Inference in a Natural Language Front End for Databases

Final Report on SERC Grant GR/C/98825; Alvey Project IKBS 019
(Natural language processing)

Ann Copestake and Karen Sparck Jones
Computer Laboratory, University of Cambridge
New Museums Site, Pembroke Street, Cambridge CB2 3QG, UK

Febuary 1989

Abstract

This report describes the implementation and initial testing of knowledge representation and inference capabilities within a modular database front end designed for transportability.

Contents

Acknowledgement

We are very grateful to Dr John Tait of ICI plc, our Monitoring Officer, for his comments, criticisms, suggestions and general help, especially in causing us to programme our work more effectively than we would otherwise have done.

Chapter 1

Preface

This grant was for a programmer to support the Computer Laboratory group working on natural language processing, for whom database access was a key test area. In the event, the need for miscellaneous, general-purpose support was somewhat less than envisaged, and we concentrated on following up a particular approach to database access that had been developed under earlier SERC projects (Sparck Jones and Boguraev 1983, Boguraev and Sparck Jones 1984, 1985). In the group as a whole there are several lines of work involving database query, but the philosophies being pursued are somewhat different, and there was no obvious way of combining them in a single project, particularly given the additional constraints represented by other factors like the projects' various goals, timings, and administrative structures. The particular line we followed had been the major focus of effort in the group's previous research, and the immediately prior project had been a design study for further work. The research we have carried out has therefore been to implement and evaluate this design, with the group's other work as a local comparative context.

Thus in the project we have taken our earlier model for a database front end aimed at transportability and involving first application-independent syntactic and semantic processing and then application-dependent processing, and extended it to allow the inference often needed to interpret natural language database questions in complex domains. Consideration of these inference needs and other requirements of transportable front ends has led us to build a new system, which retains the old modular structure but which also incorporates a knowledge base and inference component. Some difficulties with computing resources meant that we have not done as much testing as we wanted, but we have tested the new system in a not completely trivial way on two domains, including one complex one.

We have therefore demonstrated that it is feasible to use restricted domain-independent knowledge combined with domain-dependent knowledge and limited inference to extend the range of user input which can be interpreted by the system. This architecture also shows potential as the basis of a system with

expanded capabilities, such as handling user misconceptions. However we have had problems in our attempt to use an existing formalism for our knowledge base and the issue of knowledge acquisition remains problematic.

Chapter 2 of the Report summarises the framework and starting point for the project and describes the new structure for the front end which has followed from the provision of knowledge representation and inference capabilities. Chapter 3 discusses the knowledge base and inference components in detail, indicating the problems to be addressed using the proposed representation scheme and the strategies adopted to solve these, and describing the nature of the inference specialists implemented and the way these figure in input question processing. This was the core work of the project, and this chapter includes some comparisons with other approaches to representation and inference. The front end inference operations are concentrated in the translation and conversion stages in processing, where input question elements are transformed to concepts in the application domain and then to concepts in the database itself; Chapter 4 describes the other processing modules. Chapter 5 presents test examples, illustrating the performance of the new front end. The concluding Chapter 6 considers the project as a whole, compares it with other approaches and suggests some directions for future research.

Chapter 2

Introduction

This chapter summarises the essential features of our starting point for the project and outlines the overall structure of the final system. Section 2.1 characterises the existing front end design and describes the town-planning domain from which most of the examples mentioned are taken. Section 2.2 describes the inference needs to be met, the form of knowledge base we proposed to use, and the way we suggested this should be used for inference. This material is presented at length, with numerous illustrations, in Boguraev, Copestake and Sparck Jones 1986, and we have reproduced sections 1 - 3 of this paper for convenient reference as Appendix A here. Section 2.3 more explicitly specifies the requirements that a system such as ours should ultimately address and mentions some of the restrictions imposed by the application. Section 2.4 describes the effects the implementation of the knowledge representation and inference capabilities had on the overall structure of the front end.

2.1 Background: the existing system

Our research aim was to test the approach to the provision of an inference capability for natural language front ends for databases proposed in a previous study project (Boguraev and Sparck Jones 1985). In earlier work we had built a front end based on the idea that a significant part of the work of input question interpretation could be done using general semantic as well a syntactic information, with important gains for front end transportability (Boguraev and Sparck Jones 1983, 1984). The disambiguated input in the form of an explicit meaning representation would then be processed using domain- and database-specific knowledge. The overall front end design was therefore modular, with two major application-independent and application-dependent sub-systems each in turn with two constituent processors, as shown in Figure 2.1. The processing technique involved was a form of pattern matching, which allowed transformation of a varied range of input forms into a database query. The functions of the modules

and the different representations which they output, which apply to the current work, are summarised here; for further details see Section 1 of Appendix A.

The application-independent modules produce a logical form which is essentially independent of the application domain. Thus the *logic representation* output by the *extractor* has the general form of the quantified expressions produced by LUNAR (Woods 1972) but contains predicates corresponding to ordinary word senses; these are represented by labels such as 'own1' for example. The *query representation* that the *translator* produces contains domain-referring predicates which are indicated by an '&' preface and a tag indicating the particular domain; **&own-GH** for example, refers to the Green Hills domain (see below). Especially because of the severe limitations on the relational data model's ability to express the semantics of the data represented, it is necessary to provide a model of the application which is not tied to the relational model; this is what we refer to as the *domain* level description. By contrast the *schema* level incorporates the description in terms of the relational data model. The *search representation* finally produced by the *convertor* is in the syntax of a relational database query language and therefore contains expressions referring to the names of relations and columns in the particular schema. For clarity, items in the domain and schema languages are indicated by a bold font in the text of this report.

Note that *domain* is being used here, and throughout the rest of this report, in a rather restricted way, to refer to the coverage of the actual application database and not to some broader area which might be regarded as involving specialist knowledge and vocabulary (such as computing or town planning). Thus even vocabulary which might have a specialised use (eg "bus", "compiler") is not domain-specific in the sense we are using here.

Our modular design though motivated, as in other front end projects, by the need for transportability, had other advantages: in particular the different representations formed a base for different kinds of feedback to the user (Boguraev and Sparck Jones 1984).

In the earlier work the front end was developed and tested on a very simple domain, Suppliers and Parts (Date 1977), and a first pass port was done to a more complex one, involving town planning. Problems arose because the complexity of this new domain allowed for more variation and, more importantly, more indirection, in the expression of queries. It became clear in these cases that interpretation required inference on supporting and amplifying knowledge. The results of the study project, which described the inference needs and the proposed system extensions to deal with them, are summarised in the next section.

Though some of the examples in the current report refer to Suppliers and Parts, most are taken from the richer town planning domain. This application, Green Hills, is roughly based on the IBM TQA Project's White Plains (Damerau 1980, 1985); our aim was to use some real, rather than trivial, material. Dr Damerau kindly supplied the complete TQA attribute list, but as we could not, for proprietary reasons, use any actual White Plains data, we invented data for

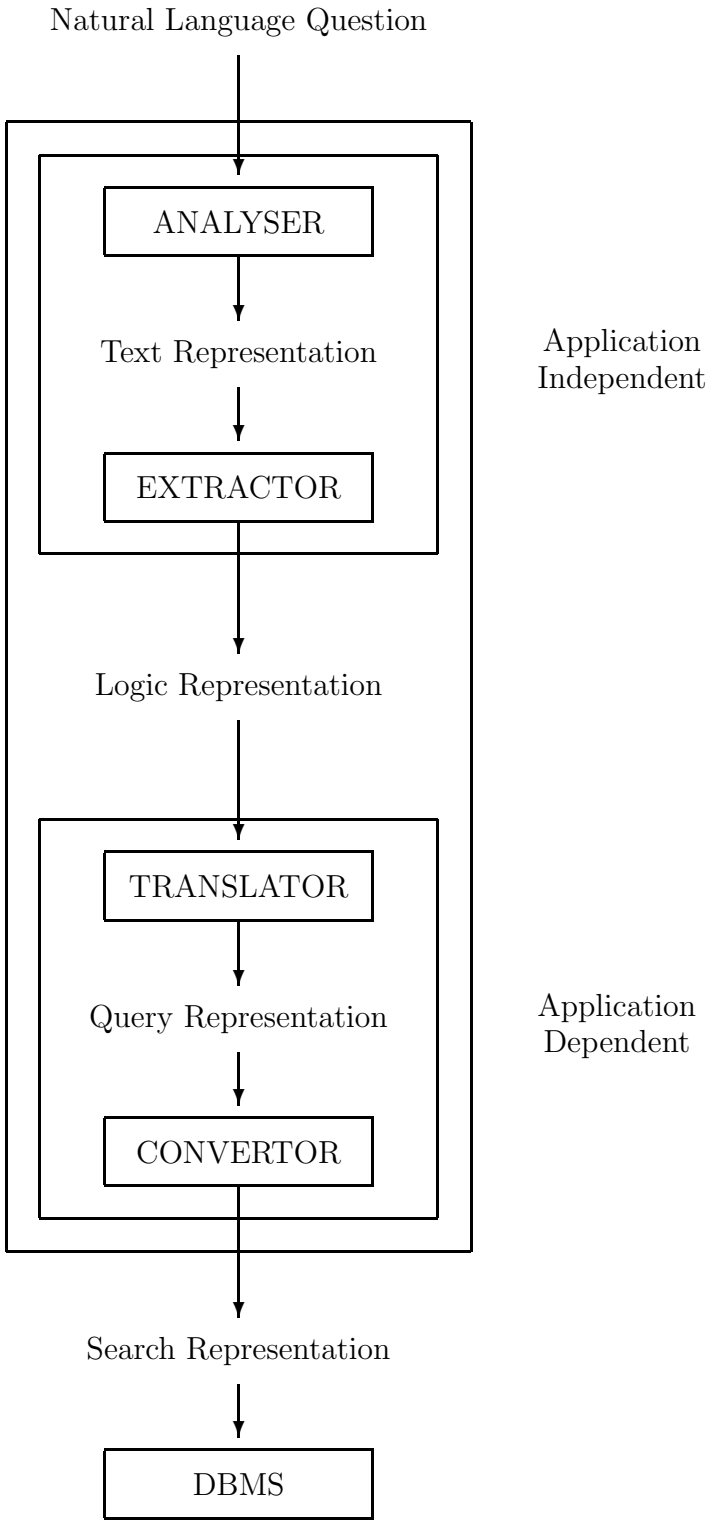

Figure 2.1: Original system structure

an imaginary English village, Green Hills. The essential structure of the Green Hills domain is that it deals with parcels of land and their owners; the parcels are grouped into blocks which are themselves organised into wards. Most of the information is about the parcels: area, number of stories, assessment for taxes, and land use codes (LUCs), for example. The LUC is a three digit code which specifies the parcel use (for example 541 might be the land use code corresponding to a supermarket).

Some of the possible queries on this domain (taken from the TQA project) include:

> How many dwelling units are on Craven Lane?
> Who owns the vacant land in Hillair Circle?
> What is the parcel area of Roane's property?
> Where are the gas stations in Fisher Hill?

The complete schema for Green Hills is included in Appendix A. A subset was created for initial test purposes which incorporated a cross-section of the different types of attributes and relationships within the domain; this is used in the examples in this report and is shown in Figures 2.3 and 2.2. Some changes were made to normalise the full schema given in Appendix A to avoid unnecessary problems with the same information being accessible by multiple paths.

2.2 The need for inference

This section summarises the material in Sections 2 and 3 of Appendix A which give examples of inference needs to be met by a database front end and describes a possible approach towards meeting them, put forward in the previous study project.

It was evident that there are what may be described informally as different types of processing requirement, although the proposal was to meet them operationally in an integrated way. In transforming one representation into another it is necessary to find equivalent elements and also to meet constraints, which may involve inferring additional information.

Consider, for example, the stages involved in transforming

> Which owners are in Market Place?

into a valid database query. The output of the extractor is:

```
(For THE VARA1/OWNER1
   : (For THE VARA3/PLACE2
        : (NAME2 VARA3 "Market Place")
        - (BE5 VARA1 VARA3))
   - (Display VARA1))
```

Wid	ward number
Bid	block number
Pid	parcel number
Oid	owner identifier
Surnam	owner surname
Inits	owner initials
Strnum	street number
Strnam	street name
Luc	land use code
Park	number of parking spaces
Dwell	number of dwelling units
Fl	number of floors
Cityv	assessment for city district
Sqft	parcel area

Figure 2.2: Attributes in the Green Hills database

OWNER	OOid OSurnam OInits
OWNERSHIP	**OWOid OWPid**
PARCEL	PPid **PBid** PStrnum PStrnam PLuc PPark PDwell PFl PCityv PSqft
BLOCK	BBid **BWid**
WARD	WWid

Keys are underlined, foreign keys are in bold type.

Figure 2.3: Relations in the Green Hills database

To translate this into domain terms it is necessary to find the domain equivalents of owner1, place2, name2 and be5. Some of these concepts may have domain equivalents which have been explicitly specified, but in other cases, such as place2 in this example, finding the domain equivalent (&street-GH) requires inference on knowledge which need not all be domain-specific. It is then necessary, since there is no information about owners' locations in the domain, to find some plausible connection between **&owner-GH**s and **&street-GH**s. The query can be interpreted as the equivalent of

Which owners own properties which are in Market Place?

which links owners and streets through the concept of property. The translated expression should therefore be:

```
(For THE VARB1/&OWNER-GH
   : (For SOME VARB9/&PARCEL-GH
        : (For THE VARB3/&STREET-GH
             : (&STREET-NAMED-GH VARB3 "Market Place")
             - (&PARCEL-IN-STREET-GH VARB9 VARB3))
        - (&OWN-GH VARB1 VARB9))
   - (Display VARB1 ))
```

Converting the translated expression to a database query involves finding the intermediate relation which represents ownership, and the correct columns to join over. The output *search representation* expressed in QUEL is:

```
range of VARC14 is OWNERSHIP
range of VARC9 is PARCEL
range of VARC1 is OWNER
retrieve unique (VARC1.ALL)

where (VARC9.PSTRNAM = "MARKET PLACE"
and (VARC1.OOID = VARC14.OWOID
and VARC9.PPID = VARC14.OWPID))
```

As this example shows different types of inference on three levels of information (domain-independent, domain-dependent and schema-dependent) are needed, and the information on the different levels has to be linked.

The primary reason for suggesting the use of a single knowledge base linking different kinds of knowledge was to avoid duplication of information in order to maintain integrity and simplify knowledge acquisition. It also appeared that cross access from one module to the type of information naturally associated with another might be needed, and specifically access from application-independent modules to application knowledge. (The general issues thus raised for the front end philosophy are mentioned later in Section 6.2.) However it was not initially

apparent exactly what these access needs would be. Furthermore a single, general purpose knowledge base seemed more likely to support further expansion of the capabilities of the system, to allow dialogue rather than single-shot questions, for example.

The need was therefore for a form of knowledge base which would allow an easy linking of one piece of information to another and of one representation level and language to another, and also a form that would naturally support what appeared to be the relatively shallow characterisation of knowledge and limited types of operation on it that seemed appropriate in the transportable front end case.

We proposed, at least as a starting point, to use Alshawi's Memory knowledge base formalism (Alshawi 1987) which had already been exploited in a front end to databases, albeit one designed for database creation rather than querying. We envisaged a set of general-purpose inference specialists built from primitive network operations, which could be invoked in a parametrised way from different processing modules. Thus the general structure of the new front end would be as shown in Figure 2.4. Each of the successive representations generated by the processing stages in question interpretation can be viewed as being couched in some meaning representation language referring to some real or model world (Sparck Jones 1983). The knowledge base would link concepts and expressions both for any one representation language and across languages, and so allow inferences required to transform one representation of an input question into another.

2.3 The (ultimate) aims of our approach

The study report did not give much attention to what was actually meant by the portability of the system. It was stated that it should be possible to transport the system to a new domain by *adding* new information, but did not suggest strategies for acquiring the new knowledge or what might be involved. The type of end user involved was to some extend implicit in the examples given but was not explicitly stated. The following section therefore attempts to summarise what we assumed in this project would be the (longer term) requirements for a portable natural language interface to databases, since these had major implications for the design of the current system. Even though actually achieving these objectives was way beyond the scope of the project, if we had produced a design that did not attempt to take them into account we could only have built a system with an inherently limited architecture and scope. Such a system would have had better performance than ours, certainly on the test domains, but the goal of this project, given its very limited human resources, was rather to investigate an approach, so the actual system was only relevant as a test-bed for ideas.

A design for a portable natural language interface must consider two classes

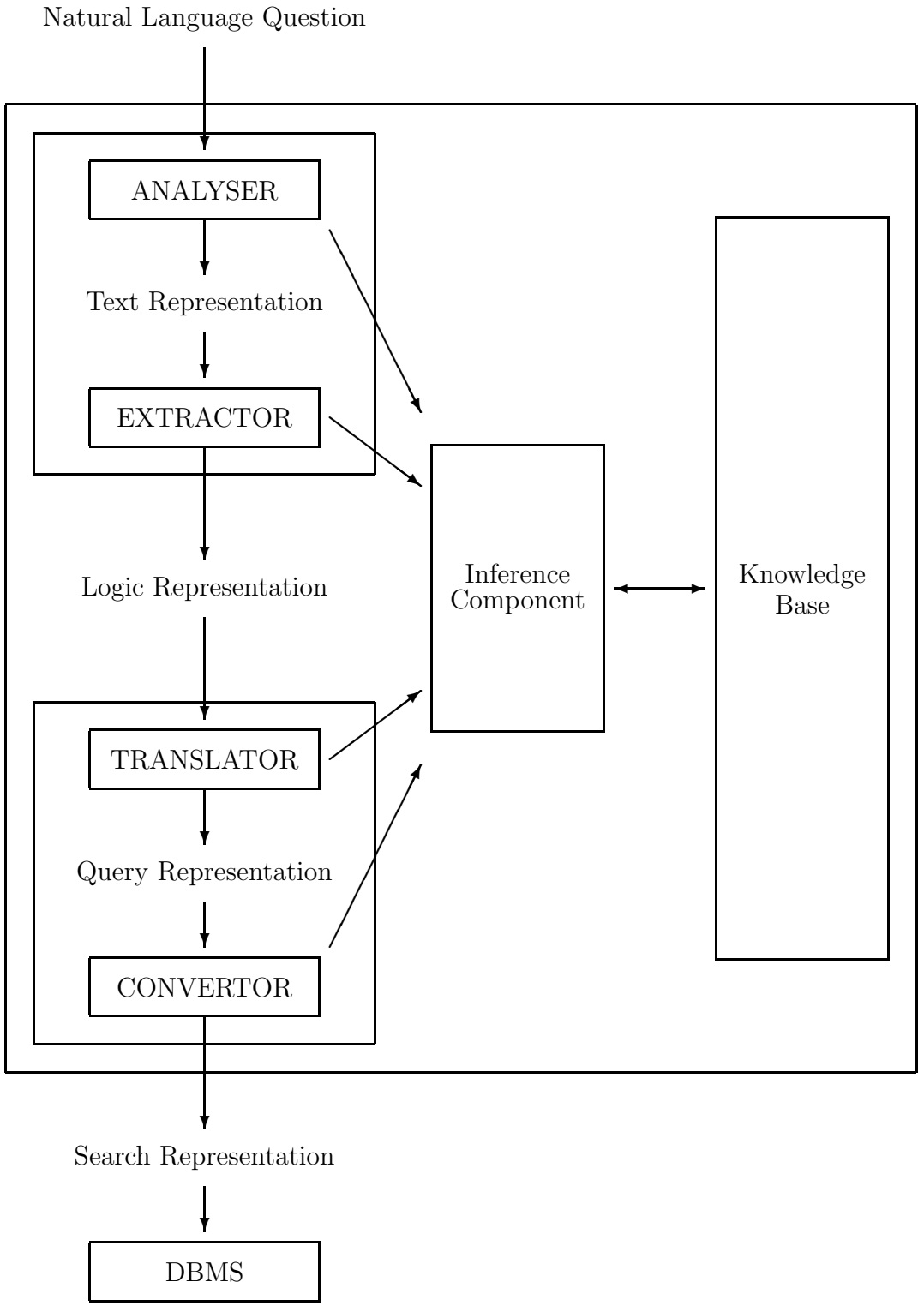

Figure 2.4: Proposed system structure

of users; end users and those responsible for transporting the system to new domains. The interface must interpret natural sentences[1] produced by an end user who has no knowledge of the database schema, and a less than perfect knowledge of the domain. This implies that the system must have some inference capability, some way of describing the domain to the user, and some means of detecting and correcting user misconceptions. The person responsible for porting the front end to a new domain can be expected to have extensive domain knowledge and knowledge of the particular database schema, but should not have to spend a large amount of time learning how to use the system and in doing the actual porting, and should not need expertise in linguistics or knowledge representation (other than that which is appropriate to databases). A problem here is that the knowledge encoded in the conventional data models (the relational model for example) is only a small fraction of the domain-specific knowledge needed to actually understand the data.

