DESIGNING A PORTABLE NATURAL LANGUAGE DATABASE INTERFACE

by

ALLAN DAVID BOOTH

B.Sc., University of British Columbia, 1979

A THESIS SUBMITTED IN PARTIAL FULFILMENT OF

THE REQUIREMENTS FOR THE DEGREE OF

MASTER OF SCIENCE

in

THE FACULTY OF GRADUATE STUDIES

DEPARTMENT OF COMPUTER SCIENCE

We accept this thesis as conforming

to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

March, 1983

In presenting this thesis in partial fulfilment of the requirements for an advanced degree at the University of British Columbia, I agree that the Library shall make it freely available for reference and study. I further agree that permission for extensive copying of this thesis for scholarly purposes may be granted by the head of my department or by his or her representatives. It is understood that copying or publication of this thesis for financial gain shall not be allowed without my written permission.

Department of Computer Science

The University of British Columbia
1956 Main Mall
Vancouver, Canada
V6T 1Y3

Date March 27 1983

Abstract

Allowing a user to inquire into a database in his native language is becoming an increasingly desirable feature. Consequently, there have been a number of attempts to attach a natural language front end to an existing database management system. However, few database systems today are competent in processing natural language queries. The main stumbling block is the adaptation of an existing natural language front end to a new domain of discourse.

The development of a domain independent natural language interface to an existing database management system is discussed here. The underlying domain independent features of natural language are examined and combined into one linguistic core. The domain specific information is gathered into an information dictionary for the linguistic core to process. Finally, the interface to the database handling routines is modularized with standard inputs and outputs.

Table of Contents

Figures

Acknowledgements

I would like to extend my thanks to Dr. Richard Rosenberg who initially created and continually increased my interest in the study of natural language understanding.

I would especially like to thank my parents and friends for their constant support - without it this thesis would undoubtedly never have been finished.

Chapter 1

Introduction

One might have thought that the use of large database systems would be widespread by today. People in every walk of life could benefit from the day to day use of such systems. But the development has failed to reach this expected level. One of the major reasons for this failure has been an inability to provide the database with a suitable natural language (NL) interface. Casual users of a database with no formal training in the use of computers invariably balk at having to learn a somewhat artificial language in which to communicate with the machine.

Branches of Artificial Intelligence (AI) have been concerned with this problem for some time. They probe into all aspects of natural language understanding from answering database queries to automatic translation and paraphrasing. The question answering (Q/A) paradigm has some rather strong restrictions on the inputs allowed and these can help to simplify the task. For example, when answering questions to a database, a system need not worry about declarative sentences. Because information in the database will be associated with one particular topic, both the number of words which must be understood and the possible meanings of those words will be

limited. This reduces the occurrence of ambiguity in the subset of the language which is being processed. But even with all of these assumptions which limit the language processing requirements of a Q/A system, there are still few commercially viable natural language database interfaces on the market.

However, this does not mean that the technology to produce such an interface does not exist. Many examples of adequate NL systems can be found in the current literature (Harris 1977a; Sacerdoti 1977; Waltz et al 1976; Woods et al 1972). The major stumbling block in applying this technology seems to be in the start-up costs of transferring a reasonable portion of any developed NL system to a new domain or database system. These start-up costs are usually comparable to the initial development cost of the entire system. By examining branches of Computer Science which have already dealt with the issues of portability (such as compiler design and operating systems research), it becomes clear that it is possible to apply present AI techniques to develop Q/A systems into useful tools. Unfortunately one cannot simply take the current systems and modify them to suit the needs. The issue of portability must be built in at the ground level if a system is to remain structurally sound.

Indeed, the problem that will be addressed here is the issue of portability. How can a NL interface be designed so that it can be transferred to a different system with minimal effort and still retain a reasonably high standard of question

answering capability? The portability issue will be viewed from two different perspectives: domain portability and database portability. Domain portability refers to the problems of applying the NL interface to a new domain of discourse. The issue of database portability deals with a change in the actual physical structure and data accessing methods of the database management system. All NL systems contain some procedures for dealing with linguistic features of the language regardless of the domain or database structure. It is these concepts we wish to exploit in the design of the system.

The method of achieving portability here has been to abstract all components of current systems which are domain independent and combine them together into one "linguistic core" (Rosenberg 1980). This component consists not only of the natural language parsing procedures but also of the user interaction and answer generation components. The linguistic core consults with a "domain definition" to retrieve information about the particular domain in which it is working and communicates with the database through a "database interface".

In Chapter 2 we will review some past and current systems with a goal toward pointing out some of their achievements as well as their shortcomings. Next is an overview of the important concepts of natural language portability in Chapter 3. A working system embodying these ideas is presented in some detail in chapters 4 and 5. Chapter 6 contains a description of

the domain modification process which was necessary to change the domain from the initial restaurant information database to a bibliography database and then to a conference registration database. The last chapter attempts to summarize the ideas presented in this thesis as well as provide some directions for further research.

