elegant algorithm.

(2) It is general and logically complete: If an answer to a question can be deduced from the text, this algorithm, without fail, will eventually produce the answer.

(3) There is a known algorithm mentioned in (2) for translating a subset of English into *fopc* statements.

One disadvantage of this algorithm is that the *fopc* is not capable of expressing the semantic content of all English sentences, particularly the semantic distinctions between various determiners and quantifiers. In some cases, it can also be very slow. This is partly because it can look for only one answer at a time; that it, it can either try to prove that the question, expressed as a theorem is true or that it is false, but not both simultaneously. By contrast, QUE2 searches for both answers at once.

Most of the other algorithms used in question-answering, although varying in many respects, all use some form of structure- and word-matching technique. They usually involve some logical inference, and many perform transformations on either the question or the text sentences in their attempt to find matches. One of the earliest of these systems was Raphael's SIR. More representative of recent research along this line is Protosynthex III (4). The algorithm employed by QUE2 also falls in this class. The author knows of no proof concerning the logical completeness of any of these algorithms, but it is unlikely that any of the present ones are complete. Their rather *ad hoc* character discourages formal mathematical treatment, and they lack the elegance of the resolution principle. However, they do seem better adapted to the problem at hand than do techniques borrowed from other fields, such as formal logic, and they at least show promise of being expanded or refined to handle the problems of equivalence and quantification peculiar to natural languages. Furthermore, they are fast enough and, for simple kernel sentences, short enough for practical implementation on current computer systems, at least for small data bases.

S E C T I O N I I I :

DESCRIPTION

Semantic Structures

As mentioned in Section II, the philosophy behind the representation of semantic content used in QUE2 is that the semantic content of a sentence is the set of *relationships* between conceptual objects (represented by the words in it), which the sentence and its structure imply. Sentences or phrases can be thought of as directed graphs. The nodes of the graph are conceptual objects denoted by words, and the edges of the graph are the relationships between the words or conceptual objects. Thus, the phrase "a brown puppy" can be represented by:

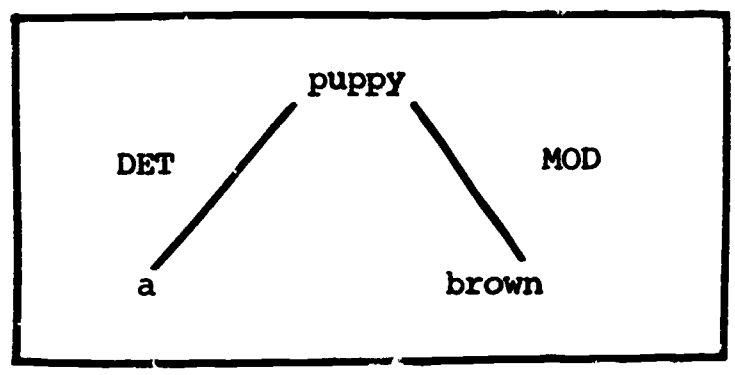

FIGURE 1

where the edge labels, DET and MOD, signify that *a* and *brown* are in the relationships *determiner* and *modifier*, respectively, to *puppy*. It is usually more convenient to represent such graphs in tabular form.

relation	node
TOK	puppy
MOD	brown
DET	a

FIGURE 2

In Figure 2, TOK indicates that *puppy* is the main word, or head, of the graph. The structures represented in Figures 1 and 2 are entirely equivalent. Whole sentences can be represented by such graphs. The sentence:

John gave Mary a brown puppy Tuesday.

is represented by:

G7	TOK	give
	TENSE	past
	AGT	John
	IO	Mary
	OBJ	(G8)
	TIME	Tuesday
G8	TOK	puppy
	MOD	brown
	DET	a
	OBJ-1	(G7)

FIGURE 3

The parentheses around G8 indicate that what stands in the direct object relationship to *give* is another whole structure rather than just one word. The verb is always the head of a sentence in this representation.

