AD698185

Best Available Copy

LOGICAL ASPECTS OF QUESTION-ANSWERING BY COMPUTER

J. L. Kuhns*

The RAND Corporation, Santa Monica, California

1. Introductory Remarks

In this paper we will consider the problem of computerized question-answering from the point of view of certain technical, although elementary, notions of logic. ~~While~~ the work reported herein has general application to the design of information systems, it is specifically motivated by the RAND Relational Data File, a data-retrieval system whose design features were proposed by R. E. Levien and M. E. Maron (1965, 1966; see also Levien, 1966, and Maron, 1966). This system, for which a prototype has been implemented (Levien, 1969), deals with the retrieval and processing of information from a large file of *relational* sentences. (In the present version there are about 70,000 sentences.) These file items concern factual information on research in the field of cybernetics--what Levien and Maron call *context* data. By that is meant such information as who wrote what paper, with what organization someone is affiliated, what

---

*Any views expressed in this paper are those of the author. They should not be interpreted as reflecting the views of The RAND Corporation or the official opinion or policy of any of its governmental or private research sponsors. Papers are reproduced by The RAND Corporation as a courtesy to members of its staff.

This paper was prepared for presentation at the Third International Symposium on Computer and Information Sciences, Miami Beach, Florida, 18-20 December 1969.

conferences were held--and where--and who attended them, etc.

We will not discuss specific details of the system but rather an abstract version of it. Our purpose is to demonstrate how notions in logic can be used to influence the design of components of an information system. A secondary purpose is to show how logic can be applied to the analysis of natural-language. We also hope the audience will benefit from an elementary exposition of some notions in the *theory of relations*.

2. Overview of an Information System

In Fig. 1 we show a simplified diagram of an information system. There are two inputs: data of some sort and queries--information requirements of users. The problem, in its broadest outline, is to match an input query to the batched input data. But this match must be done on a common ground; that is, both data and query must be cast into comparable strings of symbols. Thus the two inputs are accompanied by two representation processes which result in what we shall call *file items* and *symbolic questions*. At this stage we leave open the question of the mode of these auxiliary processes--they may be either manual or automatic. Neither will we specify the exact form of the input data. Let us however consider the given expression of the user's information problem to be a natural-language query. Now the design of the system should be guided by the *logical*

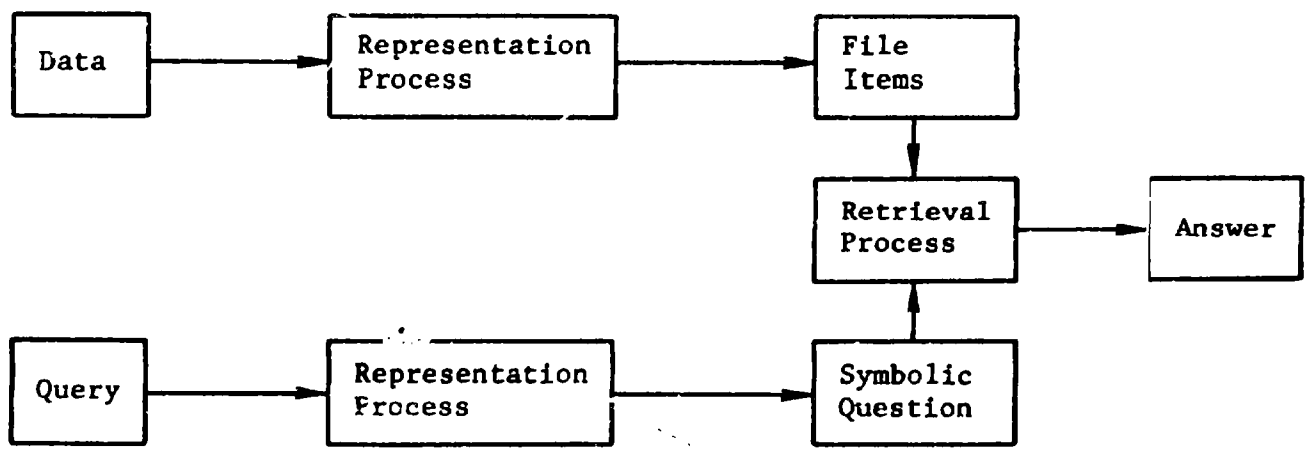

Fig. 1--Schematization of an Information System

analysis of such queries rather than an analysis restricted to traditional grammar. A specific example will illustrate this idea.

