THEORETICAL/TECHNICAL ISSUES IN NATURAL LANGUAGE ACCESS TO DATABASES

S. R. Petrick

IBM T.J. Watson Research Center

INTRODUCTION

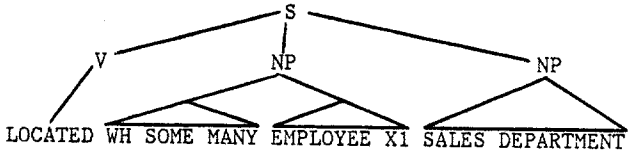
In responding to the guidelines established by the session chairman of this panel, three of the five topics he set forth will be discussed. These include aggregate functions and quantity questions, querying semantically complex fields, and multi-file queries. As we will make clear in the sequel, the transformational apparatus utilized in the TQA Question Answering System provides a principled basis for handling these and many other problems in natural language access to databases.

In addition to considering some subset of the chairman's five problems, each of the panelists was invited to propose and choose one issue of his/her own choosing. If time and space permitted, I would have chosen the subject of extensibility of natural language systems to new applications. In light of existing restrictions, however, I have chosen a more tractable problem to which I have given some attention and in whose treatment I am interested; this is the translation of quantified relational calculus expressions to a formal query language such as SQL.

AGGREGATE FUNCTIONS AND QUANTITY QUESTIONS

Questions such as "How many employees are in the sales department?" must be mapped into three radically different database query language expressions depending on how the database is set up. It may be appropriate to retrieve a pre-stored total number of employees from a NUMBER-OF-EMPLOYEES field of a DEPARTMENT file, or to count the number of records in an EMPLOYEE file that have the value SALES in the DEPARTMENT field, or, if departments are broken down into offices with which are associated the total numbers of employees employed therein, to total the values of the NUMBER-OF-EMPLOYEES field in all the records for offices in the sales department.

In the TQA System there are a number of different levels of representation of a given query. The grammar which assigns structure to a query has some core components which are essentially application-independent (e.g., the cyclic and post-cyclic transformations) and has other components that are application-dependent (e.g., portions of the lexicon and precyclic transformations). Surface structures are mapped by the application-independent post cyclic and cyclic transformations into a relatively deep structural level which is referred to as the underlying structure level. In this representation, sentence nodes are expanded into a verb followed by a sequence of noun phrases, and the representation of reference is facilitated by the use of logical variables X1, X2, ... The underlying structure corresponding to the previously cited example sentence would be something like the following (suppressing details):



Now, depending on feature information associated with the lexical items in the two NP's, application-specific precyclic transformations can be formulated to map this underlying structure into any of three query structures that directly reflect the three data structures and corresponding formal queries previously discussed. Rather than sketching query structures that could be produced for this example, let me be more specific by substituting the actual treatment of two similar sentences currently treated by the TQA System land-use application. These are the sentences:

(1) "How many parking lots are there in ward 1 block 2?"

(2) "How many parking spaces are there in ward 1 block 2?"

In the current data base, individual lots are identified as being parking lots by a land use code relation LUCF, which has attributes that include JACCN (parcel account number) and LUC (land use code). Parking lots have an LUC value of 460. Another relation, PARCFL, has attributes which include JACCN and JPRK (the number of parking spaces on a given parcel).

The underlying structures assigned to both these sentences are nearly identical, differing only in the lexical distinctions between "parking lot" and "parking space". The common structure is very much like that of the previously given tree structure except that PARKING_LOT or PARKING_SPACE (together with their associated features) replaces EMPLOYEE, and the second NP dominates the string "WARD 1 BLOCK 2". The feature + UNIT on a node that dominates PARKING_SPACE is not found in the corresponding structure involving PARKING_LOT, and this feature (together with a number of other structural prereq-

uisites) triggers a pair of precyclic transformations. The action of those two transformations is roughly indicated by the following sequence of bracketted terminal strings (the actual trees together with all their features would take up much more space):

```
(BD LOCATED
    ((WH SOME MANY) (PARKING_SPACE X3))
    ((WARD 1) (BLOCK 2)) BD)

                    TOTPUNIT
                       ->

(BD TOTAL
    ((WH SOME) (THING X46))
    (THE (X3
      (BD PARKING_SPACE
        X3
        ((WARD 1) (BLOCK 2)) BD))) BD)

                    LOTINS2
                       ->

(BD TOTAL
    ((WH SOME) (THING X46))
    (THE (X3
      (BD PARKING_SPACE
          X3
          (THE ((LOT X48)
            (BD LOCATED
                X48
                ((WARD 1) (BLOCK 2))
                BD))) BD))) BD)
```