There are also requirements that arise from interfacing to a traditional database as opposed to a Prolog knowledge base (for example). It must be possible to handle existing databases and conventional commercial DBMS; so to retain efficiency, as far as possible, a single question should only result in one query to the DBMS. No data which is stored in the DBMS should be duplicated in the front end. Large and complex databases, where natural language interfaces may really become preferable to the alternatives, must be handled. If these requirements cannot be met the advantages of a traditional database (efficiency, consistency) will be lost.

Unfortunately the requirements of the two types of user conflict directly. To cope with the naïve end user a large amount of knowledge is needed; but the domain-specific part of this at least must be provided by the user responsible for porting the interface. The need for inference is especially awkward given the limited access possible to the data; the interaction between the interface and the database is very restricted. Everything is made worse by the problems of scale; it is not possible, for example, to copy the database's contents into the interface's knowledge base and maximising efficiency of access becomes very important. However what may make this type of enterprise feasible is that the knowledge which databases handle is restricted, and it is possible to represent to scope of that knowledge without copying the entire database. The hypothesis that has to be tested eventually is that it is possible to acquire the restricted domain specific knowledge and to link it to limited general purpose knowledge (the acquisition of this must also be feasible), and that this will provide significantly greater ability to interpret user input than is possible if such knowledge is not present. The goal of the current project was to look at a particular architecture; a greatly expanded system and a much larger project would be needed to investigate the main hypothesis properly.

[1] Ideally both querying and updating the database.

The ways in which our system currently addresses some of the points mentioned above are brought out in Chapters 3, 4 and 5; the incorporation of other necessary extensions into this architecture is discussed in Chapter 6.

2.4 Overview of current system

Originally we made the assumption that inference was something to be invoked when the operation of the original modules failed, and that an inference component would be added on to the existing system. However we subsequently realised that the sort of operations that had been performed by pattern matching in the modules could be validly regarded as a type of inference. This would remove the need for specific triggers of inference and, more importantly, avoid duplicating knowledge. We also wanted to investigate the use of a completely different type of analyser, and to extend the capabilities of the convertor. It therefore seemed reasonable to completely reimplement the system based on these new approaches.

The structure of the current system is more closely described by Figure 2.5 than by Figure 2.4. The staged processing with the various intermediate representations has been retained. The convertor is explicitly split into three parts: the function of the first part is described in Chapter 3, that of the second and third parts in Chapter 4. In fact the convertor was divided in the old system, but the division is stressed here since it affects knowledge base access. The translator module and the first stage of the convertor essentially just call the inference component on their input, as is discussed in Chapter 3. The knowledge base is also accessed by the analyser, extractor and the second stage of the convertor, as described in Chapter 4, but such access is much more specific to the functioning of the individual modules and inference in the same sense is not required. The knowledge base and the different levels of information it contains are described in the next chapter.

The main testing has been done with the Green Hills town planning domain. But as a check we have also continued with the suppliers and parts material of our first project, allowing information about both domains to coexist in the knowledge base as a way of checking the hypothesis that for new applications it should only be necessary to add new knowledge. Some test examples are described in Chapter 5.

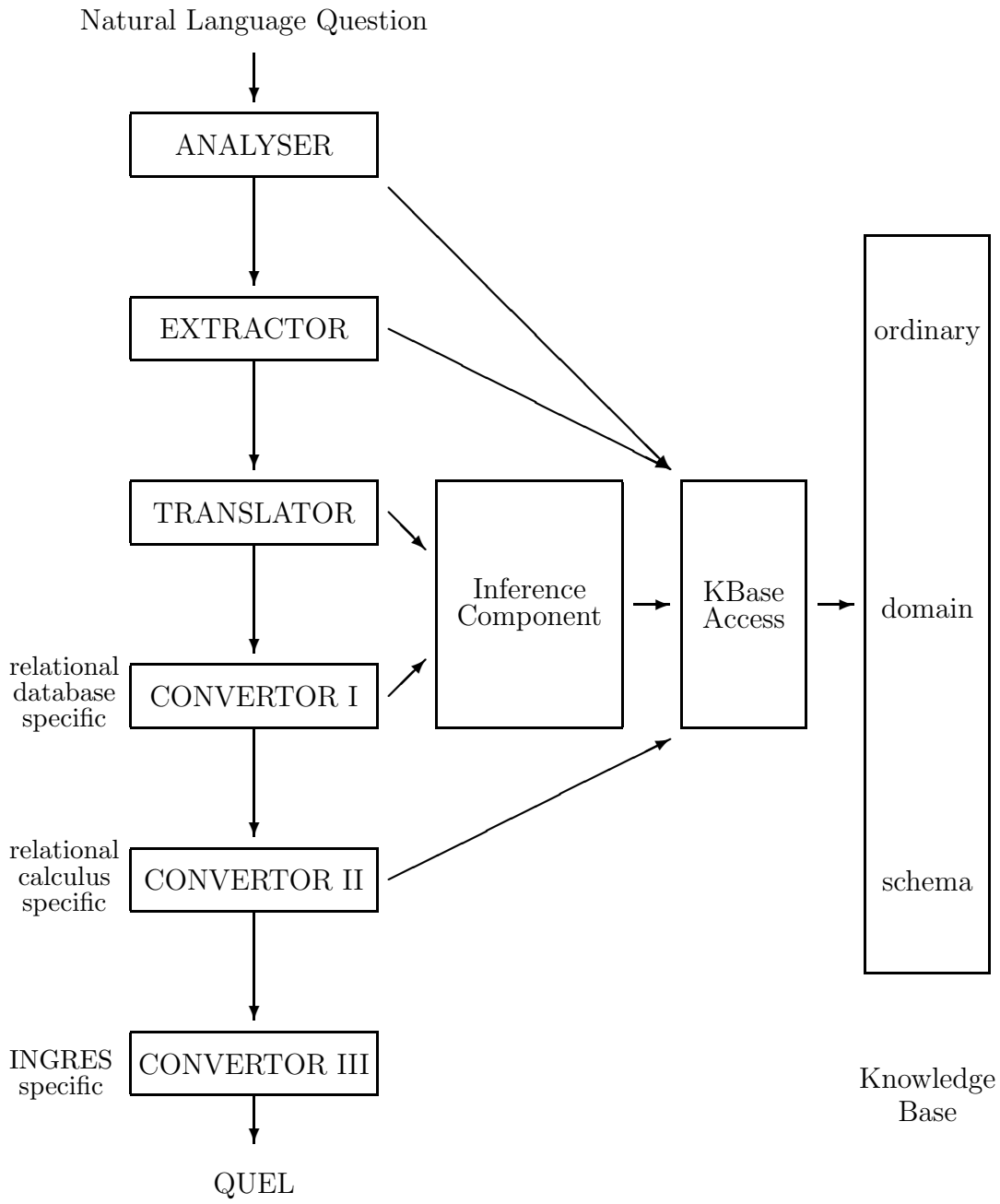

Figure 2.5: Current system structure

Chapter 3

Knowledge representation and inference components

Section 3.1 describes the general issues of the design of the knowledge base which were important for this project. Section 3.2 goes on to illustrate how the MEMORY formalism was used to implement the knowledge base. Section 3.3 covers the types of knowledge represented and considers their acquisition. Section 3.4 details the various types of inference implemented and Section 3.5 compares our strategy with other systems and mentions some of the problems of our approach. Suggestions for future work based on this experience are given in Chapter 6. Note that in this discussion, we use 'relationship' rather than 'relation' as a general term, to avoid confusion with database relations.

3.1 Knowledge base design

As was mentioned earlier an interface to a conventional DBMS must, for efficiency, produce a single database query for each input question, if this is possible. This is an important difference from a system which interfaces with a theorem prover, such as TACITUS (Hobbs et al. 1986), since access to the ground clauses in the database has to be limited; however, as will be seen later, certain of our inference specialists do perform operations with direct analogues in TACITUS. The complete system's information source had to be considered; that is the combination of the target database and whatever knowledge base we provided for the front end. It is necessary to avoid duplicating any of the contents of the database in the knowledge base; to do so would have serious consequences for the consistency of the database. We therefore make two types of distinction between the two components; the first is between the *extensional* and *intensional* parts (Reiter 1978); the database contains the extensional information and the front end's knowledge base contains purely intensional information. Secondly we also considered that the knowledge base should primarily contain *definitional* or *terminological* infor-

mation, rather than *assertions*, which should be stored in the database. The division between the front end's knowledge base and the database (or databases) is therefore analogous to the split between the T-box and A-box in KRYPTON (Brachman, Gilbert and Levesque 1985).

It should be obvious that we are strictly limiting the type of knowledge which we represent in the knowledge base. This seems to be necessary, for two main reasons. First, the portability requirement implies that we should at least start by investigating the use of knowledge which is as limited as possible, and only extend its scope if it seems essential to improve performance. The terminological constraint is well-motivated and fits in with considerations about the portability of general-purpose knowledge which are described later, in 3.3.3. Second, the single database query restriction limits the type of inference which we can carry out in principle, but we can certainly validly make some inferences based on terminological information; this is discussed further in 3.4.2.

In contrast to KRYPTON we have only a very weak coupling of our knowledge base and the database, in that a query has to be completely specified before the database is accessed. Coupling of knowledge bases and databases, to allow inference, has been approached mostly from the viewpoint of expert systems accessing databases, rather than natural language systems; this is briefly discussed in Section 6.3.

Our design for a knowledge base assumes that there are three distinct levels of information; the *schema* level which contains a description of the target database in terms of the relational model, the *domain* level which includes domain-specific information required to understand the database which cannot be encoded in the relational model, and the *ordinary* level which contains general-purpose information. We have further assumed that the knowledge involved is encoded separately for each level, with links between levels which are as simple as possible, to facilitate porting. This should be contrasted with the PHLIQA1 system (Scha 1983), which used an equivalent separation of levels but had translation rules rather than essentially self-contained descriptions. This approach seemed to us to be less suitable for a portable system and not suited to supporting front end functions other than translation. Given the requirement that the various parts of the knowledge base could be used in different ways by different modules, we had to represent them in a reasonably uniform manner and build a limited number of general inference specialists which could perform the required translations.

A further requirement was that we should avoid making the closed world assumption about the material in the knowledge base. This again follows from the portability requirement; we did not want to assume completeness where this could lead to indetectably invalid queries. However it has to be possible to specify completeness where this is known.

Finally the knowledge base had to be able to incorporate temporary assertions made during the processing of input, preferably in such a way that assertions made during processing of earlier sentences could be accessed for anaphor

resolution and so on, as required for extended question sequences or dialogue.

3.2 Use of Memory

The Memory formalism is described in full in Alshawi (1987), and is presented in the context of our proposal to use it for front end inference in Section 3 of Appendix A. This section summarises its essential relevant features.

The formalism can be treated as characterising a semantic network with two kinds of relationship between entities defined by *specialisation* and *correspondence* assertions respectively. Thus we may say

```
(Specialisation A of B)

(Corresponds P to Q as R to S)
```

If ref(X) is the set of objects in the world to which node X refers then

$$\text{ref}(A) \subseteq \text{ref}(B)$$

and if rel(Y,Z) is the set of pairs of objects to which the role-owner pair Y,Z refers then

$$\text{rel}(P, Q) \subseteq \text{rel}(R, S)$$

$$\text{rel}(P, Q) \subseteq \text{ref}(R) \times \text{ref}(S)$$

There is no restriction on the kinds of entity represented — nodes simply stand for concepts — and though there is some notion of a role-owner directionality between P and Q and between R and S, this is not reflected in the semantics of standard correspondence relationships and entities do not have a permanent role or owner status: an entity can be a role in one assertion and an owner in another.

The assertions can be flagged in ways which add additional constraints; for example in

```
(Specialisation C of D (instance))
```

the instance flag adds the condition

$$|\,\text{ref}(C)| = 1$$

to the normal subset constraint. If there is a set of specialisations of some entity F which are flagged by *distinct*

```
(Specialisation E1 of F (distinct))
...
(Specialisation En of F (distinct))
```

then:

$$\text{ref}(E_i) \cap \text{ref}(E_j) = 0 \qquad \text{if } i \neq j$$

Similarly, if

```
(Specialisation E1 of F (cover))
...
(Specialisation En of F (cover))
```

then:

$$\bigcup_{i=1}^{n} \text{ref}(E_i) = \text{ref}(F)$$

Retrieval operations are based on a combination of marking and searching operations. Some retrieval operations will be general purpose and have results which are well defined within the formalism. For example marking down the specialisation hierarchy from a series of nodes, A_1 to A_n, with different marks M_1 to M_n, and then searching for nodes marked with all the marks M_1 to M_n will retrieve all nodes Z such that

$$\text{ref}(Z) \subseteq \bigcap_{i=1}^{n} \text{ref}(A_i)$$

However other retrieval operations might be designed for use in more restricted circumstances. For example, given the following assertions:

```
(Corresponds blockBWidEntry to blockRelp
             as RelpEntry to DbRelp)

(Corresponds wardWWidEntry to wardRelp
             as RelpEntry to DbRelp)

(Specialisation blockBWidEntry of &wid-GH)

(Specialisation wardWWidEntry of &wid-GH)
```

a retrieval operation might consist of marking nodes that were roles of node **blockRelp** with one mark, marking all nodes which were below node **&wid-GH** in the specialisation hierarchy by another mark, and retrieving all nodes marked with both marks. In this context the operation corresponds to retrieving all columns in the **BLOCK** relation which have entries which are ward identifiers.

So considering the needs outlined in the previous section, although the formalism was not suitable for expressing arbitrary predicate logic expressions (and was never intended to be so), it did seem reasonable to use it for the more limited type of expressions that it was obviously essential for us to represent. (It clearly was not going to be possible to express the actual query straightforwardly

in the formalism, but as will be seen later it was not necessary to do this.) It appeared possible to implement the different levels of the knowledge base within the formalism and yet maintain distinctions between them, and since MEMORY had already been used for a database creation task by Alshawi it seemed suitable for representing information about relational databases. Since we did not know exactly what knowledge we would wish to represent the flexibility of the correspondence relationship, when compared to traditional frame representations, seemed advantageous; and similarly, although the possible retrieval operations were not fully defined, it seemed that the primitive marking and searching operations could be used, in the manner outlined above, to build up well-defined operations on larger structures.

3.3 The levels of the knowledge base

The contents of the knowledge base and the use we made of the MEMORY formalism will now be described from the perspective of the different levels of the knowledge base. As will be seen the issues of knowledge representation involved are significantly different for the individual levels. The knowledge base network itself is continuous; distinctions between the levels are maintained during processing by partitioning the knowledge base nodes using permanent marks. However modularity is possible in that the sources for the network are created and stored separately, so that domain files are independent of the ordinary level file and schema files independent of both the domain and ordinary level. Currently one domain must be preselected as the target for a particular sentence but switching domains takes a negligible time, as it is possible to have more than one domain in the memory network at once.

For each level we will attempt to characterise the type of knowledge involved, how this is represented in the system and how such knowledge is or might be acquired. Since the schema level is the simplest we discuss this first and use it to show a concrete example of the use of MEMORY assertions to build up larger structures, in this case the relations of the target database. We then discuss the domain level, and finally the ordinary level, which provides the greatest problems for our approach.

3.3.1 The schema level

The schema level characterises the structure of the database in terms of the relational model. Besides the names and columns of the individual relations, information about the key and foreign key fields is needed and about the connections with the domain level. In addition to this we classify relations according to whether they correspond to kernel or associative entities, in the terminology of RM/T (Codd 1979). We also need to maintain information about how relations

are joined to correspond to particular domain predicates — such joins may be evident from knowledge of the primary and foreign key fields, but in other cases will have to be specified explicitly.

The acquisition of such information is reasonably straightforward, although it obviously requires someone with an understanding of the terminology. The name and columns of each relation are entered (in many DBMS this information could be acquired automatically) and the primary and any foreign keys prompted for. To make the representation simpler, and connections with the domain smoother, we use schema predicates. These are generated automatically for the relationships between each kernel or associative entity and the columns in its relation or relations, and also for the relationships corresponding to associative entities. Other relationships corresponding to domain predicates which translate to single joins have to be specified explicitly. The linkage of the relation entities, columns and predicates to domain nodes will be discussed further in the next section.

In contrast with the approach used in TEAM (Grosz et al. 1987) and other systems, of specifying virtual relations, we are allowing the indirect specification of joins which correspond to particular predicates. We rejected the use of views for several reasons: with a complex database the overhead of specifying and maintaining all the views corresponding to all the predicates would be large, the view mechanism does not cater for all the possible manipulations needed, and in the longer term, update on views is problematic, so the actual relational structure would need to be accessible for this and other operations where consistency is important. However we have made some assumptions about the database structure, in particular that it is in third normal form, and if for some reason an existing database did not meet these criteria it would be necessary to provide a view of it that did for the system to access.

In effect we are assuming that domain predicates which are directly representable in the schema in the way described correspond to "felicitous joins". This mechanism does allow other joins to occur but these would only occur in questions like:

> Which suppliers are in the same city as the parts they supply?

where the columns over which the join occurs are indirectly specified in the query.

Figure 3.1 gives some of the network assertions that correspond to the OWNER relation in the Green Hills schema illustrated in Figure 2.3. As mentioned earlier the individual assertions are in effect being used to build up a larger scale structure with more specific semantics; such a structure has its own set of procedures (built up from the basic network operations) both for retrieval and creation. Network assertions are never written directly by the user in producing a schema level description — the actual form of the assertions created is largely arbitrary and the only constraint that the MEMORY formalism imposes is that the hierarchy is preserved.

```
(Specialisation ownerRelp of DbRelp)
(Corresponds owner to ownerRelp as
             RelationName to DbRelp)
(Specialisation ownerDbEntity of &owner-SGH)
(Specialisation ownerDbEntity of RelpDbKernelEntity)
(Corresponds ownerDbEntity to ownerRelp as
             RelpDbKernelEntity to DbRelp)
(Corresponds ownerOOidEntry to ownerRelp as
             RelpEntry to DbRelp)
(Corresponds OOid to ownerOOidEntry as
              RelpEntryColumn to RelpEntry)
(Specialisation ownerOOidEntry of &oid-SGH)
(Corresponds OOid to ownerRelp as
             RelationKeyColumn to DbRelp)
(Corresponds RelpownerOOid to ownerRelp as
             RelpStatement to DbRelp)
(Specialisation RelpownerOOid of &owner-identified-SGH )
(Corresponds ownerDbEntity to RelpownerOOid as
             &owner-SGH to &owner-identified-SGH)
(Corresponds ownerDbEntity to RelpownerOOid as
             RelpDbKernelEntity to RelpStatement)
(Corresponds ownerOOidEntry to RelpownerOOid as
             &oid-SGH to &owner-identified-SGH)
(Corresponds ownerOOidEntry to RelpownerOOid as
             RelpEntry to RelpStatement)
(Corresponds ownerOSurnamEntry to ownerRelp as
             RelpEntry to DbRelp)
(Corresponds OSurnam to ownerOSurnamEntry as
             RelpEntryColumn to RelpEntry)
(Specialisation ownerOSurnamEntry of &owner-surname-SGH)
(Corresponds RelpownerOSurnam to ownerRelp as
             RelpStatement to DbRelp)
(Specialisation RelpownerOSurnam of &owner-named-SGH)
(Corresponds ownerDbEntity to RelpownerOSurnam as
             &owner-SGH to &owner-named-SGH)
(Corresponds ownerDbEntity to RelpownerOSurnam as
             RelpDbKernelEntity to RelpStatement)
(Corresponds ownerOSurnamEntry to RelpownerOSurnam as
             &owner-surname-SGH to &owner-named-SGH)
(Corresponds ownerOSurnamEntry to RelpownerOSurnam as
             RelpEntry to RelpStatement)
```

Figure 3.1: Network assertions for **OWNER** relation

3.3.2 The domain level

The domain level description has to include information which cannot be made explicit in the relational model (or in any other conventional database model) but which is nevertheless essential to understand the database. In fact some such information is added to our schema level description as outlined above and it is difficult to precisely delimit the information which should be in the domain level. The specification of the felicitous joins has to be made at schema level because reorganisations of the schema with no semantic import (splitting a relation like **PARCEL** into two, each part with **PID** as the key for example) could affect such joins but should not be reflected at domain level. Furthermore, other types of information may be better specified at domain level rather than at the ordinary level even though there is some element of general world knowledge involved.

We wanted at least to start off by using a well-founded method of modelling the domain and therefore decided to use one of the 'semantic' database models as a starting point. Such models have mainly been used as aids to database design, rather than being implemented to provide DBMS. Designing a database is a process of formalising some part of the real world — going from some real information to a very restricted description of some of it. Translating natural language sentences into a database query is a similar process. It therefore seemed plausible that the methods and models used to facilitate the final description of a database might be of interest to us. Also, from a practical point of view, porting to a new domain will be made much easier if a formal description at a suitable level of abstraction exists or can easily be provided by the database manager and can be utilised by the natural language interface.

