An Efficient Easily Adaptable System for Interpreting Natural Language Queries[1]

David H. D. Warren
and
Fernando C. N. Pereira

Artificial Intelligence Center
SRI International
333 Ravenswood Avenue
Menlo Park, CA 94025

This paper gives an overall account of a prototype natural language question answering system, called Chat-80. Chat-80 has been designed to be both efficient and easily adaptable to a variety of applications. The system is implemented entirely in Prolog, a programming language based on logic. With the aid of a logic-based grammar formalism called extraposition grammars, Chat-80 translates English questions into the Prolog subset of logic. The resulting logical expression is then transformed by a planning algorithm into efficient Prolog, cf. "query optimisation" in a relational database. Finally, the Prolog form is executed to yield the answer. On a domain of world geography, most questions within the English subset are answered in well under one second, including relatively complex queries.

1. Introduction

This paper describes the results of a three-year research project carried out by the two of us. The project was directed towards the goal of providing practical computer systems that will answer questions expressed in precisely defined subsets of natural language. The results of our work are incorporated in a running prototype system, called "Chat-80".

Two issues have particularly influenced the approach we have taken, namely efficiency and portability. Given the practical objective, we wanted to achieve rapid, interactive question answering, and we wanted the techniques to be easily adaptable to a variety of applications, with as much of the implementation code as possible being application independent (cf. Konolige 1979).

There has been no intention to try to handle unrestricted natural language. Given the current state of the art, we accept that users of a practical natural language question answering system will have to learn how to use a restricted natural language subset relevant to the particular application. The important issue is whether they will find this more convenient than a more formal query language. We believe that, for many purposes, a suitable natural language subset will be much preferred, on grounds of conciseness and ease of typing alone (see the examples in Appendix III, for instance). It is fair to say that our objective is to so constrain natural language that it becomes a formal, but user-friendly, query language.

The starting point for our work was a question answering system for a small subset of Spanish implemented by Veronica Dahl 1981,1979, following the approach advocated by Colmerauer 1978. We were particularly attracted to Colmerauer's approach for the clear insight it gives into some of the essential problems involved in constructing a practical natural language answering system, especially the problem of correctly interpreting determiners. In addition, it proved very easy to adapt Dahl's program to English (and to a different domain). This was due in large part to the fact that the system is implemented in Prolog (Roussel 1975; Warren, Pereira, and Pereira 1977), a programming language based on first-order logic. To be more specific, the system is largely made

[1] This work was carried out in the Department of Artificial Intelligence, University of Edinburgh, Scotland.

0362-613X/82/030110-13$03.00

up of rules of the metamorphosis grammar (MG) formalism (Colmerauer 1978), which map directly into the Prolog subset of logic. MGs are an extension of a simpler formalism to which we have given the name definite clause grammars (DCGs) (Pereira and Warren 1980).

We called the adapted program "Chat". When running compiled under DEC-10 Prolog (Warren 1979; Pereira, Pereira, and Warren 1978), Chat's speed at analysing English sentences proved very satisfactory (under a tenth of a second of CPU time per sentence). Therefore this seemed a promising approach from the point of view of both efficiency and portability. However, we found that Chat has a number of shortcomings, the most serious of which is that the process of *answering* a question, once it has been analysed, is much too inefficient for any significant application.

We have therefore written an entirely new program, Chat-80, which, like the original Chat, is implemented entirely in Prolog. The main ways in which Chat-80 differs from its forerunner are:

- the semantics given to determiners, and the rules for determining their scopes,
- the process of planning and executing a query,
- the rather wider coverage of English,
- the use of *extraposition grammars* (XGs) in place of MGs to handle certain "transformational" aspects of English,
- the way the natural language analysis is performed in three separate phases rather than one.

In order to test the approach on a nontrivial domain, Chat-80 includes a database of facts about world geography, and a small vocabulary of English words sufficient for querying the database. This domain has the advantage, for demonstration purposes, that the facts in the database are generally common knowledge, so it is easier to appreciate what is entailed in answering different queries. The database contains basic facts about the world's countries (over 150 of them), oceans, major seas, major rivers, and major cities. The largest relation, 'borders', represents all pairs of countries, oceans, or major seas that are adjacent, and contains therefore over 850 tuples. It should be emphasised that the database is itself implemented as ordinary Prolog; it therefore resides within the normal DEC-10 virtual memory.

Chat-80 processes a question in three main stages:

English
↓ translation
logic
↓ planning
Prolog
↓ execution
answer

corresponding roughly to: "What does the question mean?", "How shall I answer it?", "What is the answer?". The planning and execution stages are discussed in detail in a companion paper (Warren 1981), so here we will concentrate on the translation stage, which is responsible for the natural language analysis. We first describe the way we represent the "meaning" of an English sentence as a logical expression, and then outline how the translation process is formalised, in logic, as a practical Prolog program. Finally, we briefly explain how the logical form is transformed into a Prolog program by the planning phase and how it is then executed.

2. A Simplified Semantics for a Basic English Subset

To answer a question, one first has to understand what it means. If question answering is to be done by computer, there needs to be some precise representation for the result of this first process, what one might call the "meaning" of the question, and also some precise way of relating the question to its meaning. It is further necessary that the meaning representation can in some way be given a precise interpretation (or "semantics") so that, in the case of questions for instance, one then knows precisely what is or is not a correct answer to the questions. It is surprising that, even for such basic features of English as the common determiners, and even within the different language camps represented by linguistics, philosophy, and artificial intelligence, there is as yet no established solution to this problem.

One way to try to articulate the meaning of a sentence is to paraphrase it into some standard, unambiguous form of English. Since these standard forms are likely to be stilted and long-winded, it will probably be convenient to represent them in a more concise notation. This is essentially the logician's approach, and we will call such meaning representations "logical forms".