3. Analysis of a Natural-Language Question

Suppose our system is to store bibliographic and related data on logic, and suppose that an inquirer, wishing to interrogate the system, poses the question

What books has Reichenbach written? (1)

How can a computerized system answer this?

First let us examine the relationship between the question and its answer; i.e., as a relationship between expressions. A member of the set of answering expressions would be the title '*Elements of Symbolic Logic*' because

Reichenbach wrote *Elements of Symbolic Logic* (2)

and

Elements of Symbolic Logic is a book (3)

Let us symbolize these two elementary sentences. The first is analyzed as expressing a *relation* holding between two individuals--a person and a written work. In logical notation this relation can be designated by 'W' associated with two argument positions. Introducing 'a' and 'b' as symbolic translations of 'Reichenbach' and '*Elements of Symbolic Logic*', respectively, then (2) receives the symbolic translation

aWb (4)

'W' (together with its associated argument positions) is called a *two-place predicate* and designates a relation; 'a' and 'b' are *argument expressions*, in this case individual constants, designating individuals. Similarly the sentence (3) can be symbolized

Bb (5)

where 'B' (with its associated argument position) is a translation of 'book' and as a *one-place predicate* designates a property, namely, the property of being a book. [The concept of one-place predicate is closely related to the concept of *class*; thus (5) can also be read

b belongs to the class B (5')

This second, *extensional*, interpretation is especially appropriate for noun phrases such as 'a book' occurring in (3).]

The method of *propositional data storage* as used in the Relational Data File (Levien and Maron, 1965, 1966) provides a method for storing sentences which is analogous to the symbolic translations given above. Thus to store the sentence (2) the English words are first translated into computer words (analogous to 'W', 'a', 'b'). Next, the three computer words are stored in the file (in our case, a disk file). Finally, a fourth computer word is

used to identify the sentence so stored. Thus, we have a data line of the design

1	a	W	b

Similarly the sentence (3) [or (5)] leads to a data line having the design

2	-	B	b

with no entry in the second place.[1]

A *data base* then consists of two parts: (1) a dictionary of individual constants and predicates (together called *descriptive names*); (2) a collection of elementary, i.e., atomic sentences, which are certain strings of descriptive names.

An example dictionary is shown in Fig. 2. We can regard it as an inventory of our universe of discourse.

An example file is shown in Fig. 3. This describes a possible state of the universe of discourse.

Returning to the input question (1), let us give it a similar symbolic translation. We have three major classes of logical "parts of speech" to work with: argument expressions, predicates, and logical signs (to be explained). The result of the analysis of the question will be to identify the lexical units that correspond to these classes.

a	Reichenbach
b	*Elements of Symbolic Logic*
c	*On Meaning*
d	Russell
e	Whitehead
f	*Principia Mathematica*
B	(the property of being a) book
P	(the property of being a) paper
W	wrote (the relation of author-ship in the broad sense)

Fig. 2--An Example Dictionary

1	a	W	b
2		B	b
3	a	W	c
4		P	c
5	d	W	f
6	e	W	f
7		B	f

Fig. 3--An Example File

The argument expressions are 'What' and 'Reichenbach'. 'What' has the status of a variable; we call it an *interrogative descriptive variable.*[2] Everything else in (1) is a complicated relation-expression; we will refer to it as 'R_3' because it turns out to have three levels of complexity. Thus the symbolization must reduce somehow to

$$x \; R_3 \; a \tag{6}$$

Since a principal concern is the possibility of the automatic creation of symbolic questions, we must be careful not to overlook any machine-recognizable clues. Thus it will be helpful to have a symbolism that mirrors the English word order as much as possible. The relational notation used in (6) already seems to be a good choice.