Note that the lot insertion transformation LOTINS2 has produced structure of the type which is more directly assigned to the input query, "What is the total number of parking spaces in the lots which are located in ward 1 block 2?". This structure is then further transformed by a transformation LOCATION that replaces the abstract verb LOCATED by a verb (WBLOCK in this instance) which corresponds to an existing data base relation.

```
                    LOCATION
                       ->

(BD TOTAL
    ((WH SOME) (THING X46))
    (THE (X3
          (BD PARKING_SPACE
              X3
              (THE ((LOT X48)
                (BD WBLOCK
                    ((WARD 1) (BLOCK 2))
                    X48
                    BD))) BD))) BD)
```

The latter structure is mapped via the TQA Knuth attribute grammar formalism into the logical form:

```
(setx 'X46
  '(total X46
    (bagx 'X3
      '(setx 'X48
        '(and
          (RELATION 'PARCFL
            '(JPRK JACCN)
            '(X3 X48)
            '(= =) )
          (RELATION 'PARCFL
            '(WBLOCK JACCN
            '('100200 X48)
            '(= =)))))))
```

This logical form is in a set domain logical calculus to be discussed later in the paper. Roughly, it denotes the set of elements X46 such that X46 is the sum of the members of the bag (like a set, but with possible duplicate elements) of elements X3 such that a certain set is not empty, namely the set of elements X48 such that X48 is the account number (JACCN) of a parcel whose number of parking spaces (JPRK) is X3 and whose wardblock (WBLOCK) is 100200. The expression

```
(RELATION 'PARCFL
  '(JPRK JACCN)
  '(X3 X48)
  '(= =) )
```

in the above logical form denotes the proposition that the relation formed from the PARCFL relation by projecting over the attributes JPRK and JACCN contains the tuple (X3 X48). The logical form is straightforwardly translated by means of a LISP program whose details we will not concern ourselves with into the SQL query:

```
SELECT SUM(A.JPRK)
FROM PARCFL A
WHERE A.WBLOCK = '100200';
```

The other structure (for the sentence with PARKING_LOT) lacks the triggering feature + UNIT, and hence transformations TOTPUNIT and LOTINS2 do not apply; furthermore, the LOCATION transformation applies to the original instance of the verb LOCATED rather than the copy of LOCATED introduced by the lot insertion transformation LOTINS2 in the analysis of the previous sentence:

```
(BD LOCATED
    ((WH SOME MANY) ((PARKING_LOT 460) X3))
    ((WARD 1) (BLOCK 2)) BD)

                    LOCATION
                       ->
(BD WBLOCK
    ((WARD 1) (BLOCK 2))
    ((WH SOME MANY) ((PARKING_LOT 460) X3)) BD)
```

This structure is mapped via the Knuth attribute grammar into the logical form:

```
(setx 'X48
  '(quantity X48
    (setx 'X3
      '(and
        (RELATION 'PARCFL
          '(WBLOCK JACCN)
          '('100200 X3)
          '(= =) )
        (RELATION 'LUCF
          '(LUC JACCN)
          '('0460 X3)
          '(= =) ) ) ) ) )
```

and this logical form is translated to the SQL query:

```
SELECT COUNT(UNIQUE A.JACCN)
FROM PARCFL A, LUCF B
WHERE A.JACCN = B.JACCN
AND B.LUC = '0460'
AND A.WBLOCK = '100200' ;
```

The points to be made with respect to this treatment are that the information indicating differential, database-specific treatment can be encoded in lexical features, and that differential treatment itself can be implemented by means of precyclic transformations which are formally of the same type that the TQA system uses to relate underlying to surface structures. The features, such as + UNIT in our example, are principled enough to permit their specification by a data base administrator with the help of an on-line application customization program. (+ UNIT is also required in lexical items such as DWELLING UNITS and STORIES).

If the database organization had been different, simple lexical changes could have been made to trigger different sequences of transformations, resulting in structures and ultimately SQL expressions appropriate for that database organization. In this way, it would be easy to handle such database organizations as that in which the total number of parking lots and/or parking spaces is stored for each wardblock, and that in which such totals are stored for each splitblock which is included within a given wardblock.

QUERYING SEMANTICALLY COMPLEX FIELDS

In posing this problem, the session chairman pointed out that natural language query systems usually assume that the concepts represented by database fields will always be expressed in English by single words or fixed phrases. He cited as an example the query "Is John Jones a child of an alumnus?" where "child of an alumnus" is a fixed phrase expressing the binary relation with attributes APPLICANT (whose values are the names of applicants) and CHILD-OF-ALUMNUS (whose values are either T or F). He further noted that related queries such as "Is one of John Jones' parents an alumnus?" or "Did either parent of John Jones attend the college?)" require some different treatment.