Chat-80 represents the meaning of a question by a logical form. The approach is a development of that proposed by Colmerauer 1978 and implemented by Dahl 1981,1979.

Words approximating to the status of "proper nouns" are represented by logical *constants,* for example:

France	france
the Soviet Union	soviet__union
wine	wine

Most verbs, nouns, and adjectives (together with any associated prepositions) are represented by *predicates,* taking one or more *arguments* (which for our purposes are called constants). A predicate with its arguments is called a *predication* (or sometimes a *goal*). Examples are:

France **exports** wine **to** Britain.
exports(france,wine,britain)
France is a **country.** country(france)
Paris is the **capital of** France. capital(france,paris)
France is **European.** european(france)

Many kinds of more complex phrases or sentences can be represented by *conjunctions* of predications, for example:

Paris is a European city.
european(paris) & city(paris).
France is a country that borders on Spain.
country(france) & borders(france,spain)

The second of these conjunctions, for example, can be read more literally as a shorthand for "France is a country and France borders Spain", which is just a paraphrase of the original sentence.

The most important class of words not covered so far are the determiners – words such as "a", "the", "every". Determiners play a particularly important role in questions, since they enable relatively complex requests for information to be expressed very concisely.

In Colmerauer's approach, each determiner is represented by what he called a "three-branched quantifier" (3BQ). For example, the logical form that would be ascribed to the sentence "The boy sleeps" would be:

the(X,boy(X),sleeps(X))

where the determiner "the" is represented by the 3BQ 'the(__,__,__)'. 3BQs are very close to the meaning representation for determiners used by Woods in the LUNAR system (1977). Colmerauer gave 3BQs a precise semantics in terms of certain operations over sets. However, we have found that this way of interpreting 3BQs fails to give a correct model of natural language in certain cases, and, worse still, it does not appear to lend itself to efficient implementation. Certainly this is the case with Dahl's program which, while being very efficient at *understanding* questions, is hopelessly inefficient at *answering* questions where the domain is of any significant size.

In Chat-80, we have addressed these problems by instead translating 3BQs directly into standard first-order logic, or rather into something as close to first-order logic as is practical. To be more exact, we translate into the Prolog subset of logic, which we have augmented with certain "meta-logical" extensions. (A similar approach has been taken by McCord 1982, who has independently been developing a system influenced by the Colmerauer/Dahl approach.) The subset of first-order logic we have chosen has the great advantage as a meaning representation that it already has a well understood semantics which is amenable to very efficient implementation. Thus the meaning representation can in principle be directly executed as a Prolog program (whereas 3BQs require a special-purpose interpreter). Furthermore, the first-order logic formulation lends itself to transformations which can greatly improve the efficiency of execution. This corresponds to what is known as "query optimisation" in relational database circles, and will be discussed in more detail later.

Our translation into logic completely ignores the presuppositions which can be implicit in a natural language question, and which Colmerauer's 3BQ semantics took pains to reflect. In most situations that we are concerned with, this simplification seems relatively harmless or even beneficial. For example, Chat-80 ignores the presupposition that there is only one answer to the question "Which ocean borders the United States?", and simply gives all three answers without further comment.

The way we translate determiners into logic is shown in Figure 1. Each determiner is translated into a *quantification,* which introduces some logic *variable* (X, N, etc.), and which links two predications involving that variable, called the *range* and *scope*, indicated by *R* and *S*.

The determiners "a", "the" (in a singular context), and "some" (whether in a singular or plural context) are all translated in exactly the same way, by a standard first-order logic *existential quantification.* Read 'exists(*X,P*)' as "there is some *X* such that *P*". The same translation is also normally used for the "empty plural" determiner, for example:

Zambia exports minerals.
exists(X,mineral(X) & exports(zambia,X)).

The determiner "no" is translated with the aid of a kind of *negation.* Read '\+**P**' as "it cannot be shown that **P**". Note that this is not the standard negation of first-order logic, which is outside the Prolog subset. Standard negation is problematic to implement and seems inappropriate for many purposes. Instead Prolog systems provide (or can easily be extended with) a partial implementation of *nonprovability.* The predication '\+**P**' is considered true if **P** is not deducible from the facts and rules which define a particular application domain. Note that nonprovability is also used in the translation of the determiners "every" and "all".

The determiner "the" in a plural context presents a number of problems, for which we do not yet feel we have a completely adequate solution. In general, we consider a plural definite noun phrase to denote a set. To cater for this, we have proposed and implemented a "meta-logical" extension to Prolog, which has been described in detail elsewhere (Warren, 1982; Byrd, Pereira, and Warren 1980). The extension allows predications of the form:

setof*(X,P,S)*

a, some, the[singular]	exists(X,**R** & S)
no	\+exists(X,**R** & S)
every, all	\+exists(X,**R** & \+S)
the[plural]	exists(X,setof(X,**R**,X) & S)
one, two, ... numeral(N)	numberof(X,**R** & S,N)
which, what	answer(X) <= **R** & **S**
how many	answer(N) <= numberof(X,**R** & S,N)

Some birds migrate.
  exists(X,bird(X) & migrates(X)).
The population of Britain exceeds 50 million.
  exists(X,population(britain,X) & X > 50000000).
There are **no** rivers in Antarctica.
  \+exists(X,river(X) & in(X,antarctica)).
Man inhabits **every** continent.
  \+exists(X,continent(X) & \+inhabits(man,X)).
Jupiter is the largest of **the** planets.
  exists(X,setof(X,planet(X),X) & largest(X,jupiter)).
The Rhine flows through **three** countries.
  numberof(X,country(X) & flows_through(rhine,X),3).
Which birds migrate?
  answer(X) <= bird(X) & migrates(X).
How many countries export oil?
  answer(N) <= numberof(X,country(X) & exports(X,oil),N).

