American Journal of Computational Linguistics *Microfiche 75: 3*

THE DERIVATION OF ANSWERS FROM LOGICAL FORMS IN A QUESTION ANSWERING SYSTEM

FRED J DAMERAU

IBM Corporation
Thomas J Watson Research Center
Yorktown Heights, New York

ABSTRACT

This paper describes how the process of generating a response given an underlying representation for an input question is accomplished in the Transformational Question Answering (TQA) system under development at IBM Research, a brief description of which is given.

The last formal level of representation in this system is called a logical form. The basic method of evaluation of logical forms is the generate and test" paradigm, used, for example in the LUNAR system (Woods, Kaplan and Nash-Webber, 1972), although the implementation must be fairly efficient in order to be practical on a moderate size data base. The basic idea is to keep track of the equivalence relationships between the variables in the logical form and associated constants, and use this information to derive from the data base the extensions of the predicates contained in the logical form. A similar proposal has been made by Reiter(1976). The logical forms and the process by which candidate sets are computed from these forms are described in considerable detail. We believe it should not be necessary for a computational linguistics project to describe operations beyond the last level of formal representation in order for an outsider to understand exactly how a system operates sufficiently well that he can predict its behavior. Although we have attempted to achieve that, we still have a considerable way to go.

INTRODUCTION

This paper describes how the process of generating a response given an underlying representaion for an input question is accomplished in the Transformational Question Answering (TQA) system under continuing development at IBM Research. TQA has been operational in a laboratory mode for several years. The system is now installed in the office of the planning department of a small city where it is used to access the file of land use for each parcel of land in the city, (about 10,000 parcels with 40 pieces of data for each parcel). The sysytem is undergoing modifications and improvement prior to a formal evaluation stage.

SYSTEM OVERVIEW

A generalized flow diagram of the TQA system is given in Figure 1. Input, from a display device or typewriter-like terminal, is fed to the preprocessor, which segments the input character string anto words and performs lexical lookup. The process of lookup is complicated somewhat by a provision for synonym and phrase replacement. Words like "car" and "automobile" are changed to "auto", and strings like "gas station" are frozen into single lexical units.

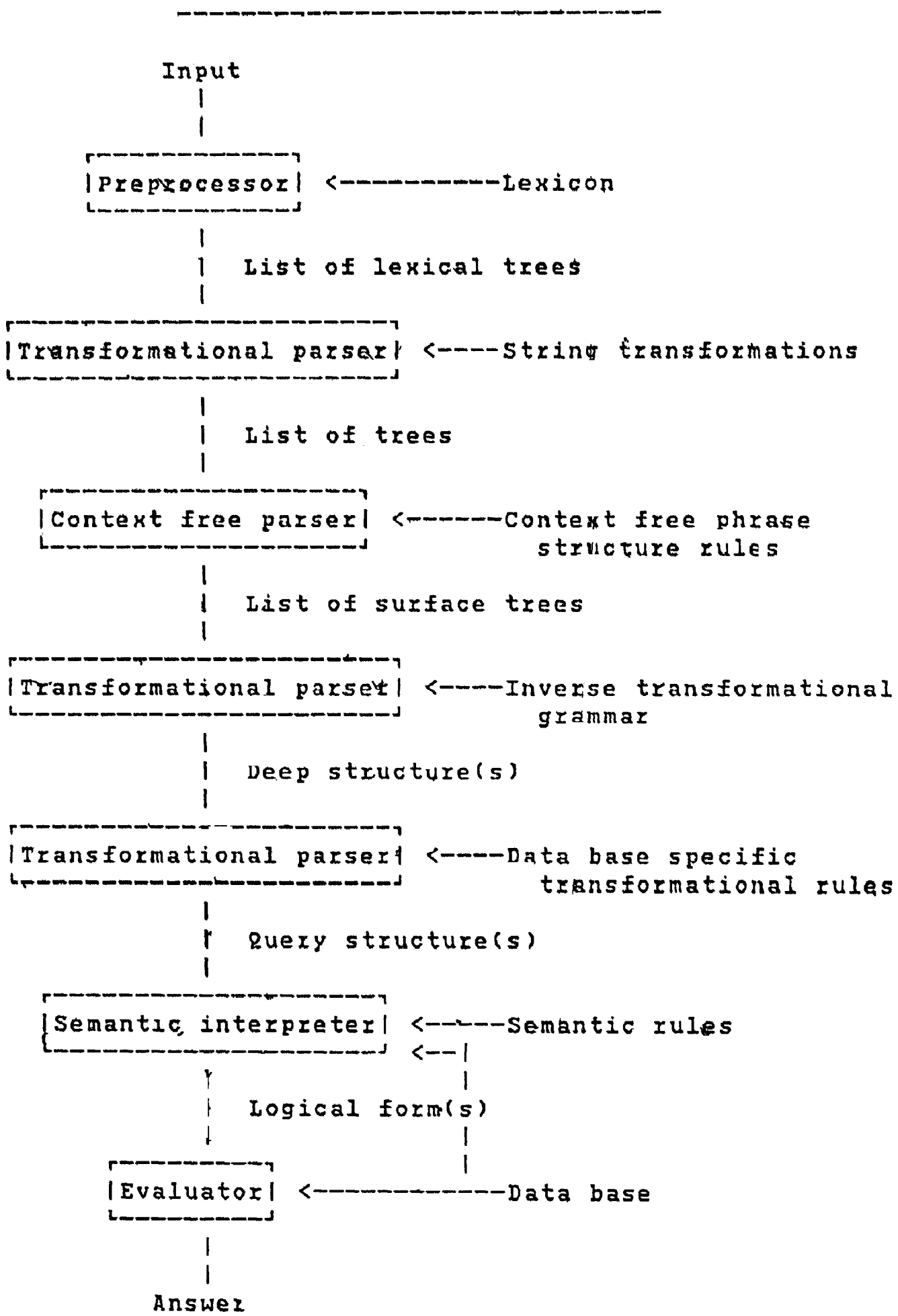
Input
Preprocessor
Lexicon
List of lexical trees
Transformational parser
String transformations
List of trees
Context free parser
Context free phrase structure rules
List of surface trees
Transformational parser
Inverse transformational grammar
Deep structure(s)
Transformational parser
Data base specific transformational rules
Query structure(s)
Semantic interpreter
Semantic rules
Logical form(s)
Evaluator
Data base
Answer