The approach we have taken in TQA is, insofar as possible, to provide the necessary coverage to permit all the locutions that are natural in a given application. The formalism by which this is attempted is, once again, the transformational apparatus. Transformations often coalesce queries which have the same meaning but differ substantially in their surface forms into common underlying or query structures. There is, however, no requirement that this always be done, so such queries are sometimes mapped into logically equivalent rather than identical query structures. In either case, the transformational formalism provides a solid basis for assigning very deep semantic structures to a wide spectrum of surface sentence structures. The extent to which we have been successful in allowing broad coverage of logically equivalent alternative statements of a query is difficult to quantify, but we believe that we have done well relative to other efforts for two reasons: (1) We have made an effort to cover as many underlying relations and their surface realizations as possible in treating a given application, and (2) The transformational formalism we use is effective in providing the broad coverage which reflects all the allowable interactions between the syntactic phenomena treated by a particular grammar.

MULTI-FILE QUERIES

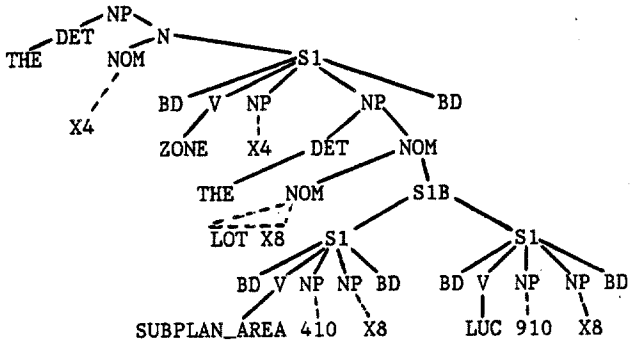
This problem deals with multi-file databases and the questions of which files are relevant to a given query and how they should be joined. This "problem" is one which is often raised, and which invariably reflects a quick-and-dirty approach to syntactic and semantic analysis. Within a framework such as that provided by the transformational apparatus in TQA, this problem simply doesn't arise. More accurately, it is a problem which doesn't arise if an adequate grammar is produced that assigns structure of the depth of the TQA System's query structures. This, of course, is no easy task, but it is one which is central to the transformational grammar-based approach, and its successful treatment does provide a principled basis for eliminating a number of potential difficulties such as this multi-file query problem.

To see why this is so, let us consider how, for a given query, relations are identified and joined in TQA. As we have already indicated, TQA underlying structures and query structures consist of sentence nodes which dominate a verb followed by a sequence of noun phrases. These simple structures are joined together to form a complete sentence structure through the use of additional phrase structure rules which indicate conjunction, relative clause-main clause connection, etc. Query structure verbs correspond, for the most part, to database relations, and the noun phrase arguments of those verbs correspond to attributes of their associated relations. Furthermore, query structures contain logical variables which serve the function of establishing reference, including identity of reference. Thus if the query structure assigned to a query identifies two (or more) relations which have attributes whose values are the same logical variable, we have an indication that it is those attributes over which the relations should be joined.

An example should make this clearer. Consider the query structure which TQA assigns to the sentence

"What is the zone of the vacant parcels in subplanning area 410?"

(We omit feature information and some structure which is irrelevant to the subsequent discussion in the structure below.)



This structure represents the set of elements X4 such that X4 is the zone of an element of the set of lots X8 such that the land use code (LUC) of X8 is 910 and the subplanning area (SUBPLAN_AREA) of X8 is 410. The structure is mapped in straightforward fashion by a Knuth attribute grammar translation procedure into the set domain relational calculus expression:

```
(setx 'X4
  '(setx 'X8
     '(and
       (RELATION 'ZONEF
         '(ZONE JACCN)
         '(X4 X8)
         '(= =) )
       (RELATION 'GEOBASE*
         '(SUBPLA JACCN)
         '('410 X8)
         '(= =) )
       (RELATION 'LUCF
         '(LUC JACCN)
         '('910 X8)
         '(= =))))))
```

Each deep (query structure) verb such as ZONE has associated with it (by means of a translation table entry) a relation, which is usually the projection of an existing data base relation. Thus instead of translating a portion of the above tree to (ZONE X4 X8), an expression which is true if X4 is the zone of the parcel whose account number is X8, the translation table is used to produce

```
(RELATION 'ZONEF
  '(ZONE JACCN)
  '(X4 X8)
  '(= =) )
```

which is true if the projection of the ZONEF relation over attributes ZONE and JACCN (account number) contains a tuple (X4 X8).