Since QUE2 is written in LISP1.5, which works only with lists, the graph of Figure 3 must be transformed into list notation. The actual representation used in QUE2 is given in Figure 4. Tenses are not distinguished by

QUE2. In addition, there is no distinction between the *agent* and *subject* relations, although one could easily be implemented.

```
(G7((TOK GIVE)(SUBJ JOHN)(OBJ (G8))(IO MARY)(TIME TUESDAY)))
(G8((TOK PUPPY)(MOD BROWN)(DET A)(OBJ-1(G7))))
```

FIGURE 4

In Figure 4, the order of the lists or of the elements in each list is arbitrary.

The functions in QUE2 are, at present, set up to handle kernel questions and kernel sentences which conform to any one of three paradigms:

1. (subject) "is" (predicate adjective).
2. (subject) "is" (predicate nominative).
3. (subject) verb (direct object)(indirect object)(time).

Elements enclosed in parentheses may be either single words or noun phrases. The last three elements in Paradigm 3 are all optional, so that, for example, *John hit Tom* and *John sing* are legal kernel sentences.* The ten sentences which currently make up list DATA are given in Table 1.

TABLE 1

Knowledge (list DATA)

John's house is green.
John's house is-south-of Mary's house.
Tom is a student.
John give Mary a brown puppy.*
Harry hit Tom Tuesday.
John's mother give a student John's new car.
A man give John his promotion Tuesday.
John's boss give John a letter.
John's girlfriend is Mary.
Susan is Mary's girlfriend.

*No conjugation of verbs is done, and no attempt is made to distinguish tenses.

The lexicon (list LEX) is a list of pairs; the first element of each pair is a word, and the second element is its definition. The definition is itself a list of pairs; the first element of each pair is a relation, and the second is a list of all things which are in the given relation to the word being defined. The relations currently implemented in LEX include TOK, SUP, SUP-1, EQV, EXCL, and CONV. The lists that follow each of these relations in the definition of a word are, respectively, pointers to sentences in DATA in which the word appears, words which are superclasses of the word, words it is a superclass of, words to which it is equivalent, words which are in sets disjoint from it, and, for verbs, verbs which are converses. For example, the lexical entry for *woman* is:

(WOMAN((TOK(G19))(SUP(PERSON))(SUP-1(GIRL MOTHER))(EXCL(MAN))))

which indicates:

--that *woman* is used in one sentence, G19,

--that it is a member of the class *person*,

--that *girl* and *mother* are subclasses of it, and

--that nothing which is a man or a subclass of *man* is a woman.

In logical notation:

woman → *person*, *girl* → *woman*, *woman* → ~ *man*, *woman* → ~ *boy*, *etc.*

These logical relations are used by function WORDMATCH, which is discussed in the following subsection.

Question-Answering Algorithm

A hierarchical set of functions is employed by QUE2. The top function merely reads the next question, passes it to function QUERY1, and prints the answer returned by QUERY1. The question is passed by QUERY1 to function RELEVANT, which returns an ordered list of pointers to some of the sentences in the text that are ordered according to their probable relevancy to the question. QUERY1 then passes the elements of this list (one by one in order) along with the question to function MATCH. MATCH divides the question and the sentence into clauses and passes these clauses to function CLAUSEMATCH, which further divides the clauses into words and passes pairs of words to function WORDMATCH. Each of these last three functions returns to the function preceding it, one of the three logical values: TRUE, FALSE, or DK. CLAUSEMATCH and MATCH place entries in a truth-table with the answers returned by lower functions until they are themselves able to return a value to the next higher function.

If MATCH discovers that the verb of the question is *is*, it transfers the complete answering job to function ISMATCH, which has a different truth-table. ISMATCH also uses CLAUSEMATCH and WORDMATCH to fill-in its table. First, however, ISMATCH attempts to answer the question from lexical information alone. DETMATCH is a specialized version of WORDMATCH which compares determiners. The truth-tables for these functions are given in Tables 2 through 6. In each of these tables, if the item for a column is missing in the question, that item in the table is set to TRUE. If, for example, a question has no indirect object, TRUE is entered in that column in function MATCH. The truth-tables, as they are shown here, do not actually appear in the functions; instead, they are incorporated in the nested conditionals to be found at the end of each function. Flowcharts for functions QUERY1 and MATCH are given in Figures 5 and 6.