Figure 1

The output from the lexical lookup is a list of trees, each tree containing part of speech information, syntactic features and semantic features, as required. A description of the lexical component, now obsolete in its detail but still valid in main outline is given in Robinson(1973). The list of trees is input to a set of string transformations, described in Plath(1974). These transformations operate on adjacent lexical items to deal with patterns of classifiers, ordinal numbers, stranded prepositions, and the like. The effect of this phase is to reduce the number of surface parses and the amount of work done in the transformational cycle. The resulting list of trees is input to a context free parser, which produces a set of surface trees, each of which is fed to the transformational recognizer.

The recognizer attempts to find an underlying structure for each surface tree, Plath(1973). Typically only one of a set of surface trees will result in an underlying structure. This structure itself is input once again to the transformational recognizer, using a (small) set of grammar rules tailored to a specific data base to produce a query structure. Query structures are similar to underlying structures in form, but reflect the particular meaning constraints resulting from the format and content of a given data base. The query structure tree is processed by a Knuth-style semantic interpreter, Petrick (1977), producing a logical form. A logical form can best be thought of, in

our context, as a retrieval expression which is to be evaluated, producing an answer to the English input query. Since the major part of this paper is concerned with processing logical forms, discussion of their specifics will be deferred until later

The process of answer extraction from the data base is accomplished by a combination of LISP and PL/I programs, described below, and an experimental relational data base management system called Relational Storage System (RSS) (Astrahan, et al. 1976). The RSS provides the capability to generate a data base of n-ary relations, with indexes on any field of the relation, and low-level access commands like OPEN, NEXT, CLOSE, with appropriate parameters, to retrieve information from such a data base.

All the processing modules are under the control of a driver module, which maintains communication with the user, calls the processors in the correct sequence, and tests for errors. An example of the processing of a question, with the intermediate outputs, is given in Figure 2.

In this example, the numbers 2945, 6535, 6635, 6975 are the numbers of milliseconds of computer time used up to the point shown, on an IBM S/370 Model 168. The structures printed are a bracketted terminal string representation of structures which are stored and manipulated as trees by the

```
what are the heights of the drug stores ?

2945 SURFACE STRUCTURES:

   (((WH SOME) (THING X1)) BE (THE ((HEIGHT X4)
     (OF (THE ((DRUG_STORE 591) X7))))) ?)

6535 UNDERLYING STRUCTURES:

1. (BD IDENTICAL (THE (X4 (* BD HEIGHT X4
    (THE ((DRUG_STORE 591) X7)) BD *))) ((WH SOME)
    (THING X1)) BD)

6635 QUERY STRUCTURES:

1. (THE (X4 (* BD HEIGHT X4 (THE ((DRUG_STORE 591)
    X7)) BD *)))

6975 LOGICAL FORM:

(setx 'X4
  '(foratleast 1 'X44
    `(setx 'X7
       '(testfct
         '591
         '('LUC X7 '1976)
         '= ) )
      (testfct
       X4
       '('JSTOR X44 '1976)
       '= ) ) )

7995 ANSWERS:
```

	NUMBER STORIES
1976	
70590001610	1
80100000710	1
80100000811	2
90430000910	1

Figure 2

processing programs. The nonterminal nodes of the tree, together with their associated complex features, represent much additional information that is not shown here. The number 591 is a land use code which, in the data base, indicates a drug store, and the long numbers in the answer are the parcel identifiers, (ward-block-lot).

From this brief description, it should be apparent that the TQA system, considered as a black box, is similar to many others. In particular, there is a designated level of meaning representation, the logical form, which is the last formal construct in the system. The remaining processing necessary to derive an answer and to format it for presentation to a user is accomplished by an unstructured set of computer programs. Two separate issues arise as a result: how efficiently can the logical form be evaluated against a real data base, and to what extent do the processing functions further specify meaning, beyond that carried by the logical form?

