HOW TO DRIVE A DATABASE FRONT END USING GENERAL SEMANTIC INFORMATION

B.K. Boguraev and K. Sparck Jones

Computer Laboratory, University of Cambridge
Corn Exchange Street, Cambridge CB2 3QG, England

ABSTRACT

This paper describes a front end for natural language access to databases making extensive use of general, i.e. domain-independent, semantic information for question interpretation. In the interests of portability, initial syntactic and semantic processing of a question is carried out without any reference to the database domain, and domain-dependent operations are confined to subsequent, comparatively straightforward, processing of the initial interpretation. The different modules of the front end are described, and the system's performance is illustrated by examples.

I INTRODUCTION

Following the development of various front ends for natural language access to databases, it is now generally agreed that such a front end must utilise at least three different kinds of knowledge to accomplish its task: linguistic knowledge, knowledge of the domain of discourse, and knowledge of the organisational structure of the database. Thus broadly speaking, a user request to the database goes through three conceptually different forms: the output of linguistic analysis of the question, its representation in terms of the domain's conceptual schema, and its interpretation in the database access language. Early natural language front ends usually did not have a clearcut separation between the different stages of the process: for example LUNAR (Woods 1972) merged the domain model and the database model into one, and systems such as the early incarnation of LADDER (Hendrix et al 1978) and PLANES (Waltz 1978) made heavy use of semantic grammars with their domain-dependent lexicons combining linguistic knowledge with domain knowledge and so merging the first two stages. None of these systems, moreover, made any significant use of general, as opposed to domain-specific, semantic information.

In an attempt to achieve portability from one database to another, most current systems adhere to a general framework (Konolige 1979), which makes a clear distinction between the different processing phases and distinguishes the domain-dependent from the domain-independent parts of the front end, and also domain operations from database management operations. However semantic processing is still essentially driven by domain-dependent semantics. Linguistic processing is therefore primarily syntactic parsing, and relating general linguistic to specific domain knowledge within the framework of a modular front end takes the form of applying domain-dependent semantic processing to the output of the syntactic parser. This may be done in a simple-minded way as in PHLIQA1 (Bronnenberg et al 1979) and TQA (Damerau 1980), or by providing hooks in the syntactic representation (domain-independent calls to semantic operators which will evaluate differently in different contexts), as in DIALOGIC (Grosz et al 1982). In either case the usual unhappy consequence of separating syntactic and semantic processing, namely the hassle of manipulating alternative syntactic trees, follows. Furthermore, changing domains implies changing the definitions of the semantic operators, which are procedural in nature, while it may be preferable to keep the domain-dependent parts of the front end in declarative form, as is indeed done in (Warren and Pereira 1981).

Thus in systems of this by now conventional type, the 'portability' achieved by confining the necessary domain-dependent semantic processing to well-defined modules is purchased at the heavy price of limiting the early linguistic processing to syntax, and, perhaps, some very global and undiscriminating semantics (see for example the scoping algorithm of (Grosz et al 1982)).

II SPECIFIC APPROACH

Our objective is to do better than this by making more use of powerful, but still non-domain-dependent semantics in the front-end linguistic analysis. Doing this should have two advantages: restraining syntax, and providing a good platform for domain-dependent semantic processing. However, the overall architecture of the front end still follows the Konolige model in maintaining a clearcut separation between the different kinds of knowledge to be utilised, keeping the bulk of the domain-dependent knowledge in declarative form, and attempting to minimise the consequences of changes in the front end environment, whether of domain or database model, to promote smooth transfers of the front end from one back end database management system to another.

We believe that there is a lot of mileage to be got from non-task-specific semantic analysis of user requests, because their resulting rich, explicit, and normalised meaning representations are a good

This work is supported by the U.K. Science and Engineering Research Council.

starting point for subsequent task-specific operations, and specifically, are better than either syntax trees, or the actual input text of e.g. the PLANES approach. Furthermore, since the domain world is (in some sense) a subset of the real world, it is possible to interpret descriptions of it using the same semantic apparatus and representation language as is used by the natural language analyser, which should allow easy and reliable linking of the natural language input words, domain world objects and relationships and data language terms and expressions. Since the connections between these do not appear hard-wired in the lexicon, but are established on the basis of matching rich semantic patterns, no changes at all should be required in the lexicon as the application moves from one domain or database to another, only expansions to allow for the semantic definitions of new words relevant to the new application.

