ISSUES IN NATURAL LANGUAGE ACCESS TO DATABASES FROM A LOGIC PROGRAMMING PERSPECTIVE

David H D Warren
Artificial Intelligence Center
SRI International, Menlo Park, CA 94025, USA

I INTRODUCTION

I shall discuss issues in natural language (NL) access to databases in the light of an experimental NL question-answering system, Chat, which I wrote with Fernando Pereira at Edinburgh University, and which is described more fully elsewhere [8] [6] [5]. Our approach was strongly influenced by the work of Alain Colmerauer [2] and Veronica Dahl [3] at Marseille University.

Chat processes a NL question in three main stages:

translation planning execution
English ----> logic ----> Prolog ----> answer

corresponding roughly to: "What does the question mean?", "How shall I answer it?", "What is the answer?". The meaning of a NL question, and the database of information about the application domain, are both represented as statements in an extension of a subset of first-order logic, which we call "definite closed world" (DCW) logic. This logic is a subset of first-order logic, in that it admits only "definite" statements; uncertain information ("Either this or that") is not allowed. DCW logic extends first-order logic, in that it provides constructions to support the "closed world" assumption, that everything not known to be true is false.

Why does Chat use this curious logic as a meaning representation language? The main reason is that it can be implemented very efficiently. In fact, DCW logic forms the basis of a general purpose programming language, Prolog [9] [1], due to Colmerauer, which has had a wide variety of applications. Prolog can be viewed either as an extension of pure Lisp, or as an extension of a relational database query language. Moreover, the efficiency of the DEC-10 Prolog implementation is comparable both with compiled Lisp [9] and with current relational database systems [6] (for databases within virtual memory).

Chat's second main stage, "planning", is responsible for transforming the logical form of the NL query into efficient Prolog [6]. This step is analogous to "query optimisation" in a relational database system. The resulting Prolog form is directly executed to yield the answer to the original question. On Chat's domain of world geography, most questions within the English subset are answered in well under one second, including queries which involve taking joins between relations having of the order of a thousand tuples.

A disadvantage of much current work on NL access to databases is that the work is restricted to providing access to databases, whereas users would appreciate NL interfaces to computer systems in general. Moreover, the attempt to provide a NL "front-end" to databases is surely putting the cart before the horse. What one should really do is to investigate what "back-end" is needed to support NL interfaces to computers, without being constrained by the limitations of current database management systems.

I would argue that the "logic programming" approach taken in Chat is the right way to avoid these drawbacks of current work in NL access to databases. Most work which attempts to deal precisely with the meaning of NL sentences uses some system of logic as an intermediate meaning representation language. Logic programming is concerned with turning such systems of logic into practical computational formalisms. The outcome of this "top-down" approach, as realised in the language Prolog, has a great deal in common with the relational approach to databases, which can be seen as the result of a "bottom-up" effort to make database languages more like natural language. However Prolog is much more general than relational database formalisms, in that it permits data to be defined by general rules having the power of a fully general programming language. The logic programming approach therefore allows one to interface NL to general programs as well as to databases.

Current Prolog systems, because they were designed with programming not databases in mind, are not capable of accommodating really large databases. However there seems to be no technical obstacle to building a Prolog system that is fully comparable with current relational database management systems, while retaining Prolog's generality and efficiency as a programming language. Indeed, I expect such a system to be developed in the near future, especially now that

Prolog has been chosen as the kernel language for Japan's "Fifth Generation" computer project [4].

II SPECIFIC ISSUES

A. Aggregate Functions and Quantity Questions

To cater for aggregate and quantity determiners, such as plural "the", "two", "how many", etc., DCW logic extends first-order logic by allowing predications of the form:

```
setof(X,P,S)
```

to be read as "the set of Xs such that P is provable is S" [7]. An efficient implementation of 'setof' is provided in DEC-10 Prolog and used in Chat. Sets are actually represented as ordered lists without duplicate elements. Something along the lines of 'setof' seems very necessary, as a first step at least.