The conjunction of three relations with a common JACCN attribute value of X8 indicates that the three relations are to be joined over the attribute JACCN.

There is, however, one complication in translating the relational calculus expression above into a formal query language such as SQL. The relations ZONEF and LUCF are existing database relations, but there is no relation GEOBASE* in the database, giving the subplanning area of specific parcels. Instead, the PARCEL relation gives the splitblock (SBLOCK) of a given parcel (JACCN) and the GEOBASE relation gives the subplanning area (SUBPLA) of all the parcels within a given splitblock (SBLOCK).

There are at least three solutions to the problem of bridging the gap between relational calculus expressions such as this and appropriate formal query language expressions. These are:

(1) Write a precyclic database-specific splitblock insertion transformation which assigns query structure corresponding to the query, "What are the zones of the vacant parcels which are located in splitblocks in subplanning area 410?"

(2) Store information that permits replacing expressions involving virtual relations such as

```
(RELATION 'GEOBASE*
  '(SUBPLA JACCN)
  '('0410 X8)
  '(= =) )
```

by existentially quantified expressions involving only real database relations such as:

```
(setx 'X111
  (and
    (RELATION 'PARCFL
      '(SBLOCK JACCN)
      '(X111 X8)
      '(= =) )
    (RELATION 'GEOBASE
      '(SUBPLA SBLOCK)
      '('410 X111)
      '(= =) ) )
```

(3) Make the data base administrator (DBA) responsible for providing a formal query language definition of the virtual relations produced. In this case that would take the form of defining GEOBASE* as the appropriate join of projections over GEOBASE and PARCFL.

All three solutions have been implemented in the TQA System and used in specific cases as seems appropriate. For a database system with the definitional facilities available in SQL, solution (3) is particularly attractive because it is the type of activity with which data base administrators are familiar. Solutions (1) and (2) were also implemented at various times for examples such as the one in question, leading to the following SQL query:

```
SELECT UNIQUE A.ZONE, A.JACCN
FROM ZONEF A, GEOBASE B, PARCFL C, LUCF D
WHERE A.JACCN = C.JACCN
AND C.JACCN = D.JACCN
AND B.SBLOCK = C.SBLOCK
AND D.LUC = '0910'
AND B.SUBPLA = '4100';
```

(We note for the careful reader that '0910' and '4100' are not misprints, but the discussion of how such normalization can be automatically achieved from DBA declarations is outside the scope of the present paper.)

TRANSLATING QUANTIFIED RELATIONAL CALCULUS EXPRESSIONS TO FORMAL QUERY LANGUAGE EQUIVALENTS

In this section we consider a problem of our own choosing. In most of the existing relational calculus formalisms, use is made of logical variables and some type of universal and existential quantifiers. Early versions of TQA were typical in this respect. The version of TQA which was tested in the White Plains experiment, for example, made use of quantifiers FORATLEAST and FORALL whose nature is best explained by an example. The logical form assigned to the previously considered sentence was, at one time:

```
(setx 'X4
  '(foratleast 1 'X112
     (setx 'X8
       '(and
          (RELATION 'GEOBASE*
            '(SUBPLA JACCN)
            '('410 X8)
            '(= =) )
          (RELATION 'LUCF
            '(LUC JACCN)
            '('910 X8)
            '(= =) ) ) )
     (RELATION 'ZONEF
       '(ZONE JACCN)
       '(X4 X112)
       '(= =) ) ) )
```

This logical form denotes (roughly) the set of zones X4 such that for at least one element X112 of the set of parcels X8 which are in subplanning area 410 and have a land use code of 910, parcel X112 is in zone X4. In simple examples such as this, where only existential quantification of logical forms is involved, there is no problem in translating to a formal query language such as SQL. However, when various combinations of existential and universal quantification are involved in a logical form, the corresponding quantification-indicating constructs to be used in the formal query language translation of that logical form is not at all obvious. An examination of the literature indicates that the arguments used in establishing the completeness of query languages offer little or no guidance as to the construction of a practical translator from relational calculus to a formal query language such as SQL. Hence, the approach used in translating TQA logical forms to corresponding SQL expressions will be discussed, in the expectation of eliciting explanations of how the translation of quantification is handled in other systems.