The approach leads to an overall front end structure as follows:

```
                 : English question :
   ANALYSIS               |
+-------------------------+-------------------------+
| +-----------------------v------ 1 -------------+  |
| | Analyser                                      |  |
| |  (uses linguistic knowledge)                  |  |
| +-----------------------------------------------+  |
|                         v                          |
|          : meaning representation :                |
|                         v                          |
| +------------------------------ 2 -------------+  |
| | Extractor                                     |  |
| |  (uses logico-linguistic knowledge)           |  |
| +-----------------------------------------------+  |
+-------------------------+-------------------------+
                          v
             : logic representation :
   TRANSLATION            |
+-------------------------+-------------------------+
| +-----------------------v------ 3 -------------+  |
| | Translator                                    |  |
| |  (uses domain world knowledge)                |  |
| +-----------------------------------------------+  |
|                         v                          |
|           : query representation :                 |
|                         v                          |
| +------------------------------ 4 -------------+  |
| | Convertor                                     |  |
| |  (uses database organisation                  |  |
| |   knowledge)            |                     |  |
| +-------------------------+---------------------+  |
+-------------------------+-------------------------+
                          v
             : search representation :
```

Each process in the diagram above operates on the output of the previous one. Processes 1 and 2 constitute the analysis phase, and processes 3 and 4 - the translation phase. Such a system has essentially been constructed, and is under active test; a detailed account of its components and operations follows.

For the purposes of illustration we shall use questions addressed to the Suppliers and Parts relational database of (Date 1977). This has three relations with the following structure: Supplier(Sno, Sname, Status, Scity), Part(Pno, Pname, Colour, Weight, Pcity), and Shipments(Sno, Pno, Quantity).

III ANALYSIS

A. The Analyser

The natural language analyser has been described in detail elsewhere (Boguraev 1979), (Boguraev and Sparck Jones 1982), and only a brief summary will be presented here. It has been designed as a general purpose, domain- and task-independent language processor, driven by a fairly extensive linguistically-motivated grammar and controlled in its operation by variegated application of a rich and powerful semantic apparatus. Syntactically-controlled constituent identification is coupled with the judgemental application of semantic specialists; following the evaluation of the semantic plausibility of the constituent at hand, the currently active processor either aborts the analysis path or constructs a meaning representation for the textual unit (noun phrase, complementiser, embedded clause, etc.) for incorporation into any larger semantic construct. The philosophy behind the analyser is that syntactically-driven analysis (which is a major prerequisite for domain- and/or task-independence) is made efficient by frequent and timely calls to semantic specialists, which both control blind syntactic backtracking and construct meaning representations for input text without going through the potentially costly enumeration of intermediate syntactic trees. The analyser can therefore operate smoothly in environments which are syntactically or lexically highly ambiguous.

To achieve its objectives the program pursues a passive parsing strategy based on semantic pattern matching of the kind proposed by (Wilks 1975). Thus the semantic specialists work with a range of patterns referring to narrower or broader word classes, all defined using general semantic primitives and ultimately depending on formulae which use the primitives to characterise individual word senses. However the application of patterns in the search for input text meaning is more effectively controlled by syntax in this system than in Wilks'.

The particular advantages of the approach in the database application context are the powerful and flexible means of representing linguistic and world knowledge provided by the semantic primitives, and the ease with which 'traps for the unexpected' can be procedurally encoded. The latter means that the system can readily deal with the kinds of problems generated by unconstrained natural language text which provoke untoward 'ripple' effects when large semantic grammars are modified. The semantic primitive foundation for the analyser provides a good base for the whole front end, since the comprehensive inventory of primitives can be exploited to characterise both natural language and data language terms and expressions, and to reconcile the user's view of the database domain with the actual administrative organisation of the database.

For present purposes, the form and content of the outputs of the natural language analyser are more important than the means by which they are derived (for these see Boguraev and Sparck Jones 1982). The meaning representations output by the analyser are dependency structures with clusters of case-labelled components centred around main verb or noun elements. Apart from the structure of the dependency tree itself, and group identifying markers like 'tns' and 'modality', the substantive information in the meaning representation is provided by the case labels, which are drawn from a large set of semantic relation primitives forming part of the overall inventory of primitives, and by the semantic category primitive characterisations of lexically-derived items.