TABLE 2

Truth-Table for Function WORDMATCH

Condition	*Value*
D missing, or Q = D, or Q EQV D, or Q → D	TRUE
Q → ~ D	FALSE
otherwise	Nil

TABLE 3

Truth-Table for Function DETMATCH

Condition	*Value*
D missing, or Q = *a*, or Q = D, or D → Q	TRUE
D = *a*, or Q → D	Nil
otherwise	FALSE

In Tables 2 and 3, Q is the word in the question, and D is the corresponding word in the text sentence.

TABLE 4

Truth-Table for Function CLAUSEMATCH

DETMATCH	*MOD WORDMATCH*	*TOK WORDMATCH*	*Value**
TRUE	TRUE	TRUE	TRUE
--	--	FALSE	FALSE
--	FALSE	--	FALSE
FALSE	--	--	FALSE
all other combinations			Nil

*In addition to a value of TRUE, FALSE, or Nil (DK), CLAUSEMATCH also returns a second value: *definite (d)*; *or indefinite (i)*. It returns *i* if either determiner is *a*, and *d* otherwise.

TABLE 5

Truth-Table for Function ISMATCH

Subj CLAUSEMATCH	*Pred Adj WORDMATCH*	*Value*
TRUE	TRUE	TRUE*
TRUE, d	FALSE	FALSE
all other combinations		Nil

Subj CLAUSEMATCH	*Pred Nom CLAUSEMATCH*	*Value*
TRUE	TRUE	TRUE*
TRUE, d	FALSE	FALSE
FALSE	TRUE, d	FALSE
all other combinations		Nil

*Unless the atom Q appeared in the question, in which case the corresponding element in the text sentence is returned.

TABLE 6

Truth-Table for Function MATCH

Subj CLAUSEMATCH	*Obj CLAUSEMATCH*	*IO CLAUSEMATCH*	*Time WORDMATCH*	*Value*
TRUE	TRUE	TRUE	TRUE	TRUE*
TRUE, d	TRUE, d	TRUE	FALSE	FALSE
TRUE	TRUE, d	TRUE, d	FALSE	FALSE
--	FALSE	--	--	Nil
TRUE	TRUE, d	FALSE	TRUE	FALSE
FALSE	TRUE, d	TRUE	TRUE	FALSE
all other combinations				Nil

*Unless the atom Q appeared in the question, in which case the corresponding element in the text sentence is returned as the answer.