Figure 1. Translations of determiners.

to be read as "the set of **X**s such that **P** is provable is S, where S is nonempty". Note that this construct behaves like a quantification in that it introduces a variable (or, more generally, a collection of variables) **X** which is *purely local* to **P**. Thus it is possible, as in the translation of the plural definite article given in Figure 1, to use the same variable name for a set and its "typical" element.

Our translation entails that certain predicates can take sets as arguments. At present, we leave it up to the definition of each individual predicate to draw any necessary correspondence between predications over sets and predications over individuals, since it is hard to fix a general rule. To see what the problem is, compare the sentences:

The boys like the girls.
The boys are married to the girls.

In certain contexts, such as:

What are the ages of the boys?

plural definite noun phrases are translated somewhat differently, as *indexed* sets. More on this later.

For convenience in translating the numerals "one", "two", "three", etc., we allow predications of the form:

numberof*(X,P,N)*

meaning "the number of **X**s such that **P** is provable is **N**". This predicate can easily be defined in terms of 'setof' by the logical *implication*:

numberof(X,P,N) <= setof(X,P,S) & sizeof(S,N)

Read '**P** <= **Q**' as "**P** if **Q**". The predicate 'sizeof' just gives the number of elements in a set.

Notice that implications are also used in the translation of questions. Read 'answer(**X**) <= **P**' as "**X** is an answer if **P**" or "I want to know **X** if **P** (is true)". Variables in the 'answer' predicate are not explicitly quantified. Such *free* variables are interpreted, following normal conventions in logic, as though they were *universally* quantified; that is, one may prefix one's reading of the entire logical form with phrases "For any X, for any Y," etc.

We have now seen how some common types of English words are translated into bits of logical structure. It remains to discuss how the bits fit together to produce the complete logical form for a whole sentence. For example, a sentence within the scope of our subset is:

Which European country exports no arms to countries in Africa?

How is it that the bits of structure corresponding to each individual work in this sentence fit together to produce the following logical form:

```
answer(C) <= european(C) & country(C) &
    \+exists(X, arm(X) &
        exists(C1, country(C1) & in(C1,africa) &
            exports(C,X,C1) ) )
```

(One should realize that this is just a shorthand for "For any C, C is an answer if C is European and C is a country and it cannot be shown that there is some X such that X is an armament and there is some C1 such that C1 is a country and C1 is in Africa and C exports X to C1".) Hopefully, the reader should already have a fair intuitive idea of what the "assembly process" must do. Basically, there are two main problems.

First, how are the appropriate variables or constants chosen to fill each predicate argument position? For example, why is the last argument of 'exports' the same variable C1 that appears as arguments to 'country' and 'in'? This "slot filling" is determined mainly by the grammatical structure of the sentence, but the correct attachment of prepositional phrases often cannot be determined on purely syntactic grounds.

Second, what determines the scope of the different quantifications? For example, why is the existential quantification corresponding to the empty determiner governing "countries in Africa" nested inside the quantification corresponding to "no"? This "scoping" is only weakly influenced by grammatical structure. Indeed, it is possible to give an alternative but much less likely reading to our example sentence, which corresponds to the same grammatical structure, but to a different logical form where the nesting of the two inner quantifications is reversed. The more likely reading is that no arms are exported to *any* countries in Africa; the less likely reading is that no arms are exported to *certain* countries in Africa.

The details of our mapping from English into logic are made precise in the program text corresponding to the first phase of Chat-80. This text itself consists of clauses of the Prolog subset of logic. It serves both as a clear formal definition of our mapping and, when executed by Prolog, as an efficient implementation of that mapping, which performs the translation of English questions into their logical forms. A fuller discussion of this implementation will be given in the next section, and further details can be found in Pereira 1982.

3. Translating English into Logic, in Logic

The translation from English sentence to logical form can be seen as involving three main functions – the slot filling and scope determination mentioned in the last section, and in addition the *parsing* function, which determines the grammatical structure of a sentence. In Chat-80, in contrast to Dahl's system, these three functions are separated into distinct program modules, which operate in sequence:

$$\begin{tikzcd}
\text{English question} \arrow[d, "\text{parsing}"] \\
\text{phrase structure} \arrow[d, "\text{interpretation}"] \\
\text{semantic structure} \arrow[d, "\text{scoping}"] \\
\text{logical form}
\end{tikzcd}$$

The parsing module consists of a set of grammar rules of the XG formalism, which the Prolog system preprocesses into Prolog clauses. The interpretation and scoping modules consist of various translation rules, expressed directly as Prolog clauses. Executing the grammar rules with Prolog leads to a straightforward top-down, backtrack parsing strategy (Pereira and Warren 1980). Although the main concern in writing the grammar was to produce a clear description of the language covered, some concessions have had to be made to achieve a reasonably efficient behaviour with the above parsing strategy; in particular, left-recursive rules have been avoided. It is somewhat contrary to current opinion that such a grammar, not carefully designed for parsing, can also be the basis of an efficient parser. One of the reasons for the efficiency in this case may be that the grammar itself makes no attempt to give the "right" modifier attachments, as discussed below.

There has been some debate on whether the different parts of language analysis, such as the three distinguished above, should be done serially or concurrently (Burton 1976; Woods 1977). Some of the arguments for concurrent operation are that early semantic interpretation limits search by bringing in relevant information at the earliest possible moment, and that piecewise generation of the interpretation is psychologically more plausible. Given that we are not proposing a *model* of language comprehension in people, the latter argument is not directly relevant. The other argument is a two-edged one: by interweaving several operations, one is multiplying together their nondeterminacies. There are two main reasons why we have chosen serial operation. The first is that serial operation is much simpler both conceptually and in programming terms, particularly if one is trying to deal flexibly with *global* properties of the input, for instance the relative scopes of determiners. The other reason is that the backtracking traditionally associated with modifier attachment can be avoided by a careful choice of the parse trees produced by the parsing module.