EVALUATION OF LOGICAL FORMS

The basic method of evaluation of logical forms is the "generate and test" paradigm used, for example, in the LUNAR

system Woods Kaplan and Nash-Webber, 1972). The simple version of this paradigm, used by Woods and implemented in our early systems, involves checking pre-selected lists of objects or, in the worst case, all the objects known to the system, to see if they satisfy the query predicates. It is computationally impractical except for small data bases. Our current variant of this method is much more efficient. The basic idea is to keep track of the equivalence relationships between the variables in the logical form and associated constants, and use this information to derive the extensions of the predicates contained in the logical form from the data base. A similar proposal has been made by Reiter(1976) We do not however, make such extensive use of query transformations as Reiter outlined.

Logical forms

In order to describe the evaluation process, it is necessary to describe the logical form in somewhat more detail, referring for an example again to Figure 2. In the first place, except for the set-forming function setx, which takes as arguments a variable name and a proposition, all other well-formed formulas are composed of predicates and their arguments. Some of the predicates are perfectly ordinary like greaterthan. Some are quantifiers, like foratleast, which takes a limit argument n, an argument

which is a set, and a proposition p, and which is true just in case n or more elements of the specified set satisfy the proposition p. Others are special application predicates like parcel, which is true just in case its single argument is a parcel identifier.

The main data base related predicate is named testfct. Referring to Figure 2, it is seen that testfct has three arguments. The first is a constant or a variable which will be replaced by a constant before evaluation, the second argument is a list whose members determine a particular data base value, and the third is an operator specifying the relation which must hold between the first argument and the data base value determined by the second argument.

The data base can be thought of as a collection of binary relations, all sharing the same key. In our application, this is the parcel identification or account number, by which any piece of property can be identified. The list which is the second argument of testfct consists of the relation name and the key which identifies a value in the relation. The key actually has two parts. The second part is a year, now unused, although since the files in which we are currently interested are changed on a yearly basis, we anticipate maintaining and accessing historical data. The first part of the key is the account number mentioned above. In general, the second argument of testfct must be

sufficient to identify a unique binary relation and value in that relation.

If the logical form is itself a proposition the system will answer either "yes" or "no". If the logical form has a top level setx, the system will print the members of the set satisfying the specified proposition, perhaps along with some identifying information:

Simplifications

A number of simplifications can be, and in part have been, carried out on logical forms prior to evaluation. Some predicates, for example, are essentially empty for purposes of evaluation, in that they always evaluate to true. As an example, the predicate dollar, for information fields referring to taxes, is empty of meaning because the processor assumes that the contents of the taxes field are always dollars. A slightly less obvious example of a possible simplification can be seen in Figure 2. The set argument of the foratleast predicate contains no free variables. It is not necessary, therefore, to evaluate the inner setx function for each evaluation of the foratleast predicate. Instead, the setx function is evaluated as soon as the semantic interpreter has discovered that it has no free variables, using the standard evaluation mechanism, and the value, i.e., a set, is substituted for the setx

expression. Our system performs simplifications of this kind in its normal mode (although it can also delay all evaluations until a complete form has been built), so that the final logical form seen by the retrieval functions during processing is usually that shown in Figure 3, where the inner setx has been replaced by the satisfying set viz the parcel identifiers of the set of drug stores.[2] After all the applicable simplifications have been done, the resulting form is passed to the evaluation function, EVALU.

The Pre-evaluator

It might seem that since the system has been written in LISP, it would only be necessary to define the appropriate functions and then call the regular LISP evaluator, instead of a special evaluator like EVALU. While this would be possible, the difficulty with such an approach can readily be seen by considering the embedded setx in Figure 2. The desired set of X7s is that set of parcel identifiers fo which the associated land use code is "591". testfct is a predicate which is true for the appropriate X7s, but what is the candidate set of X7s which should be tested? At worst, the system might consider the set of all objects it knows about. As a better choice, the system could infer from the syntax of testfct that the candidates are all members of the set of parcel identifiers, but still there are almost 10,000

```
what are the heights of the drug stores ?

2930 SURFACE STRUCTURES:

1. (((WH SOME) (THING X1)) BE (THE ((HEIGHT X4)
     (OF (THE ((DRUG_STORE 591) X7))))) ?)

6500 UNDERLYING STRUCTURES:

1. (BD IDENTICAL (THE (X4 (* BD HEIGHT X4 (THE
     ((DRUG_STORE 591) X7)) BD *))) ((WH SOME)
     (THING X1)) BD)

6599 QUERY STRUCTURES:

1  (THE (X4 (* BD HEIGHT X4 (THE ((DRUG_STORE 591)
     X7)) BD *)))

7176 LOGICAL FORM:

(setx 'X4
  '(foratleast 1 'X44
     (90430000910 80100000811 80100000710
      70590001610)
     (testfct
      X4
       ('JSTOR X44  1976)
      '= ) ) )

7385 ANSWERS:
```

	NUMBER STORIES
1976	
70590001610	1
80100000710	1
80100000811	2
90430000910	1

Figure 3

of those A much better approach is to attempt to compute the extension of those predicates for which the variable being sought is an argument Again referring to Figure 2, a

reasonable set (in fact the perfect set) of candidates for X7 can be found by looking in the data base for that set of identifiers for which the land use code is 591 If the data base is properly organized, such a search can be very fast Not all predicates are so simple however. The remainder of this section will describe in some detail how candidate sets for more complicated predicates are arrived at. Once candidate sets have been computed the EVALU function can invoke the LISP evaluator on the logical form. The alternative of including a candidate generator in the setx program and all the potential top level predicates and then applying the LISP EVAL function directly seems much less attractive.