The formulae characterising word senses may be quite rich. The fairly straightforward characterisation of 'supplier1', representing one sense of "supplier" is

```
(Supplier ...
   (supplier1
      (((*ent obje) give) (subj *org)) ...),
```

meaning approximately that some sort of organisation (which may reduce to an individual) gives entities. The meaning representation for the whole sentence "Suppliers live in cities" (with the formulae for individual units abbreviated, for space reasons, to their head primitives) is

```
(clause ........
   (v
      (live1 ... be
         (@@agent (n (supplier1 ... man)))
         (@@location (n (city2 ... spread)))))),
```

where @agent and @location are case labels. "The parts are coloured red" will be analysed as

```
(clause ......
   (v
      (be2 ... be
         (@@agent
            (n (part1 ... thing
                  (@@number many))))
         (@@state
            (st (colour1 ... sign)
                  (val (red1 ... sense))))))),
```

and "Who supplies green parts?" will give rise to the structure:

```
(clause ... (type question)
  (v
    (supply1 ... give
      (@@agent (n (query (dummy))))
      (@@cbject
        ((trace (clause v agent))
          (clause
            (v
              (be2 ... be
                (@@agent
                  (n (part1 ... thing)))
                (@@state (st (cclour1 ... sign)
                  (val
                    (green1 ...
                      (see sense)))))))))))).
```

As these examples show, the analyser's representations combine expressive power with structural simplicity. Further, the power of the semantic category primitives used to identify text message patterns means that it is possible to achieve far more semantic analysis of a question, far earlier in the front end processing, than can be achieved with front ends conforming to the Konolige model. The effectiveness of the analyser as a general natural-language processing device has been demonstrated by its successful application to a range of natural language processing tasks. There is, however, a price to pay, in the database context, for its generality. Natural language makes common use of vague concepts ("have", "do"), almost content-empty markers ("be", "of"), and opaque constructions such as compound nouns. Clearly, front ends where domain-specific information can provide leverage in interpreting these input text items have advantages, and it is not clear how a principled solution to the problems they present can be achieved within the framework of a general-purpose analyser of the kind described. To provide a domain-specific interpretation of, for example, compounds like "supplier city", an interface would have to be provided characterising domain knowledge in the semantic terms familiar to the parser, and guaranteeing the provision of explicit structural characterisations of the text constituent which would be available for further exploitation by the parser.

To avoid invoking domain knowledge in this way in analysis we have been obliged to accept question interpretations which are incomplete in limited respects. That is, we push the ordinary semantic analysis procedures as far as they will go, accepting that they may leave 'dummy' markers in the dependency structure and compound nominals with ambiguous member words and no explicit extracted structure.

B. The Extractor

While the meaning representations constructed by the natural language analyser are general and informative enough to be able to support different tasks in different applications for different domains, they are not necessarily the best form of representation for question answering, and specifically for addressing a coded database. After the initial determination of question meaning, therefore, the question is subjected to task-oriented, though not yet domain- and database-oriented, processing. Imposing domain world and database organisation restrictions on the question at this stage would be premature, since it could complicate or even inhibit possible later inference operations. The idea of providing a system component addressing a general linguistic task, without throwing away any detailed information not in fact needed for some specific instance of that task, like natural language distinctions between quantifiers ignored by the database system, is also an attractive one.

The extractor thus emphasises the fact that the input text is a question, but carries the detailed semantic information provided by the analyser forward for exploitation in the translation phase of the processing.

A good way to achieve a question formulation abstracted from the low-level organisation of the database is to interpret the user's input as a formal query. However our extractor, unlike the equivalent processors described by (Woods 1972), (Warren and Pereira 1981) and (Grosz et al 1982), does not make any use of domain-dependent information, but constructs a logic expression whose variable ranges and predicate relationships are defined in terms of the general semantic primitives used for constructing the input question meaning representation. The logic representation of the question which is output by the extractor highlights the search aspects of the input, formalising them so that the subsequent processes which will eventually generate the search specification for the database management system can locate and focus on them easily; at the same time, the semantic richness of the original meaning representation is maintained to facilitate the later domain-oriented translation operations.