In typical systems where syntactic and semantic functions operate in sequence (Woods, Kaplan, and Nash-Webber 1972), choices are made in the syntactic analysis that may be found to be inadequate on semantic grounds, as for example in prepositional phrase attachment. In the present system, the syntactic component of the grammar contains additional constraints which block all but one of the potential attachments of each postmodifier. Thus, all analyses produces are in a kind of *normal form* with respect to postmodifier attachment. From this normal form, subsequent operations can reconstruct other alternative analyses, if that is needed on semantic grounds. The use of normal form analyses has the useful consequence of making apparent other, significant, ambiguities in the input, which otherwise would be swamped by a large number of alternative analyses differing only with respect to modifier attachment.

A major limitation of the current system is that pronouns (other than interrogative and relative pronouns) are not covered at all by the translation phase. This makes the natural language subset strictly less powerful than the underlying logic. For example, the following sentence has a logical form which cannot be rendered in our natural language subset:

Which country contains a city bigger than its capital?

```
 answer(C) <= country(C) &
     exists(X, contains(C,X) & city(X) &
         exists(Y, bigger(X,Y) & capital(C,Y) ) ).
```

To each of the three modules of the translation process there corresponds a separate dictionary. The grammar dictionary contains definitions of the usual syntactic categories, both for general-purpose, or "closed-category", words, such as determiners, and for application dependent, or "content", words, such as the nouns and verbs corresponding to database predicates. The slot filling module has a dictionary of *templates,* which define the translation of words to predicates and the argument patterns required for each such translation. Each application will require a different set of templates. Finally, the scope determination module has a small dictionary defining the scope relationships for determiners and other "operator" words, and their translation as discussed in the last section. This dictionary is independent of the application domain. The dictionaries are divided in this manner for conceptual and programming reasons, but it would not be difficult to write a program to create the dictionary entries from a more user-oriented dictionary format. The complete dictionary definition for a single word is shown in Appendix II.

3.1. Phrase Structure

A grammar for any substantial language fragment needs to define grammatical relationships which we may call "transformational", that is, relationships which cannot be described directly by a small number of phrase structure (context-free) rules. Both MGs (used in Dahl's system) and DCGs have general programming power, and so can describe any "transformational" relationship, but they cannot do so by specific, well motivated grammar rules. In particular, this applies to "left extraposition", the underlying concept in most grammars for such important constructions as WH-questions, relative clauses, and auxiliary fronting. Similar comments apply to ATNs (Woods 1970), even those using the HOLD/VIR facility.

To handle "left extraposition", and some other "transformational" concepts, we have introduced the grammar formalism of *extraposition* grammars, which are described fully elsewhere (Pereira 1981). An XG, like a DCG, is no more than "syntactic sugar" for clauses of logic.

As the Chat-80 grammar is intended partly as a demonstration of the power of XGs for treating left extraposition in English, the coverage of questions and relative clauses is fairly extensive. Of course, this wide coverage is essential if complex queries are to be formulated in a single sentence.

A major limitation in the present coverage of English syntax is that the only phrases that may be conjoined (with "and", etc.) are noun postmodifiers and predications introduced by the verb 'to be'. To cover more general conjunctions would require a minor change in the XG formalism, to cope with the interaction between left extraposition and conjoined phrases.

The analysis of a sentence produced by the Chat-80 grammar is a fairly conventional annotated surface structure, where in general the subtrees for all phrases appear in the same order as in the input. This is essential for the heuristics used in the scoping operations, which rely on the left-to-right order of noun phrases.

3.2. Attaching Arguments and Modifiers

As we have seen, the translation of content words into predicates is defined by dictionary templates. Of course, certain words, like "average", "number (of)" and superlative adjectives, cannot be translated as first-order predicates, but represent some higher-order operation. Templates specify, for each argument position of a predicate which translates a word, the "case" (usually a preposition) and the most general entity *type* that can fill the position, or *slot*. Type matching helps the system to find a semantically sound argument placement, and is also used to create additional predications when attributes are referred to implicitly, as in comparatives. Templates are similar to the dictionary entries in Dahl's system, but our attachment procedure, by being separate from the syntax analysis, can be much more flexible in deciding how to fill slots.

The following are typical application dependent templates:

```
property(area,measure&area,A,region&Type,R,
                                        area(R,A)).
transitive(exceed,measure&Type,X,measure&Type,Y,
                                        exceeds(X,Y)).
```

The first template states that 'area' is a "property", a subclass of nouns which require an argument marked by the preposition "of". Variable A in this template corresponds to the "area" value, which belongs to the sub-type 'area' of the type 'measure'. Variable R corresponds to the thing having an area, and belongs to some sub-type of type 'region'. The predication for this word is 'area(R,A)'. The second template states that "(to) exceed" is a transitive verb, whose subject X and object Y are both measures of some common type Type, and 'exceeds(X,Y)' is the corresponding predication.

When Prolog accesses such templates, unification automatically does most of the work of "slot filling".

The result of the slot filling process is a tree with three kinds of nodes, quantification nodes (Quants), predication nodes (Preds) and conjunction nodes (Conjs). Quants correspond to noun phrases, Preds to verbs, and Conjs to conjoined restrictive modifiers.

As we have seen, quantifications play a crucial role in the translation of sentences into logics. The fields of a Quant are:

- the *determiner,* which can be an English determiner or one of the special determiners described below;
- the *head,* which is either the predication translating the head noun of a noun phrase, or a term denoting a higher-order operation;
- the *predication,* a tree describing restrictions on the head whose determiners have narrower scope than the determiner on this Quant;
- the *arguments,* a list of the trees for the arguments of the noun, together with the trees of those restrictions whose determiners may have wider scope than the present one;
- the bound variable for this quantification.