Many semantic data models have been proposed but we are currently using a binary relationship model. We assume that the domain can be basically described in terms of entities, attributes and predicates. Entities may be simple (kernel) or associative, identifiers are a special class of attributes and the predicates may relate an entity and an attribute or two or more entities. Associative entities must be postulated if there is a many-to-many relationship between two or more entities (eg shipments, ownership).

In the simplest case the domain description would be a straightforward analogue of the database schema with entities corresponding to relations, attributes of those entities corresponding to columns in those relations and inter-entity predicates corresponding to joins. In this case our domain model would map directly onto our schema model. However more usually the natural domain description does not match with the actual schema description. We have attempted to classify some such examples which allows us to give a static description of the domain and schema and provide a limited number of general conversion specialists rather than have to encode each case with separate rules. All these cases can be generalised as examples of objects which are naturally regarded as entities in the domain but do not have full entity status in the schema; that is they do not

correspond to a relation or relations.

Such entities may be represented in the schema by their names or identifiers, by numbers representing the cardinality of a certain set of these entities, by booleans representing their existence, or even by complex attributes which combine what are more naturally thought of as separate entities. These traces allow only limited questions to be asked about the entities concerned, but actually representing them as entities at domain level allows interpretation of queries which would otherwise cause problems. The most straightforward case is where individual entity names are present in the schema; for example **Part_City** and **Supplier_City** in the Suppliers and Parts database. The only way in which this schema is not equivalent to one in which there is a separate city relation is that the existence of a particular city in the database may be dependent on the existence of particular supplier or part tuples. Queries such as

List all the cities!

therefore have a somewhat different status from

List all the suppliers!

since suppliers are not existence-dependent on anything, and the user may need to be made aware of this. This is far more important when update is being considered, of course, since adding new suppliers will be supported by this schema, but adding new cities will not.

There are many examples of entities with more limited representation in the schema. In the Green Hills database there is an attribute which gives the number of parking spaces in a particular parcel, in the Ships database used in the TEAM project there is a boolean value indicating the presence of a doctor on a particular ship. In general we represent all these cases by a domain entity which has an attribute corresponding to its name, cardinality or existence, and connect this attribute to the schema. A query involving such an entity then has to be coerced into one involving the relevant attribute by an inference specialist invoked from the convertor module. If this is not possible then the cause of the failure will be easier to establish and convey to the user than if the entity were not explicitly represented. It is important to note that the same type of field values (ie symbolic, numerical and boolean) can also occur when denoting concepts which are naturally thought of as attributes (such as identifier, weight and presence or absence of a property such as volcanism). Any attempt to directly connect natural language and database seems to inevitably lead to an ad-hoc or sketchy treatment of such phenomena, because such distinctions cannot be made straightforwardly. These types of problems in representation are not a consequence of assuming any particular database model, but arise out of limitations on the kind of data that is covered by the database or even because of the restricted availability of the actual information.

Predicates which are not part of the schema description, but which correspond to combinations of other predicates, and will correspond to joins over several relations, can be specified at domain level. For example in the Green Hills domain **&parcel-in-ward-GH** can be defined in terms of **&parcel-in-block-GH** and **&block-in-ward-GH**. Obviously all combinations of predicates will not be specified in this way; but when such a predicate makes sense in the domain and can be simply expressed in natural language without reference to its component predicates, such definition will be useful. In some cases a natural language expression which corresponds to such a predicate which has not been predefined can nevertheless be interpreted — see 3.4.3 below.

Domain objects can be organised hierarchically; for example **&supplier-city-SP** and **&part-city-SP** can be made specialisations of **&city-SP**. In other cases it is useful to define subtypes of domain entities; for example **&house-GH** might be a specialisation of **&parcel-GH** with a value for **&num-dwelling-units-GH** of **1**. Extending the basic domain model by creating such concepts is not essential, but allows further links to be made to natural language words. In a real application it is envisaged that the transporter would provide the basic domain model and that subsidiary definitions such as that for **&house-GH** could be added incrementally, possibly by the end user, given appropriate acquisition support (TELI (Ballard and Stumberger, 1986) illustrates the feasibility of this).

Crude tools for producing the MEMORY description of the domain from a description in terms of the binary relationship model were implemented for this project. A graphical interface would make this part of the acquisition easier. Adding subsidiary definitions to this model might be done using natural language; this would not involve any significant addition of functionality to the front end — the difficulties that might arise are in maintaining the consistency of the domain description.

3.3.3 The ordinary level

The work here was intended not just to supply enough material to support the test applications' inference needs, especially as these were expected to arise in translation, but to provide sufficiently extensive knowledge to avoid the danger of bias. Thus it was necessary, for methodological reasons, to avoid supplying just enough material to obtain the desired inferences, but no further material which could make it harder to be selective in inferencing. In Alshawi's own experiments there was little more general material than was needed to organise and motivate the characterisation of his application knowledge; so lexical sense selection, for example, though tested, was not very extensively tested.

It was therefore essential to ensure that the ordinary level knowledge covered a fair range of words and not just those contained in a small sample of test sentences or corresponding to domain terms, and also covered a proper range of senses for these words: it is of course impossible to treat all words in detail

without tackling the entire vocabulary. It was at the same time necessary, to avoid rigging results, to ensure that even with a fair size sample of words and senses, there would not be too much conceptual disjunction, so it would be too easy to operate only within the parts of the network relevant to a given question.

This suggested a strategy of taking a vocabulary which would be comprehensive for the application, but allowing extensions into other subject areas as these were prompted by other senses for the 'starting' vocabulary: if at least some of these extensions naturally called for fairly solid treatment, or were given it, the result should be a body of general knowledge which would have some depth in areas other than those immediately associated with the test applications, even though there would inevitably be a fair amount of fringe material. There were, however, unfortunately some limits on the extent to which the various other senses of words with domain interpretations would be involved in processing, especially for sense selection, as for this they had to be in the analyser lexicon, and attempting to include them led to unacceptable degradation in performance due to the lexical ambiguity (see Chapter 4).

It also seemed to be necessary, as an anchor for the input material, to provide some high level structure relating abstract concepts. The natural way of providing this 'upper' structure was via the semantic primitives and primitive classes used to characterise word senses and semantic restrictions for the analyser and figuring in its output *text representation*. Wilks' own grouping of the primitives (cf Sparck Jones 1984) provided a specialisation hierarchy, but it appeared that other relationships like those associated with linguistic case structures and their analogues in the world also needed a schematic high-level characterisation, under which more specific correspondence assertions could be inserted. Again the semantic case labels, or relation primitives, of the analyser's output, had to provide the basis for this.

Essentially, whether or not inference would need to 'back up' to high levels, it appeared that some view of the essential general concepts and their relationships was needed to ensure consistency, coherence and coverage in the treatment of the more particular knowledge on which effective detailed inference would be based. Our belief was that some worked-out higher-level structure was required for the same reason that high-level frames characterising the essential semantic structure of some body of knowledge are required, given that the formalism itself, with its very abstract assertion types, did not provide much guidance here. In particular it was desirable to have some view of abstract nodes essentially functioning as node types, and of standard ways of combining these in assertions, to compensate for the fact that the formalism, when compared with KL-ONE for example, has only two link types, one of which is very unconstraining.

The strategy we adopted in building the general part of the knowledge base was to work from a dictionary, and specifically from Longman's Dictionary of Contemporary English (LDOCE) (Procter 1981). We thought it appropriate, as a general principle, to treat the knowledge base as language-oriented, ie as char-

acterising the world as this is done through language. So we sought, as far as possible, to have natural language word senses as node names in the general knowledge base, as a way of anchoring the base, though it is of course impossible to ensure that words are used in established ways, and in fact some artificial node names had to be invented.

This view was based partly on the fact that the front end is oriented towards language processing, mapping or transforming expressions in one language into those in another, and partly on the fact that the front end is designed for language interpretation of a relatively restricted kind for which we hypothesise that deep knowledge and intensive reasoning are not normally required; it is indeed not obvious how these resources, if they were needed, could be provided in a way compatible with transportability. (We assume that when the system proves inadequate to deal with a particular query this would be best dealt with by interaction with the user.) These points suggested that the knowledge base could and should be built by using dictionary definitions of word senses in a fairly straightforward way. This would mean that the knowledge of the world incorporated in the network would be of a relatively agreed and stable, ie public, kind and at an appropriate, ie plausibly middling, level of detail. The use of the basic defining vocabulary of the LDOCE definitions would in particular serve to encourage coherence and connectivity in the treatment of ordinary world knowledge.

It further appeared that it would be both natural and not too difficult to make the representation of the definitional information more transparent and more consistent if definitions were treated as case structures (either directly or indirectly via their upward links); this would also make the information more accessible to searches driven by the representations of input questions. The representations output by the analyser/extractor include case role information, following Boguraev's earlier work on the analyser; and further work had been done on extending and consolidating the initial set of cases (Boguraev and Sparck Jones 1987). The set was motivated by work on prepositional phrase attachment, so the cases were primarily suited to verb-argument structures, but allowed for some nominal relationships, and appeared to offer a good foundation for our knowledge base work. The approach we proposed to adopt, therefore, was to unpick dictionary entries into case structures, but to avoid as far as possible going further than this: we wanted to avoid any misconceived attempt at radical reductionism and even more the sort of seeking after the truth of the physical world which is too liable to turn into a long term and probably never ending search for the holy grail.

The vocabulary for which we had at least one sense in the lexicon - the starting vocabulary - is given in Appendix B.1. The set of primitives and their specialisation structure is shown in Appendix B.2, and the case list we had available is given in Appendix B.3.

The way LDOCE definitions could be partially converted to specialisation assertions for simple taxonomic cases, and into correspondence assertions for definitions with an identifiable case structure, is illustrated for some straightforward

examples in Figure 3.2. As noted, the presumption is that taxonomic links can

property1 That which is owned (and has some value): possession(s).

property2 Land, buildings or both together.

own1 To possess (something) esp. by lawful right

owner1 A person who owns something, especially one who possesses something by lawful right.

```
(Corresponds property1 to own1Object as WordForm to VerbCase)
(Specialisation property1 of possession3)
(Specialisation property2 of property1)
(Specialisation land5 of property2)
(Specialisation building1 of property2)
(Specialisation own1 of have)
[ Note - LDOCE defines possess1 in terms of own, so we
  connect own1 up to the primitive "have" instead ]
(Corresponds own1Agent to own1 as VerbAgent to Verb)
(Corresponds own1Object to own1 as VerbObject to Verb)
(Corresponds owner1 to own1Agent as WordForm to VerbCase)
```

Figure 3.2: Network assertions for some LDOCE definitions

be continued upwards to primitives, and that more specific correspondence assertions will be similarly subsumed under higher pairs of a more abstract kind, for example,

(Corresponds own1Agent to own1 as VerbAgent to Verb)

(the implications of this last assumption are considered further below). The illustrations of Figure 3.2 show successful applications of our strategy.

We did encounter a number of generic problems, some of them variants, for our representation scheme, of well-known ones. We were able to propose solutions to these, though we did not always carry our implementations very far and in general were not able to evaluate them very fully, because of system development and testing limitations (see Chapter 5).

Thus we had to develop a strategy for dealing with morphological variants as these were encountered in entries like "owner: a person who owns something". We can either construct a direct correspondence relation like 'owner1 is to own1 as VerbAgent is to Verb' or an indirect one with 'owner1' in this relation replaced by

'own1Agent' and 'owner1' being related to this new node. The second approach was adopted since although it proliferates nodes it means that the procedures for accessing the knowledge base can be made simpler and that the knowledge base is less dependent on the particular words which happen to be morphologically well-formed. This leads to structures like that shown in Figure 3.2. Note that we cannot rely on the sometimes arbitrary sets of forms lexicographers have chosen to list; instead we attempt to include as nodes all words that can be regularly derived from a stem. It is also necessary to allow for structures representing linguistic as well as substantive relations, so that, for example, 'assessment1 is to assess1 as NounAction is to Verb'.

We also had some problems (as one always does) with missing case roles. The case set we used had been developed partly following earlier work with the analyser, which focussed on sentence analysis, and partly through a systematic analysis of English prepositions, so it did not always provide the means of handling definitions consisting of noun phrases which implicitly only invoked a weak verb like "be", for example ones containing "of". An example of this is

> floor: a lower limit of prices (sense 4 in LDOCE)

Existing cases like 'poss-by' and 'state' needed interpreting as necessary rather than optional relations, and some new cases, like 'part-of' had to be supplied.

It was further not always easy to handle definitions necessarily linking several concepts with forms like "x: p with a q for doing r".

> block: a piece of wood or metal with words or line drawings cut into the surface of it, for printing (sense 3 in LDOCE)

For the assertions to be connected to the correct concept it may be necessary to adopt arbitrarily-labelled nodes as in Figure 3.3, which conflicts with the general strategy of having actual word-senses as node labels, though experience suggested common patterns for such structures.

Finally, though the emphasis was on the more specific part of the ordinary level base, ie that involving word-senses, since the structures there 'hang off' higher-level ones some thought had to be given to the latter. But it was difficult, given the heterogeneity of the concepts involved (which ranged from ones covering objects, relations and properties in the 'real world' represented by Wilks' primitives and their higher-level group labels to language-oriented ones like 'Verb'), their generally abstract character, and the very general relationships available, to develop any clear idea about what might be desirable, even on the Hobbsian basis that concepts like 'beast' and 'predicate' are equally real. So apart from simply taking over Wilks' hierarchy, we adopted a somewhat schematic and place-holding approach to the higher-level structure, putting in a few relationships as was felt necessary to justify proposed lower structures from a descriptive point of view: our approach indeed means that an elaborate high-level structure is not

```
(Corresponds petrol-selling-garage to petrol-sold-by-garage
             as sell3Loc to sell3Object)
(Specialisation petrol-sold-by-garage of petrol1)
(Specialisation petrol-selling-garage of garage2 (cover))
(Specialisation car-repairing-garage of garage2 (cover))
(Corresponds car-repairing-garage to cars-repaired-by-garage as
             repair1Loc to repair1Object)
(Specialisation cars-repaired-by-garage of car1)
```

These assertions are part of an attempt to define garage2 as a place where cars are repaired or petrol is sold. Note that nodes like cars-repaired-by-garage are necessary to avoid stating that all cars are repaired by some garage.

Figure 3.3: Network assertions for the definition of garage

appropriate. But the weakness of the formalism made it difficult to see whether what was provided was in principle adequate.

There were also problems which are independent of the knowledge representation. Since we have restrict our coverage to the most straightforward concepts possible, there are many aspects of the LDOCE definitions that we had to exclude; it would be pointless to include entries for words such as "belief", for example. In many cases it was not clear what we should attempt to represent and what we should leave out. But the converse problem also applies; extra knowledge, not just interpretation of the definition, may have to be provided; for example to explicitly connect sell1 to buy1. LDOCE definitions cannot be more than a skeleton which has to be filled in by the knowledge base constructor. There are also some peculiarities of LDOCE itself; the number of senses given for a particular word sometimes seems excessive, and in a few cases at least the number of word senses has been halved between the first and second editions.

We felt that our provision of general knowledge for the base was very much a first pass, though we believe we have built enough general network to withstand accusations of total triviality or undue bias. The 120 open class words in the starting vocabulary typically had several senses, some words as many as 20, the starting vocabulary was also not an aggressively technical one. We covered morphological variants of the starting words having their own entries in LDOCE. It is true that we did not have the range of starting words outside the application areas we would have wished, though we did some preliminary work on a larger vocabulary; but the senses of the words in the starting vocabulary ranged far outside the applications, and we imported many senses of other words via the starting definitions. We did not, therefore, build only relevant net, though our

experiments in selecting the correct senses of words occurring in input sentences were, as noted earlier, limited. Overall, the work with LDOCE that was fully incorporated into the knowledge base generated some 1500 assertions involving 840 nodes. The extent to which we actually exploited this information in our tests is discussed in Chapter 5.

We would have liked to have build more of the ordinary knowledge base, to see what kind of, and how much, work was involved. But we found the work very time consuming, and we had to concentrate on the starting words relevant to the application, without having any time left for more. We wanted in any case to see how well what we did construct performed, before continuing the effort. It was also clear that some net building tools, for producing the standard network assertions to describe case structures for example, and more generally for displaying created material, were required to support large scale work, but we did not have the time to provide them. This should be contrasted with the provision of the domain and schema levels, where we were able to completely define the knowledge base structures that were to be used and so to provide tools to construct the network assertions from higher-level descriptions. We envisage that tools based on machine-readable dictionaries would be provided to support large scale acquisition of the type of knowledge that we are using; however, as mentioned above, these can only be tools; it is not feasible to construct a complete knowledge base automatically from a dictionary and therefore building the ordinary level of the knowledge base for a useful system would be a significant undertaking.

In Section 3.5 below we consider the lessons we learnt about the adequacy and convenience of Alshawi's formalism when building our knowledge base and inference specialists; in Chapter 6 we examine its capacity to meet our needs, and the implications of our work with it for our generic front end design.

3.4 Inference operations

This section describes how inference on the knowledge described in the previous sections is carried out, in order first to translate the *logic representation* into the *query representation* and then to convert this into terminology appropriate for the schema. Inference specialists to carry out particular inference operations were built out of combinations of retrieval operations of the type described in Section 3.2. The expressions they transform may arise either directly from the query or from the result of previous application of a specialist. The first three parts of this section describe our three classes of inference specialist; the fourth describes the control of specialist application.

3.4.1 Transformations between levels

An essential feature of our system is the division of the knowledge base into levels. As mentioned above this is motivated by the desire for portability of as much of the knowledge as possible. The most important function of the inference component is to transform the query into an expression that is appropriate for the particular target level. Links between levels have to be specified by the person responsible for porting the database; the approach adopted is to keep such links to a minimum. If **O** is the set of all objects in the higher level and **D** is the set of all those in the lower level, the meaning of links between concept **O1** and the concepts **D1 ... Dn** is

$$D1 \cup D2 \cup \ldots Dn \equiv O1 \cap D$$

.

Such links are indicated by specialisation relationships between nodes which refer to sets of concepts; for example

```
(Specialisation &supplier-SP of supplier1)
```

where supplier1 is marked as part of the ordinary level and **&supplier-SP** is part of the domain level. If a one-to-one link is not possible several domain concepts may be linked to a single ordinary level concept. For example

```
(Specialisation &supplier-name-SP of name1)
(Specialisation &part-name-SP of name1)
```

It can be seen that transformations between levels based on these direct links are valid provided the assumption that the query does in fact pertain to the lower level is valid. For example a transformation of name1(X) to **&supplier-name-SP** (X) $\vee$ **&part-name-SP(X)** is valid if the query is actually intended to be of the Suppliers and Parts domain. (We will assume here that the database to be queried is in fact always known.) The set of objects may of course be further restricted by other parts of the query. Furthermore it is possible to make valid transformations from concepts that are more general than those linked directly to the lower level. For example in the Green Hills domain there is a concept **&street-GH** which would be linked to street1. The concept street1 is a subset of road1 and no other subsets of road1 are specified as being in the domain. The transformation road1(X) to **&street-GH(X)** is therefore valid, if we assume that the knowledge base is complete for hierarchies involving domain concepts. In large complex domains the validity of this assumption will be dubious, but the assumptions being made are weaker than the full closed world assumption.

This mechanism also handles the transformation of cases, such as LOC (location), into the appropriate domain predicate. The extreme case of this mechanism is involved in the interpretation of compound nominals; in the general case nothing may be known about the relationship and so the set of possibilities is all

the two-place predicates in the domain, constrained because the entities involved must match the restrictions on the arguments.

Since the representation of the intra-level subset relationships is also by specialisation links, this is all implemented straightforwardly, by marking down the specialisation hierarchy from the concept to be transformed and finding the most general concept(s) so marked which are also marked as being at the appropriate level, and which meet any other known constraints. This general constraint satisfaction mechanism is therefore the most basic part of the operation of the translator and convertor and simple queries will be transformed without any other operations being involved. The operations that follow will only be invoked if it fails.

3.4.2 Terminological transformations

The second type of transformation is between concepts that have been specified to be equivalent. This is where the definitional aspect of the knowledge base introduced in Section 3.1 is most evident. Transformations based on straight implication are not made: though for example, in Green Hills

$$\text{ref}(\&\text{shop-GH}) \subseteq \text{ref}(\&\text{parcel-GH})$$

we would not transform a query about all the parcels into a query about all the shops. We wish to avoid making the closed world assumption about the knowledge base so if **&shop-GH** and **&postoffice-GH** are the only specialisations of **&parcel-GH** this does not imply that **&shop-GH** and **&postoffice-GH** are the only types of **&parcel-GH** that exist. The specialisation flag type, cover, is intended to allow this type of assertion where it is needed, for example in the Suppliers and Parts domain to specify that

$$\text{ref}(\&\text{supplier-city-SP}) \cup \text{ref}(\&\text{part-city-SP}) = \text{ref}(\&\text{city-SP})$$

Restricting ourselves to such definitional assertions allows us to guarantee that we can maintain the same retrieval set and thus produce a single database query. For example an expression involving **&shop-GH** is transformed into one about **&parcel-GH**s with a **&luc-GH** of **566**; and a query about **&city-SP**s, such as,

> List all the cities!

where no other constraints implied that the retrieval set should be further restricted, would be transformed into a query about **&supplier-city-SP**s $\vee$ **&part-city-SP**s.