The question of how to treat explicitly stored aggregate information, such as "number of employees" in a department, is a special case of the general issue of storing and accessing non-primitive information, to be discussed below in section D.

B. Time and Tense

The problem of providing a common framework for time instants and time intervals is not one that I have looked into very far, but it would seem to be primarily a database rather than a linguistic issue, and to highlight the limitations of traditional databases, where all facts have to be stored explicitly. Queries concerning time instants and intervals will generally need to be answered by calculation rather than by simple retrieval. A common framework for both calculation and retrieval is precisely what the logic programming approach provides. For example, the predication:

```
sailed(kennedy,july82,D)
```

occurring in a query might invoke a Prolog procedure 'sailed' to calculate the distance D travelled, rather than cause a simple data look-up.

C. Quantifying into Questions

Quantifying into questions is an issue which was an important concern in Chat, and one for which I feel we produced a reasonably adequate solution. The question "Who manages every department?" would be translated into the following logical form:

```
answer(M) <= \+ exists(D, department(D) &
                          \+ manages(M,D))
```

where '\+' is to be read as "it is not known that", i.e. the logical form reads "M is an answer if there is no known department that M does not manage". The question "Who manages each department?", on the other hand, would translate into:

```
answer(D-M) <= department(D) & manages(M,D)
```

generating answers which would be pairs of the form:

```
accounts - andrews ;
sales - smith ; etc.
```

The two different logical forms result from the different treatments accorded to "each" and "every" by Chat's determiner scoping algorithm [8] [5].

D. Querying Semantically Complex Fields

My general feeling here is that one should not struggle too hard to bend one's NL interface to fit an existing database. Rather the database should be designed to meet the needs of NL access. If the database does not easily support the kind of NL queries the user wants to ask, it is probably not a well-designed database. In general it seems best to design a database so that only primitive facts are stored explicitly, others being derived by general rules, and also to avoid storing redundant information.

However this general philosophy may not be practicable in all cases. Suppose, indeed, that 'childofalumnus' is stored as primitive information. Now the logical form for "Is John Jones a child of an alumnus?" would be:

```
answer(yes) <=
   childof(X,johnjones) & alumnus(X)
```

What we seem to need to do is to recognise that in this particular case a simplification is possible using the following definition:

```
childofalumnus(X) <=>
   exists(Y, childof(Y,X) & alumnus(Y))
```

giving the derived query:

```
answer(yes) <= childofalumnus(johnjones)
```

However the logical form:

```
answer(X) <=
   childof(X,johnjones) & alumnus(X)
```

corresponding to "Of which alumnus is John Jones a child?" would not be susceptible to simplification, and the answer to the query would have to be "Don't know".

E. Multi-File Queries

At the root of the difficulties raised here is the question of what to do when the concepts used in the NL query do not directly correspond to what is stored in the database. With the logic programming approach taken in Chat, there is a simple solution. The database is augmented with general rules which define the NL concepts in terms of the explicitly stored data. For example, the rule:

```
lengthof(S,L) <=
   classof(S,C) & classlengthof(C,L).
```

says that the length of a ship is the length of that ship's class. These rules get invoked while a query is being executed, and may be considered to extend the database with "virtual files". Often a better approach would be to apply these rules to preprocess the query in advance of actual execution. In any event, there seems to be no need to treat joins as implicit, as systems such as Ladder have done. Joins, which are equivalent to conjunctions in a logical form, should always be expressed explicitly, either in the original query, or in other domain-dependent rules which help to support the NL interface.

III A FURTHER ISSUE - SEMANTICS OF PLURAL "THE"

A difficulty we experienced in developing Chat, which I would propose as one of the most pressing problems in NL access to databases, is to define an adequate theoretical and computational semantics for plural noun phrases, especially those with the definite article "the". It is a pressing problem because clearly even the most minimal subset of NL suitable for querying a database must include plural "the". The problem has two aspects:

(1) to define a precise semantics that is strictly correct in all cases;

(2) to implement this semantics in an efficient way, giving results comparable to what could be achieved if a formal database query language were used in place of NL.