The distinction between the predication and the argument list of a Quant corresponds to one of the main scoping heuristics in the system: full relative clauses are the only subordinated phrases whose determiners cannot "move up" to dominate determiners in higher tree nodes. Thus, when a Quant is mapped into a logic quantification, the head, predication, and some of the arguments will translate into the range of the quantification, whereas the scope will be made from the rest of the arguments and some quantifications which are higher in the tree but whose determiners have a narrower scope than the present one.

The determiner of a Quant may not be the original English determiner of the corresponding noun phrase. In nested plural definite noun phrases, only the highest determiner gets translated into a set expression following the translation table of Figure 1; lower determiners are understood to *index* that set, and are represented in Quants by an index token. For example, the noun phrase:

the children of the employees

translates into a set of sets of children, indexed by employees:

setof(E-S, employee(E) & setof(C,child(E,C),S), S1).

3.3. Meaning and Scope of Quantifications

Given the tree of Quants, Preds, and Conjs produced by slot filling, the final module specifies the relative scopes of determiners, of negation, and of question markers. The main information for this module is a set of rules of thumb about what determiners usually "govern" other determiners. The relationship of "governing" is not a total order, or even a partial order: it is only meaningful for pairs of operators, one in a subordinating and the other in a subordinated position. Relative scopes not decided by "governing" are decided by the left-to-right order of phrases in the input sentence.

Our scope rules are more accurate than those proposed by Colmerauer 1982 and used in Dahl's system. Colmerauer's rules determine relative scope strictly on the basis of predicate-argument relationships defined in syntactic terms. Our rules are related to the ideas for improving LUNAR discussed by Woods 1977 and Vanlehn 1978. Unfortunately, except for set and higher-order operations, we have no means of using the distinct roles of different argument places to help decide the scopes of their fillers.

As an example of our scope rules, the determiner 'each' is assumed to have a "distributive" role, and governs most other determiners, and also question markers, so the reply to a question containing 'each' will be an indexed list of values, one for each entity satisfying the conditions in the noun phrase with that determiner. A negated verb, however, will prevent an 'each' in any of its arguments from assuming its distributive role. In fact, no determiner governs a negation except 'any', which is seen as a universal quantification of wide scope relative to negation.

Apart from deciding on relative scopes, this module also specifies how set expressions are built from plural determiners and index determiners, and what are the first-order predications which make the arguments of higher-order functions like 'average' and 'number of'.

4. Query Planning and Execution

We have now seen how an English question is mapped into its logical form. Since the logical form has a precise semantics, it is *in principle* possible to determine the answer to the question. However, there is a big difference between principle and practice. Can the process of finding the answer really be made fast enough for this to be useful for practical question answering purposes? The evidence of Chat-80 is that it can.

The key question here is how to avoid combinatorial explosion. Naive question-answering strategies typically take a time that is exponential in the size of the query. Even for relatively small databases, such as that in Chat-80, exponential behaviour will be disastrous for non-trivial queries (and non-trivial queries are what a natural language interface encourages). A query over a small database that can be answered almost instantaneously with the right question-answering strategy may easily take hours with a poor algorithm. We had actual experience of this phenomenon when we experimented with mounting the Chat database on the relational database system Ingres (Stonebraker, Wong, Kreps, and Held 1976). Simple queries Ingres handled easily, but on queries involving more than two relations, which were no problem for Chat, Ingres usually bogged down completely. We therefore think that the question-answering part of natural language question-answering is at least as worthy of attention as the natural language part, and the two should be studied hand in hand.

A complete description of the way Chat-80 processes logic queries, together with a fuller discussion of the efficiency issues, is given in a separate paper (Warren 1981), which we urge the interested reader to consult. Here we just give a brief summary, illustrated by one example. Basically, Chat-80 augments the logical form of a query with extra *control* information, to make it into an efficient piece of Prolog program, which can then be directly executed to produce the answer. The control information is computed by a general planning algorithm, applicable to any query in the logic subset, not just to those derivable from the present natural language subset. The planning process makes use of certain statistics about the size, etc., of the domain relations, and is analogous to "query optimisation" in a relational database system.

The control information that is generated takes two forms:

- the ordering of predications within a query, which will determine the order in which Prolog will attempt to satisfy them;
- the marking of "independent subproblems" by enclosing them in braces, to limit (or "cut") the amount of backtracking performed by Prolog.

For example, here is an English question with the logical form produced by the natural language analysis phase of Chat-80:

"Which countries bordering the Mediterranean border Asian countries?"

```
answer(C) <= country(C) &
             borders(C,mediterranean) &
             exists(C1,country(C1) & asian(C1) &
             borders(C,C1))
```

After planning, the logical form is transformed into:

```
answer(C) <= borders(C,mediterranean) &
             {country(C)} & {borders(C,C1) &
             {asian(C1) & {country(C1)}}}
```

When executed by Prolog, this produces a behaviour equivalent to the following procedural interpretation:

```
To generate an answer C:
   generate a C bordering the mediterranean, and then
   check that C is a country, and then
   check that it is possible to:
      generate a C1 bordered by C, and then
      check that C1 is asian, and then
      check that C1 is a country.
```

Thus Prolog is led to answer the query in an obviously sensible way; it iterates through the countries bordering the Mediterranean, and for each one, it iterates through the things bordering that country until it finds something that is an Asian country. In fact the DEC-10 Prolog compiler can in principle compile the transformed query into code which is comparable in efficiency with iterative loops in a conventional language (Warren 1977). However, in Chat-80 the transformed query is actually just interpreted.

5. Performance and Portability

Chat-80 at its current stage of implementation, covers a limited but useful subset of English. A fair idea of the range of the present subset is given by the examples in Appendix I. It will be seen that the subset includes nouns, verbs, adjectives, prepositions, and determiners, with a fairly full coverage of interrogative and relative constructions. We have concentrated on features of English that seemed essential for simple question answering; there are many directions in which the subset could usefully be extended and which do not appear to pose any particular difficulties.