A variety of network structures are involved in expressing such information, as was indicated in 3.3.2 and 3.3.3, and it is difficult to produce a stable set of operations on them. Actually encoding this class of specialists has therefore proved to be unnecessarily complicated, and there seems little point in detailing the operations involved here.

3.4.3 Heuristics

Some of the inference specialists actually embody heuristics which are pragmatically motivated[1]. All of these operations correspond to those used in other systems (for example TEAM's pragmatic specialists) — we have attempted to incorporate them within our standard framework of knowledge representation and inference rather than dealing with each case in isolation.

The most important group of heuristics are those associated with coercions. There is a class of questions where interpretation can be made by inserting an additional predicate or predicates, on the basis that in this particular domain there is only one way to connect the entities that are represented in the query. The example discussed in the introduction is of this type, where the question

> Which owners are in Market Place?

is transformed into the equivalent of

> Which owners own properties which are in Market Place?

This appears to be a very standard method but cannot be said to be very well motivated because, although it will allow transformations of questions which may be genuine examples of metonymy (possibly the example above) and ellipsis, it will also allow transformations which may be the result of a user misconception about the contents of the database (again, possibly the example above). Furthermore the use of this heuristic has masked the need for genuine deductive inference, for example in Grosz(1982) it is stated that inference would be required to transform

> Is there a doctor within 200 miles of Philadelphia?

to

> Is there a doctor on board a ship which is within 200 miles of Philadelphia?

yet Ginsparg(1983) suggests using this heuristic to perform this transformation. This will obviously fail on more complex domains, where there may be many connections between the entities for example. Unexpected results also arise; for example,

> Is every supplier white?

will be transformed into the equivalent of

> Does every supplier supply white parts?

[1] ie hacks

It does not seem useful to classify all examples of this type as metonymy (Hobbs and Martin, 1987), and this approach is not going to work well for complex knowledge bases. However we have not found any well motivated way of restricting the application of this heuristic.

In TEAM, words (like "France") that are associated with a sort which is a subsort of *NAME* in the conceptual schema can be coerced into the sort of the object named. Hobbs and Martin also apparently regard this as metonymy. In our system such words have to be assumed to be ambiguous, because of the semantic constraints in the analyser; the invalid interpretation is excluded either by the general-purpose selectional restrictions or later during translation by the domain predicate constraints.

A heuristic which may be slightly better motivated involves avoidance of some tautologies. Consider the example query

Where is the post office in Market Place?

It is obviously inappropriate to give the answer "Market Place" rather than the block identifier or full address, for example. In this case it is simple to detect that the query produced is of something for which the answer is already known. This heuristic could, of course, be given an air of respectability by reference to Grice, but since we are so far off understanding how an interface might observe his maxims this would be somewhat bogus.

3.4.4 Control of application

The classification of inference specialists given above is a result rather than a starting point of our project. Originally the nature of the type of inference that was involved in processing examples such as those given in Appendix A was far less clear and in implementing the new system it therefore seemed appropriate to adopt a control strategy that was as flexible as possible. The blackboard model (Nii 1986) influenced this: each type of inference operation was encoded as a separate specialist which could be applied to part of the expression to be transformed. Intermediate results and control information are kept in the network for ease of access by other specialists — the network is therefore analogous to the blackboard. Several simple control strategies were tried; currently each specialist contains a test to check whether it is appropriate to invoke it; if more than one specialist can be invoked specialists which make transformations between levels of the knowledge base (see 3.4.1) have priority over those which make normal transformations within a level (see 3.4.2) which have priority over the heuristics (in 3.4.3). The specialists only access information in the network itself (via the lower level access functions) plus a flag which indicates whether translation or conversion is occurring.

It would be possible to implement a better defined control structure in the light of our experiences. It would also seem sensible to use a more conventional

language for representing the queries output by the extractor, translator and first stage of the convertor than that of Woods, since we are not using the procedural aspect of his representation at all and it is somewhat over-restrictive and cumbersome to manipulate.

3.5 Evaluation of our approach to knowledge representation and inference

Our system's knowledge base is designed to be more extensive than those of previous natural language interfaces to databases, both in allowing a wider range of domain-specific knowledge to be given and especially in attempting to utilise more general-purpose knowledge. Compared with the TEAM "conceptual schema" and "database schema", for example, the knowledge base contains more detailed information (although it is not entirely clear what types of knowledge could be encoded in TEAM's "pragmatic properties") and is also more general. Ginsparg's (1983) system is more similar because of the attempt to use a more general-purpose form of knowledge representation and to support inference, but the details are too unclear to make a detailed attempt at comparison worthwhile. The recent use of the NIKL knowledge representation in IRUS (Stallard 1986) to support terminological transformations addresses a more limited class of inference than we have attempted to, in a more rigorous manner. It does seem as though it would be possible to expand the use of the domain model in IRUS to fill many of the needs which we are considering. IRUS does not contain an equivalent of our ordinary level, but some of the functions of this are filled by the IRules. The tools provided for knowledge acquisition for complex domains (Ayuso, Shaked and Weischedel, 1987) indicate the sort of support that would be needed before any practically useful experiments could be carried out to evaluate the utility of general purpose knowledge within our architecture. Even with these tools customising IRUS for a new domain is obviously a considerable effort; the question that would have to be considered is how many such transfers would be needed before any benefit was seen from adopting the use of extensive domain-independent knowledge.

In contrast, customising TEAM has been reported to be a relatively trivial operation, but it is far from clear how well this would scale up to complex domains. Customisation is driven by the database schema, and therefore the conceptual schema is very dependent on it. As mentioned earlier a database schema can be very unnatural, furthermore this approach is tied to the relational data model, and so we think that the approach (adopted in IRUS) of concentrating on the provision of a good domain model is preferable.

It is not possible for us to draw any hard conclusions as to whether the limitations we have imposed on the type of knowledge we encode and the types of deductive inference we make are too severe. Since the capability of our system

in this respect is in theory greater than previous systems it seems plausible that a system with these restrictions could be practically useful, even though it is also possible to think of situations that cannot, in principle, be dealt with. It is not obvious how to increase the scope of the inferences without removing the single query requirement. However what seems to be the more important issue is the acquisition of the knowledge; we ourselves have found making the knowledge base to be a very time-consuming operation and although this might be alleviated in various ways there is really no way of telling whether the cost of acquiring extensive domain-independent knowledge would be acceptable without conducting experiments on a greatly enhanced system. Although we cannot make any strong conclusions as to the validity of our general approach we have encountered problems with the particular formalism we have used, as follows.

While the formalism we have been using is not necessarily inadequate in principle for the task, our use of it has become very strained as we have attempted to utilise it for purposes for which it was not designed. In fact since no limitations are imposed on the possible operations on the network, it is difficult to see how MEMORY could be formally inadequate, which has made it difficult to decide at what point we really should not attempt to represent something, and very difficult to decide on an optimum representation. As stated earlier we have in effect been using the knowledge representation to build up other structures which themselves have (implicitly if not explicitly) defined semantics. But achieving modularity of operations on these substructures efficiently is difficult, if we rely on the marking and searching primitives which are designed to be efficient for operations with a much larger scope, and it has proved necessary to allow direct retrieval without marking. As illustrated in 3.3.3 the actual structures that have to be produced if the specialisation hierarchy is maintained can become very complex and unintuitive, and often involve the introduction of nodes with no natural name.

In general the problem is that the flexibility that we thought we would gain to investigate different types of knowledge representation and inference strategy by building on MEMORY as a base has not materialised. Both because the representations built up in this way are not inherently completely defined and because individual operations cannot be combined efficiently, it is in fact more difficult to modify the operations that are performed on the knowledge base (in order to experiment with adding new specialists, for example) than it would have been if there were no underlying structure. As discussed in Chapter 6 any subsequent work should involve producing a knowledge base which would allow representation of the knowledge we have ended up with, using a well-defined language, which could be expanded later as required.

Chapter 4

The rest of the system

In this chapter we discuss the analyser, extractor and the second and third stages of the convertor. These modules are discussed here mainly from the viewpoint of their needs for knowledge base access (see Figure 2.5 above).

4.1 The analyser

In the current system Boguraev's original ATN analyser (Boguraev 1979) was replaced by a parser based on PATRii (Shieber et al. 1983) modified to use a case structure grammar incorporating semantic selectional restrictions based on Wilks' (1975) semantic primitives which were used in the original analyser. Despite the fact that Boguraev's analyser allowed lexical entries to be arbitrarily complex groupings of primitives it had been found in practice, especially when it was used for applications like the database one, that nearly all the disambiguation was dependent on either the head primitive alone or on the head primitive plus a single modifier in a few standardised cases. Therefore only this information was extracted from the old lexical entries. The approach adopted was to incorporate selectional restrictions in the grammar rules and lexical entries so that the semantic checks were applied as part of each grammar rule.

The lexical entry for each word consists of one or more syntactically different entries each of which has one or more semantically different word-senses associated with it. The semantic senses are assumed to be in some preference order. The unifications specified by each grammar rule are in two parts; the first is applied to the syntactic component of each daughter and if unification succeeds the second set of unifications is applied to the semantic components in order of preference. If no successful unification is found the grammar rule application fails. Information about the less preferred structures is stored so that if a rule application subsequently fails because of selectional restrictions it is possible to backtrack. This approach avoids duplicating syntactic information that may be common to several different word senses.

Originally the information needed to allow unification of semantic primitives was encoded in the DAG associated with each word sense. However since Wilks' primitive classes do not form a tree-structured hierarchy this was extremely cumbersome (each primitive class had to have a separately specified binary value). The unification algorithm was therefore modified so that the knowledge base was accessed when attempting to unify primitives.

The knowledge base is also used to allow the preference order of the word senses to be modified depending on the domain being accessed. Thus when the domain is changed word senses which correspond directly to some domain concept are marked and preferred over unmarked word senses. The analyser passes the highest priority structure to the extractor; if subsequent processing fails lower preference structures are tried.

Both these uses of the knowledge base improve speed but are not essential for the functioning of the system. The earlier work identified some cases in which it was essential to access the domain specific information if the query was to be processed; in general the need is to identify domain specific 'words' (for example "3/2" is a block number in the Green Hills domain, "541" is a land use code). It is not possible to include all such items, and also proper nouns, in the lexicon; acquisition by reading the database is hardly viable if update is frequent and anyway failure to parse sentences which include proper nouns that are not database values leads to rejection of some valid queries. The approach adopted here is to allow the morphology of domain-specific strings to be specified as part of the knowledge base, so this is accessed by the parser to check whether such items are valid in the domain and also to retrieve enough information to construct the equivalent of a lexical entry for them. Assertions incorporating the given string are then made in the network so that the subsequent process of translating such items simply involves retrieving this previously determined information. Proper nouns are assigned a dummy lexical entry with a rather general primitive class (*ent) as the only semantic information. Unification will often result in a more specific instantiation of this head primitive slot and this extra information may be used by the translator, which will determine which sort of domain entity the name refers to.

Another use for the knowledge base suggested in the earlier report was to allow disambiguation of structures for compound nouns, by access to the domain specific knowledge. In fact we do not attempt such disambiguation at this stage; instead, as in the old system, a single structure is produced, in effect packing the ambiguous structures, which are then resolved if possible at translation.

In general this work has revealed problems with attempting to achieve fully disambiguated meaning representations using general purpose semantic information of this type. If a separate entry is produced for each word sense in LDOCE it is very unclear how it is possible to disambiguate them using a limited range of semantic primitives. For example five of the senses which are given for 'land' are identical as far as the parser is concerned, so at least five structures could

eventually be produced for any sentence including 'land' (and having no other ambiguity); these structures would be identical apart from the sense number of land.[1] Passing such structures to the translator obviously leads to a great deal of repetition of processing. Applying a preference order alleviates the problem slightly but it has proved impractical in general, in this project, to put all the different word senses in the lexicon. There is also the problem of PP-attachment ambiguity which cannot be resolved by general-purpose selectional restrictions. This is seen quite well in the Green Hills domain which is mostly about places and locations.

Who owns a house in a street with parcels in Block 3/2?

is unambiguous in the domain, but highly ambiguous to the analyser. Possible solutions to these problems will be discussed in Section 6.2.

4.2 The extractor

The extractor transforms the output of the analyser (the text representation) into one or more quantified expressions (the logic representation). The expressions which can be displayed to the user, as illustrated in Section 2.2 for example, do not show the case structure information, as indeed was the case with the earlier system; this information is retained for subsequent processing by making temporary assertions in the network at this stage.

The algorithm used by the extractor is based on that used in LUNAR and described in Woods (1978), in that the process of transforming the unscoped logical form into a scoped logical form involves collecting the quantifier terms into a "store" from which they are extracted when the scope is determined. However we have only implemented very simple scoping rules since very little of what we were primarily interested in investigating depended on input questions with complex quantified structures. As determining correct scoping is such a difficult problem in general it seemed that it would be an unnecessary distraction from the main aims of the project to attempt to deal with it more fully. Compared with systems such as TEAM and even LUNAR, therefore, our treatment of quantifiers is rudimentary. However, as is mentioned in the next section, the limited support for quantification found in certain so-called relational database systems suggests that even a practically useful system could be built with little attention being paid to quantifier scoping in natural language.

We have not tested the assertion, made in the earlier study report, that domain-specific knowledge could be used to disambiguate possible alternative structures in the extractor. However, it seems improbable that it would be sensible to attempt such disambiguation before translation, since transformation of

[1] From the viewpoint of the selectional restrictions they would have been identical even if the fuller lexical entries of the old analyser were used.

the ordinary word senses into their domain equivalents would have to occur before the expressions could be compared with the domain-specific part of the knowledge base. Any domain-dependent inference for scoping would therefore have to be done as part of translation. This would require additional, specific inference processes and also require that the current translator be modified to accept unscoped logical forms, and so it would, in effect, take over much of the function of the extractor. At the moment translation would fail on structures where the quantifier scope violates the domain restrictions, but this is an inefficient way of dealing with ambiguities.

Currently therefore the extractor only accesses the knowledge base in a very straightforward way and does not use any of the inference functions accessed by the translator and the first stage of the convertor.

4.3 The convertor: stages II and III

The structure produced by the first stage of the convertor can be regarded as mixing domain- and tuple- oriented relational calculus in that the variables can range over relations or domains. The relations and columns over which joins have to be made have not been directly specified. For example, the query which has been used in previous examples

> Which owners are in Market Place?

results in

```
(For THE VARC1/OWNERDBENTITY
   : (For SOME VARC9/PARCELDBENTITY
        : (RELPPARCELPSTRNAM VARC9 "Market Place")
        - (RELPOWNS VARC1 VARC9))
   - (Display VARC1 ))
```

being output by the first stage of the convertor. The second stage of the convertor produces a generalised form of tuple-oriented calculus with a syntax similar to that given in Date (1977).

```
range (VARC14 OWNERSHIP)
range (VARC9 PARCEL)
range (VARC1 OWNER)
retrieve ((VARC1 ALL))
where (exists VARC9
        (and ((VARC9 PSTRNUM) = "MARKET PLACE")
             (exists VARC14
               (and ((VARC1 OOID) = (VARC14 OWOID))
                    (VARC9 PPID) = (VARC14 OWPID)))))
```

Two main operations are involved: the explicit introduction of variables ranging over any intermediate relations involved in particular relationships (such as VARC14 in the example above), and the transformation of any variables ranging over domains. An algorithm for all of this had to be devised; access to the knowledge base is necessary but the requirements are quite different from those of the translator and the first stage of the convertor. No attempt is made to optimise the query and sometimes redundant variables are introduced. Obviously any practical natural language interface would have to address the general problem of query optimisation; although DBMS like INGRES perform some optimisations, "semantic" transformations (King 1981) are outside their scope, because they require knowledge of semantic constraints which are not expressible in the standard relational model. In an ideal system a semantic optimiser would utilise the same knowledge base as the natural language front end, but since optimisation needs are not peculiar to natural language interfaces they should be investigated in a wider context. An issue which is related because of the requirement for frequency information, is the detection of queries which are infelicitous because of the processing cost or amount of data involved. For example if a user of the Green Hills database asks

What parcels are there?

and output is being sent to a terminal it is desirable to block the query. This is more likely to be a problem with natural language interfaces than others, not only because of the more naïve user expected, but because such queries may have more restricted interpretations in the context of a dialogue.

Unlike the earlier convertor which could only deal with the select-project-join subset of relational algebra, the current convertor is believed to cope with any valid relational expression. Algorithms for producing relational algebra expressions from the relational calculus are well-known ("Codd's reduction algorithm") and so little point would have been served in this project by reimplementing one of them; therefore relational algebra queries are not produced, although they were in the earlier system.

The relational calculus expression is then converted to a QUEL query. Since QUEL does not really support universal quantification[2] some queries cannot be correctly transformed. Although this is annoying it does suggest that for practical database access a detailed treatment of quantifiers does not have the importance that is sometimes supposed.

[2]See Stonebraker et al. (1976). The example given in Date (1977) page 221 of such a query using ANY(...)= 0 apparently does not work in University INGRES.

Chapter 5

Testing the system

Testing was aimed at investigating the range of examples that could be processed with varying types of information in the knowledge base and with different inference specialists, rather than attempting to get as wide a coverage as possible on the specific test domains. In this chapter we discuss first the various limitations on system coverage and second the contribution various types of knowledge and particular inference specialists make to processing; the discussion is illustrated with examples of questions actually handled. We conclude the chapter by comparing the coverage achieved with the proposals in the study report.

5.1 Limitations on coverage

The most obvious limitation of the system is the restriction to single sentences; no dialogue capabilities were implemented. (This was simply because the project was not intended to do this; more dialogue capabilities are clearly a desideratum for front ends.) The extent of the lexicon has already been discussed; grammar coverage was aimed at providing a reasonably well-motivated treatment of a sufficient fragment of English to cope with fairly natural test examples, but is not nearly as complete as that in TEAM, for example. From the point of view of this project it is more important to discuss the types of sentences which were mentioned as requiring inference in the earlier study report but which could not be handled by the current system.

The provision of facilities for calculation was considered but not implemented so queries like:

> What is the average height of Norwich Road?

cannot be handled. This type of question does not seem to raise any problems which will require radically different or difficult techniques to solve; for example the above question has to be transformed into the equivalent of

> What is the average height of the properties on Norwich Road?

but the query

What are the heights of Norwich Road?

while less natural, poses much the same difficulties without involving calculation. Working out the correct expression in a particular database query language would involve access to the schema level of the knowledge base, so the third stage of the convertor which currently only involves a very simple syntactic transformation would have to be more complex; unfortunately since computational facilities are not part of the relational data model and differ considerably between DBMS, such code could not be general purpose.

Some more interesting sentences are those which were given as examples in the study report, but which we would not now suggest our system should translate. We considered it impossible to deal with some questions in a sufficiently general manner without great extensions to the capability of the system; it is possible that these questions should be treated as user misconceptions for which the system should not attempt to produce a database query, but should react in some other appropriate way. For example it was suggested in the study project that the query

Has the second school in Ward 2 been demolished yet?

could be transformed into the equivalent of

Is the number of schools in Ward 2 equal to 1?

Handling this sort of query in a remotely motivated manner would require very considerable extensions to the knowledge base and inference mechanisms to allow reasoning about time, for example. What is at issue here is not the theoretical basis for such reasoning, but whether such extensions would lead to an unreasonable increase in complexity both of the implementation and, more importantly, of the knowledge to be acquired. However it is not obvious that the system should make an attempt to transform such a query, even if the user is given the feedback of a natural language sentence generated from the transformed query (Boguraev and Sparck Jones 1984). The point is that although the eventual database query may be as close as possible to the original query, if, as in this case, the results might very well be misleading, the user must be told explicitly what the coverage of the database with respect to the question asked actually is, and what aspects of the input query do not correspond with the domain. In the case above, for example, the user needs to be told that the database only contains information about the current situation. In general, given such information, it seems likely that the user will be better at reformulating the question than the system. Handling user misconceptions is a (very difficult) area for future research (cf. Sparck Jones 1988); in general in the current system inference is not designed to cope with user input that probably indicates a serious misconception of the coverage of the domain.

5.2 Examples of question types handled

This section illustrates question types that the system does process successfully. The set of test sentences was partly adapted from some sentences handled by the TQA system (Damerau 1980, 1981) and partly from sentences used to illustrate different types of inference in the earlier study report.

Which parcels are in 2/1?
Who supplies green parts?
What is the value of Blake's parcel?
What is in Baker Street?

are examples of the sort of straightforward query which can be handled by the system when a minimal ordinary level knowledge base is present, when the domain description is straightforwardly analogous to the database schema (see 3.3.2), and only simple transformations between levels are used (as described in 3.4.1). By a minimal ordinary level knowledge base we mean one which connects up the words which are defined as having straightforward domain equivalents to the semantic primitives and cases. Some compound nouns can be interpreted with the same knowledge base and a slight extension of the transformation mechanism to allow translation of a dummy predicate. Examples are

Which 541 properties are in 2/1?
Who are the bolt suppliers?