Chapter 2

The Development of Question Answering Systems

The developmental path of natural language question answering (Q/A) systems has taken many twists and turns. Influenced greatly by Chomsky's research in transformational grammars (Chomsky 1965), researchers initially expected that linguistics was going to play a major role in the field. Early systems were developed within the transformationalist approach; that is, they adhered to a linear paradigm (Rosenberg 1980) in which the structural description of a query was first obtained before any semantic processing was initiated (Woods 1967; Woods et al 1972). However, since a syntactically directed parse tends to generate a large number of ambiguous parses from a natural language query, many researchers attempted to find a more data directed method of parsing (Schank 1973; Waltz et al 1976; Marcus 1979).

It was soon realized that, at least in the limited Q/A paradigm, the "meaning" of a query could be extracted by tailoring a system towards the semantics and paying only little attention to syntax (Brown et al 1974; Waltz et al 1976; Sacerdoti 1977). The major disadvantage of these systems was that they were built entirely around the vocabulary of the domain in which they were working and to change the domain meant

to rewrite most of the code.

Today researchers are exploring the middle ground where both aspects of natural language understanding coexist. Systems are being built which retain the framework of the purely syntactic parse but include intermediate interaction with the semantic component to determine meaning and weed out unwanted parses early (Harris 1977a; Bobrow and Webber 1980).

2.1 Syntax Without Semantics (or Structure Without Meaning)

Work on syntax directed parsing has, for the most part, been based on the transformational approach of linguists such as Chomsky (Chomsky 1965), Katz and Postal (Katz and Postal 1964). An input sentence was transformed from its input "surface structure" into a syntactic "deep structure" before any semantic interpretation was attempted. The deep structure of a sentence is the level at which the meaning can be obtained, under the 1965 "Aspects" (Chomsky 1965) theory. The surface structure, on the other hand, is the uttered form of the sentence. In general, one deep structure could correspond to many different surface structures. For example, the two queries:

Which restaurants take reservations ?

and

Reservations are taken by which restaurants ?

although they have different surface structures, have the same deep structure (Figure 2.1).

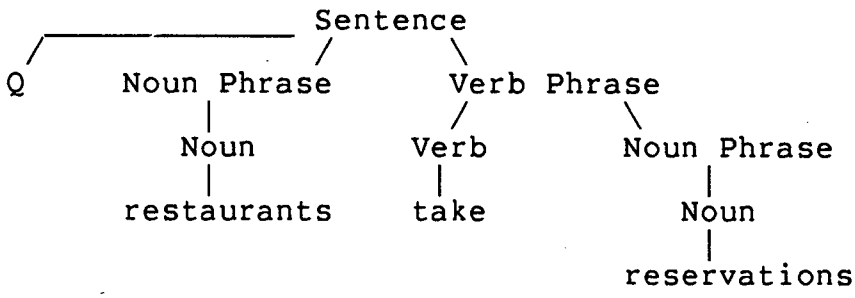


Figure 2.1: Sentence Deep Structure

The Augmented Transition Network (ATN) parser (Woods 1970) is the prime example of work in this direction. The ATN itself was modelled on the finite state transition graph, which is a network of nodes representing states and directed, labelled arcs governing the conditions for transition from one state to another. For the purpose of natural language understanding, the transitions were based on syntactic categories and a variety of conditions.

The reason for producing the deep structure was to capture as many of the regularities of a natural language as possible, thereby reducing the number of possible structures which the semantic component would have to consider. After producing its deep structure, control would be passed to the semantic processor and the "meaning" extracted. Although there were

distinct advantages to this method, a few major disadvantages developed.

Because the syntactic processor (usually the ATN parser) incorporated no semantic information, it could only generate pure syntactic parses which had no semantic grounding on which to determine sensibility. Since, even when employing semantic information, there is still ambiguity in the English language, without it, it became an impossible task to generate the one correct parse. In fact, many spurious parses were usually generated. Consequently, a "generate and test" strategy was adopted by some. All possible deep structures were generated and presented, in turn, to some decision component (Harris 1977a) until the intended one was found. For example, the query:

Find a car with a trailer which is red.

is ambiguous because there is no way to decide from syntax alone whether "red" is the colour of the car or of the trailer.

An even more important problem was that syntactically sound queries which had no possibility of success semantically had to be completely parsed before the semantic decision component could be consulted. An example from the PLANES system (Waltz et al 1976) is:

How many engine repairs required maintenance in May ?

If parsed by a purely syntactic processor there would be no way to discover the subtle fact that "engine repairs" could never require maintenance.

Additionally, since there were so many possible parses, it became necessary to show the user that the system had indeed selected the correct one. This either required the implementation of a paraphraser (Harris 1977a) or, more often, required the user to have a working knowledge of the system's internal syntactic sentence representation (Woods et al 1972).