We begin by observing that a logical form

```
(foratleast 1 X1
  (setx X2 (f X2))
  (g X1))
```

(which denotes the proposition that for at least one X1 which belongs to the set of elements X2 such that f(X2) is true, g(X1) is true) is equivalent to the requirement of the non-emptiness of the set

(1) `(setx 'X1 '(and (f X1) (g X1)))`

Similarly,

```
(forall X1
  (setx X2 (f X2))
  (g X1))
```

(which denotes the proposition that for all X1 in the set of elements X2 such that f(X2) is true, g(X1) is true), is equivalent to a requirement of the emptiness of the set

(2) `(setx 'X1 '(and (f X1) (not (g X1))))`

Conversion of expressions with universal and existential quantifiers is then possible to expressions involving only set notation and a predicate involving the emptiness of a set. The latter type of expressions are called set domain relational calculus expressions.

Fortunately, SQL provides operators EXISTS and NOT EXISTS which take as their argument an SQL SELECT expression, the type of expression into which logical forms of the type (setx 'X1 ...) are translated. A recursive call to the basic logical form-to-SQL translation facility then suffices to supply the SQL argument of EXISTS or NOT EXISTS.

It is worth noting that, under certain circumstances which we will not explore here, the "(setx X2" portion of an embedded expression (setx 'X2 (f X2)) can be pulled forward, creating a prefix-normal-form-like expression of the type (setx 'X1 (setx 'X2 ...)), and the logical variables that can be pulled all the way forward correspond to information implicitly requested in English queries. The values which satisfy these variables should also be printed to satisfy users' implicit requests for information. For example, in our previously considered query

"What are the zones of the vacant parcels in subplanning area 410?"

one probably wants the parcels identified in addition to their zones. Translation to the form of set domain relational calculus used in TQA then provides a basis for either taking the initiative in automatically printing these implicitly requested values or for engaging in a dialog with the user to determine whether they should be printed.

As a final example of this method of translating quantified logical forms, consider the sentence

"What gas stations are in a ward in which there is no drug store?"

The logical form initially assigned by TQA to this sentence is

```
(setx 'X2
  '(and
    (RELATION 'LUCF
      '(LUC JACCN)
      '('0553 X2)
      '(= =) )
    (foratleast 1 'X81
      (setx 'X7
        (forall 'X80
          (setx 'X13
            (RELATION 'LUCF
              '(LUC JACCN)
              '('0591 X13)
              '(= =) ) )
            (RELATION 'PARCFL
              '(WARD JACCN)
              '(X7 X80)
              '(¬= =) ) ) )
       (RELATION 'PARCFL
         '(WARD JACCN)
         '(X81 X2)
         '(= =) ) ) ) )
```

which is translated to the set domain logical form:

```
(setx 'X2
  '(setx 'X7
    '(and
      (RELATION 'PARCFL
        '(WARD JACCN)
        '(X7 X2)
        '(= =) )
      (not
        (setx 'X13
          '(and
            (RELATION 'PARCFL
              '(WARD JACCN)
              '(X7 X13)
              '(= =) )
            (RELATION 'LUCF
              '(LUC JACCN)
              '('0591 X13)
              '(= =) ) ) ) )
      (RELATION 'LUCF
        '(LUC JACCN)
        '('0553 X2)
        '(= =) ) ) ) )
```

The latter form translates easily into the SQL expression:

```
SELECT UNIQUE A.JACCN, A.WARD
FROM PARCFL A, LUCF B
WHERE A.JACCN = B.JACCN
AND B.LUC = '0553'
AND NOT EXISTS
(SELECT UNIQUE C.JACCN
FROM PARCFL C, LUCF D
WHERE C.WARD = A.WARD
AND C.JACCN = D.JACCN
AND D.LUC = '0591');
```

REFERENCES

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

Damerau, F.J., "Advantages of a Transformational Grammar for Question Answering," Proc. 5th IJCAI, Vol. 1, 1977, p. 192.

Damerau, F.J., "Operating Statistics for The Transformational Question Answering System," American Journal of Computational Linguistics, Vol. 7, No. 1, January-March 1981, pp. 30-42.

Petrick, S. R., "Semantic Interpretation in the Request System," in Computational and Mathematical Linguistics, Proceedings of the International Conference on Computational Linguistics, Pisa, 27/VIII-1/IX 1973, pp. 585-610.

Petrick, S.R., "On Natural Language Based Computer Systems," IBM Journal of Research and Development, Vol. 20, No. 4, July 1976, pp. 314-325.

Petrick, S.R. "Field Testing the Transformational Question Answering (TQA) System," Proc. 19th Ann. Mtg. of the ACL, June 1981, pp. 35-36.

Plath, W.J., "REQUEST: A Natural Language Question-Answering System," IBM Journal of Research and Development, Vol. 20, No. 4, July 1976, pp. 326-335.