The syntax of the logic representation closely follows that defined by (Woods 1978):

```
(For <quantifier> <variable> / <range>
   : <restrictions on variable>
   - <proposition> ),
```

where each of the restrictions, or the proposition, can themselves be quantified expressions. The rationale for such quantified expressions as media for questions addressed towards an abstract database has been discussed by Woods. As we accept this, we have developed a transformation procedure which takes the meaning representation of an input question and constructs a corresponding logic representation in the form just described. Thus for the question "Who supplies green parts?" analysed in Section A, we obtain

```
(For Every $Var1 / query
   : (For Every $Var2 / part1
      : (colour1 $Var2 green1)
      - (supply1 $Var1 $Var2))
   - (Display $Var1)),
```

where the lexically-derived items indicating the ranges of the quantified variables ('query', 'part1'), the relationships between the variables ('supply1') and the predicates and predicate values ('colour1', 'green1') in fact carry along with them their semantic formulae: these are omitted here, and in the rest of the paper, to save space.

The extractor is geared to seek, in the analyser's dependency structures, the simple propositions (atomic predications) which make up the logic representation. Following the philosophy of the semantic theory underlying the analyser design, these simple propositions are identified with the basic messages, i.e. semantic patterns, which drive the parser and are expressed in the meaning representations it produces as verb and noun group clusters of case-related elements. In order to 'unpack' these, the extractor looks for the sources of atomic predicates as 'SVO' triples, identifiable by a verb (or noun) and its case role fillers, which can be extracted quite naturally in a straightforward way from the dependency structure.

Depending both on the semantic characterisation of the verb and its case arguments, and on the semantic context as defined by the dependency tree, the triples are categorised as belonging to one of two types: [$Obj $Link $Obj], or [$Obj $Poss $Prop], where the $Obj, $Link, or $Prop items are further characterised in semantic terms. It is clear that the 'basic messages' that the extractor seeks to identify as a preliminary step to constructing the logic representation define either primitive relationships between objects, or properties of those same objects. Thus the meaning representation for "part suppliers" will be unpicked as a 'dummy' relationship between "suppliers" and "parts", i.e. as

```
[$Obj1(supplier1) $Link1(dummy) $Obj2(part1)],
```

while "green parts" will be interpreted as

```
[$Obj2(part1) $Poss(be2) $Prop(colour1=green1)].
```

Larger constructs can be similarly decomposed: thus "Where do the status 32 red parts suppliers live?" will be broken down into the following set of triples:

```
   [$Obj1(supplier1) $Link1(live1) $Obj3(query)]
 & [$Obj1(supplier1) $Link2(dummy) $Obj2(part1)]
& [$Obj1(supplier1) $Poss1(be2) $Prop1(status=32)]
 & [$Obj2(part1) $Poss2(be2) $Prop2(colour1=red1)].
```

It must be emphasised that while there are parallels between these structures and those of the entity-attribute approach to data modelling, the forms of triple were chosen without any reference to databases. As noted earlier, they naturally reflect the form of the 'atomic propositions', i.e. basic messages, used as semantic patterns by the natural language analyser.

For completeness, the triples underlying the earlier question "Who supplies green parts?" are

```
[$Obj1(query=identity)
    $Link1(supply1) $Obj2(part1)]
 & [$Obj2(part1)
      $Poss1(be2) $Prop1(colour1=green1)]
```

The sets of interconnected triples are derived from the meaning representations by a fairly simple recursive procedure. The next stage of the extraction process restructures the triples tree into a skeleton quantified structure, the logic representation, to be passed forward to the translator generating the formal query representation. Whenever more explicit information regarding the interpretation of the input as a question can be extracted from the meaning representation, this is incorporated into the logic representation. Thus the processing includes identification and scoping of quantifiers following the approach adopted by Woods, and establishing the aspect, modality and focus of the question. Like anyone else, we do not claim to provide a comprehensive treatment of natural language quantifiers, and indeed in practice have not implemented processes for all the quantifiers handled by LUNAR.

The logic representation defines the logical content and structure of the information the user is seeking. It may, as noted, be incomplete at points where domain reference is required, e.g. in the interpretation of compound nouns; but it carries along, to the translator, the very large amount of semantic information provided by the case labels and formulae of the meaning representation, which should be adequate to pinpoint the items sought by the user and to describe them in terms suited to the database management system, so they may be accessed and retrieved.