A definite advantage of the "two pass" (syntactic/semantic) system, although never fully realized, was the ability of the initial processor, using only syntactic knowledge to remain relatively independent of the domain in which it was working. This means that to change the domain would merely require modifications to the second, semantic processor. But because the deep structure produced by the syntactic processor had to contain all of the syntactic information, it was a complex representation. And because the representation was complex, the semantic processor had to be complex to "understand" it. Also, the representation created by the syntactic parse contained little information which would prove helpful in extracting the meaning of the original query. Consequently, most of the "useful" processing had to be done in the semantic phase of the program. Therefore, the idea that the syntactic phase of the program should never have to be rewritten was obscured by the

fact that the semantic phase was itself incredibly large and complex (Woods 1967; Woods et al 1972). Even an aspect such as anaphora which should ideally be included within the domain independent syntactic portion could not be because of its need for semantic interpretation.

For the naive user to be comfortable with a computer system it must be fairly flexible. The creation of completely grammatical queries is both verbose and difficult - especially "on the fly". Using a strict syntactic parser for the first pass demanded that the grammar be reasonably inflexible since the only information which the parser could employ was syntactic. If the constructs were not strictly adhered to, many ambiguities might be introduced and the parser would become unable to parse the query. One method adopted which helped to partially alleviate this problem was to try to parse the query as a "noun phrase utterance" (Woods et al 1972) or a "sentence fragment" (Harris 1977a) if it could not be parsed as a complete query. This, like all "add-on" fixtures, only minimally reduced problems which were caused by the basic design of the method. Furthermore, the problem of non-grammatical inputs was still not addressed.

Some of the parsers which used complete backup (e.g. the ATN parser) would construct a sentence component such as a noun phrase and then, if the parse failed, the component would be dissolved. Later, at a different stage of the same parse, the

same component might have to be reconstructed. This is clearly a waste of time and energy and again some work has been done to remedy this situation (Bobrow and Webber 1980).

The semantic portion - the second pass of the two pass system - has been handled in a number of ways. The most common is the "procedural" semantics (Woods 1967; Woods et al 1972) where patterns in the deep structure trigger the use of certain procedures. Unfortunately, to update or add a new construct was a complex task in itself.

2.2 Semantics Without Syntax (or Meaning Without Structure)

Semantically oriented systems are typically data directed, one pass systems. Much of the justification for this has come from introspection to find the methods which we ourselves use for natural language understanding (Schank 1973). The term data directed (as opposed to syntax directed) means that rather than searching for, say, a prepositional phrase at each stage of a parse, one would only be looked for after a preposition has first been found. The concept of one pass means that the semantic content of each word or phrase is introduced immediately as the word is parsed and, therefore, after just one parse, the meaning of the entire query should have been found.

The main thrust of the semantically oriented Q/A systems

has been in the area of so called semantic grammars. Semantic grammars use semantic concepts as the basic building blocks for developing a sentence representation rather than syntactic ones. As examples, the SOPHIE system (Brown et al 1974) parses concepts such as voltages and resistor types, the PLANES system (Waltz et al 1976) can understand plane types and damage types and SRI's Naval database called LADDER (Sacerdoti 1977; Hendrix et al 1978) deals competently with ship types and parts. Since the semantic concepts which semantic grammars look for tend to be relatively unambiguous within a particular domain, many of the natural ambiguities in the natural language can be ignored. This type of system has demonstrated a high proficiency in simple domains (Brown et al 1974) and even simple sentence fragments and non-grammatical input can be handled.

When the semantic units have been formed, they are usually fitted into slots in a predefined pattern to determine acceptability (Waltz et al 1976; Brown et al 1974). These patterns define the range of questions which the system can answer and any deviation from them will usually result in an unanswerable query. This is not necessarily worse than the problems arising with misunderstandings during a syntactic parse because at least the system did not generate unwanted parses; however, the system still did not really have a grasp of where it went wrong and therefore, could rarely provide the user with any guidance as to why the error may have occurred.

Semantic grammars themselves are a method of representing the domain-specific knowledge in a Q/A system. Unfortunately to create or modify them, as is the case with any coded or "procedural" semantics (Woods 1967), one requires either a complete knowledge of the intimate details of the system's workings or the use of a semantic grammar "generator". LIFER (Hendrix 1977), used in conjunction with the LADDER (Sacerdoti 1977) system at SRI is an example of such a generator. NETEDI (Waltz et al 1976) is used in the PLANES system to modify already created semantic grammars. However even with the help of the grammar generator, it is still an unstructured and somewhat ad hoc process to insert new grammatical structures into the original code.

Since little notice is paid to syntax in a semantically driven system, some of the regularities in English (or any natural language) cannot be easily captured. Without the benefit of syntactic structures, the semantic grammar can degenerate to the level of having to specify or expect every possible query.

Regardless of the complexity of the representation, however, a major flaw in the philosophy of semantically oriented systems is that, being developed around the semantics of a particular domain, they must be virtually rewritten when applied to a new situation.