As a preliminary, notice that we need only insure that candidate sets have been established for all the setx variables in a logical form. This is so because, while each quantifier has an associated variable, the domain of that quantifier is either given explicitly as a list of constants, or implicitly by a setx expression. Secondly, since the object of pre-evaluation is merely to find efficient, not necessarily optimal, candidate sets for the setx variables, we need not keep track of the structure of a complex predicate. As an example, consider Figure 4, which is the logical form for the question,

"What drug stores are located in ward 2?"

The predicate of the setx is "and", but for purposes of

```
(setx 'X2
  '(and
    (testfct
      '591
      '('LUC X2 '1976)
      '= )
    (testfct
      '2
      '('WARD X2 '1976)
      '= ) ) )
```

Figure 4

determining a candidate set we can consider each term of the "and" individually. Evaluation of the form with a given candidate set will ensure that a particular member satisfies both terms of the "and".

Operation of the pre-evaluation function. Pre-evaluation is accomplished by a function EVALUA, which takes a logical form, i.e., a setx expression or a proposition as its argument. It determines the type of form with which it is dealing and calls an appropriate specialist routine If as in the case of the "and" of Figure 4, the logical form being considered contains more than one component form, EVALUA calls itself recursively. Consequently, pre-evaluation is a depth-first, left-to-right process. The function always returns nil, all work being accomplished by changes to global variables. Among these are a LISP variable which

contains a list of all setx variables in the logical form, a LISP variable which lists each query variable for which a value has been found, and its value, and a LISP variable which keeps track of the equality relationships which have been discovered between query variables for which a value is yet to be found.

Operation of the algorithm can be better understood by considering somewhat more complicated examples than those seen previously. When EVALUA is given the logical form of

---

What parcels have an area exceeding 550000
square feet ?

7524 LOGICAL FORM:

```
(setx 'X2
  '(and
    (foratleast 1 'X39
      (setx 'X5
        '(testfct
          X5
          '('PARAREA X2 '1976)
          '= ) )
      '(greaterthan X39 '550000) )
    (parcel X2) ) )
```

Figure 5

Figure 5, it calls the setx specialist, which adds X2 to the (null) list of set variables and the (null) list of query variables, and calls EVALUA with the associated setx predicate, "and". As mentioned, this simply results in two

calls to EVALUA, the first of which causes the quantifier specialist to be invoked. (The second call, when made, will not cause any change to the global lists of candidate values for variables, since a candidate set of all parcel identifiers is not useful for purposes of retrieval.) X39 is added to the list of query variables, and the domain argument of the quantifier is inspected. When this is seen to be an instance of setx rather than a list of constants, two actions are taken. Notice that whatever the domain of X39 is, it is a subset (perhaps not a proper subset) of the domain of X5, i.e., the candidate set for X5 must include at least all of the elements of X39. Further, any restrictions which can be imposed on X39 can also be imposed on X5, since the proposition associated with the quantifier is the one to be satisfied, and any candidate not meeting this criterion would be superfluous. Therefore, we can 1) enter into the list of variable relationships the information that for purposes of the pre-evaluator, X39 and X5 are equivalen and 2) call EVALUA once more with the setx associated with X5 as an argument.

X5 is added to the list of set variables, and reinvocation of EVALUA with the setx predicate causes a call to the specialist for testfct. Since there are two variables in testfct, X5 and X2, for which values are unknown, a call to the data base cannot yet be made. The instance of testfct is placed on a list of pending data base calls,

preceded by the variables which require values. (Each time a value for a variable is found, that list is inspected, and any data base calls which can then be made are executed.) Return is made to the quantifier specialist, which calls EVALUA with the predicate over whose arguments quantification is made, viz., greaterthan.

The specialist for numeric predicates, finding that one argument is a variable and the other a constant, causes a change in the variable list to show that X39 and consequently X5 are greater than 550,000. A value like ">550,000" can be used by the data base component to narrow its search just as well as a constant or list of constants, and is therefore acceptable as the value of a candidate list. These changes to the variable lists cause the list of pending data base calls to be inspected and, since only one variable is now unknown in the stacked testfct, a call to the data base is made for those parcels with an area greater than 550,000 square feet.

The specialist for testfct instructs the data base search routine to return as a value a list corresponding to the remaining variable in the form, i.e., X2. In the present example, that is a list of parcel numbers, viz., those parcels which have an area exceeding 550,000 square feet. This list is then assigned as the value of the candidate set for X2.