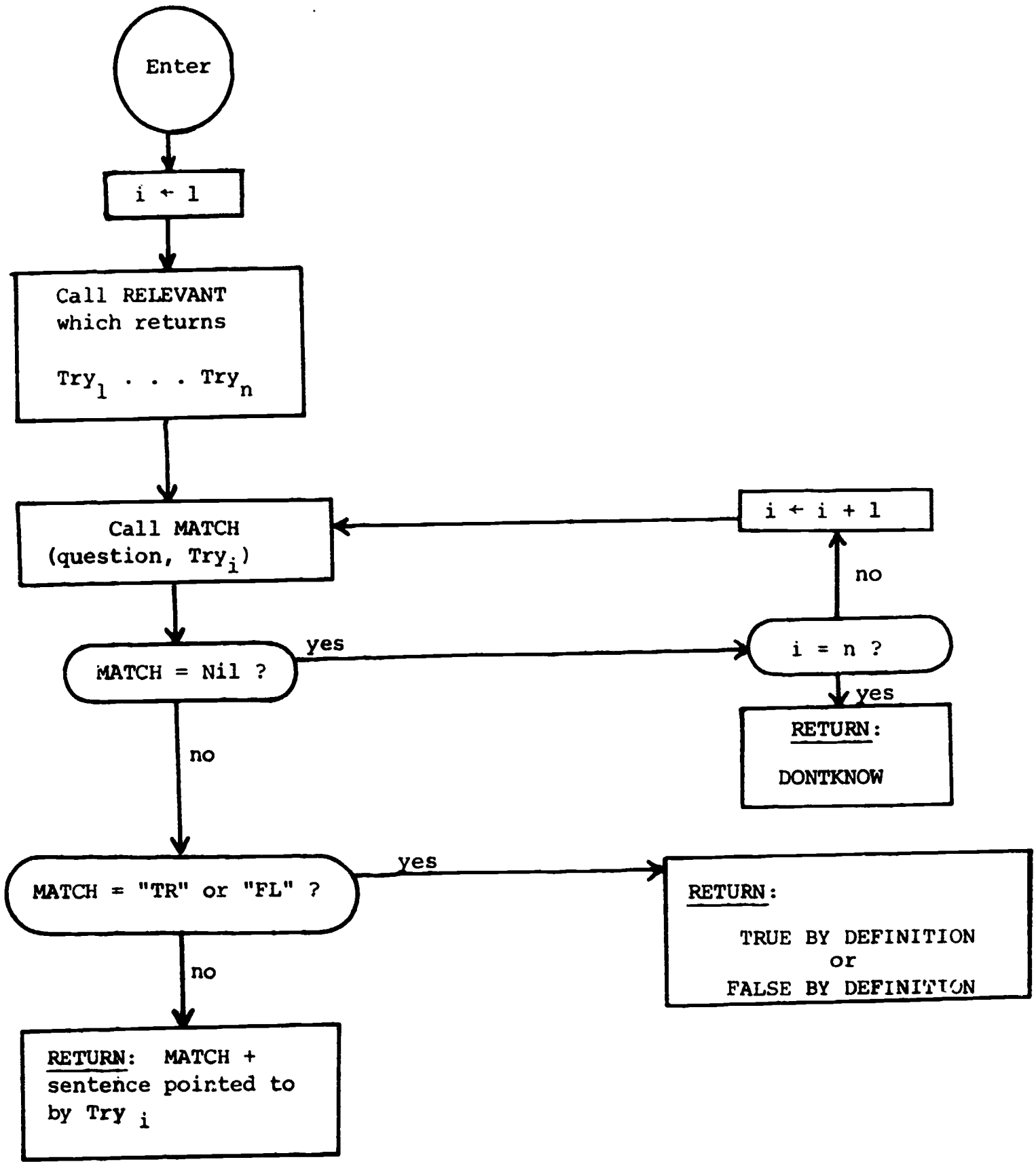

FIGURE 5.--Flowchart for Function QUERY1.