Returning to the example question (1), we note that the candidates for answers are restricted to the class of books. Thus R_3 is a *relation with a restricted domain.* The operation for forming this (in a slightly modified notation of Whitehead and Russell (1950)) is indicated by writing the predicate which defines the restriction (in this case 'B') as a left subscript to the relation-expression. Thus we have

$$R_3 = {}_B R_2 \tag{7}$$

where R_2 is to be further analyzed. (The formal definition of a relation with restricted domain is

$$x\ {}_F R\ y =_{Df} Fx.xRy \tag{8}$$

where '.' is the sign of conjunction.)

Now R_2 is close to the relation W of the example file. It is not the same however because of the reversal of the argument expressions. Apparently this is due to the splitting of the relation expression in (1) by the individual constant. The effect is that a proper symbolization requires the use of the *converse* of a relation. The converse relation is indicated in English sometimes by the passive voice. Thus *has been written by* is the converse of *has written*. Sometimes it is indicated by another expression; for example, *child of* is the converse of *parent of*. In questions, the splitting of the relation expression flags its appearance. The formal definition of the converse $\check{R}$ of R is

$$x\check{R}y = yRx \tag{9}$$

Thus we have

$$R_2 = \check{R}_1 \tag{10}$$

and, finally,

$$R_1 = \text{has written} \tag{11}$$

If we identify R_1 with W of the file, then, by using (6), (7), (10), (11), the symbolization of (1) becomes

$$x\ {}_B[\check{W}]\ a \tag{12}$$

But let us replace the abbreviations by the English expressions and compare the result with the original question. We have

$$\text{What}_{\text{books}}[(\text{has written})^{\smile}]\ \text{Reichenbach} \qquad (13)$$

Note how the analysis closely matches the English word order; only the second argument expression has been shifted to another position.

By studying a small collection of natural-language questions we have derived some general principles of symbolization and then assimilated these principles into a computer program. The program analyzes an input string--a question--by applying a series of *rewrite rules* (explained in the example to follow). The analysis terminates when either the entire string has been reduced to the symbol 'Q' (for 'question') or if no more rules apply. The present program is written in the IBM 7044/7044 FORTRAN IV language and has 30 rules. The root of the program consists principally of a large *computed GO TO statement* which directs control to numbered FORTRAN statements, each heading a body of code representing a single analysis rule. Each rule is in four parts: (1) an identification number; (2) a rewrite instruction; (3) a symbol definition (in case a rewrite involves an abbreviation); (4) a transfer condition (giving the next rule).

The output is in two parts: a *symbol definition table* which constitutes the analysis of the question; and a

rewrite table which is for research purposes and gives the sequence of string modifications.

The printouts for the analysis of question (1) are shown in Figs. 4 and 5. (In this experimental program the English words are truncated after six characters.) Figure 4 gives the symbol definition table. The printout is read from the bottom up. Thus, in line 9, the question is analyzed as having the form of a variable 'V(1)' followed by a relational expression 'R(3)' followed by an individual constant 'C(1)'. (Compare with (6).) R(3) is identified in line 8 as being a *relation with restricted domain* (RD); namely, the *converse* (CNV) of *has written* (lines 6, 5, 2) with domain restricted to the class of *books* (line 4).

In Fig. 5 we show the rewrite table. String 1 is the input question. The only input markings we use are asterisks to set off individual constants. Rule 1 was first applied--it determined 'what' as an interrogative descriptive variable by a dictionary look-up.

Rule 3 determined the auxiliary 'has'--also by a dictionary look-up. The individual constant 'Reichenbach' was next recognized (Rule 4) by the input markings. In the next rule application (Rule 6) we determine the one-place predicate 'books' not by a dictionary look-up but by its position between a variable and an auxiliary (this seems to be a common structure in questions). By Rule 15, which is next to be applied, we determine the relational

	SYMBOL		DEFINITION				
1	V(1)	=	WHAT				
2	A(1)	=	HAS				
3	C(1)	=	REICHE				
4	P(1)	=	BOOKS				
5	R(1)	=	A(1)	WRITTE			
6	R(2)	=	CNV	(	R(1)	)	
7		=					
8	R(3)	=	RD	(	R(2)	,	P(1))
9	Q(1)	=	V(1)	R(3)	C(1)		

Fig. 4--Symbol Definition Table for the Computer Analysis of the Natural-Language Question 'What books has Reichenbach written?'