IV TRANSLATION

A. The translator

In the process of transforming the semantic content of the user's question into a low-level search representation geared to the administrative structure of the target database, it is necessary to reconcile the user's view of the world with the domain model. Before even attempting to construct, say, a relational algebra expression to be interpreted by the back-end database management system, we must try to interpret the semantic content of the logic representation with reference to the segment or variant of the real world modelled by the database.

An obvious possibility here is to proceed directly from the variables and predications of the logic representation to their database counterparts. For example,

```
 (supply1 (give)
    $Var1/supplier1 (man) $Var2/part1 (thing))
```

can be mapped directly onto a relation Shipments in the Suppliers and Parts database. The mapping could be established by reference to the lexicon and to a schedule of equivalences between logical and database structures.

This approach suffers, however, from severe problems: the most important is that end users do not necessarily constrain their natural language to a highly limited vocabulary. Even in the simple context of the Suppliers and Parts database, it is possible to refer to "firms", "goods", "buyers", "sellers", "provisions", "customers", etc. In fact, it was precisely in order to bring variants under a common denominator that semantic grammars were employed. We, in contrast, have a more powerful, because more flexible, semantic apparatus at our disposal, capable of drawing out the similarities between "firms", "sellers", and "suppliers", as opposed to taking them as read. Thus a general semantic pattern which will match the dictionary definitions of all of these words is (((*ent obje) give) (subj *org)). Furthermore, if instead of attempting to define any sort of direct mapping between the natural language terms and expressions of the user and corresponding domain terms and expressions, we concentrate on finding the common links between them, we can see that even though the domain and, in turn, database terms and expressions may not mean exactly the same as their natural language relatives or sources, we should be able to detect overlaps in their semantic characterisations. It is unlikely that the same or similar words will be used in both natural and data languages if their meanings have nothing in common, even if they are not identical, so characterising each using the same repertoire of semantic primitives should serve to establish the links between the two. Thus, for example, one sense of the natural language word "location" will have the formula (this (where spread)) and the data language word "&city" referring to the domain object &city will have the formula (((man folk) wrap) (where spread)), which can be connected by the common constituent (where spread).

One distinctive feature of our front end design, the use of general semantics for initial question interpretation, is thus connected with another: the more stringent requirements imposed on natural language to data language translation by the initial unconstrained question interpretation can be met by exploiting the resources for language meaning representation initially utilised for the natural language question interpretation. We define the domain world modelled by the database using the same semantic apparatus as the one used by the natural language front end processor, and invoke a flexible and sophisticated semantic pattern matcher to establish the connection between the semantic content of the user question (which is carried over in the logic representation) and related concepts in the domain world. Taking the next step from a domain world concept or relationship between domain world objects to their direct model in the administrative structure of the database is then relatively easy.

Since the domain world is essentially a closed world restricted in sets if not in their members, it is possible to describe it in terms of a limited set of concepts and relationships: we have possible properties of objects and potential relationships between them. We can talk about &suppliers and &parts and the important relationship between them, namely that &suppliers &supply &parts. We can also specify that &suppliers &live in &cities, &parts can be &numbered, and so on.

We can thus utilise, either explicitly or implicitly, a description of the domain world which could be represented by dependency structures like those used for natural language. The important point about these is the way they express the semantic content of whole statements about the domain, rather than the way they label individual domain-referring terms as, e.g. "&supplier" or "&part". It is then easy to see how the logic representation for the question "What are the numbers of the status 30 suppliers?", namely

```
 (For Every $Var1/supplier1 : (status1 $Var1 30)
     - (Display (number1 $Var1))),
```

can be unpacked by semantic pattern matching routines to establish the connection between "supplier1" and "&supplier", "number1" and "&number", and so on. In the same way the logic representations for "From where does Blake operate?" and "Where are screws found?" can be analysed for semantic content which will establish that "Blake" is a &supplier, "operate" in the context of the database domain means &supply, and "where" is a query marker acting for &city from which the &supplier Blake &supplies (as opposed to street corner, bucket shop, or crafts market); similarly, "screw" is an instance of &part and the only locational information associated with &parts in the database in question is the &city where they are stored. All this becomes clear simply by matching the underlying semantic primitive definitions of the natural language and domain world words, in their propositional contexts.