At present, the system accepts a small vocabulary of about 100 domain dependent works (not counting proper nouns, but including alternative word forms such as plurals). This vocabulary can very easily be extended (as indicated below). In addition there are some 50 domain independent words. On the whole, any question that can be expressed using this vocabulary is correctly understood and answered by the system.

The sizes of the different components of the system are indicated below in terms of the approximate number of Prolog clauses comprised, and the approximate number of DEC-10 (36-bit) words occupied:

	clauses	DEC-10 words
NL analysis	580	24000
Query planning and execution, etc.	290	8500
Geographical vocabulary	180	4000
Geographical database	1590	22500

The speed of Chat-80 on some sample queries relating to the geographical database is shown in Appendix I. Generally speaking, any query in this domain that can comfortably be expressed in a single sentence of the English subset is answered in well under one second of CPU time. Note that the domain dependent vocabulary could be much extended without having any significant impact on these times (because of the way the dictionary is indexed).

It is also worth noting that, for all but the simplest queries, natural language analysis represents only a small proportion of the total time. This suggests that, as far as the efficiency of natural language question answering systems is concerned, it is the answering process rather than the natural language analysis to which most effort needs to be directed. Certainly this has been our approach, although it appears to be somewhat contrary to the prevailing view in artificial intelligence. In particular, parsing does not seem to pose any major efficiency problem, provided one does not expect the grammar to do too much.

As regards portability, we think Chat-80 should be relatively easy to adapt to different applications – for the same reasons that we found it easy to adapt Dahl's program to English and to a different domain.

Partly this is due to the fact that (in both systems) the domain dependent parts are clearly separated from the rest of the system, and are broken down into small units which can be added incrementally as "data" (see Appendix II). Thus our natural language analysis modules deal exclusively with general features of English, in contrast to the "semantic grammar" approach (Burton 1976).

Now there are other practical systems which have not taken the "semantic grammar" approach but are, we feel, less easy to modify than Chat-80; LUNAR (Woods, Kaplan, and Nash-Webber 1972) is a good example. The reason lies in the way "meanings" are attached to words. In LUNAR, a meaning is simply a procedure. For nouns it is a procedure to generate objects in a certain class; for most other words it is a procedure to test whether some property is true of given objects. This entails that "meanings" can be executed in only one way, and precludes the kind of query planning done in Chat-80. But such a simple-minded approach to query execution is not viable in most practical situations, as Woods 1977 recognises with his "smart quantifiers". The only alternative, given the procedural approach to meaning, is to represent the meaning of a word by a set of alternative procedures to be used in different circumstances. But this makes life very difficult for someone wanting to introduce a new word or concept into the system.

In Chat-80, the meaning of a new word is in principle just a set of facts and general rules that define the predicate corresponding to that word. The procedural aspect is on the whole taken care of by the planning algorithm, and by Prolog's flexible handling of predications in which only certain arguments are instantiated. However, with the present system, the definer of a new word must be responsible for ensuring that the predicate definition he supplies is not only correct, but is also reasonably efficient when executed by Prolog.

6. Conclusion

We have shown how questions within a limited subset of English can be translated into a certain subset of logic which, when suitably transformed, is executable as efficient Prolog code. Although this mapping between English and logic may seem "obvious" (since logic is, after all, usually motivated in terms of its correspondence with natural language), it is surprising that a mapping like ours does not appear to have been implemented, or even precisely defined, before. Parts of the mapping overlap with Montague's 1974 formalisation of aspects of English, but many of the basics were not covered in his work (for instance, the treatment of questions, plurals, and determiner precedence), and of course he was not concerned to produce a practical implementation. Among practical implementations, the closest work is that of Woods 1977 and that of Colmerauer 1982 as implemented by Dahl 1981. Both these efforts are similar in that they map English into a nonstandard (and more elaborate) logic, where the quantifiers are more directly modelled on the determiners of natural language. These nonstandard quantifiers are called "FOR expressions" by Woods and "three-branched quantifiers" by Colmerauer. We have kept much closer to standard predicate logic, for the very down-to-earth reason that the traditional formalism seems to make "query optimisation" much easier (see the companion paper, Warren 1981).

Our mapping from English to logic, as well as the processes of query planning and answering, have all been implemented entirely in Prolog. The result is a prototype natural language question answering system, Chat-80, which we think probably has the best combination of efficiency and portability of any comparable system at the present time, due principally to the use of Prolog as the implementation language.

Acknowledgements

This work was supported by a British Science Research Council grant, and owes much to the work of Alain Colmerauer and Veronica Dahl.

References

Burton, R.R. 1976 Semantic grammar: an engineering technique for constructing natural language understanding systems. Report 3453. Bolt Beranek and Newman Inc. (December).

Byrd, L., Pereira, F., and Warren, D.H.D. 1980 A guide to version 3 of DEC-10 Prolog. Occasional Paper 19. Dept. of Artificial Intelligence, University of Edinburgh (July).

Colmerauer, A. 1978 Metamorphosis grammars. In Bolc., L., Ed., *Natural Language Communication with Computers.* Springer-Verlag. First appeared as an internal report, Les Grammaires de Metamorphose, in November 1975.

Colmerauer, A. 1982 An interesting subset of natural language. In Clark, K.L. and Tarnlund, S.-A., Ed., *Logic Programming.* Academic Press.

Dahl, V. 1979 Quantification in a three-valued logic for natural language question-answering systems. *IJCAI-79,* Tokyo (August) 182-187.

Dahl, V. 1981 Translating Spanish into logic through logic. *AJCL* **7,** 3, 149-164.