This actually involves finding all the domain predicates that can apply between two known entities first, rather than restricting a set of predicates found by transforming some item that exists in the network. This is just done for efficiency; the general technique could be used if some most general predicate **DUMMY** was in the network. This level of capability is approximately equivalent to that of the earlier system.

Adding to the domain description and allowing terminological transformations (see 3.4.2) allows the use of a wider vocabulary in queries such as

Who owns the houses on Baker Street?
Which shops are in Ward 1?

A more extensive ordinary level description allows vocabulary to be interpreted even though it is not anticipated by the person responsible for porting the system to a new domain.

Which roads are the shops on?
Where is petrol sold?

The first example relies on road1 being a more general term than street1 (which is connected to the domain concept of **&street-GH**), and thus only requires

straightforward transformations; the second example relies on recovering the concept of garage2, which is connected to the domain.

The heuristics mentioned in 3.4.3 depend on the domain description alone and allow processing of queries such as

> Who are the post office block shop owners?
> What are the assessments on Baker Street?

by finding indirect connections between entities. Redundant parts of queries can be discovered;

> Where is the shop in Baker Street?

generates a query about the block that the shop is in or about its street position rather than just the street; in some cases parts of a query may be removed altogether,

> Which houses in Market Place have land use codes?

will be transformed into the equivalent of

> Which houses are in Market Place?

since all parcels have land use codes. The user is told about the redundancy and may abort the query. As can be seen from these examples multiple applications of different inference specialists may be necessary to transform a query into an appropriate form. However the specialists in the set we have built are able in combination to handle a wide range of 'indirect' input questions.

These questions are examples of large classes of sentences; however our classification of types of inference requirement now differs considerably from that of the original study report which was somewhat woolly and heterogeneous. We conclude this chapter by discussing the coverage of our system compared with that envisaged in the original report.

5.3 Comparison with the study report proposals

Inference during analyser and extractor operations was discussed in Sections 4.1 and 4.2. In contrast to what was originally suggested we do not carry out any domain-dependent disambiguation of compound nominals, quantifier structures and so on at these stages, since this is not possible before translation. Disambiguation is carried out by excluding incorrect structures as a normal part of the translation process (ie no special inference is needed); "packing" structures would make this more efficient. As suggested previously certain domain-specific

lexical items (called "value words" in the study report although actually they could in principle be words of any type) have to be accessed by the analyser; however we would not now regard this as a type of inference. The way in which we handle proper nouns and the use of the knowledge base in order to assign a domain-dependent preference order to word senses were not suggested originally, but are very straightforward extensions of the architecture.

As can be seen from the examples we can handle questions which were originally categorised as needing "coercion" and "conceptual completion". We can also quite straightforwardly make some "negative" inferences in that we can block queries which violate domain constraints, but we would not attempt to block queries which appeared contrary to common sense knowledge, on the grounds that the user is more likely to know whether a question makes sense in the real world than the system is! Although we handle some types of question that the study report suggested needed "causal reasoning", we do not think it sensible to attempt to treat some others, as was discussed in the previous section.

The study report mentioned inferences in the convertor which might arise because the schema model was less rich than the domain model. As illustrated in 3.3.2 we have investigated this in some depth, and we can handle, in a general way, several types of questions that some previous systems have dealt with in a somewhat ad-hoc manner. The study report suggested that inference might be needed in the convertor to determine such things the specification of joins and columns which particular domain concepts map onto. This does not actually seem to be necessary; our current schema model is slightly more than a simple expression of the relation and column structure of the database, as was described in 3.3.1, and so once the query is transformed into the schema terminology, by the first stage of the convertor, subsequent processing, by the later convertor stages, does not require inference. As mentioned earlier, calculations are not supported in the current system, but we do not regard providing this facility as requiring any significant alterations to our design.

So of the questions considered as requiring inference in the original report, the only significant types which we could not translate, in a transportable way, within our current architecture are some of those involving "causal reasoning"; and, as just mentioned, we think that attempts to produce search queries from such questions would be misguided.

Chapter 6

Conclusions

This small-scale project was an attempt to investigate a particular design for a natural language interface to databases and not a full test of our main hypothesis about the utility of general purpose knowledge. We can therefore only evaluate our design, and in conclusion attempt this, comparing it briefly with some previous work on natural language interfaces to databases. We then discuss the problems that have arisen and conclude with some suggestions for future work.

6.1 An evaluation of our system's design

We can conclude that the general design of the system is a good foundation on which to base future work. Splitting the knowledge base into levels seems to work well and the explicit provision of an independent domain level based on semantic data modelling techniques has several advantages. It makes it much easier to tackle schemata which are in some sense unnatural or incomplete, and therefore to handle queries which do not map directly onto the database structure. The domain model is independent of particular database models (eg the relational model), and the interface could therefore be adapted for different types of database systems by replacing the schema level of the knowledge base and the convertor module. It could not, however, be adapted in this way to provide a front end for an expert system, which carries out inference on its own knowledge base, because there would then be no principled way of making the distinction between the two knowledge sources. Because the concepts of the domain model are mostly those which are commonly used in semantic database models, and we strictly limit the type of knowledge involved, the provision of the domain level of the knowledge base by someone with database expertise, but limited training in other forms of knowledge representation, ought to be feasible. As will be mentioned later a domain level of this type is necessary if the natural language interface for querying databases is to be seen in the wider context of extended database interfaces.

The utility of limited, definitional, general-purpose knowledge has been demonstrated, although our testing did not have the extent which we would have liked. The limitations on the general knowledge which we incorporated were partly due to the greater emphasis we have put on the feasibility of knowledge acquisition than was done in the study report, and partly due to problems with the knowledge representation that we used, as discussed in the next section. However we have shown that limited general-purpose knowledge allows a wide range of queries to be processed even though only a minimal number of word senses are directly linked to the domain. As the knowledge which we provided reflects dictionary entries from LDOCE we think it reasonable to assume that it was not specific to a particular application. We have demonstrated the system with two applications, meeting the original transportability requirement that new knowledge should only have to be added when moving to a new domain. (In fact we have gone further than this; the system can be set up to act as an interface to both our test databases at the same time.)

We have demonstrated the utility of limited inference on these types of limited knowledge. Of course the whole enterprise is only possible because we assume that questions are to be interpreted with respect to specific databases, and that the knowledge that the databases contain can be delimited by the domain models. It is very important that we do not abuse this and make heavy use of heuristics which will fail with complex domains. Examples from the LOKI project (Binot et al, 1988) illustrate the effect that a small domain can have on limiting the interpretation of input.

The concepts of the study report have been refined in several ways. We have a much better idea of what sort of information should be in the knowledge base, and have classified the types of inference to be carried out on it, as was explained in Chapter 3. The notion of regarding the current system's knowledge base in conjunction with the actual database as a hybrid system was very important in developing our ideas. We have shown that inference is not in fact required from all modules, as was originally envisaged. The access to the domain-dependent part of the knowledge base from the analyser is limited to the recognition of domain-specific lexical items (such as **541** which is a land use code) and the principle of initial domain-independent analysis of the input question is therefore compromised as little as possible.

Our two main problems have been with ambiguity and our chosen knowledge representation. Discussion of this, and some suggested solutions, are in the next section.

We cannot attempt to compare the capabilities of our system directly with those of TEAM and IRUS for example, since we have not been attempting to produce a practically usable front end. This was inevitable, given the small scale of the project, but may also have been advantageous, since we were not constrained by the need to produce a full-scale system for a particular application, which might have led us to adopt some techniques which are not generally applicable.

Since we are advocating the use of a large body of general-purpose knowledge, our design obviously differs from most other systems. Various comparisons have been made in the text of the report, and other systems were discussed in the report on the previous study project from the viewpoint of their utilisation of inference. We therefore just attempt to summarise what we see as the most revealing points here.

Our system has more similarities with IRUS than TEAM since IRUS emphasises the domain model far more, and permits limited inference to occur on it. Although our implementation of knowledge acquisition tools is limited, the approach we envisage is again similar to that of IRUS, where KREME is used to aid the provision of the domain model, as opposed to TEAM, where acquisition is driven by the database schema. We also assume the availability of syntactic information from machine-readable dictionaries, and so do not think that the complex techniques used in TEAM for getting this from the user would be needed. Ginsparg's (1983) system might seem to be more similar to ours, both in the claim to use general-purpose knowledge, and in the use of a semantic network knowledge representation. Because of the paucity of information about this system we have found it difficult to evaluate; the use of an expert system to carry out inference is mentioned but no details are given, for example. It is far from obvious how portable the system actually is, or how well it could cope with complex domains. The rejection of the use of any logical representation certainly leads to problems; the treatment of universal quantifiers suggested will not deal with all cases, for example.

There is little point in attempting comparisons between this work and natural language systems which evaluate logical forms against knowledge bases, such as that used to illustrate the functioning of the SRI Core Language Engine (CLE) (Alshawi et al., 1988). As has been mentioned earlier, especially in Chapter 3, there are significant differences between accessing conventional databases and accessing knowledge bases; there is also a considerable difference between attempting to interface to some body of information which has been collected and organised for independent purposes and providing one's own information. Although our analyser and extractor are analogous to the CLE, and have some similarities with it, as their provision was not the main part of this project we shall not discuss this further here.

6.2 Problems

6.2.1 The knowledge representation

As discussed in Section 3.5 we have not found MEMORY to be an easy knowledge representation to work with. Essentially the problem is that it is underspecified; although we had hoped to avoid having to design our own knowledge represen-

tation, in order to use MEMORY for our purposes we have in effect built up other knowledge representations on top of it. While this was fairly successful in the case of the domain and schema levels, where we decided at a relatively early stage what sort of knowledge we were attempting to represent, it caused severe problems with the ordinary level. It should be emphasised that Alshawi did not regard MEMORY as being suitable for inference and that the original study only suggested taking it as a starting point, which is what we have, in effect, done. Given that the study report did not specify the types of knowledge needed to support inference, using a very flexible knowledge representation might have been reasonable, allowing experimentation with different types of knowledge while keeping the same underlying formalism. But, because of these tactics, the semantics of the system that we built up were insufficiently declarative to support work even on the relatively small-scale which we were attempting. The problems are of usability, not of power, and we do not see MEMORY being used in any future work along these lines.

As mentioned previously we have much more concrete ideas about the needs for inference and the knowledge to support it than we did originally. The flexibility of access to inference processes from each module, suggested in the study report, is not in fact required. We have limited the type of knowledge that we attempt to provide to relatively straightforward definitional information, not because we cannot think of examples which fuller knowledge might be necessary to interpret, but because of the impossibility of providing extensive knowledge, in a portable manner, in the foreseeable future. We would therefore suggest that future work could make use of NIKL, at least as a starting point; NIKL seems to meet many of our requirements, at least for the ordinary and domain levels of the knowledge base. It has been used to implement the domain model in IRUS and is designed to represent the sort of terminological information which we are considering. The schema level, since it contains very specific knowledge, might be better implemented in a special purpose knowledge representation, rather than attempting to force NIKL into a use for which it is not designed.

6.2.2 Ambiguity

Ambiguity is a serious problem for any natural language interface which produces an initial interpretation of the input sentence (such as a logical form) using a grammar and a lexicon which are not specific to the application. As mentioned in 4.1 using a lexicon with senses corresponding to those in LDOCE vastly increases the number of representations output by the analyser, since the general-purpose selectional restrictions are not sufficient for disambiguation. This problem would also have arisen with the old system if an attempt had been made to incorporate such fine sense distinctions in the lexicon.

The suggestions in the study report about carrying out domain-specific disambiguation in the analyser and extractor were not implemented, since to do this

translation would have to have been interwoven with parsing. As mentioned in Section 4.1 word senses which are specified as having a straightforward domain equivalent are preferred over those which do not; it would be possible to extend this to allow the domain-specific selectional restrictions to overwrite the general purpose ones and backtrack if parsing failed (cf Ginsparg 1983). This would not work well if much of the vocabulary being used were not directly translatable; since this is what we are assuming may in fact be the case, this approach is unacceptable. Less straightforward translation of part of a sentence is frequently heavily dependent on accumulation of constraints from other sentence elements, and so genuine interweaving of parsing and translation would pose difficulties. (It would not necessarily compromise modularity or transportability, but would increase the complexity of the interface between modules.) There are limits to the extent to which disambiguation can be postponed; it would not be efficient to do a database search to allow disambiguation of a structural ambiguity, for example.

We therefore propose a simpler approach, which should at least be tried before changing the architecture of the system to the extent suggested above. Word senses which are distinguished in LDOCE and therefore in the network could be "packed" together in some way and full disambiguation only attempted during translation. If the general-purpose selectional restrictions were retained this would involve amalgamating in the lexicon all the senses of a word which have the same semantic primitive, to avoid replicating structures differing only in lexical sense numbering; however this seems very messy. Since it is already far from evident that having selectional restrictions of the sort with which we have been working in the analyser is worthwhile (see Section 4.1) it might be preferable to only maintain distinctions between syntactically different word senses at the analyser level and combine this with techniques for representing structural ambiguity.

6.3 Future work

The needs of a natural language front end should be considered in the context of the more general requirements of systems which access large databases. There is a need for more semantic information about databases to be provided to increase the capabilities of interfaces of all types. Natural language interpretation (for querying and update), dealing with defective input, response and description generation, query optimisation, consistency maintenance, interfacing with expert systems, database design: all may require more extensive knowledge of the database than can be provided in existing, commercial, DBMS. It is absurd to consider these needs in isolation from one another. The utility of a natural language interface without other extended capabilities is dubious, and if a domain-specific knowledge base, once set up, can be used for a variety of different functions, the

cost is much more likely to be acceptable. The idea of such integrated systems is of course not new, Kellogg, Klahr and Travis (1978) describe a deductive retrieval system with a natural language component, though of course the portability requirement, which they do not address, complicates the problem. However the AI literature on natural language interfaces almost completely ignores even the directly relevant work in the database literature (see Brodie 1988). It is very important to decide which issues are specific to natural language interfaces to databases, and which can be investigated in the more general contexts, either of non-natural language portable interfaces to databases, or of general natural language processing. Because of the restrictions on the database query task, natural language interfaces to databases may not be suitable for investigating many issues in computational linguistics, but they can obviously adopt some techniques that are investigated in a more general context.

Future work on portable natural language interfaces to databases with inference capabilities should probably initially involve a detailed investigation of existing work on coupling knowledge base systems and databases. The type of loose coupling we envisage is similar to that described in Jarke (1986). A new representation language for describing the domain, drawing on the results of the current project, might be based on NIKL; the combination of this and a database model would again form a hybrid knowledge representation. A system implemented on this basis would provide a suitable starting point for investigating an integrated interface system.

But we are mainly interested in investigating the utility of general-purpose knowledge for natural language processing. There is obviously no point in attempting to use general-purpose knowledge if adding knowledge for a new domain is more difficult than replacing the existing knowledge would be, so we would have to control the experiment by comparing the two strategies on a number of domains. Experiments on multiple, truly different, domains are also essential to ensure that the ways in which we take advantage of the restrictions on the knowledge covered by a database are, in fact, generally applicable. It is obvious that we could not hope to genuinely interpret unrestricted natural language sentences, so we would have to find out whether the degree of competence we can achieve is acceptable given the effort that has to be put into knowledge acquisition. Since there will be a trade-off between processing ability and ease of acquisition of knowledge, we would need to investigate different extents of knowledge provision.

The system outlined above would be a suitable basis for this. Natural language processing capability would first be added using minimal general-purpose knowledge. We would then have to provide the tools necessary to aid knowledge acquisition, before attempting to carry out experiments on the lines suggested above. By attempting to separate the issues of general inferential access to databases from those of natural language interpretation, we would hope to make it easier to see how deficiencies could best be corrected; whether expanding the

ordinary level knowledge or the domain-specific knowledge was necessary or, orthogonally, whether the natural language processing or the database interfacing modules needed alteration. Our existing architecture has gone some way towards making these distinctions clear, but it will be necessary to make them even clearer if in providing natural language interfaces to databases we take advantage of both database and computational linguistic research.

References

Alshawi, H.(1987) *Memory and Context for Language Interpretation,* Cambridge University Press

Alshawi, H. et al(1988) *Research Programme in Natural-Language Processing,* Annual Report, Cambridge Computer Science Research Centre, SRI International, Cambridge, England

Ayuso, D.M., Shaked, V. and Wieschedel, R.M.(1987) 'An Environment for Acquiring Semantic Information', *Proceedings of the 25th Annual Meeting of the ACL,* Stanford University, California, pp.32–40

Ballard, B. and Stumberger, D.(1986) 'Semantic Acquisition in TELI', *Proceedings of the 24th Annual Meeting of the ACL,* Columbia University, New York, pp.20–29

Binot, J. et al(1988) *LOKI: A Logic orieted Approach to Data and Knowledge Bases supporting Natural Language Interaction,* Scicon Ltd., London

Boguraev, B.K.(1979) *Automatic Resolution of Linguistic Ambiguities,* Technical report No. 11, Computer Laboratory, University of Cambridge

Boguraev, B.K. and Sparck Jones, K.(1983) 'How to Drive a Database Front End Using General Semantic Information', *Proceedings of the Conference on Applied Natural Language Processing,* Santa Monica, CA, pp.81–88

Boguraev, B.K. and Sparck Jones, K.(1984) 'A Natural Language Front End to Databases with Evaluative Feedback' in G.Gardarin and E.Gelenbe (eds.), *New Applications of Databases,* Academic Press, New York, pp.159–183

Boguraev, B.K. and Sparck Jones, K.(1985) *A Framework for Inference in Natural Language Front Ends to Databases,* Technical Report 64, Computer Laboratory, University of Cambridge

Boguraev, B.K., Copestake, A.A. and Sparck Jones, K.(1988) 'Inference in Natural Language Front Ends' in Meersman, R.A. and Sernadas, A.C. (eds.), *Data and Knowledge (DS-2),* North Holland, Amsterdam, pp.41–70

Brachman, R.J., Gilbert, V.P. and Levesque, H.J.(1985) 'An Essential Hybrid Reasoning System: Knowledge and Symbol Level Accounts of Krypton', *Proceedings of the 9th IJCAI,* Los Angeles, pp.532–539

Brodie, M.L.(1988) 'Future Information Systems: AI and Database Technologies Working Together' in Mylopolous and Brodie (eds.), *Readings in Artificial Intelligence and Databases,* Morgan Kaufman, San Mateo, pp.

Codd, E.F.(1979) 'Extending the Database Relational Model to Capture More Meaning', *ACM Transactions on Database Systems, vol.4,* 397–434

Damerau, F.(1980) *The Transformational Question Answering (TQA) System: Description, Operating Experience and Implications,* Report RC8287, IBM Thomas J.Watson Research Center, Yorktown Heights, NY

Damerau, F.(1981) 'Operating Statistics for the Transformational Question Answering System', *American Journal of Computational Linguistics, vol.7,* 30–42

Damerau, F.(1985) 'Problems and Some Solutions in Customization of Natural Language Database Front Ends', *ACM Transactions on Office Information Systems, vol.3,* 165–184

Date, C.J.(1977) *An Introduction to Database Systems,* Addison-Wesley

Ginsparg, J.(1983) 'A Robust Portable Natural Language Database Interface', *Proceedings of the Conference on Applied Natural Language Processing,* Santa Monica, CA, pp.25–31

Grosz, B.(1982) 'Transportable Natural Language Interfaces: Problems and Techniques', *Proceedings of the 20th Annual Meeting of the ACL,* Toronto, Ontario, pp.46–50

Grosz, B.(1987) 'TEAM:, an Experiment in the Design of Transportable Natural-Language Interfaces', *Artificial Intelligence, vol.12,* 173–243

Hobbs, J.R. et al(1986) 'Commonsense metaphysics and lexical semantics', *Proceedings of the 24th Annual Meeting of the ACL,* Columbia University, New York, pp.231–240

Hobbs, J.R. and Martin, P.(1987) 'Local Pragmatics', *Proceedings of the 10th IJCAI,* Milan, pp.520–523

Jarke, M(1986) 'Control of Search and Knowledge Acquisition in Large-Scale KBMS' in On Knowledge Base Management Systems (eds.), *Brodie and Mylopolous,* Springer Verlag, New York, pp.507–522

Kellogg, C., Klahr, P. and Travis, L.(1978) 'Deductive Planning and Pathfinding for Relational Databases' in Gallaire and Minker (eds.), *Logic and Data Bases,* Plenum Press, New York and London, pp.