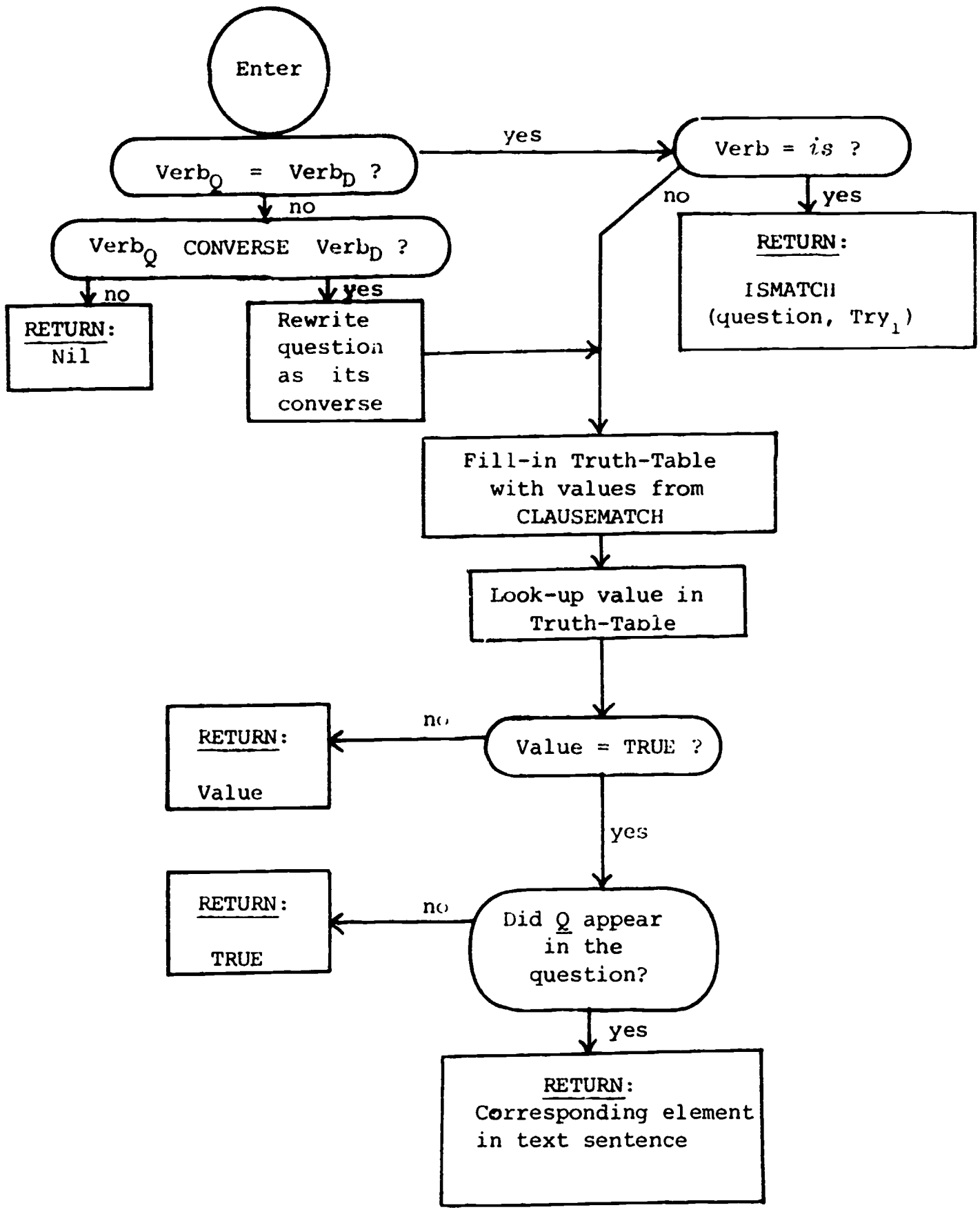

FIGURE 6.--Flowchart for Function MATCH

18

If functions MATCH and ISMATCH cannot arrive at an answer of TRUE or FALSE, they try transformations on the questions before giving up and returning a value of Nil (DK). ISMATCH rewrites the question, reversing the subject and predicate nominative, and then tries to answer this new question as before. If MATCH discovers that the verbs in the question and text sentence currently under consideration are converses, it switches the subject with the indirect object, or with the object if there is no indirect object in the question. It can also rewrite the question leaving the verbs as converses, in which case it returns TRUE if the truth-table indicates FALSE, and vice-versa. Having the question rewritten in any of these ways does not impair QUE2's ability to answer Who-What-When questions correctly.

Function RELEVANT, which produces an ordered list of probably-relevant text sentences for a given question, assigns points to text sentences by locating in the lexicon each word in the question. Taking the words in the question (one by one), one point is assigned to a sentence if there is a pointer to that sentence in the lexical entry for the word. A sentence also gets one point for every word EQV to, EXCL to, SUP to, or SUP-1 to the question word which has pointers to that sentence. The sentences are then ordered according to their point total. If there are sentences with three or more points, sentences with two or less points are deleted from the list of sentences to be considered. This heuristic greatly reduces running time for questions to which the correct answer is DONTKNOW, and it has never caused QUE2 to overlook a definite answer. A better heuristic would probably improve running time even more. A complete listing of the questions answered by QUE2 is given in Table 7.

TABLE 7

Questions (read in)	*QUE2's Answers*
Based on: John's house is green.	
1 John's house is green?	TRUE*
2 John's home is brown?	FALSE
3 John's home is what?	GREEN
4 Whose home is green?	JOHN
5 John's car is green?	DONTKNOW
6 Tom's home is green?	DONTKNOW
7 A house is a home?	TRUE BY DEFINITION
Based on: John's house is-south-of Mary's house.	
8 John's house is-south-of Mary's house?	TRUE
9 Mary's house is-south-of John's house?	FALSE
10 Whose house is-south-of Mary's house?	JOHN
11 John's house is-north-of Mary's house?	FALSE
12 Mary's house is-north-of John's house?	TRUE
13 Whose house is-north-of Mary's house?	DONTKNOW
14 Whose house is-north-of John's house?	MARY
Based on: Tom is a student.	
15 Tom is a student?	TRUE
16 Tom is the student?	DONTKNOW
17 Is Tom a old student?	DONTKNOW
18 Is Susan a student?	DONTKNOW
19 Is Harry a student?	DONTKNOW
20 Is Tom a man?	TRUE BY DEFINITION
21 Is Tom the man?	DONTKNOW
22 Is Tom a girl?	FALSE BY DEFINITION
Based on: John give Mary a brown puppy.	
23 John give Mary a dog?	TRUE
24 Mary give John a puppy?	FALSE
25 John give Mary a what puppy?	BROWN
26 John give Mary what?	A BROWN PUPPY
27 John give who a dog?	MARY
28 John give Susan a dog?	DONTKNOW
29 Tom give Mary a dog?	DONTKNOW
30 John give Mary the dog?	DONTKNOW