The stack of recursive calls to EVALUA will now unwind, until a return is made to the evaluation function EVALU. This function determines that candidate lists for all the setx variables have been found, and creates a new list of variable-candidate set pairs for use by the setx function itself. Finally, EVALU can call the LISP evaluator, with the original logical form as an argument.

The case of negatives. The predicate "not", denoted in our system by not* to distinguish it from the LISP not, presents special problems for the kind of system outlined above. A simple example of the difficulty can be seen in

```
What drug stores are not in traffic
zone 6 ?

5651 LOGICAL FORM:

     (setx 'X3
       '(and
         (not*
           (testfct
             '6
             '('TRAFZ X3 '1976)
             '=) ) )
         (testfct
           '591
           '('LUC X3 '1976)
           '= ) ) )
```

Figure 6

Figure 6, which corresponds to the question

"What drug stores are not located in traffic zone 6?"

and variants thereof. When the testfct specialist is given the first half of the and in this form, along with information that there is a dominating not*, it could in principle generate a data base call, since there is only one unassigned variable. The effect would be the retrieval of all parcel identifiers of parcels not located in traffic zone 6. This is a substantial fraction of the data base, and would require inordinate amounts of time and storage space to handle Notice that the other half of the and will also provide a candidate list for the variable X3, presumably much smaller in size. It appears to be the case, from our so far limited experience, that questions containing only a single negated search clause hardly ever occur. The evaluator therefore puts a testfct call of this type on the stack mentioned earlier, indexed by the variable(s) corresponding to the parcel identifier. When the second half of the and of Figure 6 is processed, and a value found for X3, the deferred testfct will be unstacked, resulting in a data base call, and causing a retrieval based on that list of identifiers rather than on the negated value. This data base search is necessary, since we must find the traffic zones for the parcels contained in the candidate list.

This example is also an illustration of why, as was mentioned above, the logical form as a whole must in general be evaluated by the LISP evaluator. In this case, the candidate set for X3 derived from the second clause of the

and is a superset of the answer set which can only be derived by evaluating the whole conjunction. Some efficiencies could doubtless be gained by skipping evaluation in those cases where it is unnecessary, but that is purely an implementation decision

The not* of Figure 7 presents a different kind of problem

```
How many banks have a height not exceeding
 5 floors ?

7966 LOGICAL FORM:

(setx 'X1
 '(quantity X1
   (setx 'X3
     '(and
       (not*
         (foratleast 1 'X45
           (setx 'X6
             '(testfct
               X6
               '('JSTOR X3 '1976)
               '= )
             '(greaterthan X45 '5) ) )
       (testfct
          611 *
         '('LUC X3 '1976)
         '= ) ) ) ) )
```

Figure 7

---

from the previous example. Firstly, notice that the negative must be passed inside the quantifier since the alternative of finding all buildings greater than 5 stories in height and then getting the complement set with respect

to all buildings is extremely unattractive computationally. In the second place, a search qualifier of "<= 5" does not intuitively seem to be much worse than "> 5", at least in the absence of data base distributional statistics. One might, therefore, generate a search with such a qualifier. Our present system does this, although experience may show that all instances of testfct dominated by not* should be deferred, as are the cases of "¬=", for efficiency reasons.

Other specialists Most of the important specialist routines in the pre-evaluator have already been mentioned. There are a few others which should be noted. One is a generator function which, given a predicate, will produce its extension from a stored list. This feature was heavily used in our early system, which had a small data base, but is currently hardly used at all, though it remains available. In principle, one could, given a predicate like "SCHOOL(X)", generate a list of schools. In the present application, this would not be useful, but might in some other. The sole uses at present are a generator for the predicate RANK, for which a list of numbers from 1 to 100 is produced, and for the predicate YEAR, which produces a list of the numbers 1960 to 1985.

The proposition "(QUANTITY x s)" is true if x is equal to the cardinality of the set s. The associated specialist has the obvious function of determining x when s is an instance

of setx.

Equality between variables can be inferred where the domain of a quantified variable is given by an instance of setx, as was illustrated above. Certain predicates also allow this inference to be made. It is clear that predicates like "EQUAL", "SAMEREF", (for "same reference"), and "IDENTICAL" should belong to this class. Since variables can only refer to individuals, the predicate "MEMBER" also is in this class, e.g., given (MEMBER X3 (SETX ....)), a candidate set for X3 can be derived by evaluating the setx expression.

Further efficiency considerations. It has already been noted that generation from instances of testfct with an operator of "¬=" are deferred until enough information is available to execute the query using a list of parcel identifiers. Some other steps have also been taken to reduce data base access time and subsequent evaluation time For one thing, the semantic interpreter has a preferred ordering for instances of the predicate testfct. For example, the relation "WARD" divides the parcels of the city into 6 classes, while the relation "LUC" (Land Use Code) divides the parcels into several hundred classes. If there is no intrinsic reason for ordering the instances of testfct differently, the one with "LUC" will occur earlier in the logical form, (cf. Figure 4). The pre-evaluation specialist

for testfct makes use of this ordering in two ways. If a variable has been assigned a list of identifiers containing fewer members than some threshold x, (x is currently set to 25, but can easily be changed), then a retrieval will always be made using the list of identifiers rather than by a constant compared to data base values. In Figure 4, the second call to the testfct specialist will look up the ward of the four drug stores instead of finding the hundreds of parcels in ward 2. In some instances, particularly for relations like Land Use Code, this may result in more data base accesses than retrieving a new set of keys depending on value, but the improvement cannot be large. In many other instances, there is a big reduction in accesses.