King, J.L.(1981) 'QUIST: A System for Semantic Query Optimization in Relational Databases', *Proceedings of the 7th International Conference on Very Large Databases,* Cannes, France, pp.510–517

Nii, P.(1986) 'Blackboard Systems, Part 1', *The AI Magazine, vol.7(2),* 38–53

Nii, P.(1986) 'Blackboard Systems, Part 2', *The AI Magazine, vol.7(3),* 82–106

Procter, P.(1981) *Longman Dictionary of Contemporary English,* second edition, Longman, Harlow, England

Reiter, R.(1978) 'Deductive Question Answering on Relational Databases' in Gallaire, H. and Minker, J. (eds.), *Logic and Data Bases,* Plenum Press, New York and London, pp.149–178

Scha, R.J.H.(1983) *Logical Foundations for Question Answering,* PhD thesis

Shieber, S.M. et al(1983) 'The Formalism and Implementation of PATR-II' in Grosz, B. and Stickel, M. (eds.), *Research on Interactive Acquisition and Use of Knowledge,* SRI Final Report 1894, SRI International, pp.

Sparck Jones, K.(1983) 'Shifting Meaning Representations', *Proceedings of the 8th IJCAI,* Karlsruhe, Germany, pp.621–623

Sparck Jones, K.(1984) *Basic Semantics Information,* Computer Laboratory, University of Cambridge

Sparck Jones, K.(1988) *A Note on Robustness in Front Ends,* Computer Laboratory, University of Cambridge

Sparck Jones, K. and Boguraev, B.K.(1983) *Final Report on SERC Grant GR/B27159: Natural language query processor for database access,* Computer Laboratory, University of Cambridge

Sparck Jones, K. and Boguraev, B.K.(1987) 'A note on a study of cases', *Computational Linguistics, vol.13,* 65–68

Stallard, D.G.(1986) 'A Terminological Simplification Transformation for Natural Language Question Answering Systems', *Proceedings of the 24th ACL,* New York, pp.241–246

Stonebraker, M. et al.(1976) 'The Design and Implementation of INGRES', *ACM Transactions on Database Systems, vol.1,* 189–223

Wilks, Y.(1975) 'An Intelligent Analyser and Understander of English', *Communications of the ACM, vol.18,* 264–274

Woods, W.(1972) *The Lunar Sciences Natural Language Information System,* Final Report, Bolt, Beranek and Newman, Cambridge, Mass

Woods, W.(1978) 'Semantics and Quantification in Natural Language Question Answering' in M.Yovits (eds.), *Advances in Computers,* Academic Press, New York, pp.1–87

Appendix A

Extract from Boguraev, Copestake and Sparck Jones (1986)

This appendix reproduces, for convenience of reference, Sections 1–3 of the paper by B.K. Boguraev, A.A. Copestake and K. Sparck Jones presented at IFIP TC2 Working Conference on Knowledge and Data (DS-2) organised by Working Group 2.6 and held in Aldeia das Açoteias, Portugal, November 1986. Proceedings published in 'Data and Knowledge (DS-2)', editors R.A. Meersman and A.C. Sernadas, North-Holland, 1988.

A.1 The original Cambridge front end

A.1.1 System structure

This front end has been very fully described elsewhere (see Boguraev and Sparck Jones 1983, 1984), particularly from the language processing point of view. The description below is therefore intended only to outline the essential, relevant features of the front end. In this account *domain* will be used to refer to the world of objects, properties and relations of which the database is an instantiation. *Domain description* and *database schema* (or *description* and *schema* for short) will be used as very general terms for the characterisation of the nature of the domain and database respectively; *domain model* and *data model* will refer to corresponding formal types of characterisation. We have so far used only the relational data model: the issue precisely raised by the need for inference has been that of an appropriate domain model, and more generally, as will be explained below, that of the type of formalism suited to representing and relating knowledge about ordinary and special worlds and ordinary and special languages. *Database* will be used to refer both to the data model and the actual data, unless an explicit

distinction between model and data is required. Finally, we shall use the term *application* to cover both domain and database, to allow for the fact that the same domain information and data may have alternative database embodiments.

Our front end design takes linguistic transportability further than systems like TEAM (Grosz 1983, Martin et al 1983) and IRUS (Bates and Bobrow 1983). Semantic grammar-based front ends can be very effective for individual applications, but are not transportable; linguistic transportability is increased, as in TEAM and IRUS, by separating the syntactic and semantic components of the language processor, since the syntactic component can be made application independent, and an independent component is useful when the variety of allowed input forms makes it worth exploiting syntax. The Cambridge front end design was based on the belief that a substantial amount of the semantic processing of an input question could also be done using application-independent semantics. Specifically, if an initial syntactic *and* semantic interpretation could be carried through without reference to the application, to provide an unambiguous and normalised representation of the input, working on this to derive the required application representation of the question would be easier than working on the raw input; at the same time processing could be carried through with a more restricted application-dependent semantic apparatus than is required to handle the variety of natural language inputs directly, and one which is therefore easier to supply. The possible penalties are in not getting enough interpretive leverage from domain knowledge in working on the initial linguistic input, and in having to provide a more careful domain characterisation and processor to eliminate independently plausible, but irrelevant, interpretations of the input.

The front end therefore has two major components, the first application independent and the second application dependent. The input question is interpreted in two stages, to determine its meaning as an ordinary expression of English, and then to determine its more specialised database reading. The two stages in fact take two steps each, as shown in Figure A.1, so the front end as a whole involves four processors, each delivering its own characteristic representation of the input. The first processor within the application-independent analysis component, the *analyser*, combines syntactic parsing using an augmented transition network and conventional grammar with semantic procedures exploiting semantic primitives, to achieve lexical and structural disambiguation of the input question. The output *text representation* is a dependency tree whose components are case structures linking word senses, defined by formulae using semantic category primitives, through semantic relation primitives. The second application-independent processor, the *extractor*, works over the dependency tree to pull out the 'atomic' propositions it contains, and to determine the question's quantification structure. The resulting *logic representation* has the general form of the quantified expressions of LUNAR (Woods 1972), with the detailed semantic information given by the formulae and case labels embedded in it as hooks for the subsequent domain-dependent operations. This subcomponent can be viewed as task-oriented in the

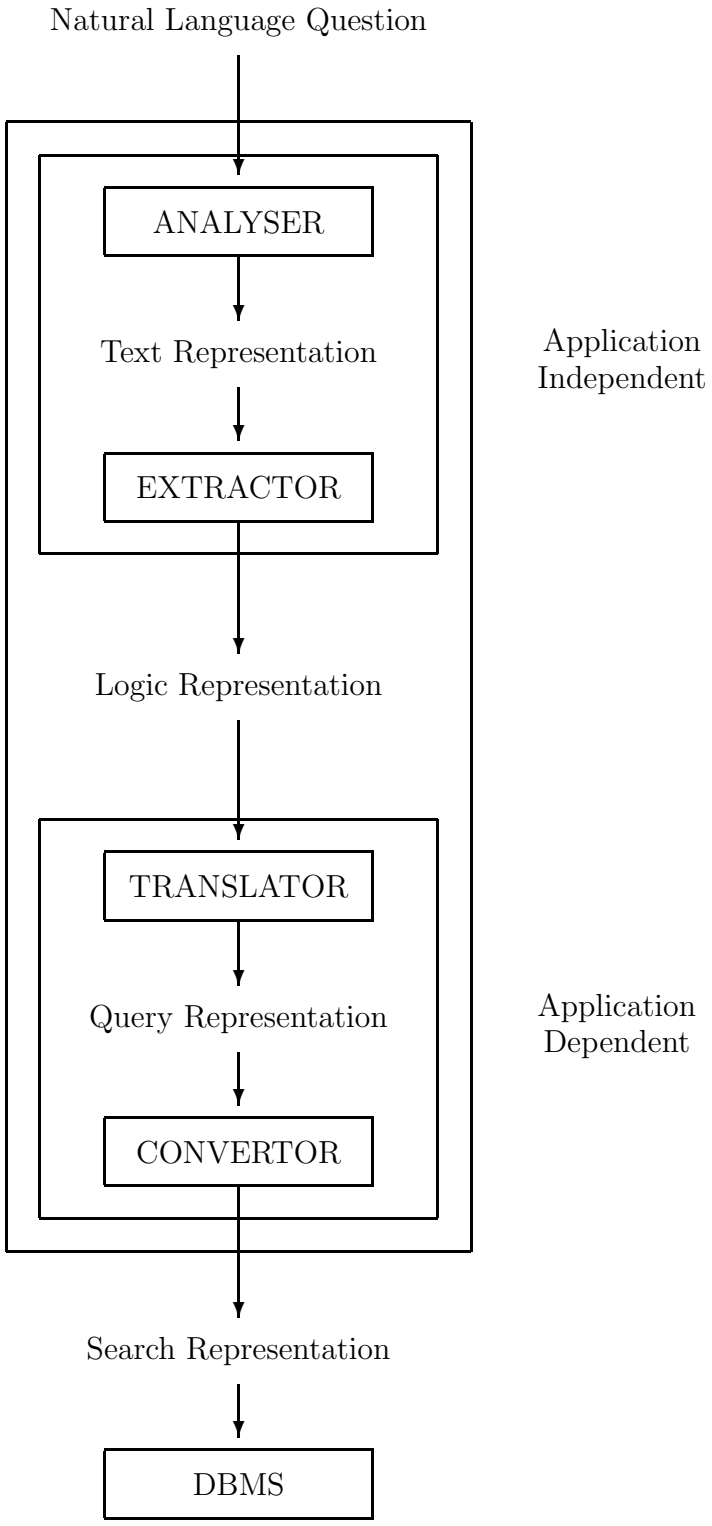

Figure A.1: System structure

sense that the item at the focus of the question dominates the representation; but this is only in the broad sense that question-answering in general, without reference to coded databases, is deemed a natural language processing task.

In the second major stage of interpretation, involving application-dependent operations, the logic representation for the question is processed first to establish its domain meaning, and then to derive a suitable form for the database instantiation of this domain. The initial *translator* substitutes domain-referring terms and expressions for those referring to the ordinary world that were handed to it by the extractor. As explained in detail in Boguraev and Sparck Jones (1983), the grounds for this substitution are the semantic primitive characterisations of the elements and the explicit and implicit relations determining the structure of the question. As the terms and expressions of the high-level data, or domain, language used for the translator's output *query representation* can in principle be similarly characterised, implicit commonalities in the characterisations of sentences in the two languages motivate the translation. The second application-dependent processor, and final front end module, the *convertor*, in turn relates domain-referring terms and expressions to those of the low-level language used to search the actual application database. This processor in fact first produces a relational algebra form of the query, and subsequently the final *search representation* of the input question in the specific language accepted by the local DBMS (in our case primarily SALT — see King 1983, and QUEL — Stonebraker 1976).

The essential properties of this front end are a strongly modular design with each processor undertaking a well-defined interpretive task and generating a correspondingly 'oriented' representation of the question; and the uniform application of *pattern matching* as the processing technique driving each module. It might be more appropriate to use the expression "pattern mapping" rather than "pattern matching", reserving "pattern matching" as the term for the wholly abstract, formal operation. "Pattern mapping" in the present context would make the contrast, in linguistic interpretation, between transformations mapping one linguistic expression onto another without provision for inference and those for which inference may be invoked. However we will continue to use "pattern matching", albeit in the sense of pattern mapping just described.

The sequence of representations produced for one input natural language question is illustrated in Figure A.2. The details of the various forms of representation are given in Sparck Jones (1984). For present purposes it is sufficient to notice that in the text representation items such as 'own1' are word sense labels; the expressions following these are the primitive formulae (with head elements at the right emphasised in bold) — thus

((*org subj) ((*ent obje) **have**))

is the formula for 'own1', denoting the "possess" sense of *"own"*; and items

User Query: *Which Norwich Road properties do International Stores Ltd own?*

Text Representation:

```
(clause (type question) (query identity) (tns present)
   (v (own1 ((*org subj) ((*ent obje) have)))
        (@@agent (n (International Stores Ltd (this (*org ent))))
        (@@object
           (n (property1
                 (((*org subj) have)
                  (obje ((where spread) thing))))
              (@@number many)
              (@@nmod
                  (((n (Norwich Road
                          (this (line (where spread))))))))))))
```

Logic Representation:

```
(For Every $Var1/property1
   :(AND (For The $Var2/Norwich.Road - (dummy $Var1 $Var2))
       (For $Var3/International.Stores.Ltd - (own1 $Var3 $Var1)))
   -(Display $Var1))
```

Query Representation:

```
(For Every $Var1/&property
   :(AND (For The $Var2/Norwich.Road - (&location $Var1 $Var2))
       (For $Var3/International.Stores.Ltd - (&own $Var3 $Var1)))
   -(Display $Var1))
```

Search Representation:

```
(Range of (Q1-var2, Q1-var1) is (Parcel, Owners)
Retrieve into Terminal (Q1-var2.Pid)
  where (Q1-var2.Pid = Q1-var1.Pid)
  and   (Q1-var2.Strnam = 'Norwich Road')
  and   (Q1-var1.Surnam = 'International Stores Ltd').
```

Reformulated Query:

Show me the properties in Norwich Road which International Stores Ltd own.

Figure A.2: Sequence of representations

prefaced by @@, for example @@agent, are case labels, i.e. semantic relation primitives. In the logic representation the formulae and case labels associated with word senses are not given, but it must be emphasised that they are retained by the system. In the query representation domain-referring items in the high-level query language are prefaced by '&', as in '&own'. The various elements of the search representation are of course those of the DBMS query language: thus 'Owners' is a relation name and 'Surnam' an attribute. Note that for clarity in the text of this paper, items in specialised languages, like the domain and data languages, are indicated by the use of bold, along with e.g. underlining for data attributes. Thus we write &own, **Owners** and **Surnam.**

The representations of Figure A.2 are not merely the means of communication between one module and the next. They embody different views of the question which are of value in their own right and can be exploited, for example, as sources of feedback to the user showing how the question is being handled (see Boguraev and Sparck Jones, 1984). Thus generating an English equivalent of the initial text representation can show how lexical and syntactic ambiguities have been resolved, while generating from the final search representation can show how the database access path has been constructed (see also Figure A.2). It should be noted, however, that feedback is still within the context of single-shot questions and answers; we are not attempting extended dialogue where the interpretation of questions is context-dependent.

A.1.2 System testing

Our initial tests were with a toy database, Date's (1977) **Suppliers and Parts**. It should be emphasised that we were not concerned with any DBMS operations, and so with the implications of database scale: we were therefore not interested in the kind of query optimisation issues addressed in the natural language context by Warren and Pereira (1982). It should also be emphasised that while the Suppliers and Parts domain is trivial, very varied input forms of the same search query can occur when unrestricted natural language is allowed, as Figure A.3 shows. Even

Which is Blake's city?
Give me the city for Blake!
Where is Blake located?
Where does Blake live?
I want the place Blake lives?
Where does Blake operate from?
Where is Blake?

Figure A.3: Varied input forms of the same search query

a front end relying on domain semantics in processing would have a good deal

of syntactic and semantic work to do to map the varied sentences of Figure A.3 onto a domain predicate "have" with arguments "Blake" and "city". However we recognise that our initial processing involves extra work because we are in principle allowing any ordinary meanings of *"live"* and *"operate"*, for example, to hold, and hence have to allow for analysis effort and text representations which would not be allowed possibilities in the domain-specific case. From this point of view Suppliers and Parts is a more serious test of our front end than it would be for those following a more conventional approach: our return for the extra effort in any individual case should be the simple transportability of the first component across applications (with additional lexical entries as needed).

However it was necessary, both to test the front end more fully, and to begin to investigate the transportability claim, to look at a second more serious application. For this we chose **Green Hills,** an English reflection of the IBM TQA Project's White Plains application (Damerau 1980). Actual White Plains data could not be made available to us for proprietary reasons, but we are using the same form of database description, i.e. set of attributes, applying them to an imaginary village in East Anglia. The database has been set up in relational form, rather more thoroughly than the TQA Project was able to do, given the practical constraints of their use of a large, existing database. A few of the attributes, especially identifying numbers, have also been treated slightly differently, to avoid some arbitrary low-level coding features of the White Plains implementation.

The detailed descriptions of the attributes and relations of the full Green Hills database are given in Figures A.4 and A.5. (We have used only a subset of the attributes, and some in a simple form, in our tests so far, but this is irrelevant here.) The essential structure of the domain is that it deals with parcels of land. These are located in the progressively more embracing areas of block and ward, and themselves may include lots. Most of the attributes are those distinguishing parcels: they include area properties like square feet, volume properties like number of stories in buildings, valuation properties like the assessment for school taxes, and use properties like the number of residential floors and land use codes (LUCs). Blocks also have a range of properties including physical area and a variety of administrative areas, for example census tract and traffic zone. The most difficult attribute is LUC (strictly attribute type, since there are separate LUC assignments for different floors). Individual building functions e.g. pets hospital, school, etc are subsumed under specific codes via an independent list, presenting many problems for the recognition of equivalent forms e.g. "pets hospital"/"animal hospital"/"veterinary centre". There are also complex implicit interactions between different attributes, for example different area measurements; and there are some nasty incomplete interactions hinging on the fact that LUC3 can refer not only to floor 3 specifically, but to floor 3 and all the higher floors.

The kinds of question that the IBM TQA system could process are illustrated in Figure A.6 (see also Damerau 1981). The TQA implementation clearly involved a

Wno/Bno/Pno/Lno	ward/block/parcel/lot number
Strnum/Strnam	street number/name
Zone	planning zone class code
Type	assessor's property type No.
Stcode	state code
Stor	number of stories
LUCi	land use code for the i-th floor ($i \leq 3$)
Park	number of parking spaces
Dwell	number of dwelling units
Resfl	number of residential floors
Comfl	number of commercial floors
Landv	land assessment (ie valuation)
Imprv	improvement (building) assessment
Exemv	assessment exemption
Cityv	assessment for city district
Schv	assessment for school district
Sewv	assessment for sewer district
Sqft	parcel area
Gflsqft	ground floor area
Censt	census track number
Censb	census block number
Plarea	planning area number
Schzone	school zone number
Nbarea	(1962 plan) neighbourhood planning area number
Xgrid	X grid coordinate
Ygrid	Y grid coordinate
Drarea	drainage area number
Splarea	subplanning area number
Trzone	traffic zone number
Neigh1	neighbourhood association 1 number
Neigh2	neighbourhood association 2 number
Bid	combines Bno and Pno
Pid	combines Wno, Bno and Pno
Lid	combines Wno, Bno and Lno
Inits	initials
Surnam	surname

Figure A.4: Green Hills attributes

BLOCK	Bid Wno Censt Censb Plarea Schzone Nbarea Xgrid Ygrid Drarea Splarea Trzone Neigh1 Neigh2
PARCEL	Pid Wno Bno Pno Strnum Strnam Zone Type Stcode Stor LUC1 LUC2 LUC3 Park Dwell Resfl Comfl Landv Imprv Exemv Cityv Schv Sewv Sqft Gflsqft
LOT	Lid Wno Bno Lno Pno
OWNERS	Pid Inits Surnam

Figure A.5: Green Hills relations structure (keys in bold)

How many two family houses are there in the Oak Ridge Residents Assn.?
What is the account number of the parcels that DelVecchio owns?
How many dwelling units are on Craven Lane?
What parcels in Planning Area 8 have more than 79 dwelling units?
Who owns the vacant land in Hillair Circle?
What is the average assessment of the apartments on Lake St.?
Where are the gas stations in Fisher Hill?
What is the gross floor area of the commercial buildings?

Figure A.6: Sample TQA queries

great deal of very specific program unique to that application, sufficient to resolve many of the problems we shall consider, albeit in an ad hoc way. The user could also be assumed to be very knowledgeable about the domain, and indeed the database, so questions leading to processing difficulties might well not get asked.

Many of the questions which would ordinarily be addressed to Green Hills, like those addressed to White Plains, would not require inference. But Green Hills questions can produce inference problems, serving to emphasise the fact that inference is required, so we must be prepared for it. Thus though it is evident that interpretation requirements we would have to meet by inference could often be finessed by the type of application-dependent semantics and very powerful lexical apparatus used in the TQA implementation, inference needs *can* arise within the framework of a front end relying on application semantics, so it is necessary to address the question of how inference mechanisms can be provided for natural language front ends. These needs can arise even where nothing more than the literal interpretation of the input question is required; as soon as any attempt to make more powerful database front ends dealing, e.g., with meta- as well as

object-level questions, or with more complex domains, inference is increasingly needed (see, e.g. Bates 1984). This is even more the case as we progress from the simple database case to consultative interfaces to software and expert systems (see e.g. CONSUL — Mark 1981 — and HAM-ANS — Hoeppner et al 1983).

A.2 Inference needs

In general we assume that inference on information about the 'ordinary' or non-specialised world will be needed to support full text interpretation, as it has been found necessary in other work on language processing. Thus we may suppose that the analyser in particular, and possibly the extractor, may call for inference on general world knowledge in the same way as these modules call on general linguistic knowledge. However the problems presented for our front end design stem essentially from the fact that the analyser and extractor modules also need to access information about the application. The translator and even convertor may also require more application information than a purely application-specific front end. Our primary focus will therefore be on the way application information is needed, and on how this need can be met. In fact there is no hard and fast line between general and application knowledge, so the proposals we make for handling the application case involve the general case, as discussed in Section 3. The same applies to linguistic inference, i.e. inference on linguistic information: the proposals we put forward for dealing with non-linguistic information involve dealing with linguistic information.