*For all answers except DONTKNOW and TRUE- and FALSE BY DEFINITION, QUE2 also returns a copy of the text sentence from which it deduced the answer.

TABLE 7 (continued)

	Questions (read in)	*QUE2's Answers*
31	John give Mary a red dog?	DONTKNOW
32	John give Mary a car?	DONTKNOW
33	Mary get a puppy from John?	TRUE
34	Mary get a red dog from John?	DONTKNOW
35	John get a puppy from Mary?	FALSE
36	Mary get what from John?	A BROWN PUPPY
37	Mary get a puppy from whom?	JOHN
38	Mary get a dog?	TRUE
39	Boy give girl dog?	TRUE
40	Mary is John's puppy?	FALSE BY DEFINITION
	Based on: Harry hit Tom Tuesday.	
41	Harry hit Tom Tuesday?	TRUE
42	Harry hit Tom?	TRUE
43	Harry hit Tom Wednesday?	FALSE
44	Harry hit John?	DONTKNOW
45	Harry hit John Tuesday?	DONTKNOW
46	John hit Tom Wednesday?	DONTKNOW
47	A man hit Tom Tuesday?	TRUE
48	The man hit Tom Tuesday?	DONTKNOW
49	A man hit Tom Wednesday?	DONTKNOW
	Based on: John's mother give a student John's new car.	
50	John's mother give a student John's new car?	TRUE
51	A mother give a student John's new car?	TRUE
52	A man give a student John's new car?	FALSE
53	John's mother give Tom John's new car?	DONTKNOW
54	John's mother give the student John's new car?	DONTKNOW
55	John's mother give a animal John's new car?	FALSE
56	John's mother give a student Tom's new car?	DONTKNOW
57	John's mother give a student John's old car?	DONTKNOW
58	John's mother give John's new car to whom?	A STUDENT
59	Susan's mother give a student John's new car?	FALSE
60	A woman give a person whose car?	JOHN
	Based on: A man give John John's promotion Tuesday.	
61	A man give John John's promotion Tuesday?	TRUE
62	John's boss give John John's promotion Tuesday?	DONTKNOW
63	A woman give John John's promotion Tuesday?	FALSE
64	A man give John John's promotion?	TRUE
65	A woman give John John's promotion?	FALSE

TABLE 7 (continued)

Questions (read in)	*QUE2's Answers*
66 A man give John John's promotion Wednesday? ----------	FALSE
67 A man give Tom John s promotion Tuesday? ----------	FALSE
68 A man give Tom Tom's promotion Tuesday? ----------	DONTKNOW
69 A man give John John's promotion when? ----------	TUESDAY
70 John get John's promotion from a man? ----------	TRUE
71 John get a promotion from a woman? ----------	DONTKNOW
72 John get John's promotion from a woman? ----------	FALSE
73 John get John's promotion Wednesday? ----------	FALSE
74 John get a promotion? ----------	TRUE
Based on: John's boss give John a letter.	
75 John's boss give John a letter? ----------	TRUE
76 Tom's boss give John a letter? ----------	DONTKNOW
77 Susan's boss give John a letter? ----------	DONTKNOW
78 John's boss give John the letter? ----------	DONTKNOW
79 Susan's boss give Tom a letter? ----------	DONTKNOW
80 John's boss give Tom a letter? ----------	DONTKNOW
Based on: John's girlfriend is Mary.	
81 John's girlfriend is Mary? ----------	TRUE
82 John's girlfriend is a girl? ----------	TRUE BY DEFINITION
83 Mary is a girlfriend? ----------	TRUE
84 Mary is John's girlfriend? ----------	TRUE
85 Mary is Tom's girlfriend? ----------	FALSE
86 Susan is John's girlfriend? ----------	FALSE
Based on: Susan is Mary's girlfriend.	
87 Susan is Mary's girlfriend? ----------	TRUE
88 Susan is a girlfriend? ----------	TRUE
89 Susan is a girl? ----------	TRUE BY DEFINITION
90 Sally is Mary's girlfriend? ----------	FALSE
91 Mary's girlfriend is Susan? ----------	TRUE
92 Mary's girlfriend is Sally? ----------	FALSE