As a first approximation, Chat treats plural definite noun phrases as introducing sets, formalised using the 'setof' construct mentioned earlier. Thus the translation of "the European countries" would be S where:

```
setof(C,european(C) & country(C),S).
```

The main drawback of this approach is that it leaves open the question of how predicates applied to sets relate to those same predicates applied to individuals. Thus the question "Do the European countries border the Atlantic?" gets as part of its translation:

```
borders(S,atlantic)
```

where S is the set of European countries. Should this predication be considered true if all European countries border the Atlantic, or if just some of them do? Or does it mean something else, as in "Are the European countries allies?"?

At the moment, Chat makes the default assumption that, in the absence of other information, a predicate is "distributive", i.e. a predication over a set is true if and only if it is true of each element. So the question above is treated as meaning "Does every European country border the Atlantic?". And "Do the European countries trade with the Caribbean countries?" would be interpreted as "Does each European country trade with each Caribbean country?".

Chat only makes this default assumption in the course of query execution, which may well be very inefficient. If the 'setof' can effectively be dispensed with, producing a simpler logical form, one would like to do this at an earlier stage and take advantage of optimisations applicable to the simpler logical form.

A further complication is illustrated by a question such as "Who are the children of the employees?". A reasonable answer to this question would be a table of employees with their children, which is what Chat in fact produces. If one were to use the more simple-minded approximations discussed so far, the answer would be simply a set of children, which would be empty (!) if the 'childof' predicate were treated as distributive.

In general, therefore, Chat treats nested definite noun phrases as introducing "indexed sets", although the treatment is arguably somewhat ad hoc. A phrase like "the children of the employees" translates into S where:

```
setof(E-CC,employee(E) &
          setof(C,childof(E,C),CC),S).
```

If the indexed set occurs, not in the context of a question, but as an argument to another predicate, there is the further complication of defining the semantics of predicates over indexed sets. Consider, for example, "Are the major cities of the Scandinavian countries linked by rail?". In cases involving aggregate operators such as "total" and "average", an indexed set is clearly needed, and Chat handles these cases correctly. Consider, for example, "What is the average of the salaries of the part-time employees?". One cannot simply average over a set of salaries, since several employees may have the same salary; an indexed set ensures that each employee's salary is counted separately.

To summarise the overall problem, then, can one find a coherent semantics for plural "the" that is intuitively correct, and that is compatible with efficient database access?

REFERENCES

1. Clocksin W F and Mellish C S. Programming in Prolog. Springer-Verlag, 1981.

2. Colmerauer A. Un sous-ensemble interessant du francais. RAIRO 13, 4 (1979), pp. 309-336. [Presented as "An interesting natural language subset" at the Workshop on Logic and Databases, Toulouse, 1977].

3. Dahl V. Translating Spanish into logic through logic. AJCL 7, 3 (Sep 1981), pp. 149-164.

4. Fuchi K. Aiming for knowledge information processing systems. Intl. Conf. on Fifth Generation Computer Systems, Tokyo, Oct 1981, pp. 101-114.

5. Pereira F C N. Logic for natural language analysis. PhD thesis, University of Edinburgh, 1982.

6. Warren D H D. Efficient processing of interactive relational database queries expressed in logic. Seventh Conf. on Very Large Data Bases, Cannes, France, Sep 1981, pp. 272-281.

7. Warren D H D. Higher-order extensions to Prolog - are they needed? Tenth Machine Intelligence Workshop, Cleveland, Ohio, Nov 1981.

8. Warren D H D and Pereira F C N. An efficient easily adaptable system for interpreting natural language queries. Research Paper 156, Dept. of Artificial Intelligence, University of Edinburgh, Feb 1981. [Submitted to AJCL].

9. Warren D H D, Pereira L M and Pereira F C N. Prolog - the language and its implementation compared with Lisp. ACM Symposium on AI and Programming Languages, Rochester, New York, Aug 1977, pp. 109-115.