2.3 Coexistence

Some current Q/A systems try to take advantage of syntactic and semantic processing simultaneously. A basically syntactic parse is performed with intermediate calls to the semantic routines to determine acceptability and extract meaning (Bobrow and Webber 1980). The units formed are then combined into the semantic sentence representation. In this way, the systems gain the syntactic framework of the syntactically oriented systems as well as the "single pass" advantage of the semantically oriented ones. Rather than having a distinct syntactic-semantic processing split, there is a syntactic-semantic knowledge split. The syntactic knowledge is that which is always true for all domains, being based on the syntax of the natural language, whereas the semantic knowledge depends on the domain involved. The important problem with these systems lies in the search for an adequate and easily modifiable representation for the specification of the domain dependent knowledge.

As mentioned before, both semantic grammars and procedural semantics are means of representing semantic knowledge. The major disadvantage of these, along with Woods' cascaded ATNs (Woods 1980), is that they are basically program code and to alter them usually requires unstructured programming changes. Clearly, it would be simpler to modify information that was structured in a dictionary-type, declarative format.

The Graceful Interaction (GI) system (Hayes and Reddy 1979) is an attempt to segregate many different levels of knowledge. The domain specific (or task specific (Ball and Hayes 1980)) knowledge is represented in a "schema" which is patterned after Minsky's frame structures (Minsky 1975). These schema capture the idea that the domain independent information in a system should not only be separable from the overall system, but also be formally defineable. The GI schema reduce the definition of a new database to the level of slot filling, thereby reducing the effort involved in the process while also minimizing the chance of error. However, because these schema have been designed to describe computer programs and not real world situations, they do not deal with incomplete descriptions.

Most database systems are forced to deal with incomplete world definitions. Problems usually occur when one tries to define the possible database elements since in many instances a field can contain virtually any value. This makes the possible range of values infinite and, consequently, hard to define. The ROBOT system (Harris 1977a) uses an inverted index of the database as the world knowledge for the system. It is treated as an extension of the dictionary and so addition of a new element to the database simply requires an update of the inverted index. This method allows actual values in the database to take on meanings even if the elements, possibly jargon, may not be found in a real dictionary.

In capturing and exploiting the regularities in a natural language like English, no system seems to have more potential than the case driven systems (Taylor and Rosenberg 1975). A case grammar (Fillmore 1968) attempts to capture the purpose of a word or phrase in a sentence by determining its role in terms of a system of cases. For example in the query:

Who serves chicken ?

"who" is the agent of the action, "serves" is the action and "chicken" is the patient (or the thing being acted upon). Case systems combine both syntactic structures and semantic knowledge into one unit which can be specified in a concise, understandable and easily modifiable way. In addition, there is a structure imposed by the case "frames" which provides a basis for a formal, structured change to the semantic knowledge. The case system proposed by Fillmore in 1968 did not define the actual number of cases needed to specify a natural language but attempts have yielded numbers ranging from a mere five (Celce-Murcia 1979) on up. The actual number of cases is irrelevant, however, because it is the ability to specify a domain simply by specifying the cases that gives the system its power and elegance.

2.4 Summary

There currently does not appear to be a truly domain or

database independent natural language question answering system. It does appear that the closest thing to one is the ROBOT† (Harris 1977a) system, mainly because of its use of an inverted index of the database as the basic semantic knowledge for the domain. The RUS parser attached to the PSI-KLONE system (Bobrow and Webber 1980) at BBN appears to be making headway by departing from both the syntactically oriented and the semantically oriented parsing methods to combine the two into one general parse. But even these systems still appear to try to keep the syntactic and semantic knowledge somewhat separate. A remedy for this seems to lie in the case driven systems which allow the combination of syntax and semantics not only at the processing level but also at the knowledge level.

---

†The ROBOT (Harris 1977a) system is now being marketed by the Artificial Intelligence Corp. under the name Intellect. For further details see Johnson (1981).

Chapter 3

Concepts in Natural Language Portability

Certainly the idea of a transportable computer program is not a new one. Researchers in compiler design and operating systems have studied this problem for some time now (Johnson and Ritchie 1978; Richards 1969). These studies have shown that to make a system transportable, the changeable part must be separated from the system core. The two disciplines do not, however, split up their projects in the same way. The compiler design split follows the flow of the program. Usually the "code generation" phase is separate, following the syntactic and semantic processing phases. The operating systems split, on the other hand, is one of functional interfaces. The system dependent routines may be called upon at any time during the operating system's processing.

Attempts in AI to follow this simple separation idea have met with modest success. Earlier we examined systems which follow the compiler design portability method of "flow separation" where syntactic processing of a question is completed before any semantic processing is started. This is not necessary. Just as in the operating system where system dependent routines may play an important role at any time, domain and database dependent routines may play an important

role at any time during NL processing. The important feature of this type of separation is the structured, well-defined interface between the system dependent and independent routines.