S E C T I O N I V :

EXPERIMENTS

QUE2 is currently provided with a data text of the 10 sentences listed in Table 1. Its lexicon contains 42 words; it was given the set of 92 questions listed in Table 7; and it returned the answers also listed in that table. The program was run under the LISP1.5 interpreter on a CDC6600 computer. It required about 3 seconds to set up and about 50 seconds of central processing time to answer all of the 92 questions. This represents an average of about 0.54 seconds per question. This figure can be reduced slightly by either providing more memory or by reducing the number of questions per run so that no "garbage collections" are necessary. It was provided with 55000_8 words of memory, and there were three "garbage collections" during the run. A more significant reduction in the average time per question could be achieved by a more sophisticated cut-off heuristic to limit the number of text sentences QUE2 looks at before giving up and returning DONTKNOW, since QUE2 generally spends less than 0.5 seconds for answering questions to which it can find a definite answer and, in some cases, a full second or more for arriving at DONTKNOW. In a few places, more efficient code could probably be written. Some of the questions in Table 7, as well as QUE2's answers, are in the following discussion.

Questions 3 and 4 illustrate QUE2's ability to answer "what" and "who" questions. It is not limited to returning one-word answers, as is illustrated by Questions 26, 36, and 58. Question 36 also demonstrates that QUE2's ability to answer such questions is not impaired by having first to transform the question.

Questions 7, 20, 22, 40, 82, and 89 were all answered on the basis of lexical information alone. Notice that the form of such questions is identical to the form of all other questions. Also, compare Question 20 with 21: The distinction resulting in the two different answers is based solely on the determiner preceding "man." While the determiner of the predicate nominative must be "a" for QUE2 to answer TRUE BY DEFINITION, this is not necessary for deriving FALSE BY DEFINITION, as is shown in Question 40. The answer to 40 is based on a rather long chain of lexical information: Mary is SUPed to girl-person, girl-person to girl-class, girl-class to woman, and woman to person; person excludes animal, animal is SUP-led to dog, and dog to puppy.

Questions 11-14, 33-38, 70-74, 83-86, and 91-92 all required QUE2 to make some transformation on the question before an answer could be given. As demonstrated by Questions 34 and 71, QUE2's ability to avoid unwarranted answers is not impaired by having to make such transformations.

Text Sentences 4, 6, 7, and 8 all have the same basic structure, and all involve the verb "give." In 6 and 7, a definite object is given, while the direct objects in 4 and 8 are indefinite. Questions on these sentences demonstrate the use of the definite-indefinite value returned by CLAUSEMATCH which appears in lines 2, 3, 5, and 6 of the truth-table of function MATCH (Table 6). The direct objects of both the text sentence and of the question must be definite before QUE2 will return an answer of FALSE. This provision is based on observation of common usage in English: *John gives Mary* a *puppy* does not preclude John

giving Mary anything else, John giving anyone else anything, or anyone else giving Mary anything. But *A man gives John John's promotion Tuesday* precludes practically any other event described by a change in only one element of the text sentence. For example, compare Question 28 with 67, or 29 with 65. In these examples, the intent was to reflect ordinary usage of language. In some cases, decisions were made rather arbitrarily. For example, Questions 24 and 35 illustrate the fact that QUE2 assumes that X giving Y to Z precludes Z giving Y to X. This is, of course, not logically the case and is not always the case in ordinary usage, as is obvious if Y is a "kiss." To distinguish the very different logical implications of *John give a kiss to Mary* and *John give a puppy to Mary*, QUE2 would need a great deal of additional lexical information.

Another rather arbitrary decision was made in the second and third lines of Table 6 concerning the logical implications of time. Notice that QUE2 answered FALSE to both Questions 43 and 66. Most people would agree that the answer to 66 is reasonable, but there might be some question about 43. If Harry is a bully, he might well hit Tom every day. The problem is even more obvious for verbs like "sleep" and "eat," which are generally done by people every day. Obviously, the verbs in the lexicon should be classified according to repeatability, as well as by many other classifications.