RULE NUMBER	STRING NUMBER	WORD STRING									
	1	WHAT	BOOKS	HAS	*	REICHE	*	WRITTE	.		
1											
	2	V(1)	BOOKS	HAS	*	REICHE	*	WRITTE	.		
3											
	3	V(1)	BOOKS	A(1)	*	REICHE	*	WRITTE	.		
4											
	4	V(1)	BOOKS	A(1)	C(1)	WRITTE	.				
6											
	5	V(1)	P(1)	A(1)	C(1)	WRITTE	.				
15											
	6	V(1)	P(1)	CNV	(	R(1)	)	C(1)	.		
16											
	7	V(1)	P(1)	R(2)	C(1)	.					
17											
	8	V(1)	RD	(	R(2)	,	P(1)	)	C(1)	.	
18											
	9	V(1)	R(3)	C(1)	.						
27											
	10	Q(1)									

Fig. 5--Rewrite Table for the Computer Analysis of the Natural-Language Question 'What books has Reichenbach written?'

expression by the occurrence of an argument between an auxiliary and a string without arguments; this yields the symbol definition in line 5 of Fig. 4; at this point 'C(1)' is transferred to the end, and as a part of this transformation the converse is introduced. Rule 16 executes an abbreviation, as does Rule 18. Rule 17 restricts the domain because of the adjacency of 'P(1)' with 'R(2)'. Finally, Rule 27 recognizes string 9 as a permissible form of a question.

4. Answering the Question

In the transformation of (1) into (13) we have given a *gross semantic analysis* of the natural-language question. This is meant in the following sense: we have stipulated that 'has written' corresponds to a relation, 'book' to a property, 'Reichenbach' to an individual, and 'what' to an unknown. But to answer the question requires us to have a trick for representing the finer meanings to a computer.

Philosophy tells us there are two aspects of meaning to be considered; *extension* and *intension* (see Carnap, 1956). The intension of a one-place predicate is the property designated by the predicate, and the extension is the class of things having that property. Similarly the intension of a two-place predicate is the relation designated by the predicate and the extension is the class of ordered pairs of individuals which stand in that relation.

Now the example file of Fig. 3 gives, relative to our universe of discourse, the extensions of certain "primitive"

predicates. Thus the extension of 'W' is given in lines 1, 3, 5, and 6 as {(a,b), (a,c), (d,f), (e,f)}; the extension of 'B' in lines 2 and 7 as {b,f}; and the extension of 'P' in line 4 as {c}. For this reason the file is called the *extensional file*.

In order to answer the input question (1) the meanings of the English phrases must be traced back to these stored lists. That is, the answer is to be found in the extension-list of '$_B[\check{W}]$' (a list of pairs) among the entries in the first members corresponding to 'a' in the second. This "calculation" can be done by working with *arrays*; each logical operator of the symbolic question corresponds to a certain *array manipulation*. For example, the array for the converse W is obtained from the array for W by interchanging the columns; the array for a restriction of domain, say of R to B, is obtained by deleting those rows in the array for R whose first place members are not in the array for B.[3]

In the foregoing, we have assumed that 'has written' can be identified with 'wrote' (and hence with 'W'). This is the *intensional aspect* of the problem. We must have a relational sentence stored stating the synonymity of the two expressions. Similarly we must have a store of logical definitions such as (8) (for restriction of domain), and (9) (for the converse). This second file is called the *intensional file*. The dictionary itself can be regarded as part of the intensional file.

Synonyms are the simplest example of a more general class of representations of intensions called *meaning postulates* (Carnap, 1956). These define certain relations in the terms of more primitive predicates, or express in some way relations between relations, or between relations and properties, or between properties and properties. Consider for example

Did Russell co-author with Whitehead? (14)

The computer analysis of this is given in Figs. 6 and 7. The input question is analyzed as a sentence (line 8 of Fig. 6) which is to be affirmed. The analysis reduces to the meaning of the predicate '*did co-author with*' (line 4 of Fig. 6). Now it is clear that the example file contains all the information necessary to answer this. The problem is to relate 'did co-author with' with 'wrote'. This is done by forming the *relative product* of W with its converse. The relative product of two relations R and S, in symbols '$R|S$', is defined formally by

$$x(R|S)y =_{Df} (\exists z)(xRz.zSy) \tag{15}$$

where '$(\exists z)$' is the existential quantifier. Thus we have

$$\text{did co-author with} = W|\breve{W} \tag{16}$$

Perhaps we may wish to modify (16) so that no person is co-author with himself; i.e., we intersect $W|\breve{W}$ with the