In the past, NL systems which attempted to incorporate any transportability features have been split into different phases (Figure 3.1). Each phase of the program, syntactic, semantic and retrieval, was forced to operate on its own. Since the syntactic structural description had to contain any information which the later phases might require, the structure developed became extremely complex and convoluted. Furthermore, because the semantic component had to process this relatively complex structure it had to be fairly complex in design.

The review of various natural language question answering systems has shown the similarities of their tasks. Whether the domain involved was for moon rocks or a company payroll, there were always a number of common functions to perform. This chapter will examine some of these functions and try to show the need for simultaneous syntactic and semantic processing. Additionally it will be shown that each of the tasks contains a domain dependent as well as a domain independent portion. It is this difference which is important in a portable question answering system since it is only the domain dependent information which need be modified when changing the domain of discourse.

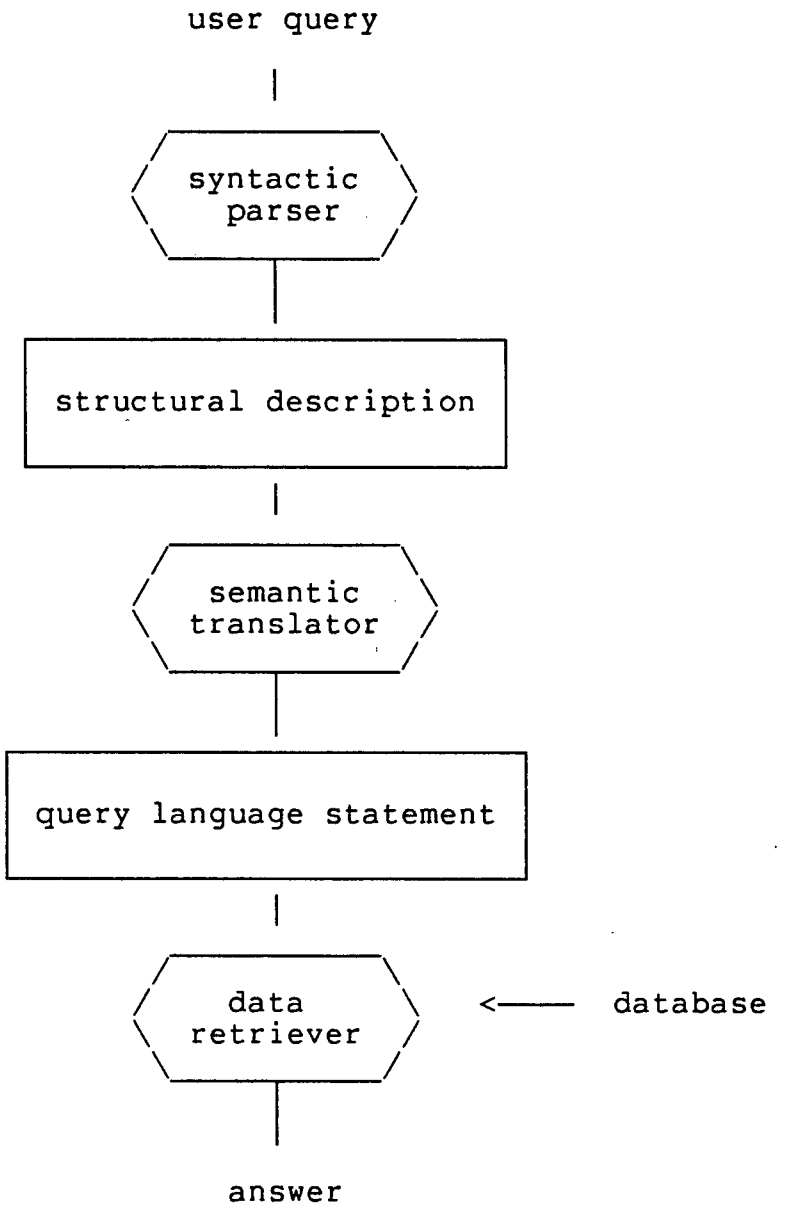


Figure 3.1: Current Question Answering Systems†

† taken from Rosenberg (1980) p. 5.

3.1 The Language Structure

From the beginning researchers have noticed that structural features of natural languages help to categorize sentences and limit the total number of different variations possible. However, even though the English language imposes a reasonably strict structure on sentence components and the roles they must play in a query, there is still much ambiguity to be found among these components. Solving this ambiguity to determine the correct sentence structure turns out to be a difficult task. Efficiently using the structure after it has been built is again difficult. However, using the structure as it is developed during the building process to provide clues for future additions will make both processes simpler.

3.1.1 Building The Sentence Structure

The syntax of the language can provide many clues as to the meaning of a sentence without even taking into account the actual meanings of the individual words. Examining a typical question answering session might produce a number of queries of the form:

How many warchen blinges are there?

Which is the best frugle?