Konolige, K. 1979 A framework for a portable natural language interface to large data bases. Technical Note 197. AI Center, SRI International (October).

McCord, M.C. 1982 Using slots and modifiers in logic grammars for natural language. *Artificial Intelligence* **18,** 3, 327-367.

Montague, R. 1974 The proper treatment of quantification in ordinary English. In Thomason, R.M., Ed., *Formal Philosophy.* Yale University Press.

Pereira, F.C.N. 1981 Extraposition grammars. *AJCL* **7**, 4, 243-256.

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

Pereira, F.C.N. and Warren, D.H.D. 1980 Definite clause grammars for language analysis – a survey of the formalism and a comparison with augmented transition networks. *Artificial Intelligence* **13,** 231-278.

Pereira, L.M., Pereira, F., and Warren, D.H.D. 1978 User's guide to DECsystem-10 Prolog. Dept. of Artificial Intelligence, University of Edinburgh.

Roussel, P. 1975 Prolog: manuel de reference et d'utilisation Group d'Intelligence Artificielle, UER de Luminy, Université d'Aix-Marseille II.

Stonebraker, M., Wong, E., Kreps, P. and Held, G. 1976 The design and implementation of INGRES. *ACM Trans. on Database Systems* **1,** 3 (Sept.) 189-222.

Vanlehn, K.A. 1978 Determining the Scope of English Quantifiers. Master's thesis. Published as Report AI-TR-483. M.I.T. (June).

Warren, D.H.D. 1977 Implementing Prolog – compiling predicate logic programs. Technical Research Reports 39 and 40. Dept. of Artificial Intelligence, University of Edinburgh (May).

Warren, D.H.D. 1979 Prolog on the DECsystem-10. In Michie, D., Ed., *Expert Systems in the Micro-Electronic Age.* Edinburgh University Press.

Warren, D.H.D. 1981 Efficient processing of interactive relational database queries expressed in logic. Seventh International Conference on Very Large Data Bases.

Warren, D.H.D. 1982 Higher-order extensions to Prolog – are they needed? In Hayes, Michie, and Pao, Ed., *Machine Intelligence 10.* Ellis Horwood.

Warren, D.H.D., Pereira, L.M., and Pereira, F.C.N. 1977 Prolog – the language and its implementation compared with Lisp. ACM Symposium on AI and Programming Languages (August).

Woods, W.A. 1970 Transition network grammars for natural language analysis. *Comm. ACM* **13** (Oct.) 591-606.

Woods, W.A. 1977 Semantics and quantification in natural language question answering. Report 3687. Bolt Beranek and Newman Inc. (November).

Woods, W.A., Kaplan, R.M., and Nash-Webber, B. 1972 The Lunar science natural language information system: final report. Report 3438. Bolt Beranek and Newman Inc. (June).

Appendix I. Sample Queries

The Chat-80 examples below show the original English query, its logical form, the executable form after planning, and the actual answer. (The logical expressions have been "tidied", and superfluous quantifiers dropped, to make them easier to read.) Also shown, preceding the corresponding output, are the separate times (in CPU milliseconds on a DEC KL-10) for natural language analysis, for planning, and for execution. Time spent in producing output to the user is excluded.

Does Afghanistan border China?

```
 38 ms.  ans(yes) <= borders(afghanistan,china).
 12 ms.  ans(yes) <= {borders(afghanistan,china)}.
  0 ms.  yes.
```

Which country's capital is Ouagadougou?

```
 41 ms.  ans(C) <= country(C) & capital(C,ouagadougou).
 15 ms.  ans(C) <= capital(C,ouagadougou) & {country(C)}.
 22 ms.  upper_volta.
```

Which is the ocean that borders African countries and that borders Asisan countries?

```
 91 ms.  ans(X) <= ocean(X) & country(C) & african(C) & borders(X,C) &
                          country(C1) & asian(C1) & borders(X,C1).
 51 ms.  ans(X) <= ocean(X) &
            {borders(X,C) & {african(C)} & {country(C)}} &
            {borders(X,C1) & {asian(C1)} & {country(C1)}}.
102 ms.  indian_ocean.
```

What is the capital of each country bordering the Baltic?

```
 81 ms.  ans(C-X) <= country(C) & borders(C,baltic) & capital(C,X).
 12 ms.  ans(C-X) <= borders(C,baltic) & {country(C)} & capital(C,X).
 29 ms.  denmark-copenhagen, east_germany-east_berlin, finland-helsinki, poland-warsaw,
         soviet_union-moscow, sweden-stockholm, west_germany-bonn.
```

What are the lattitudes of the countries north of the United Kingdom?

```
102 ms.  ans(C-LL) <= country(C) & northof(C,united_kingdom) &
                          setof(L,latitude(C,L),LL).
 26 ms.  ans(C-LL) <= northof(C,united_kingdom) &{country(C)} &
                  setof(L,latitude(C,L),LL).
141 ms.  canada-60 degrees, denmark-55 degrees, finland-65 degrees, iceland-65 degrees,
         norway-64 degrees, soviet_union-57 degrees, sweden-63 degrees.
```

Which country is bordered by two seas?

```
 42. ms.  ans(C) <= country(C) & numberof(X,sea(X) & borders(C,X),2).
 11 ms.  ans(C) <= numberof(X,sea(X) & borders(C,X),2) & {country(C)}.
206 ms.  egypt, iran, israel, saudi_arabia, turkey.
```

How many countries does the Danube flow through?

```
 48 ms.  ans(N) <= numberof(C,country(C) & flows(danube,C),N).
  3 ms.  ans(N) <= numberof(C,flows(danube,C) & {country(C)},N).
 21 ms.  6.
```

From what country does a river flow into the Persian Gulf?

```
 69 ms.  ans(C) <= river(R) & country(C) & flows(R,C,persian_gulf).
 12 ms.  ans(C) <= flows(R,C,persian_gulf) & {river(R)} & {country(C)}.
 23 ms.  iraq.
```