	SYMBOL		DEFINITION
1	A(1)	=	DID
2	C(1)	=	RUSSEL
3	C(2)	=	WHITEH
4	R(1)	=	A(1) COAUTH WITH
5	R(2)	=	CNV (R(1))
6		=	
7	S(1)	=	C(2) R(2) C(1)
8	Q(1)	=	S(1)

Fig. 6--Symbol Definition Table for the Computer Analysis of the Natural-Language Question 'Did Russell co-author with Whitehead?'

RULE NUMBER	STRING NUMBER	WORD STRING											
	1	DID	*	RUSSEL	*	COAUTH	WITH	*	WHITEH	*	.		
3													
	2	A(1)	*	RUSSEL	*	COAUTH	WITH	*	WHITEH	*	.		
4													
	3	A(1)	C(1)	COAUTH	WITH	*	WHITEH	*	.				
4													
	4	A(1)	C(1)	COAUTH	WITH	C(2)	.						
15													
	5	CNV	(	R(1)	)	C(1)	C(2)	.					
16													
	6	R(2)	C(1)	C(2)	.								
23													
	7	C(2)	R(2)	C(1)	.								
25													
	8	S(1)	.										
28													
	9	Q(1)											

Fig. 7--Rewrite Table for the Computer Analysis of the Natural-Language Question 'Did Russell co-author with Whitehead?'

relation of *diversity* to complete the definition.[4] The *intensional definition* (16) thus relates the meaning back to the extensional file.

5. Symbolic Questions and Value Sets[5]

We have seen that the problem of answering a question by computer involves processing a formula of the predicate calculus. This formula may either stem from the conversion of a natural-language question to a symbolic question or it may be input directly as a formulation of the user's information requirement. (For example, in the current version of the Relational Data File an information requirement is expressed by means of a special programming language called INFEREX (see Levien, 1969, pp. 17-23); INFEREX instructions involve relational sentences which are essentially equivalent to formulas of the predicate calculus.) We have outlined the logical relationship between the query and the "answer" as embedded in the data base. Let us now take a closer look at the actual mechanism of the answering process.

The formulas[6] to be considered are first classed according to the presence of *free variables*. For example, there are sentence-like expressions such as

$$Bx.aWx \tag{17}$$

in which a free variable occurs; and there are formulas such as

$$(\exists x)(dWx.eWx) \tag{18}$$

with no free variables (the variable 'x' is *bound*). The first is an *open sentential formula*; the second a *closed sentential formula* or simply a *sentence*. The number of free variables in a formula is called its *degree*. Thus (17) is a formula of degree one, while (18) is of degree zero.

The "answers" to a question leading to an open formula such as (17) are those descriptive names which when substituted for the free variable yield a true formula relative to the data base. These names comprise what we call the *value set* of the sentential formula.[7]

[The value set is analogous to the extension, but the value set consists of names, while the extension consists of things. We introduce this new notion for a further reason. We want the members of a value set to have a certain form; e.g., in general, we would not want a description, as for example, 'Reichenbach's 10th book', to be a member of a value set for this would be counter to our intuitions regarding the character of an answer. The issue is further complicated by the fact that although in some cases the value set coincides with an extension-list for *some* predicate,[8] we may want to have a predicate itself in a value set; e.g., as in "answers" to questions such as 'What relation holds between a and b?'[9] For these reasons the concept of value set of a sentential formula seems justified.]

For a formula of degree zero, i.e., a sentence, we take its value set (or simply the value) to be an expression dencting its truth value--say a numeral, '1' for true, '0' for false.

Thus we assume a system consisting of a data base and a sentential formula to be processed. The problem is to calculate its value set.