Where is the ploon?

When did the muddel frump?

Find all of the blintogs which have punded.

In order to build the sentence structure, it is not necessary to know what the warchen blinges, frugles, ploons, muddels or blintogs are or even to know what it means to frump or to pund. It is, however, mandatory to know which are the nouns and verbs, and which is the subject and object. This information is, for the most part, either positional (dependent upon the location of the word in the sentence) or morphological (dependent upon the structure of the word).

It is during the examination of more complex cases that problems arise. In the sentence:

Have the blintogs punded the frugle with the ploon?

it is impossible to immediately decide on the correct sentence structure without involving the meanings of the individual words. For example with one set of meanings:

Have the bullies hit the boy with the brick?

the prepositional phrase "with the brick" probably modifies the verb phrase "hit", while in:

Have the bullies hit the boy with the glasses?

the assumption would be that the prepositional phrase "with the glasses" is modifying the noun phrase "the boy".

This problem with prepositional phrase modification causes much ambiguity in the English language. Actually there is no

way to determine which is the correct interpretation from the one sentence alone and possibly none even when the entire dialogue is taken into account. Either interpretation is possible. However, humans would usually prefer one interpretation over the other and this preference should somehow be taken into account.

Regardless of which interpretation the individual words have, the components noun phrase, verb phrase and prepositional phrase can be joined into separate units for further processing. The ambiguity lies outside the bounds of the individual components but rather in the relationships among them. The semantics of these units can be used to fit the final sentence structure together. If information gathered earlier in the sentence structure building process is used along with the semantic interpretation of the current component to help resolve the ambiguities, there is a large probability that the correct overall interpretation of the sentence will be made without the need for backup. Even if some backup is required, the components can be reorganized without any need to dissolve them.

3.1.2 Using The Sentence Structure

Once the sentence structure has been built, methods developed in linguistics can be used to find pronoun antecedents (Hirst 1979). Simple verb phrase ellipsis can also be quite easily handled once there is a comprehensive sentence structure

to work with (Hendrix et al 1978).

There exist universal concepts (not domain dependent) which can be recognized by their structure. For these particular linguistic constructs, the idea behind "semantic" grammars may be helpful. These grammars try to recognize specific constructs rather than general ones. One of the semantic grammars in the PLANES (Waltz et al 1976) system is used to recognize "amounts" (Figure 3.2). It has been designed to recognize constructs such as "more than three", "more than three but less than five", and "three or fewer times". If a system tries to recognize "amounts" (as in the PLANES (Waltz et al 1976) system) or "quantifiers" (as in this system), as concepts which are universal in nature and not tied to any domain, the power of the semantic grammar can be obtained without having to take along with it its inherent domain specificity.

3.1.3 Relaxation of Grammatical Rules

Of course when working with humans, one must remember that they are fallible. For this reason it is quite important that all of the grammatical and structural rules be relaxed when they are not absolutely necessary. For example, lack of number agreement can usually be accomodated by a human in the process of understanding the sentence. An error such as:

Find a books about natural language understanding.

```
(defatn AMOUNT

   ((*AMOUNT (wrd (any some) t
                  (setr rel '>) (setr # 0) (to AM:END))
             (wrd between t
                  (setr rel '<>) (to AM:<>))
             (cat comp (not (wrd between))
                  (setr rel (selectq *
                                 (atleast     '>=)
                                 (atmost      '<=)
                                 (lessthan    '<)
                                 (greaterthan '>)
                                 (exactly     '=)
                                  nil))
                  (to AM:REL))
             (jump AM:REL t (setr rel '=)))

    (AM:<>   (cat integer t (setr # *) (to AM:<>:1)))

    (AM:<>:1 (wrd and t (to AM:<>:2)))

    (AM:<>:2 (cat integer t
                  (setr rel '<)
                  (setr # (max (list $# *)))
                  (to AM:END)))

    (AM:REL  (cat integer t (setr # *) (to AM:#)))

    (AM:#    (wrd (time times) t (to AM:#))
             (cat conj t (eg $rel '=) (to AM:CONJ))
             (jump AM:AMT))

    (AM:AMT  (cat conj t (to AM:AMT1))
             (jump AM:END))

    (AM:AMT1 (push *AMOUNT t (setr pred (list *)) (to AM:END)))

    (AM:CONJ (wrd (fewer less) t (setr rel '<=) (to AM:END))
             (wrd more t (setr rel '>=) (to AM:END)))

    (AM:END  (wrd (time times) t (to AM:END))
             (pop (append (list (buildq (+ +) rel #)) $pred)))))
```

Figure 3.2: The AMOUNT ATN Network†

† taken from Waltz et al (1976) p. 116. The code has been somewhat abbreviated from its original form.

should be able to be parsed even though there is a certain amount of ambiguity. Similarly, sentence fragments such as:

Books by Chomsky.

should be handled. Simple verb phrase ellipsis would be quite common in a question answering system. An example of this is:

Who serves spaghetti?
THE "OLD SPAGHETTI FACTORY".
steak?