The translator is also the module where domain reference is brought in to complete the interpretation of the input question where this cannot be fully interpreted by the analyser alone.

The semantic pattern-matching potential of the translation module can be exploited to determine the nature of the unresolved domain-specific predications (both 'dummy' relationships and those implicit in compound nominals), and vacuously defined objects ('query' variables). Thus the fragment of logical form for "... London suppliers of parts ...", namely

```
(For <quant> $Var1/supplier1
   : (AND
       (For <quant> $Var2/part1
          - (dummy $Var1 $Var2))
       (For <quant> $Var3/London
          - (dummy $Var1 $Var3)))
   - ........ ),
```

is broken down into the corresponding domain predications

```
(&supply $Var1(&supplier) $Var2(&part))
```

and

```
(&live $Var1(&supplier) $Var3(&city)),
```

while translating the logic representation for the example question "Who supplies green parts?" gives the query representation

```
(For Every $Var1/&supplier
   : (For Every $Var2/&part
        : (&colour $Var2 green)
        - (&supply $Var1 $Var2))
   - (Display $Var1)).
```

Apart from the fact that semantic pattern matching seems to cope quite successfully with unexpected inputs ('unexpected' in the sense that in the alternative approach no mapping function would have been defined for them, thus implying a failure to parse and/or interpret the input question), having a general natural language analyser at our disposal offers an additional bonus: the description of the domain world in terms of semantic primitives and primitive patterns can be generated largely automatically, since the domain world can be described in natural language (assuming, of course, an appropriate lexicon of domain world words and definitions) and the descriptions simply analysed as utterances, producing a set of semantic structures which can subsequently be processed to obtain a repertoire of domain-relevant forms to be exploited for the matching procedures.

B. The Convertor

Having identified the domain terms and expressions, we have a high-level database equivalent of the original English question. A substantial amount of processing has pinpointed the question focus, has eliminated potential ambiguities, has resolved domain-dependent language constructions, and has provided fillers for 'dummy' or 'query' items. Further, the system has established that "London" is a &city, for example, or that "Clark" is a specific instance of &supplier. The processing now has to make the final transition to the specific form in which questions are addressed to the actual database management system. The semantic patterns on which the translator relies, for example defining a domain word "&supplier" as (((***ent obje) give) (subj *org**)), while adequate enough to deduce that Clark is a &supplier, are not informative enough to suggest how &suppliers are modelled in the actual database.

Again, the obvious approach to adopt here is the mapping one, so that, for instance, we have:

```
&supplier ==> relation Supplier
Clark     ==>
     tuple of relation Supplier
       such that Sname="Clark"
........
```

But this approach suffers from the same limitations as direct mapping from logic representation to search representation; and a more flexible approach using the way the database models the domain world has been adopted.

In the previous section we discussed how the translator uses an inventory of semantic patterns to establish the connection between natural language and domain world words. This inventory is not, however, a flat structure with no internal organisation. On the contrary, the semantic information about the domain world is organised in such a way that it can naturally be associated with the administrative structure of the target database. For example in a relational database, a relation with tuples over domains represents properties of, or relationships between, the objects in the domain world. The objects, properties and relationships are described by the semantic apparatus used for the translator, and as they also underlie, at not too great remove, the database structure, the domain world concepts or predications of the query representation act as pointers into the data structures of the database administrative organisation.

For example, given the relation Supplier over the domains Sname, Sno, Status and Scity, the semantic patterns which describe the facts that in the domain world &suppliers &have &status, &numbers, &names and &live in &cities are crosslinked, in the sense that they have the superstructure of the database relation Supplier imposed over them. We can thus use them to avoid explicit mapping between query data references and template relational structures for the database. From the initial meaning representation for the question fragment "... Clark, who has status 30 ..." through to the query representation, the semantic pattern matching has established that Clark is an instance of &supplier, that the relationship between the generic &supplier and the specific instance of &supplier (i.e. Clark) is that of &name, and that the query is focussed on his &status (whose value is supplied explicitly). Now from the position of the query predication (&status &supplier 30) in the characterisation of the relation Supplier, the system will be able to deduce that the way the target database administrative structure models the question's semantic content is as a relation derived from Supplier with "Clark" and "30" as values in the columns Sname and Status respectively.