Let us now look at a source of difficulty. Suppose an inquirer asked

Who did not write *Elements of Symbolic Logic?* (19)

i.e., the symbolic question is

$$\sim(xWb) \tag{20}$$

where '$\sim$' is the negation sign. A human would reject this question as unreasonable, but what should a machine do with it? Should it print every name in the dictionary except 'Reichenbach' or should it somehow prohibit the question? With directly input sentential formulas the problem is even worse--there could be mistakes which cause logical combinations leading to nonsense. For example, consider the disjunctive formula of degree two:

$$(xWb) \vee (yWb) \tag{21}$$

(where '$\vee$' is the sign of logical disjunction). Now any substitution instance which makes either component of (21) true will make the entire formula true. Consequently (21)

leads to a value set of ordered pairs corresponding to the free variable sequence 'x, y' in which a pair has either the form '(a, ...)' or the form '(..., a)' with any descriptive name replacing '...'.

We will attack this problem of "unreasonable" questions by first defining a precise concept called *definite formula*. This will then be used to explicate[10] the vague notion of "reasonable" question. The basic consideration in our definition is this: the difficulty with (20), or (21), is that its value set will change if, without changing the file, a new descriptive name is added to the dictionary. A formula that does not have this objectionable property for any data base is called *definite*.

The formal definition is as follows. We first develop the notion for a given data base D.

Definition. Given a data base D and a formula $\underline{s}$ on D, we define $\underline{s}$ to be semi-definite with respect to D by means of the following *logical test procedure*: Calculate the value set of $\underline{s}$, call it $\omega(\underline{s})$. Form a "pseudo" data base D_* by adding a new descriptive name, say '*', to the dictionary of D (this, of course, leaves the file unchanged). Calculate $\omega_*(\underline{s})$, i.e., the value set of $\underline{s}$ on D_*. If

$$\omega_*(\underline{s}) = \omega(\underline{s})$$

then we say $\underline{s}$ is *semi-definite* on D.

Finally, so that definitude is a logical property, i.e., independent of any particular data base, we define:

Definition. $\underline{s}$ is definite if and only if $\underline{s}$ is semi-definite on every data base.

For example, let $\underline{s}$ be the formula

$$Bc.\sim(xWc) \tag{22}$$

This is semi-definite on our example data base because $\omega('Bc') = 0$, and hence

$$\omega_*(\underline{s}) = \omega(\underline{s}) = \text{the null set}$$

On the other hand, (22) is not definite because, in a world where Bc was the case, '*' would belong to $\omega_*(\underline{s})$; i.e.,

$$\omega_*(\underline{s}) = \omega(\underline{s}) \cup \{'*'\}$$

Let us next turn to the problem of characterizing the definite formulas. The first result is that atomic formulas, i.e., those without operators, are definite. (Note that the atomic formulas include both the sentences in the file and those sentential formulas with variables whose value sets can be determined by a direct match with file items--the value set given by the sequence of variable replacements that produce the match.)

Consider then formulas with operators--the molecular formulas. An inventory of the operators is as follows: the *singulary* operators are the signs for negation ($\sim$), existential quantification ($(\exists x)$), and universal quantification ($(\forall x)$); the *binary* operators are the signs for conjunction (.), disjunction ($\vee$), implication ($\supset$), equivalence

($\equiv$), and for reasons to be given later we also include a special operator for the negation of implication (-). (This last operator can be read 'but not'; it is equivalent to the combination '.$\sim$'.) A sentential formula then has either one or two *components* depending on whether the major operator is singulary or binary.

Let us now characterize the definitude of formulas with *definite components*. The first step is to calculate the *free variable set* of the components. The free variable set of an atomic formula is simply the set of distinct variables occurring in it (without regard for their order); that of a binary formula is the set union of the free variable sets of the components, that of a negation is the same as that of the component; and finally, the free variable set of a quantification is the free variable set of the component less the quantified variable. A formula is of degree zero if and only if its free variable set is null.