What is the total area of countries south of the Equator not in Australasia?

```
115 ms.  ans(T) <= setof(A-C,area(C,A) & country(C) &
                          southof(C,equator) & \+in(C,australasia),S) &
            aggregate(total,S,T).
 23 ms.  ans(T) <= setof(A-C,southof(C,equator) & area(C,A) &
                          {countryof(C)} & \+in(C,australasia),S) &
            aggregate(total,S,T).
182 ms.  10228 ksqmiles.
```

What is the average area of the countries in each continent?

```
101 ms.  ans(X-Av) <= continent(X) &
                  setof(A-C,area(C,A) & country(C) & in(C,X),S) &
                  aggregate(average,S,Av).
 33 ms.  ans(X-Av) <= continent(X) &
                  setof(A-C,in(C,X) & area(C,A) & {country(C)},S) &
                  aggregate(average,S,Av).
759 ms.  africa-233 ksqmiles, america-496 ksqmiles, asia-485 ksqmiles,
         australasia-543 ksqmiles, europe-58 ksqmiles.
```

How many countries are there in each continent?

```
 50 ms.  ans(X-N) <= continent(X) & numberof(C,country(C) & in(C,X),N).
 19 ms.  ans(X-N) <= continent(X) & numberof (C,in(C,X) & {country(C)},N).
352 ms.  africa-48, america-31, asia-39, australasia-6, europe-32.
```

Is there some ocean that does not border any country?

```
 68 ms.  ans(yes) <= ocean(X) & \+exists(C,borders(X,C) & country(C)).
  9 ms.  ans(yes) <= {ocean(X) & \+exists(C,borders(X,C) & {country(C)})}.
 14 ms.  yes.
```

What does border the ocean that does not border any country?

```
 59 ms.  ans(Y) <= ocean(X) & \+exists(C,country(C) & borders(X,C)) &
             borders(Y,X).
 20 ms.  ans(Y) <= ocean(X) & \+exists(C,borders(X,C) & {country,(C)}) &
             borders(Y,X).
 24 ms.  antarctica, atlantic, indian__ocean, pacific.
```

Which are the continents no country in which contains more than two cities whose population exceeds 1 million?

```
160 ms.  ans(X) <= continent(X) &
             \+exists(C,exists(N,
                country(C) & in(C,X) &
                numberof(Ci,exists(P,
                   city(Ci) & population(Ci,P) &
                   exceeds(P,1000000) & in(Ci,C)),N) &
                N > 2)).
 58 ms.  ans(X) <= continent(X) &
             \+exists(C,exists(N,
                in(C,X) & {country(C)} &
                {numberof(Ci,exists(P,
                   in(Ci,C) & {city(Ci)} &
                   {population(Ci,P) & {exceeds(P,1000000)}}),N) &
                {N > 2}})).
754 ms.  africa, antarctica, australasia.
```

Which country bordering the Mediterranean borders a country that is bordered by a country whose population exceeds the population of India?

```
144 ms.  ans(C) <= country(C) & borders(C,mediterranean) & country(C1) &
             country(C2) & population(C2,X) & population(indea,Y) &
             exceeds(X,Y) & borders(C2,C1) & borders(C,C1).
 57 ms.  ans(C) <= population(india,Y) &
             borders(C,mediterranean) & {country(C)} &
             {borders(C,C1) & {country(C1)} &
                {borders(C2,C1) & {country(C2)} &
                   {population(C2,X) & {exceeds(X,Y)}}}}.
204 ms.  turkey.
```

Appendix II. Sample Word Definition

To illustrate how a new word, and the concept associated with it, are added to the system, we show a slightly simplified version of the actual Prolog clauses defining the verb "to drain into." These clauses represent elementary facts, giving the different forms of the verb, its meaning in terms of the predicate 'drains(R,X)' (including the types of the arguments R and X), data about the size of the 'drains' relation and of its argument domains, and finally the definition of the predicate 'drains' itself.

```
verb_root(drain)
regular_present(drain).
regular_past(drained,drain).
verb_form(drains,drain,present+finite,3+singular).
verb_form(draining,drain,present+participle).
intransitive(drain, drains(R,X), river,R, [slot(preposition(into),region,X)]).
predicate_statistics(drains,41,41,12).
drains(amazon,atlantic).
drains(amu_darya,aral_sea).
...
etc.
```

Appendix III. Comparison with a Formal Query Language

It is interesting to compare, from a user's point of view, the Chat-80 subset of English with current relational database query languages. The examples below show some Chat-80 queries with their equivalents in Quel, the query language of the relational database system Ingres (Stonebraker, Wong, Kreps, and Held 1976). Quel is arguably one of the most concise and user-friendly of current database query languages.

Which countries bordering the Atlantic border countries bordering the Pacific?

```
range of C1, C2 is countries
range of B, B1, B2 is borders
retrieve (C1.name)
where      C1.name = B1.side1
and        B1.side2 = "Atlantic"
and        C1.name = B.side1
and        B.side2 = C2.name
and        C2.name = B2.side1
and        B2.side2 = "Pacific"
```

How many countries are there in each continent?

```
range of C is countries
range of Cont is continents
range of I is inclusions
retrieve (Cont.name, count(C.name where C.name = I.inside and I.outside = Cont.name))
```

Which are the continents no country in which contains more than two cities whose population exceeds 1 million?

```
range of C is countries
range of Cont is continents
range of City is cities
range of I1, I2 is inclusions
retrieve (Cont.name)
where      0 = count(C.name
where      C.name = I1.inside
and        I1.outside = Cont.name
and        2 < count(City.name
           where    City.name = I2.inside
           and      I2.outside = C.name
           and      City.population > 1000000))
```