The preceding discussion does not attempt to defend any of these choices. The important point is that if QUE2 does not perform according to a particular user's notion of usage, the truth-tables can be changed, and/or additional information can be included in the lexicon.

S E C T I O N V :

CONCLUSIONS

From the results presented in Section IV, it can be seen that QUE2 returns reasonable answers to a wide variety of simple questions based on the given text. In fact, the author has been unable to construct a question to which QUE2 does not give a reasonable answer. As stated in the introduction, the purpose of this project is to demonstrate that the matching and truth-table methods employed in QUE2 are sufficiently powerful to make this a reasonable approach to answering questions based on kernel sentences. It is not claimed that the present QUE2 can answer all questions based on arbitrary kernel sentences. For instance, there is the problem discussed in Section IV in connection with Question 43. The lexicon of QUE2 obviously needs to include more sophisticated definitions, at least for verbs. Although the term "kernel sentence" is not rigidly defined, it would surely include simple sentence patterns or paradigms other than the three that QUE2 is presently able to handle. But different additional patterns could be recognized either by the presence or absence of elements or by the class of the verb, and then it is simply a matter of including additional truth-tables. Since a given algorithm for generating a set of meaning-preserving kernel sentences from natural language text can generate only a finite, presumably small, number of such patterns, the problem of expanding QUE2 to handle them should not be overwhelming. Adding certain relations would merely be a matter of adding columns to existing truth-tables. For example, the relation LOC, for "location," could be added to

Paradigm 3 simply by adding a column to Table 6; the entries in the column would correspond exactly to those in the "Time" column.

One obviously desirable addition to QUE2 would be the ability to answer a question based on two or more sentences. For example, in the present implementation, Sentences 3 and 5 are

Tom is a student.

Harry hit Tom Tuesday.

A good question-answerer should be able to answer

Harry hit a student Tuesday?

Actually, it would be easy to enable QUE2 to handle this particular question. Upon discovering that the question matches Sentence 5 perfectly, except for a DK value in the object position, QUE2 could generate a secondary question:

Tom is a student?

using the question-building techniques of function REWRITE. It would then pass this new question to itself recursively as a subgoal. The strategy would be excactly the subgoal strategy of Newell, Shaw, and Simon's GPS. The problem, as in GPS, would be to develop heuristics for determining when to generate new subgoals and when to give up. This is a difficult problem, but essentially the same problem arises in any question-answering system.

Due to the number of sentence patterns occuring in natural languages, it is doubtful that the methods employed in QUE2 represent the best basis for a completely general question-answerer based on arbitrary text. Furthermore, it is not at all clear that the semantic content of any arbitrary text can adequately be represented by a set of simple kernel sentences. For example, it is difficult to produce a set of simple kernel sentences for:

One of the first plants to appear on newly-formed tropical islands is the stately and graceful coacoa-nut palm.

The author does believe, however, that for text which can be represented as a set of kernel sentences, QUE2 illustrates a promising approach to question-answering.

R E F E R E N C E S

(1) E Charniak. Computer Solution of Calculus Word Problems. Proc. Int. Jt. Conf. Art. Int., Washington D. C, 1969, 303-316.

(2) C C Green & B Raphael. The Use of Theorem Proving Techniques in Question-Answering Systems. Proc. 1968 ACM Nat. Conf., 169-181.

(3) M R Quillian. The Teachable Language Comprehender. Comm ACM, Aug. 1969, 459-476.

(4) R M Schwartz, J F Burger, and R F Simmons. A Deductive Question-Answerer for Natural Language Inference. Comm. ACM, March 1970, 167-183.

(5) R F Simmons. Natural Language Question-Answering Systems: 1969. Comm. ACM, January 1970, 15-30.

(6) _____. Class notes, CS387K. Spring 1970.

(7) _____. Notes on Semantic and Discourse Structures. Design Note, Dec. 20, 1969.

(8) _____, J F Burger, and R E Long. An Approach Toward Answering English Questions from Text. Proc. FJCC, 1966, 357-363.