Let $\bar{\phi}(\underline{r})$, $\bar{\phi}(\underline{s})$ be the free variable sets of the sentential formulas $\underline{r}$ and $\underline{s}$, respectively. We have the following:

Characterization Theorem. If $\underline{r}$ and $\underline{s}$ are definite, then;

<u>a</u>. $(\exists x)(\underline{r})$ and $(\underline{r}).(\underline{s})$ are definite

<u>b</u>. If $\bar{\phi}(\underline{s}) \subseteq \bar{\phi}(\underline{r})$, then $(\underline{r})-(\underline{s})$ is definite

<u>c</u>. If $\bar{\phi}(\underline{r}) = \bar{\phi}(\underline{s})$, then $(\underline{r}) \vee (\underline{s})$ is definite

<u>d</u>. If $\bar{\phi}(\underline{r}) = \bar{\phi}(\underline{s})$ = the null set, then $\sim(\underline{r})$, $(\underline{r}) \supset (\underline{s})$, and $(\underline{r}) \equiv (\underline{s})$ are definite.

Thus the free variable sets lead to certain *sufficient* conditions for definitude. We next have a companion theorem on *necessary* conditions.

Theorem. The conditions listed in the characterization theorem are necessary for definitude with the following additional provisions:

a. For $(\underline{r})-(\underline{s})$ providing $\underline{r}$ is not contradictory[11]

b. For $(\underline{r})\vee(\underline{s})$ providing neither $\underline{r}$ nor $\underline{s}$ is contradictory

c. For $(\underline{r})\supset(\underline{s})$, providing $\underline{r}$ is not a tautology[11]

d. For $(\underline{r})\equiv(\underline{s})$, providing neither $\underline{r}$ nor $\underline{s}$ is a tautology.

(It turns out that the additional provisions are logically interesting for it has been shown by Di Paola (1968) that the class of definite formulas is not recursive.)

How do we apply the characterization theorem? If the free variable conditions are met, then the value sets can be calculated in terms of the value sets of the components. The specific rules for doing this are given in Kuhns (1967, pp. 67-80).[12] From a standpoint of machine processing, therefore, the most desirable type of formula is one that is definite in every part of its structure. This class of formulas we will call *proper*. More precisely, a formula is proper if 1) it is atomic (and hence definite), or 2) it is definite and its components are proper. The characterization theorem and its companion theorem can therefore be read with 'proper' replacing 'definite'.

But what of formulas that are definite but improper? Consider for example the natural-language question:

Are all of the publications of Reichenbach books? (23)

We can symbolize this as

$$(\forall x)(aWx \supset Bx) \tag{24}$$

Now the component of this formula is indefinite because any substitution for 'x' that makes the antecedent 'aWx' false makes the implication true. On the other hand, the entire formula is definite. This can be seen by eliminating the universal quantifier; i.e., (24) is equivalent to

$$\sim(\exists x)(aWx - Bx) \tag{25}$$

and this is definite by parts (b), (a), and (d) of the characterization theorem.

The problem now arises: Is it possible to transform a definite but improper formula into proper form? If it can be so transformed, we say it is *admissible*.[13] We have proved (Kuhns, 1967, pp. 92-95) that every definite formula without quantifiers is indeed admissible. We have also shown the admissibility of certain special, but nevertheless important, universal formulas. However, the *general* admissibility of definite formulas is an open question.

6. A Concluding Remark

We believe that automatic question answering provides a fertile field for applied logic. We have seen how notions

in logic can be applied to the analysis of natural-language questions and to the design of data bases. We have also seen how the information systems themselves involve new and interesting concepts which lead to problems of both theoretic and practical importance.

ACKNOWLEDGMENTS

I wish to thank my colleagues at RAND for their help in this work. In particular, my thanks go to N. D. Cohen, J. Economos, R. E. Levien (Project Leader for the Relational Data File Project), G. Levitt, M. E. Maron (formerly of RAND and former project co-leader), and G. R. Martins, who reviewed the manuscript of this paper.

FOOTNOTES

1. In the actual file this data line is also stored in relational form, i.e., as a logical relation between a thing and a property (class). Thus the stored data line would be similar to (5').
2. The meaning of 'What' depends on the kind of question in which it occurs. Thus in question (1) 'What' has an *extensional* connotation--it asks for an inventory. However, in a question such as 'What is an apple?', it asks for an *intension* (see §4).
3. A more efficient procedure for evaluation is to process the symbolic question after first transforming away the domain restriction and converse; i.e., to process

   Bx.aWx

   and hence to "intersect" two columnar arrays.
4. Note that the co-authorship relation is *symmetric*. The introduction of the converse in the computer analysis (line 5 of Fig. 6) is therefore redundant; it is necessary however from the linguistic structure of the question and the generality of Rule 15. To see this replace 'co-author with' by 'study under'. Line 7 of Fig. 6 could then be interpreted as 'Whitehead taught Russell'.
5. For a more detailed discussion of the ideas in this section see Kuhns, 1967.