The convertor thus employs declarative knowledge about the database organisation and the correspondence between this and the domain world structure to derive a generalised relational algebra expression which is an interpretation of the formal query in the context of the relational database model of the domain. We have chosen to gear the convertor towards a generalised relational algebra expression, because both its simple underlying definition and the generality of its data structures within the relational model allow easy generation of final low-level search representations for different specific database access systems.

To derive the generalised relational algebra form of the question from the query representation, the convertor uses its knowledge of the way domain objects and predications are modelled in the database to establish a primary or derivable relation for each of the quantified variables of the query representation. These constituents of the algebra expression are then combined, with an appropriate sequence of relational operators, to obtain the complete expression.

The basic premise of the convertor is that every quantified variable in the formal representation can be associated with some primary or computable relation in the target database; restrictions on the quantified variables specify how, with that relation as a starting point, further relational algebra computations can be performed to model the restricted variable; the process is recursive, and as the query representation is scanned by the convertor, variables and their associated relational algebra expressions are bound by an 'environment-type' mechanism which provides all the necessary information to 'evaluate' the propositions of the query. Thus conversion is evaluating a predicate expression in the context of its semantic interpretation in the domain world and the environment of the database models for its variables.

For example, given the query representation fragment for the phrase "... all London suppliers who supply red parts ...", namely

```
(For Every $Var1/&supplier
  :(AND
     (For The $Var3/London - (&live $Var1 $Var3))
     (For Every $Var2/&part : (&colour $Var2 red)
        - (&supply $Var1 $Var2))) ....,
```

$Var1 will initially be bound to the primary relation <u>Supplier</u>, which will be subsequently restricted to those tuples where Scity is equal to "London". Similarly, $Var2 will be associated with a partial relation derived from <u>Part</u>, for which the value of Colour is "red". Evaluating the proposition (&supply $Var1 $Var2), whose domain relationship is modelled in the database by <u>Shipments</u>, will in the environment of $Var1 and $Var2 yield the relational expression

```
(join
  (select Supplier where Scity equals "London")
  (join Shipments
    (select Part where Colour equals "red"))).
```

At this point, the information that the user wants has been described in terms of the target relational database: names of files, fields and columns. The search description has, however, still to be given the specific form required by the back-end database management system. This is achieved by a fairly straightforward application of standard compiling techniques, and does not deserve detailed discussion here. At present we can generate search specifications in three different relational search languages. Thus the final form in the local search language Salt of the example question "Who supplies green parts?" is

```
list (Part:Colour="green"
        * (Supplier * Shipments))
```

V IMPLEMENTATION

All of the modules have been implemented (in LISP). The convertor is at present restricted to relational databases, and we would like to extend it to other models. The system has so far been tested on Suppliers and Parts, which is a toy database from the point of view of scale and complexity, but which is rich enough to allow questions presenting challenges to the general semantics approach to question interpretation. To illustrate the performance of the front end, we show below the query representations and final search representations for some questions addressed to this database. Work is currently in progress to apply the front end to a different (relational) database containing planning information: this simulates IBM's TQA database (Damerau 1980). Most of the work in this is likely to come in writing the lexical entries needed for the new vocabulary. Longer term developments include validating each step of the translation by generating back into English, and extending the front end, and specifically the translator, with an inference engine.

Clearly, in the longer term, database front ends will have to be provided with an inference capability. As Konolige points out, in attempting to insulate users, with their particular and varied views of the domain of discourse, from the actual administrative organisation of the database, it may be necessary to do an arbitrary amount of inferencing exploiting domain information to connect the user's question with the database. An obvious problem with front ends not clearly separating different processing stages is that it may be difficult to handle inference in a coherent and controlled way. Insofar as inference is primarily domain-based, it seems natural in a modular front end to provide an inference capability as an extension of the translator. This should serve both to localise inference operations and to facilitate them because they can work on the partially-processed input question. However the inference engine requires an explicit and well-organised domain model, and specifically one which is rather more comprehensive than current data models, or than the rather informal conceptual schema we have used to drive the translator.

We hope to begin work on providing an inference capability in the near future, but it has to be recognised that even for the restricted task of database access, it may prove impossible to confine inference operations to a single module: doing so would imply, for example, that compound nouns will generally only be partly interpreted in the analysis and extraction phases. Starting with inference limited to the translation module is therefore primarily a research strategy for tackling the inference problem.