If the candidate set is larger than 25, retrieval will be made using the constant, but the length of the current candidate list is used to limit the number of accesses. Thus, if the current candidate list is 50, the data base access program will terminate if it finds more than 50 identifiers with the value being used. A re-access is then made using the list of identifiers. Again, this may result in inefficiency in some cases where searches are ended just before normal termination, but it does provide a guarantee against excessively long retrievals.

Any number of other efficiency measures could be adopted, and more may be necessary than we now have. For the moment,

these seem to provide acceptable retrieval times.

The Evaluator

For the most part, evaluation of logical forms is quite straightforward. Hidden semantic effects are discussed in the next section; here we are mainly concerned with computation.

Each instance of setx searches the list of variable-candidate set pairs to find the candidate set associated with its own variable and substitutes the members of the set for the variable one by one into its associated predicate. Those members of the candidate set for which the predicate evaluates to true are placed in the solution set. Operation of the quantifier predicates is similar to that of setx, except that, as in Figure 5, it may be necessary to evaluate an instance of setx to find the domain of the quantification variable.

Evaluation of the other predicates consists simply of applying a corresponding LISP function to the arguments. Sometimes the final logical form to be evaluated bears no obvious relation to the input question, as in Figure 8. The usual reason is that a large amount of evaluation was done

```
Are there more than 25 parcels in the Carhart
 neighborhood ?

36229 LOGICAL FORM:

(greaterthan '176 '25)
```

Figure 8

---

during interpretation. because form contained no free variables. The full logical form corresponding to Figure 8

```
Are there more than 25 parcels in the Carhart neighborhood.?

15986 LOGICAL FORM:

(forall 'X115
  (setx 'X38
    '(quantity
      X38
      (setx 'X34
        '(and
          (testfct
            '9
            '('NEIGH X34 '1976)
            '= )
          (parcel X34) ) ) ) )
   (greaterthan X115 '25) )
```

Figure 9

---

is given in Figure 9.

The evaluation of the predicate testfct is not as obvious as that of the others One of the design goals in the project has been to make it relatively easy to move from one data base to another. As part of that effort, we have attempted to make the LISP programs, as contrasted to the PL/I programs, insensitive to the structure of the data base. Our approach to this has been to define a list structure, essentially nested binary relations, into which the real data structure is mapped. Restructuring is accomplished by the PL/I program which serves as the LISP - RSS interface. At the same time as the PL/I program returns values to the testfct specialist during the pre-evaluation phase, it formats the corresponding data base items into the standard structure and writes them onto a disk file, in effect creating a sub-data base for the particular query. Only the sub-data base is used during evaluation of logical forms, to find values corresponding to keys in the instances of testfct. In addition to isolating the LISP programs from the real data structure, this tactic makes it unnecessary for any programs called by the evaluator to re-access the full data base, with a consequent efficiency gain.

Creation of the standard LISP data base into which the real data is translated has meant that the set of ISP functions has undergone the least modification in our change of data base from business statistics to planning data. Except for improvements made to increase the efficiency of

programs, these routines are almost the same as they were before.

SEMANTIC EFFECTS OF EVALUATION

In principle the processes which will bw used to compute the answer to a query should be obvious at the level of either the query structure or the logical form. We have not, however, been completely successful in accomplishing this. In some cases, we can see how it might be done and have not gotten around to doing it because of more urgent concerns. In other cases, we can see how to do it, but not how to do it efficiently. In a few cases, it is not clear what to do.

Approximation. Consider the sentence and corresponding logical form shown in Figure 10. The precise system meaning of "about" is clearly hidden in the program corresponding to the operator APPROX. In the present implementation, APPROX of x and y is true if:

1) when $y<10$, $x>y-2$ and $x<y+2$,

2) when $10<y<40$, $x>y-3$ and $x<y+3$,

3) when $y\geq 40$, $x>y-.05y$ and $x<y+.05y$.

I.e., 6 and 8 are approximately equal to 7, 14 and 18 are approximately equal to 16, and 951 and 1049 are approximately equal to 1000. Whether this definition is

```
What parcels are assessed at about $ 1000000 ?

6168 LOGICAL FORM:

(setx 'X2
   (and
    (testfct
      '1000000
      '('VNCITY X2 '1976)
      'APPROX )
    (parcel X2) ) )

6373 ANSWERS:
                                    ASSESSMENT-
                                     CITY_$
  1976
   7059000260O                         977,750
   70430000609                       1,000,000
   70400000600                       1,033,425
   70310000602                         955,900
```

Figure 10

satisfactory or not clearly depends on a variety of contextual factors. It should also be clear that the semantic interpreter could produce a logical form in which this meaning was expressed directly. We have chosen to express the meaning in our processing programs primarily for convenience, i.e. it was easiest to do it in this way, and there was no obvious reason to do it elsewhere.