At this point the system should be able to infer that the question was really:

Who serves steak?

and act accordingly.

If this relaxation is not done, the strict grammatical rules will remove any freedom the user once had in specifying his query - possibly to less than that of an "artificial" query language. In the situations where the system makes allowance for a grammatical error, there should be some way to tell the user which interpretation of the sentence was being used. A common method has been to return a corrected version of the input sentence to the user for verification. A preferred way would be to develop an answer generation component which would somehow incorporate the original question into the final answer.

3.2 Vocabulary

As a result of reviewing the semantically driven systems it can be seen that they performed quite well considering that they were primarily keying on the meanings of specific words and almost completely ignoring the sentence structure. The vocabulary used in any one particular domain is an extremely important source of knowledge which cannot be ignored. There are many facets to the structure and meaning of words and word groups within any one domain from the simple meanings we attach to proper nouns to the inferred meanings of idiomatic phrases.

3.2.1 Morphing

Word morphology is the study of the structure of words. In a natural language parser a "morpher" usually refers to a routine which systematically removes prefixes and suffixes to find the root of a given word. Because of the structure of English words, this can be done usually by knowing the syntactic category and use of the word and ignoring the meaning. Then with the combination of root word meaning and the function of the prefixes and suffixes, the meaning of the entire word can be determined. This process allows a system to understand many words with only a limited dictionary of regular and irregular words.

3.2.2 Idioms and Jargon

In contrast, aspects of natural language such as idioms and jargon are totally semantically oriented. Some idioms are too complex to handle even with current AI technology but most become complicated only if they are parsed with the conventional methods. Therefore, it would probably be advantageous to consider these as semantic concepts and handle them before any normal parsing is applied. To do this, access to the semantic domain knowledge must be provided during the parse.

3.2.3 Verbs

In the Q/A paradigm, the verb performs many different functions. The first, and most obvious, is to designate the action of the query. For example, in the sentence:

Who serves chicken?

the case frame of the verb "serve" provides slots for the component noun phrases.

The second common function of the verb is to designate the operation which is to be performed by the system. When parsing the input:

Find a cheap Japanese place.

the verb "find" is interpretted as a command to return the name

of a restaurant which satisfies the constraints "cheap" and "Japanese". In the sentence:

Total the salaries of the managers.

"total" is taken to be a command to the system. Obviously, to process this particular command properly, the system must have the capability of "totalling" a field, either within the database system or within the NL interface itself. This function of the verb will be dependent only on the functions available in the database and not on the domain.

The other major use of the verb is strictly syntactic. In the sentence:

Where is White Spot?

the verb "be" is used to designate the constraint.

Auxiliary verbs are used at the beginning of a sentence to indicate that the query requires a yes-no response. This can be shown in the example:

Does Yangtzee open on Thursdays?

where the main verb is "open" and the auxiliary verb "do" is used to designate the type of answer desired.

3.2.4 Nouns

In this type of system, nouns are usually tied to the domain in some way. Proper nouns will almost always be found as values in the database whereas common nouns will be found not only as database values but also as general domain jargon. In the query:

What is on the menu at White Spot?

the proper noun "White Spot" will be found in the database but the common noun "menu" probably will not. However, both of these nouns form a part of the domain specific information.

3.2.5 Adjectives and Adverbs

Many adjectives and adverbs appear, on the surface, to be domain independent but, in reality, are not. In the query:

Which is the cheapest Greek restaurant?

the adjective "cheapest" would have a mixture of properties which would include a domain independent as well as a domain dependent part. In a Q/A system, a superlative would usually indicate that the greatest or least value of a field was desired. This would be the domain independent portion of the meaning. The domain dependency comes in deciding which field is to be examined and which ordering of field values will be used.

When the adjective or adverb is not designating a field, as in:

Find at least 4 . . .

then the word can be assumed to have only the domain independent portion of the meaning. Therefore, some adjectives and adverbs can reside in the syntactic dictionary while others must form a part of the domain specific information.

3.2.6 Prepositions

Prepositions play an extremely important role in English, especially when a case theory is being implemented. By using these words as flags it is possible to determine a limited number of uses of a phrase before actually examining the entire phrase - in some domains there may be only one reasonable meaning. For example, if we have a LOCATION field in our domain but no TIME field, then:

at the . . .

will most likely be designating some form of location. Obviously the process is not as simple as this one example illustrates because in general there are many different "flags" for any one idea (or case) as well as many different ideas for any one flag. However this method will, at the very least, provide a starting point for determining the correct interpretation.

3.3 Sophistication of Design

To design a working system is not difficult. To design a usable system, however, is. Among what are classed as aspects of design sophistication are such elements as spelling correction, general user interaction, metaquestions and knowledge acquisition.