* Green parts are supplied by which suppliers?

  + query representation:

```
(For Every $Var1/&supplier
  :(For Every $Var2/&part : (&colour $Var2 green)
      -(&supply $Var1 $Var2))
  -(Display $Var1 ))
```

  + search representation in Quel:

```
Range of Ql-var1 is Part
Range of Ql-var2 is Supplier
Range of Ql-var3 is Shipments
Retrieve into Terminal (Ql-var2.Sname)
         where (Ql-var1.Pnc = Ql-var3.Pnc)
         and   (Ql-var2.Snc = Ql-var3.Snc)
         and   (Ql-var1.Colour = "green")
```

* From where does Blake operate?

  + query representation:

```
(For The $Var2/&city
  :(For The $Var1/Blake - (&live $Var1 $Var2))
  -(Display $Var2 ))
```

  + search representation in Quel:

```
Range of (Ql-var1) is (Supplier)
Retrieve into Terminal (Ql-var1.Scity)
         where (Ql-var1.Sname = "Blake")
```

* What is the status of the Paris part suppliers who supply blue parts?

  + query representation:

```
(For Every $Var1/&supplier
  :(AND
    (For Some $Var2/&part - (&supply $Var1 $Var2))
    (For The $Var3/Paris - (&live $Var1 $Var3))
    (For Every $Var4/&part
      :(&colour $Var4 blue)
      -(&supply $Var1 $Var4)))
  -(Display (&status $Var1) ))
```

  + search representation in Quel:

```
Range of Ql-var1 is Part
Range of Ql-var2 is Supplier
Range of Ql-var3 is Shipments
Retrieve into Terminal (Ql-var2.Status)
         where (Ql-var1.Pnc = Ql-var3.Pnc)
         and   (Ql-var2.Snc = Ql-var3.Snc)
         and   (Ql-var2.Scity = "Paris")
         and   (Ql-var1.Colour = "blue")
```

VI CONCLUSION

The project results so far suggest that developing a natural language front end to databases based on a general semantic analyser which constructs rich and explicit meaning representations offers distinct advantages in at least two respects: it makes all subsequent processing cleaner than would be the case with a representation dominated by conventional syntax, and enhances portability by encouraging the declarative description of domain-specific knowledge.

VII REFERENCES

Boguraev, B.K. "Automatic resolution of linguistic ambiguities", Technical Report No.11, Computer Laboratory, University of Cambridge, 1979.

Boguraev, B.K. and Sparck Jones, K. "A natural language analyser for database access", Information Technology: Research and Development, 1, 23-39, 1982.

Bronnenberg, W.J.H.J. et al. "The question answering system PHLIQA1", in Natural language question answering systems (Ed. Bolc), London: Macmillan, 1979.

Damerau, F.J. "The transformational question answering (TQA) system: description, operating experience, and implications", Report RC8287, IBM Thomas J. Watson Research Center, Yorktown Heights, N.Y., 1980.

Date, C.J. An introduction to database systems, Reading, Mass.: Addison-Wesley, 1977.

Grosz, B. et al. "DIALOGIC: a core natural-language processing system", in Proceedings of the Ninth International Conference on Computational Linguistics, Prague, 1982.

Hendrix, D.G. et al. "Developing a natural language interface to complex data", ACM Transactions on Database Systems, 3, 105-147, 1978.

Konolige K. "A framework for a portable natural-language interface to large data bases", Technical Note 197, Artificial Intelligence Center, SRI International, 1979.

Waltz, D. "An English language question answering system for a large relational database", Communications of the ACM, 21, 526-539, 1978.

Warren, D.H.D. and Pereira, F.C.N. "An efficient easily adaptable system for interpreting natural language queries", Research Paper 155, Department of Artificial Intelligence, University of Edinburgh, 1981.

Wilks, Y. "An intelligent analyser and understander of English", Communications of the ACM, 18, 264-274, 1975.

Woods, W.A. "The lunar sciences natural language information system", Final Report, Bolt, Beranek and Newman Inc., Cambridge, Mass., 1972.

Woods, W.A. "Semantics and quantification in natural language question answering", Advances in Computers, 17, 1-87, 1978.