A similar but slightly different example is shown in Figure 11, where the output rather than the input is to be an approximation to the true value. In this instance, a function called FUZZUP is applied to a data base value to

```
About how many square feet do the drug
stores have ?

7227 LOGICAL FORM:

  (setx 'X4
    '(foratleast 1 'X52 '(18570 24814 8440
 5465) '(approx X52 X4)) )

7479 ANSWERS:

                                   PARCEL
                                    AREA-SQ_FT
  1976
   70590001614                        19,000
   80100000714                        25,000
   80100000814                         8,400
   90430000914                         5,500
```

Figure 11

find that number with the maximum number of trailing zeros which satisfies the APPROX relation. The fuzzed value rather than the true value becomes the output.

A more subtle case is illustrated by Figure 12. It seems clear that what is really wanted are those parcels with an area of a million square feet or more, rather than exactly 1,000,000 square feet. If the latter result is wanted, the question is better phrased "exactly 1,000,000", (and must be phrased in this or a similar way in our system.) On the other hand, a value like 1,000,205 seems to imply that exact equality is wanted. This intuition is captured in our system

```
what parcels have an area of
1,000,000 square feet?

8416 LOGICAL FORM:'

(setx 'X2
  '(and
    (foratleast 1 'X45
      (setx 'X5
        '(testfct
          X5
          '(PARAREA X2 '1976)
          '= ) )
      '(equal X45 '1000000) )
    (parcel X2) ) )

8789 ANSWERS:

1:  70880000900
2:  70790000100
3:  70790000100
          . .
22:  80300000101
MORE PARTICULARS DESIRED?
YES OR NO?
yes
EXPLANATIONS TO THE ANSWERS:
FOR 70880000900 MORE - 13590410
FOR 70790000100 MORE - 5977500
FOR 70790000100 MORE - 5583085

FOR 80300000101 ALMOST - 958320
```

Figure 12

by having the testfct predicate inspect its numeric arguments with a function called ROUNDNM, which is true if an argument is a round number, defined in our system to be a number greater than 99 in which at least the rightmost half of its digits are 0. In the case of round numbers, it seems reasonable to give as an answer the identifier of a parcel

whose area is only slightly less than 1,000,000 square feet, as well as greater.. In our implementation, we use the same lower limit as for APPROX, but this may be too low. In order to insure that the answer is correctly understood by the user, the system saves the exact values retrieved and displays them on request, as shown in Figure 12.

Equality of character values. A problem analagous to that of numerical approximations occurs also in comparing character string values. Consider the question and answer pair shown in Figure 13. The contents of the OWNER field

---

```
What parcels does Shell own ?

4244 LOGICAL FORM:

(setx 'X2
  '(and
    (testfct
      'SHELL
      '('OWNER X2 '1976)
      '= )
    (parcel X2) ) )

4432 ANSWERS:
                                  OWNER
  1976
   70600009501          SHELL OIL COMPANY
   80220003300          SHELL OIL CO
```

Figure 13

---

have not been standardized, so that parcels could be owned

by "Shell Oil", "Shell Oil Co.", etc. Fortunately, for names of persons, last names are listed first, so that the strategy of assuming equality if the input argument and the field value match up to a comma or a blank is generally successful. Problems do arise; for example, properties belong both to "The City of ..." and "City of ...", where the left match fails to find all the relevant data items. The opposite situation, i.e., over-generalization, can of

---

```
what parcels does Gluck own ?

4525 LOGICAL FORM:

(setx 'X2
  '(and
    (testfct
      'GLUCK
      '('OWNER X2 '1976)
      '= )
    (parcel X2) ) )

4742 ANSWERS:

                                OWNER

  1976
   90400000100          GLUCK, DE & ORS
   90410000900          GLUCK, CP
```

Figure 14

---

course also occur, cf. Figure 14. In any event, the decision as to what constitutes sameness of reference is buried in computer code in this instance in the PL/I program as well as in the LISP definition of the function

SAMEREF.

Definitions. The extensional definition of most predicates can be derived from the data base. A few predicates are defined by the system code. Examples are RANK and YEAR. which as mentioned above have associated generators An additional example is LASTYEAR which is defined to be the previous year. Many other definitions of this kind have been eliminated in the current version of the system.

Answers. It is not always obvious what constitutes the answer to a question. Consider the example in Figure 15. Both the English question in its literal reading and the logical form would seem to imply that the question would be answered by presenting only the numbers in the right hand column of the table which is actually printed as an answer. Yet it is quite clear that a simple list would generally be useless without the parcel identifiers printed on the left, and indeed that identification would be expected by the person entering such a question. The example of Figure 16

---

what is the gross floor area of the drug stores ?