This section illustrates the way inference needs can arise in the processing carried out by the front end's different modules, and the forms of these needs. As the modules use distinctive kinds of information, when we refer to inference *types* we are labelling inference needs by their source modules and associated information. Some other distinctions, for example between the *functions* inferences serve, or the *bases* for the links established, are also necessary, as will be explained.

A.2.1 Analyser inferences

Analyser inference arises from the need to resolve ambiguities in the user's input. Most of these ambiguities are linguistic, but analysis can only be completed through reasoning about the domain. Constructions like compound nominals (*"... city school district gas station owners..."* — owners of gas stations in the district with the city school in it), and post-modifier phrase attachment (*"Do any owners who own properties located in Market Place also own parcels in ward 2?"* — is it the owners or the properties that are located in Market Place?) belong here.

One approach to dealing with this type of problem is to interface inference

using the domain description or database schema with parsing, so alternative interpretations are weeded out during analysis: this is the strategy represented by the TEAM 'pragmatic functions' (see Grosz 1983, or for a similar approach, Ginsparg 1983). Another approach is to pass all the alternatives forward to the domain translation phase, where an application-based selection can be made, as it was by the pattern-matching processes of our initial front end.

Apart from the general problems presented by having to forward many alternative initial interpretations (or, using another technique, under-interpreted chunks of input representation), there are particular problems with the postponement strategy in the database case. These are connected with the explicit use of *value* words. The same database value can take many linguistic forms, or figure in many linguistic constructions, and in any particular case what may look like a simple character or digit string can have a very complex semantics. A non-committal analysis strategy may in some cases allow for the later specialised interpretation, but there is a danger with our analyser strategy that an ordinary world interpretation could be obtained which would, exclude the possibility of any alternative. Consider, for example, the Green Hills request

Which 541 properties have parking lots?

Unless the analyser is aware that properties in the database are identified, among other things, by Land User Codes and "541" is a valid LUC which means "supermarket", it will produce, by default, the obvious interpretation leading to the subsequent output of a list of 541 properties which satisfy the specified condition, i.e. have parking lots; the question will not retrieve, as intended, only certain supermarkets.

Similar problems are exemplified by:

Print the parcel areas of Land User Codes 300 to 399.
Print the parcel areas of Norwich Road 300 to 399,

where the semantic impact of the numbers, and therefore the structural analysis of the questions, can only be established on the basis of specific knowledge of the properties of the domain. If the analyser does not take these into account, it may completely fail to deliver the correct analysis of the input question; and subsequent domain reasoning cannot be relied on to recover from failures like this.

Thus in general, as Grosz (1982) notes, it may be necessary to interleave domain reasoning with the linguistic analysis of the input to determine the particular way in which what in final database terms is a field value (or any of its synonyms) figures in the user question. It may appear as a noun phrase modifier (consider the problems of analysis of *"shop buildings"* vs. *"school buildings"*, where there is a complete LUC reserved for the concept of a "shop building", but the database offers separate encoding for "school" only); or as a head noun

referring to entities which, viewed from the angle of a particular attribute, have this particular value in the corresponding field (e.g. *"Market Place"* really means *properties for which the street-name attribute is equal to* Market Place).

This problem also affects referring pronouns and similar expressions, which can occur even within 'single shot' questions. Consider, for example,

> *List all the properties with parking spaces together with their assessed values.*

What makes this question unambiguous is the fact that in the database it is properties (in fact, parcels) which have assessed values. There is a different field specifying how many parking lots (if any) are associated with a parcel, but no assessment for these is available. (If, on the other hand, the question was

> *List all the properties with parking lots together with their assessed values.*

it would be genuinely ambiguous, since there are fields both for the value of the land and for its improvement, i.e. the building(s) on it.)

A.2.2 Extractor inferences

It is evident that domain-referring inference may be required during the extractor processing to sort out quantifier scoping. This is particularly obvious with "each". While

> *Who owns each property?*

is genuinely ambiguous, since it may properly mean either *For each property, who owns the property*, or *Who is the owner such that he owns each property*, domain inference would choose only one scoping for

> *Where is each school in the zone?*

if there could be only one school per zone.

There are many difficulties in achieving correct quantification (see, e.g. Woods 1978 and Moore 1982). Our treatment of quantifiers has so far been a rudimentary application of Woods' approach, but improving it would clearly lead to the need to tackle the kind of difficulty illustrated. The same applies to definite and indefinite referring expressions: thus inference problems can arise in extraction in connection with definite descriptions (Berry-Rogghe 1984, 1985; see also Woods 1978) as for example in

> *What is the assessment for Smith's properties?*

which may mean either what is the assessment for each of Smith's properties or what is the total of the assessments for all of them.

A.2.3 Translator inferences

As noted earlier, the requirement for inference can arise even in application-oriented front ends; and as the application-oriented approach has been the most common one, most discussions of inference in the database and broader inquiry and interface context have focussed on the way application, and particularly domain, information is supplied and handled. Then as it is clear that the domain characterisation has to be a rich one, embodying a good deal of common-sense knowledge both of a general kind underpinning the domain and of a more domain-specific sort, it is also clear that intensional reasoning is required, as well as the extensional reasoning of typical database implementations.

In general we can identify two inference functions, *coercion* and *conceptual completion.* In the particular context of the translator these arise as follows.

The translation processes connect natural language expressions with domain concepts (or, strictly, substitute domain language for natural language expressions). They build a conceptual structure centred round a *domain verb* whose case (or slot) fillers are objects defined in, and known to, the domain description. This structure expresses certain relationships between the objects in the domain and the translation process exploits requirements on the types of argument that the domain verb expects, and will accept, as slot fillers.

The need for coercion arises when the domain verb's case preferences are violated, though all the slots in the domain verb case frame have been filled, and their fillers are valid domain objects. One obvious reason for this is that the user has taken a short-cut, so an easily inferred relationship between domain objects is left implicit. A question like

Where is the Potters Lane car park?

in fact in Green Hills means *Where in Potters Lane is the property with a car park on it*, and a coercion process is required to infer the implied relationship between "car park" and "property". Similar reasoning has to be carried out to understand the meaning (in this database context) of questions like

Are there any schools in Ward 2?
What is the average height of Norwich Road?
Which owners in Market place aren't in Ward 1?

(The last example demonstrates that coercion cannot be reduced to simple expansions of conceptual slot fillers — what the question really means by "owners" is "properties".)

A different class of problems arise when there is an empty slot in the conceptualisation of the domain verb frame, since this typically has to be filled to obtain a valid query. This calls for conceptual completion inference whose aim is to deduce a likely filler for the slot. Thus in order to answer a question like

What's in Norwich Road?

the system would have to infer that the properties of parcels are sought.

Both coercion and conceptual completion may require arbitrary long chains of domain reasoning, and specifically of *causal* reasoning; extended causal reasoning, in particular, is likely to be needed when there is no obvious domain verb to hand to guide the whole process. In these cases there is no clear starting point for the inference needed to establish the relevant domain predicate. The problem of controlling long chains of (weak) causal reasoning (as in the style of Rieger, 1975) then becomes acute. Consider, for example,

Who could afford to buy International Stores Ltd?

This could be answered by evaluating a query meaning *Whose property is assessed at more than the total assessed value of International Stores Ltd parcel* against the database. The missing conceptual filler here is the "money" (or its equivalent) required for the purchase, and inferring this gives rise to a chain of reasoning connecting assets with personal worth and personal worth with buying power. Similar processes would be required to derive an answer to questions like

How wealthy is Colonel Cribbin?
Can I refuel my car in the Market Place?

The inference engine must be able to handle 'vague' concepts (such as "profitability" and "wealth") and to reason about their relationships in the world.

Causal reasoning, i.e. reasoning defined in terms of the base for the pragmatic relationships it exploits rather than in terms of the system function it performs, may be stimulated as a means of achieving coercion or conceptual completion. However for the kinds of reasons just illustrated, causal reasoning may also be required to allow question interpretation outside the specific functional contexts associated with coercion and conceptual completion, and indeed may be the main form of inference employed.

Being able to deal with vague concepts in causal style appears to be particularly important in connection with *negative* inferences. Here inference is needed not to deduce the real question to be asked of the database, but to intercept questions where common-sense reasoning suggests that there is no point in translating the input, because it does not make sense (at least in the world of the application). Thus domain reasoning by the translator about e.g. time, place and part-whole relationships would conveniently block further processing of questions like

How many parcels have land values larger than city values?

It should also be emphasised that if the DBMS has restricted facilities, for example if it cannot respond directly to yes/no questions, it may be necessary to transform an input question into a derived form with the appropriate syntax, which can involve causal or other reasoning (see Boguraev and Sparck Jones 1985). Thus inference would be needed for

Is there a lawyer in the village,

(meaning *Is there a property identifiable as a law office*), to derive a transformed form meaning *Show me any property which is a law office.*

This kind of problem is unavoidable as long as DBMSs are as restricted as they currently tend to be, so even with an application-oriented front end, natural language question transformations would be required.

There are also specific problems stemming, within our model, from the fact that, if transportability is to be achieved, the only changes to the lexicon allowed will be additions. One problem here is that concepts represented by the same natural language word would be instantiated differently in different domains: thus a "city" may be an **OperationsBase** in a Suppliers and Parts database, a **Port** in a Naval database, **Residence** in a Personnel database, and so forth. Esoteric and/or ambiguous words may also be used to refer to these domain objects, for example "habitat", "headquarters", "domicile" or "residency". Appropriate interpretations can be produced with an application-oriented front end; they cannot be guaranteed with a translator like ours, so coercive inference is more likely to be needed, especially for the second class of case.

The inference requirements to be met by the translator clearly show the need, in serving both coercion and completion functions, for inference links which can only be established by extended causal reasoning. One important base for inference is thus the causal one. But it is evident that the inference functions to be supplied for the two previous modules, if not explicitly the same as those of the translator, can be described as facilitating coercion and completion. Equally, it is likely that the inference involved will sometimes have a causal base. The same applies, in a fairly abstract way, to later inference supporting the convertor's operations.

A.2.4 Convertor inferences

It is difficult in practice to draw a precise line between reasoning within the domain and reasoning involving the database (both schema and, in the limit, actual data). The normal assumption is that if something is present in the domain it is supported in some way in the schema. If &company is a domain entity then the translator can assume that it is a concept supported by the schema without having to access the details of the way in which it is supported. However in some cases a question will make sense in the domain but cannot be answered because the actual database schema will not support the generation of a corresponding search query. In this case the query is blocked by the application of database knowledge in the convertor.

Consider, for example,

Which shops are adjacent to the Guildhall?

The concept of adjacency of must be supported in the domain, not only because it is a natural relationship between areas of land, but more specifically because it is possible to derive adjacency information about blocks, as block coordinates are known. However the schema provides no such support for deducing whether or not two parcels are adjacent, and the question above cannot be answered.

The domain description has to be given in general enough terms to support the translation of very varied input expressions of queries which the database can handle, so it is unsurprising that it will sometimes accept questions which the more impoverished schema cannot support. In some cases the schema may be obviously conceptually incomplete (for example the lack of land use codes for specific floors higher than the second in Green Hills). It would be difficult (and probably unwise) to attempt to construct a domain description that would accept *"Is the first floor of the Guildhall residential?"* and *"How many residential floors does the Guildhall have?"* but not *"Is the fourth floor of the Guildhall residential?"* Such questions should be blocked by the convertor.

Much of the inference carried out with the application-specific component of the front end can be described as planning inference, designed to identify the raw data to be extracted from the database, and to determine what operations on this will derive the required answer. Thus for the convertor, since the actual domain characterisation of what is wanted has been established, inference may be required to establish the appropriate database schema mapping of the domain concepts and structures determined by the translator. For example in the Green Hills database a decision has to be made whether to map &parcel identifier (**Pid**) onto relation **Parcel** or relation **Owners** and in some cases inference may be needed to do this.

Many of the kinds of problem discussed in connection with the translator have obvious parallels in conversion. In other cases difficulties arising in the translation stage will probably be only partly handled and will propagate forward to the convertor. Indeed the assignment of inference problems to one or the other is ultimately arbitrary as it depends on the specific characteristics of the domain description and database schema. (The same applies in general to allocation between the analyser and extractor, of course.)

Further, there may be inference requirements to be met in determining, for example, the exact role of a value word (and thus its mapping onto the database schema), or the appropriate relationship between several domain objects with associated information stored in different files (and thus requiring complex 'joins' — sometimes known as the "multipath problem" — see Moore 1979).

There are, however, also convertor inference problems associated with determining specific operations on individual database entries to complete the interpretation of the user's question. Thus, assuming that the DBMS is capable of supporting some calculation capabilities, for example those required to handle [$age = today - birthday$], [$totalworth$ = sum of all assets], [$distance = finish - start$], [$average$ = sum of n over n]) then a certain amount of reasoning

will be required in order to generate DBMS programs to carry out operations like giving data summaries:

List all the properties according to their parcel numbers!
What is the total area of Ward 2?

or carrying out implied calculations (e.g. sums, averages, percentages):

What is the average value of a property in Norwich Road?

In conclusion it will be evident that inference becomes even more complex when more than one inferential process, of the variety presented, is required to interpret the question. Thus

How wealthy are the 541 owners?

would require analyser inference to handle *"541"*, extractor reasoning to resolve *"the"*, coercion in the translator to handle *"541 owners"* and also causal inference for *"wealthy"*, and finally convertor reasoning for the calculation of *totalworth*. Many of the examples in this section are in fact of this mixed sort.

A.3 An approach to inference

A.3.1 Background

There has been relatively little work on incorporating inference facilities in database front ends in a motivated way. Though inference using both semantic and domain knowledge has been invoked to support a variety of language processing tasks, including interface-based tasks, there are few projects which address the question of inference within the framework provided by the constraints of access to conventional DBMSs. TEAM and KNOBS (Grosz 1983; Pazzani and Engelman 1983), and Ginsparg's (1983) project are notable exceptions.

One reason for the lack of interest in inference is the common limitation to single-shot operation. But a more important one is the fact that, for the individual application, engineering approaches relying heavily on close, low-level interaction between domain description, and even database schema, and linguistic analyser can work sufficiently well without inference: this is the strategy embodied in commercial systems like INTELLECT (Harris 1984). This approach can be pushed so far as to incorporate pragmatic information and inference-type operations in the system's semantic descriptions and pattern-matching strategies. In any case in practical contexts users can rapidly learn to adapt to a system's limitations, provided it is reasonably habitable. More importantly, if the limitations of the back end DBMS are severe, for example in discriminating quantifiers, and these limitations are known, there may be little practical call for powerful natural language processing capabilities whose subtleties are thrown away.

Thus though the failures of even very modest database front ends may be non-trivial (see Bates 1984), work on inference to interpret and respond to natural language inputs has been mainly in the context of more sophisticated tasks calling for more 'intelligence' and 'cooperation', as, for example, in Zarri's (1984) RESEDA, Gershman's (1981) Yellow Pages Assistant, Palmer's (1983) interpreter and the CONSUL mail system interface (Mark 1981). But even in these cases the approaches adopted rely heavily on domain-flavoured or domain-specialised inference processes. This is a natural response to the greater demands of the task, as in CONSUL, where, though the formalism used in CONSUL is quite general (as the system uses KLONE), the knowledge base the system exploits integrates linguistic and world knowledge in a completely undifferentiated and also domain- and task-oriented way. The front end's major knowledge resources are therefore not transportable. The crux is whether the more effective database front ends that are required, seen as representing progress towards more general inquiry systems with the back end DBMS as a deliverer of tuples, can only be provided by similar strategies. Specifically, in the context of the Cambridge front end design, the questions to be answered are whether the modular structure of the front end can be genuinely retained, and whether the form of knowledge base exploited to support inference, and the inference operations themselves, allow a clear separation for construction and transportation purposes between application-independent and application-dependent resources.

A.3.2 The front end structure

Maximum separation between resources could clearly be achieved by providing each module with its own knowledge and inference component, each dealing, if necessary, with both linguistic and non-linguistic information, and each exploiting, if appropriate, its own representation formalism. But this strategy is not feasible, for the reasons represented by the examples discussed in Section 2. One possible way of overcoming this difficulty could be to allow each module to call any component, but if the components relied on distinct representation schemes, this would imply intervening component-dependent 'transformers'. The real problem with this solution, however, is the duplication of information over resources, and also, potentially, that of heterarchical control. A better structure could be achieved by maintaining separate components (for the different kinds of information naturally associated with their main user modules), but having a single transforming 'differential': this improves control, but still leaves the problem of duplication of information. The solution to this, in turn, is to have a common knowledge base and with this a single inference engine; working out the implications of Section 2, this engine would subsume a set of inference specialists serving different inference functions. The assumption behind this scheme is that the identity of the calling module, and the character of the inputs to inference it offered, would be sufficient to select the relevant kinds of information from the

common knowledge base and to invoke, and parametrise, the appropriate inference specialists. The idea of a single knowledge base and inference engine is of course not novel; the point of interest here is whether it fits a specific front end architecture with a particular independent motivation.

Operationally, the question is whether the assumption about module correlation is justified. More generally, the question is whether, with an integrated knowledge base and single engine, the separation between application-independent and application-dependent information required for transportability can be readily achieved. We will return to this question after detailing the kind of representation to be used: this has to be a powerful enough formalism to encode both the system's permanent knowledge and the temporary information associated with a specific input, and flexible enough to be accessed from the distinct starting points represented by the different modules. Otherwise the proposed architecture appears to have the merit of simplicity, with the successive processor modules operating on a common collection of knowledge about the input question. (This is not a common representation of the input, since the system's various representations are distinct; see Sparck Jones 1983.)

A.3.3 The knowledge base

It is obvious that a well-founded representation formalism is needed. The general requirements such a formalism has to satisfy have been rehearsed, for example, by Brachman (1979). The relevant point here is that the kinds of natural language processing purposes described here call for a formalism supporting *classification* operations and allowing *hierarchical* relationships. A hierarchical structure is required not simply because it provides a base for inference in interpreting the inputs for any particular application, i.e. for the exploitation of an existing *categorisation*, to find existing relationships or to classify new objects and specifically new instances by assigning them to existing categories. A hierarchy also provides a means for incorporating application-specific information when the front end is transported. As noted earlier, the assumption made is that as the front end is transported new application is incorporated without its being necessary to remove old information. With a hierarchy it should be easy, if not automatic, to integrate new information, through either categorisation or class assignment. (We assume that tangled hierarchy will be needed.) With a properly based formalism, moreover, it should be possible to do this consistently. Finally, with a hierarchical knowledge base it may be possible to derive collocational constraints which can be deductively exploited in coercion or, if appropriate, in carrying out 'higher-order' inference procedures like those exemplified by the "classification" and "realisation" of Brachman (1982) and Schmolze (1983).

The requirement that is hard to meet is that of making a principled, or at least visible, distinction between application-independent and application-dependent, and more particularly between domain-independent and domain-dependent facts.

The problems presented by the need to have links between the two to support the translation of the extractor's output logic representation into the translator's query representation, and also the importance for the input question processing of domain information can be summarised by the example of processing

Which are Blake's stores in Market Place?.

Beyond translating this into something like

(&own Blake &shop Market_Place),

we have to go further to extract all the constituent propositions the question involves in order to be able to obtain a full query representation of the input content. Thus the question is: what does it mean for Blake to be an agent in an ownership relationship with a shop for an object and the Market Square as location? The answer lies in the expansion of the representation of the domain predicate to spell out its "implications". Thus the example predicate implies, at least in the Green Hills domain, that the statements in Figure A.7 can be assumed to be true.

```
(is_parcel X)
(eq (&luc X) 566)
(eq (&str_name X) Market_Place)
(is_property_owner Y)
(eq (&surname Y) Blake)
(&own Y X)
```

Figure A.7: Explicit propositional content

In practice, some of these statements will be inferable by coercion (&shop is clearly coercible to &parcel without too much trouble, using both general semantic knowledge and the categorisation hierarchy); some of them ought to be pre-specified as value words (e.g. the land-user code for a shop: "566"); and some of them ought to be interpretable as such, together with their generalisations in the concept hierarchy ("Market Place" and "Blake" may fall in this category). The remaining statements will be instantiated on matching a general inference rule for &own, and all of the derived predications will be recorded in the temporary knowledge base as facts associated with the current input. As this example shows, the needs to be met by formalism capable of supporting this kind of processing are not trivial.

A.3.4 A knowledge base formalism

The formalism we are using to represent and manipulate knowledge is that developed by Alshawi (1983). As this is fully described by Alshawi, we will simply

summarise its main features, and illustrate how the formalism can be used with Green Hills examples.

Alshawi's MEMORY represents knowledge in a way that can be realised as a semantic net. The formalism is not claimed to be a functionally complete one in the sense intended by Brachman's (1978) Structured Inheritance Networks, but it was designed to support non-trivial tasks and has a properly defined set-theoretic semantics. Alshawi's own implementation was strongly influenced by efficiency considerations.