6. From this point on we will consider formulas to be without special operators; that is, we suppose that restrictions of domain, converses, relative products, etc., have been eliminated through definitions (e.g., (8), (9), (10)).

7. If a formula is of a degree greater than one, each member of the value set will be a *sequence* of descriptive names, the length of the sequence corresponding to the degree.

8. Thus, for (17), the predicate would be formed by applying the λ-operator (see Carnap, 1956, p. 3) or be given directly by '$_B[\check{W}]$'.

9. Remarks on the subject of questions which are of particular interest for our purposes are to be found in Carnap (1937, p. 296); Reichenbach (1947, pp. 339-342), Jesperson (1965, p. 303). Remarks on the notion of "giving an extension" are to be found in Carnap (1956, p. 82).

10. We use 'explicate' in the technical sense of Carnap (1956, p. 8).

11. By 'contradictory' we mean a formula which is either false on every data base (if of degree zero) or has a null value set on every data base (if of degree greater than zero). By 'tautology' we mean a *sentence* which is true on every data base.

12. The characterization theorem identifies four classes of situations regarding free variable sets. In the computation rules for binary formulas these situations,

corresponding to parts a, b, c, and d of the theorem, lead to four kinds of procedures. We term these *cartesian* (only for '.'), *restrictive* (for '.' and '-'), *set* (for '.', '-', and 'V'; corresponding to set intersection, difference, and union, respectively), and *boolean arithmetic* (for all binary operators and negation).

13. The example (25) uses the operator '-'; this is why it was introduced--to widen the class of proper formulas.

REFERENCES

Carnap, R., *The Logical Syntax of Language*, (English Edition), Routledge, and Kegan Paul, Ltd., London, 1937.

-----, *Meaning and Necessity* (Enlarged Edition 1956), University of Chicago Press, Chicago, Illinois, 1956.

Di Paola, R. A., *The Recursive Unsolvability of the Decision Problem for the Class of Definite Formulas*, The RAND Corporation, RM-5639-PR, April 1968. (Also, published in *J. ACM*, Vol. 16, No. 2, pp. 324-327, April 1969.)

Jespersen, O., *The Philosophy of Grammar*, W. W. Norton and Co., Inc., N.Y., 1965.

Kuhns, J. L., *Answering Questions by Computer: A Logical Study*, The RAND Corporation, RM-5428-PR, December 1967.

Levien, R. E., *Relational Data File II. Implementation*, The RAND Corporation, P-3411, July 1966. (Also published in George Schecter (ed.), *Information Retrieval: A Critical View*, Proc. of 3rd Annual National Colloquium on Information Retrieval, May 1966, Thompson Book Co., Washington, D.C., 1967, pp. 225-241.)

-----, *Relational Data File: Experience with a System for Propositional Data Storage and Inference Execution*, The RAND Corporation, RM-5947-PR, April 1969.

Levien, R. E., and M. E. Maron, *Relational Data File: A Tool for Mechanized Inference Execution and Data Retrieval*, The RAND Corporation, RM-4793-PR, December 1965.

-----, *A Computer System for Inference Execution and Data Retrieval*, The RAND Corporation, RM-5085-PR, September 1966. (Also published in *Comm. ACM*, Vol. 10, No. 11, 1967, pp. 715-721.)

Maron, M. E., *Relational Data File I: Design Philosophy*, The RAND Corporation, P-3408, July 1966. (Also published in George Schecter (ed.), *Information Retrieval: A Critical View*, Proc. of 3rd Annual National Colloquium on Information Retrieval, May 1966, Thompson Book Co., Washington, D.C., 1967, pp. 211-223.)

Reichenbach, H., *Elements of Symbolic Logic*, Macmillan Company, N.Y., 1947.

Whitehead, A. N., and B. Russell, *Principia Mathematica* (Second Edition), Cambridge University Press, 1950.