7245 LOGICAL FORM:

```
(setx 'X4
   (foratleast 1 'X44
     (90430000910 80100000811 80100000710
      70590001610)
     (testfct
      X4
      '('PGFAREA X44 '1976)
      '= ) ) )
```

7465 ANSWERS:

	GROUND-FLOOR AREA-SQ_FT
1976	
70590001610	7,078
80100000710	10,125
80100000811	6,500
90430000910	1,800

Figure 15

---

is less clear. An enumeration of the three wards in which the four drug stores were located might have been a sufficient answer. The answer given would be correct for

"In What ward is each drug store located?"

Moreover, given the question

"What are the wards which have drug stores?"

it is clear that only a list of wards should be the output, and given

"What is the combined floor area of the drug stores?"

only a single number representing the total is the desired

```
In what wards are the drug stores located ?

9403 LOGICAL FORM:

(setx 'X3
  '(foratleast 1 'X64
    '(90430000910 80100000811 80100000710
      70590001610)
    '(testfct
      X3
      '('WARD X64 '1976)
      '= ) ) )

9597 ANSWERS:
                                    WARD
  1976
   70590001610                                2
   80100000710                                3
   80100000811                                3
   90430000910                                5
```

Figure 16

answer. (Our system does not as yet answer this question or its analogues, although this is planned for later in the year.) Since the ambiguity exhibited by the question of Figure 14 is so pervasive in an application of this kind, we have chosen to present a maximally general answer, including identifications, when we are unable to resolve the ambiguity directly. An exchange with the user could be devised to elicit the information for resolution, but would apidly become tedious for questions of this type. For yes/no questions, and for questions in which there is only one object in the answer set, this problem naturally does not

arise, and the appropriate answer is easily produced..

CONCLUSIONS

We have not yet concerned ourselves with adding an English response generator to the TQA system. In the applications envisioned at present, such a capability does not seem to be critical. We are able to manage with short answers from the data base and with canned information and error messages. In spite of this omission, it should also be apparent that our computational component has a considerable amount of linguistic knowledge embedded in it, more than we would like. Whether it is possible to achieve a level of formal representation which would make this unnecessary is still unclear. Moreover, even if it were possible, it is not clear whether such a solution would be efficient enough, or even if it would be more perspicuous than the current system We intend to proceed as far as we are able in this direction, out of conviction that practically useful systems must be easily adaptable to new applications, and that such adaptation is much more difficult when computer code, even high-level computer code, must be changed, rather than tables. This is not to imply that we regard modification of a table whose size is on the order of a grammar as trivial; quite the contrary. Nonetheless, we believe it is easier to change a grammar or

a semantic interpreter expressed in table form than it is to change a special parser or a special interpreter. In essence, we believe it should not be necessary for a computational linguistics project to describe operations beyond the last level of formal representation in order for an outsider to understand exactly how a system operates.

FOOTNOTES

This system was formerly called REQUEST.

The form of Figure 3 is, in fact, subject to another syntactic transformation prior to execution. Normally, foratleast needs to be executed once for each potential value of the setx variable. However, in the case where the quantificational range of foratleast 1 is a constant, repeated evaluation of the quantifier is quite inefficient. Instead, a special retrieval function called MAPFIELD, which can accept a list of arguments, replaces forms like those of Figure 3. In this example the replacement takes the form

(MAPFIELD 'x77 'JSTOR '(5043... ...00) '1976 ')

Although this transformation arises quite often in practice, it is sufficiently non-general that we have not augmented our inventory of logical forms by including MAPFIELD. Instead, we look on it as an implementation measure only.

References

Astrahan, M.M.; Blasgen, M.W.; Chamberlin, D.D.; Eswaran, K.P.; Gray, J.N.; Griffiths, P.P.; King, W.F.; Lorie, R.A.; McJones, J.; Mehl, J.W.; Putzolu, G.R.; Traiger, I.L.; Wade, B.W., Watson, V.(1976). System R: Relational Approach to Database Management. ACM Transactions on Database Systems, Vol. 1, No. 21 June, 1976, pp. 97-137.

Petrick Stanley R.(1977). Semantic Interpretation in the Request System. In in Cõmputational and Mathematical Lingustics, Proceedings of the Internamtional Conference on Computational Linguistics, Pisa, 27/VIII-1/IX 1973, pp. 585-610.

Plath, Warren J.(1973). Transformational Grammar and Transformational Parsing in the Request System. IBM Research Report RC 4396. Thomas J. Watson Research Center, Yorktown Heights, N.Y.

Plath, Warren J.(1974). String Transformations in the REQUEST System American Journal of Computational Linguistics, Microfiche 8.

Reiter, Raymond(1976). Query Optimization for Question-Answering Systems. In: COLING 76, Proceedings.

Robinson, Jane J.(1973). An Inverse Transformational Lexicon. In Natural Language Processing. Randall Rustin, ed. Algorithmics Press, Inc., New York, N.Y., 1973 pp. 43-60.

Woods, W.A.; Kaplan, R.M.; Nash-Webber, B.(1972). The Lunar Sciences Natural Language Information System: Final Report. BBN Report No. 2378. Bolt Beranek and Newman, Inc., Cambridge, Massachusetts. June 15, 1972.