MEMORY defines relations between *entities* by two types of *assertion*: *specialisations* and *correspondences*, defining hierarchical and role relations respectively. Entities can be *concepts* of any kind, objects, properties, or relationships, and there is no rigid classification of entities as either role owners or roles: an entity can be viewed either as a concept having its own roles, or as a role in the description of other entities which may themselves be role-owning concepts or role concepts. The fact that there can be 'multiple views' of entities means that concepts can be used as primitives to create composites, and that complex composites can be analytically defined.

The typical MEMORY knowledge base can be viewed as a tangled hierarchy of small, overlapping frame-like structures, each of which represents some concept in terms of other concepts represented by other nodes elsewhere in the network. The organisational relationships — i.e. the links between the nodes, defining the global structure of the knowledge base — map onto specialisations or correspondences. As noted, specialisation assertions impose a hierarchical order on the concepts known to the system; correspondence statements further classify the associations between concepts, essentially by analogy: the relationship that concept **A** has to concept **B** is like the relationship that concept **C** has to concept **D**, or **A** stands to **B** as **C** stands to **D**, where 'like' or 'stands to' are primitive notions. Clearly, such a structure can be seen as saying that **A** has the same role with respect to **B** as **C** has to **D**, so in this specific structure **A** and **C** are concepts functioning as roles and **B** and **D** are concepts functioning as role owners.

Illustrating Alshawi's scheme for the Green Hills application, as the arguments for assertions can be entities of any kind, we may have some, e.g. `own1`, `agent` and `&owner`, which are atoms denoting concepts already known to the analysis and translation components, and others, e.g. `PropertyOwner`, denoting more general concepts of potential use in inference.

A specialisation assertion, such as

```
(Specialisation: PropertyOwner of LegalOrganisation)
(Specialisation: PropertyOwner of HumanIndividual)
```

states that, depending on which way the classification hierarchy is traversed, a `PropertyOwner` may be viewed as a specialisation of a `LegalOrganisation`, and, conversely, a `LegalOrganisation` may be viewed as a generalisation of a

`PropertyOwner`. (The example shows why the specialisation hierarchy is tangled.)

A correspondence assertion, for example

```
(Corresponds:    OwnsAgent to own1 as
                 agent to VerbStatement)
```

causes certain associations to be established in the system's hierarchy of role-concept pairs, thus further augmenting the classification of the relationships between the known concepts. Starting from the most general association between two entities — that defined by [role, thing] — the example specifies that the association relationship between `OwnsAgent` and `own1` can be regarded as a refinement (where refinement is analogous to, but not the same as, specialisation) of the one between `agent` and `VerbStatement`. As noted earlier, though this correspondence statement can be interpreted as "`OwnsAgent` fills the `agent` role of `own1` frame", MEMORY does not impose any rigid distinction between concepts and roles, so `OwnsAgent` need not only be a role. Correspondence assertions can rather be seen as a pair of specialisation assertions, one providing context for the other. Thus if **A** is to **B** as **C** is to **D**, there is an implicit specialisation relation between **A** and **C** and between **B** and **D**, and the relation between **C** and **D**, presumed understood, provides a broader context for the more specific relation between **A** and **B**.

As the explicit constraints on structures in MEMORY are very weak, and the explicit relations between concepts are very abstract, it is possible to use the MEMORY formalism to encode a variety of different kinds of knowledge, as well as knowledge at different levels of abstraction.

Thus looking at the knowledge base from the point of view of the different processor modules, it is possible to represent linguistic concepts and relationships of the kind used by the analyser in the way illustrated in Figure A.8, which represents a portion of network concerned with the representation of verb/argument structure. It is similarly possible to encode properties of and relations between quantifiers relevant to the operations of the extractor. (Linguistic information may thus refer to the natural language of input questions, the text representation language, or the logic representation language.) General world knowledge about land assumed relevant to the application-independent processors could be encoded as illustrated in Figure A.9

The MEMORY formalism can be used in the same way for the application-dependent processors, to encode linguistic information about the data language (of the query representation) and the search language, and non-linguistic information about the domain world and its database subworld. Figures A.10 and A.11 show some data language and domain world assertions.

The fact that distinct classes of processor-relevant knowledge can be represented with the same formalism is only one side of the use of MEMORY. The other is that it is easy to link knowledge of one sort with another. Indeed it is

```
(Specialisation: TaggedStatement of Statement)
(Specialisation: VerbStatement of TaggedStatement)
(Corresponds:    agent to VerbStatement as TagRole to TaggedStatement)
(Corresponds:    location to VerbStatement as TagRole to TaggedStatement)
```

Figure A.8: Descriptions of analyser constructs

```
(Corresponds: area1 to land1 as
              Measurement to MeasuredEntity)
......
(Corresponds: ContainedArea to ContainingArea as
              SmallerArea to LargerArea)
```

Figure A.9: General world knowledge

```
(Specialisation: GHConcept of thing)
(Specialisation: GHPredicate of GHConcept)
(Corresponds:    GHPredicate to GHArgument as
                 Predicate to Argument)
```

Figure A.10: Data language assertions

```
(Specialisation: &street of GHPlace)
(Specialisation: &parcel of GHPlace)
```

Figure A.11: Domain world assertions

necessary, particularly in relation to world knowledge, to provide a comprehensive characterisation, involving higher-level concepts of the larger world within which a particular world like that of an individual application domain is embedded. An adequate characterisation of a domain thus involves linking domain concepts with more broadly applicable ones. At the same time, specifically linguistic concepts can only be given a motivated characterisation through links to underlying non-linguistic ones (this is seen most obviously in the indication of word meaning). Thus though it may be convenient, from one point of view, to regard the knowledge base built with the MEMORY formalism as including knowledge of different kinds, it is equally natural simply to regard the knowledge base as providing characterisations of concepts and conceptual structures that are put to particular uses by the calling modules. This also serves to underline the fact that any kind of knowledge need not, and probably must not, be exclusive to a

particular processor.

The way in which linguistic and world information may be linked is illustrated in Figure A.12. Figure A.13 shows how general and domain world information may be connected so as to allow a natural transition from one to the other.

```
(Corresponds:    containSubject to contain1 as
                 ContainingEntity to ContainsRelationship)
(Corresponds:    containObject to contain1 as
                 ContainedEntity to ContainsRelationship)
```

Figure A.12: Linking linguistic and world information

```
(Specialisation: &own of own1)
(Specialisation: &owner of owner1)
(Corresponds:    ownArg1 to &own as ownAgent to own1)
```

Figure A.13: Linking general and domain information

Thus domain-specific concepts can be 'rooted' in domain-independent ones via specialisation. The fragment shown in Figure A.14 states, in essence, how the Green Hills Land User Code, which is an attribute of parcels, has restricted values for particular types of properties.

```
(Corresponds:    &parcel to &luc as
                 GHEntity to GHAttribute)
(Corresponds:    &luc to (100..600) as
                 GHAttribute to GHAttributeValue)
......
(Corresponds:    property-in-parcel to parcel as
                 ContainsRelationship to ContainingEntity)
(Corresponds:    property-in-parcel to property as
                 ContainsRelationship to ContainedEntity)
......
(Specialisation: &school of &property)
(Corresponds:    &school to 248 as
                 &property to (100..600))
```

Figure A.14: Structural description of a "Land User Code"

In the same way, the domain description can be linked to the database schema, via a characterisation of the general properties of relational databases which is

then specialised to the particular relations in the Green Hills database. Thus assertions about the concept `DbRelp` — for example that a relation encodes a number of relationships (`RelpStatement`) between objects (`RelpDbEntity`), and is defined as a set of entries over a range of columns — can be "instantiated" to specific domain predications as illustrated in Figure A.15.

```
(Specialisation: OWNERRelp of DbRelp)
(Specialisation: OWNERSurname of column)
........
(Corresponds:    OwnerDbEntity to OWNERRelp as RelpDbEntity to DbRelp)
(Specialisation: OwnerDbEntity of &owner)
........
(Corresponds:    RelpOwns to OWNERRelp as RelpStatement to DbRelp)
(Specialisation: RelpOwns of &own)
(Corresponds:    OwnerDbEntity to RelpOwns as ownArg1 to &own)
........
(Corresponds:    OWNERSurnameEntry to OWNERRelp as RelpEntry to DbRelp)
(Corresponds:    OWNERSurnameValue to OWNERSurnameEntry as
                 RelpEntryValue to RelpEntry)
(Corresponds:    OWNERSurname to OWNERSurnameEntry as
                 RelpEntryColumn to RelpEntry)
(Specialisation: OWNERSurnameValue of &surname)
```

Figure A.15: Description of the Owner relation

These examples illustrate the hospitability and potential power of the MEMORY formalism, which suggest it is well suited to our front end needs. In particular, as the formalism allows the construction of proper hierarchies, and provides a mechanism for the specification of node *rolesets*, it is relatively easy to specify domain-imposed constraints on the arguments of domain verbs, so that much of the information required to carry out some inference tasks — namely coercion and conceptual completion — can be encoded as a matter of course. The scheme also makes it easy to incorporate information about relationships between the arguments of domain predicates, so reasoning about e.g. part/whole, mass and measures, becomes more tractable, without loss of simplicity and efficiency. Other relationships between domain predicate arguments, inferable from the predicate itself and "spelling out" its meaning in the context of the database could also be specified. Finally there should be no difficulty about upgrading the knowledge base continuously, adding new facts on the fly during the course of the system's operations, as more of the propositional content of input text(s) is made explicit and so is available for incorporation in the dynamic part of the knowledge base.

There are also more mundane reasons for adopting MEMORY. First, it was tested by Alshawi for a language processing task (see Alshawi 1983; in press).

Second, Alshawi also showed that information contained in the base could be conveniently and efficiently accessed using sophisticated indexing mechanisms. Third, because the information for the base can be supplied declaratively, it would be possible, when transportation is needed, to offer distinct 'blocks' of information as required by individual applications, for automatic integration in the whole.

A.3.5 Inference with the MEMORY formalism

There is nothing inherent in the MEMORY formalism that forces the adoption of a particular way in which inferences ought to be drawn. Alshawi himself exploited MEMORY to support storage and retrieval operations implemented via marker passing. Such network operations can be regarded as a very weak form of inference, perhaps one too weak for all our purposes. We nevertheless felt that the type of representation offered by MEMORY could provide a sensible starting point both because the inference processes it could support directly would be useful ones, and because if more powerful mechanisms were required they could be combined with this formalism (taking KRYPTON (Brachman et al 1985) as an analogy).

We can exploit Alshawi's framework directly to meet some inference needs because drawing simple inferences is essentially establishing new links in the network, and this is like the classification needed to achieve coercion and completion inference. By combining MEMORY's standard marker-based retrieval operations, for example, we could achieve the inferences needed to process

Is there a school in 3/2?

where coercion is needed for the necessary replacement of *school* by *parcel with LUC 248*.

In Figure A.14 we introduced the concept of a Land User Code by its description as an attribute of &property which has restricted values for various property types including, in particular, &school. The fragment of network illustrated also indicates that &school is a specialisation of &property which in turn is something contained by a &parcel.

Thus a retrieval operation like "start from a given entity, find all ways in which that entity has restricted values of attributes of other entities, which are marked in a given way" (in this case — a domain entity which can be converted into schema terms and which is a valid first argument to &location) will easily recover the concept of a &parcel. The process amounts to starting from the translator derived predicate

(&location &school 3/2),

and establishing a new link between &school and &parcel, the very existence of which completes the coercion with a result, explicitly recorded in the dynamic part of the knowledge base as

(&parcel X)
(&luc X 248)
(&location X 3/2) [1]

(cf. the earlier detailed example in Section 3.3 and Figure 3.1).

Of course it does not follow that all types of inference can be accomplished in this way, through the standard basic network operations, though we believe appropriate developments can be carried through. MEMORY can, moreover, be used to support causal reasoning. As the CONSUL project demonstrated (Mark 1981), individual causal inference rules can be encoded and inference chains constructed, without losing generality, within a hierarchically organised knowledge base which contains structured definitions of individual concepts. In Alshawi's scheme individual causal rules are conveniently represented simply as specialisations of the general concept of mapping (as exemplified in Figure A.16.) With rules encoded

```
(Specialisation: Mapping of Statement)
(Specialisation: Rule of Mapping)
(Corresponds:    LhsRule to MappingSource as RhsRule to MappingTarget)
```

Figure A.16: Fragment of a structural description of "Rule"

in the knowledge base in this way, the construction of causal inference chains (in the sense described in section 2.3) is triggered when a predicate without a direct domain interpretation is encountered, for example `wealth` derived from the question *How rich is Colonel Cribbin?*. Appropriate rules are applied (with relevant intermediate results recorded) until a related predicate, which is directly interpretable in the domain, is generated. (The example is too long to illustrate in full here.)

Moreover as the MEMORY formalism supports the explicit construction of structures in the knowledge base that resemble frames, these can be used to guide and control potentially explosive causal chains (an approach initially introduced in SAM (Cullingford 1978) and subsequently applied to the database interface task in KNOBS (Pazzani and Engelman 1983)).

A.3.6 Organising inference

One of the major problems of language interpretation is knowing when to make a decision: e.g. to select from a set of alternative interpretations available for an input. The example of *"Which of the 541 properties have parking lots?"* discussed

[1] This illustration only indicates the content of the propositions recorded in the knowledge base, and not their format. Also note that further inferences, of essentially similar character, will be required to establish the exact domain interpretation of "3/2".

in Section 2.1 illustrates this problem, and also shows that it is not eliminated by substituting inference for pattern matching: in extending or replacing a pattern-matching system with an inference-drawing system, we have still to decide when to apply inference and, more importantly, use its results.

Given the general strategy which motivates our front end, of proceeding from general to specific, it would seem rational to invoke inference to see what it suggests, but to delay committing ourselves to one possible interpretation rather than another until this is required, in the hope that by then all the relevant information will have been gathered and that the requirements to be met are sufficiently specific to be properly selective. A *non-committal* strategy of this sort can be seen as accumulating *constraints* to be satisfied when decisions must be made.

This strategy follows from the use of successive processor modules: in principle if the information is available in the knowledge base to make a selection, the selection could be made immediately; but the front end architecture means that the interpretive needs to be satisfied by later processors are not yet indicated, so it is not reasonable to expect the analyser, for example, to do convertor work, even if it could access the relevant information. The penalty to be paid for this strategy, on the other hand, is carrying forward many alternatives.

There are also problems with knowing exactly when to do inference, given that at any point in the processing being done by any module, one might seek information via inference. Thus while inference may always be possible, it may not be clear what type of inference, and with what goal, is needed. This implies that the processors will have to be extended to incorporate 'triggers', for example in the analyser the attempt to attach a modifier could be a trigger calling for inference to allow interpretation or to proffer alternative interpretations to those straightforwardly given by semantic pattern matching. The related problem of inhibiting inference explosions is well known: the natural way of trying to control it in the front end context is by demand driving, seeking a goal determined by a trigger.

Solutions to these problems are one aspect of implementing an inference capability. The other is providing the inference specialists for the engine, to carry out whatever types of inference are defined in terms of the bases for or functions of reasoning.

As indicated in the introduction, implementation of an inference capability for our front end is currently in progress, so we cannot yet claim to have provided adequate solutions to the problems just listed. What we have done is considered after the description of our implementation of MEMORY in the next section.

Appendix B

Lexical Information

Material relating to the construction of the ordinary level knowledge base.

B.1 English vocabulary

This appendix lists the 'starting vocabulary' used for work on the ordinary level knowledge base, consisting of words that happened to be in the existing lexicon. We investigated all of the senses for all of these words and their morphological variants as given in LDOCE, and made a first pass over writing assertion sets for them. But we only incorporated material for at least some of the senses of the words (and categories) marked with * in the actual knowledge base, so the larger investigation, though useful, was primarily a training exercise. The assertion set for the selected vocabulary was still quite large, and was sufficiently varied to provide exercise for our front end processing. We did not proceed further in base construction partly because we did not want to get involved in providing new lexical entries, partly because the selected vocabulary was sufficient for a wide range of test questions, and partly because we needed to consider how fine sense distinctions should be handled by the analyser. But note that the whole vocabulary listed below could be used for analysis since the head primitives provided in the lexicon's semantic entries automatically link the word senses in question into the knowledge base, though the sense coverage for the unstarred words is not complete or fine, and our category coverage for ambiguous words is not complete.

address * N	adjacent *	afford
area *	ask	assess *
assessment *	be	believe
big	block * N	blue
bolt	box	build *
building *	business	buy
car *	car park *	carry

castle
code
control
die
doctor
engineering
farm *
fastening
first
fourth
gas
goods
grocer *
hand
height *
house * N
kiss
large
live
lot
manufacture
money
number
own *
parcel * N
parking space *
person
place * N
post office *
promise
receive
reside
river
sash
screw
shop * N
sleep
status
storm
supply * V
town
vicinity
cheap
colour
cost
disc
drug
exist
farmer *
find
fix
fruit
get
green
grocery *
have
high *
identifier
knight
lawyer *
locate
love
manufacturer
name
nut
owner *
park *
part
persuade
position
price
property *
red
residence
road
say
sell
shopkeeper
space
store * N
street *
sword
upset
village *
city *
contain *
customer
do
eat
expensive
farming *
firm
floor * N
garage * N
give
greengrocer *
hammer
heavy
hospital
item
land * N
list
location *
man
measurement
neighbourhood
occupation
ownership
parking
people
petrol
post *
print
propietor
region
residential *
run
school
shipment
show
station
storey
supplier *
tell
value * N
ward *

washer * weight

B.2 Semantic primitives

This appendix lists the semantic primitives used by the analyser and their specialisation structure, derived with modifications from Wilks as given in Sparck Jones, 'Basic semantics information', as incorporated in the ordinary level knowledge base. The interpretation of the list is that in part B.2.1 each primitive is a specialisation of the classes given for it, if any, eg 'act' is a specialisation of '*mar' and '*ac'. In addition, the division of the primitives into action and substantive primitives is indicated by the category code A or S, respectively. In part B.2.2 each left-hand class is a specialisation of its right-hand classes, if any.

B.2.1 Semantic primitives

PRIMITIVE	CAT	CLASS	PRIMITIVE	CAT	CLASS
act	S	*mar *ac	ask	A	*do
be	A	*do	beast	S	*hum *physob
can	A		cause	A	*do
change	A	*do	count	A	*do
do	A	*do	notdo	A	*do
drop	A	*do	evnt	S	*ac
feel	A	*do	flow	A	*do
folk	S	*hum	force	A	*do
func	A	*do	get	A	*do
give	A	*do	grain	S	*org *inan
grasp	A	(*do)	grow	A	(*do)
hapn	A	*do	have	A	*do
keep	A	(*do)	let	A	
life	S	*any	line	S	*ent
make	A	*do	man	S	*hum *physob
may	A		move	A	*do
must	A		pair	A	*do
part	S	*pot *inan *physob *inst *sof *pla	pick	A	*do
plant	S	*pot *physob *inst *sof	please	A	*do
point	S	*pla	self	S	*any
sense	A	*do	sign	S	*ani
spread	S	*pla *physob *inan	state	S	*mar
strik	A	*do	stuff	S	*inan *inst

tell	A	*do	thing	S	*physob *inan *inst
think	A	*do	this	S	*any
use	A	*do	want	A	*do
whole	S	*sof	will	A	
work	A	(*do)	world	S	*any
wrap	A	*do			

B.2.2 Semantic classes

SPECIFIC CLASS	GENERAL CLASS(ES)	SPECIFIC CLASS	GENERAL CLASS(ES)
*ac	*any	*act	*ent *ac
*ani	*animar *ent	*animar	*pot
*any		*do	
*ent	*any	*hum	*ani *org
*inan	*any	*inst	*any
*mar	*animar	*org	*ent
*physob	*ent	*pla	*ent
*pot	*any	*sof	*any

B.3 Cases

This appendix lists all the cases used by the analyser; for further details see Boguraev and Sparck Jones, 'Material concerning a study of cases', Technical Report 118. In constructing the ordinary level knowledge base we found it necessary to add a few cases, given at the end of the list.

acc	ACCOMPANIMENT	act	ACTIVITY
adest	ABSTRACT-DESTINATION	aft	AFTER
ag	AGENT	aloc	ABSTRACT-LOCATION
asour	ABSTRACT-SOURCE	attr	ATTRIBUTE
bef	BEFORE	comp	COMPARISON
dest	DESTINATION	dire	DIRECTION
force	FORCE	goal	GOAL
inst	INSTRUMENT	loc	LOCATION
man	MANNER	mobj	MENTAL-OBJECT
obj	OBJECT	poss	POSSESSED-BY
quant	QUANTITY	reas	REASON
rec	RECIPIENT	sour	SOURCE
state	STATE	subj	SUBJECT

tloc TIME-LOCATION tspan TIME-SPAN

Extra cases:

part-of
result