3.3.1 Metaquestions

A major feature of any robust system is its ability to handle questions and explanations about itself, i.e. "metaquestions". If asked:

How many records are there?

it would be unreasonable for a system to retrieve every record from the database and then count them, but it should instead simply have a count ready. Similarly if asked:

What do you know about?

the system should not dump the contents of the database.

These kinds of questions should be recognized and processed with little or no database interaction. The component which identifies these questions should be domain independent because the questions themselves will be the same regardless of which domain the system is working in. The answers to the questions

are, however, both domain and database dependent. But answers to these questions will probably not be found in the database directly and therefore they must be considered part of the domain dependent information.

3.3.2 User Interaction and Communication

Any system which is designed to communicate with even partially naive users must have some way to inform the user when it is confused or needs additional information. This component of the system need only use the relevant portions of the semantic meanings of the domain dependent words in its attempts to extract the required information from the user. This type of user interaction can be guided by the system and restrict the possible user answers so that it will obtain the information it is seeking quickly. Assume a system contains inventory information for a bookstore and has never been told that "purple" is a colour. The question:

Are there any purple pens?

might produce a confusion in the system and a reasonable response would be:

I don't understand the meaning of "purple".

Is it a :

1. quality
2. colour
3. manufacturer
4. something else

This pattern could be generated by knowing the possible fields in which the word can belong or by making a prediction based on the part of the sentence processed so far. For example it might be used if the word could only be found in a few, equally probable fields and none of the fields had been chosen as the default. This answer could then be stored in the dictionary for later reference.

This type of menu driven dialogue has been shown to work well enough to allow the user to see where the system is confused, to allow the user to give a correct, understandable and concise response without forcing the user to understand the system's internal representation for queries. Additionally, the menu driven method gives the user some guidance and assistance in deciding what an appropriate answer would be. In contrast, a question such as:

What does "purple" mean?

would provide no guidance for the user at all.

Most of this menu driven dialogue can be generated by the component requiring the answer and then passed to a "user interaction" component to extract the answer from the user. This allows the user to see one consistent interface regardless of which part of the system needs the information. The controlling user interaction component, again, is not dependent upon domain.

3.3.3 Spelling Correction

The art of spelling correction is still a very ad hoc, time consuming and unreliable process. Nevertheless, it should be done if at all possible in a reasonable (not noticeable to the user) amount of time. Many software systems now employ some form of spelling correction procedure in their makeup, from simple text processing systems to complex "programmer's workbench" systems. The algorithms range from simple lookup of common spelling errors to complicated procedures where a user's typical mistakes are "remembered" by the system.

3.3.4 Knowledge Acquisition

There are many levels of knowledge acquisition, even in a simple question answering system. Some of the new knowledge comes from within the system, such as when a new word is broken apart and subsequently "understood", while other knowledge comes directly from the user, such as when a new term is defined.

Still other information can be derived from the dialogue. Some work is being done in building a psychological model of the user as the dialogue progresses. However, all of this learning - whether simple or complex - requires some use of a dynamic knowledge acquisition component which may be involved at any stage of the dialogue. A simple "add-on" feature is not enough. In our previous example, once the system has found an answer to its question and now knows what "purple" is, it should be able to save this information to use at a later date. Any future references to "purple" should not have to result in a query to the user.

A reasonable system should learn from its mistakes and thereby never make the same mistake twice. This requires modification of either some part of the program or the data. The simpler solution is to allow the program to modify its world definition. In the above example, the knowledge that purple is a colour should be easily stored in this world definition.

Since many of the "meanings" of words will be found in the database itself, it makes sense to allow the system to query the database if confused about a term. If the Q/A component is interfaced to a sufficiently fast database system, and there is only a narrow range of possible meanings of the term, this strategy could be adopted. This method has been shown to work when coupled with an inverted index of the database as discussed earlier (Harris 1977a) but with the current level of database

management system technology, it would be too slow to use as the sole source of semantic knowledge.

3.3.5 Making Assumptions

To allow the use of the system with a minimal amount of effort, assumptions must be made. Pronouns and idioms which people frequently use when communicating with each other - usually without thinking much about it - must be handled if the system is to be robust. Overall, the system must make a number of assumptions so that the user does not get bogged down by the unnatural restrictions which computer systems usually impose on their human users. Since the computer can not currently make these assumptions on its own, they must be somehow predetermined. Finding pronoun antecedents is a general enough task that it can be contained in the domain independent portion of a program. However, the interpretation of an idiom is usually tied quite closely to the domain.

3.3.6 Answer Generation

By correcting spelling errors, allowing loose and improper grammar and generally making unconfirmed assumptions, a system might suffer from one obvious problem. It is possible that the system will answer a question different to the one that was originally asked. For this reason, the original question (or what the system believes the question to be) must somehow be

incorporated into the answer. If the user has asked:

How many purple pens are there?

then, rather than a response of:

42.

a preferable answer would be:

There are 42 purple pens.

This may seem a trivial point with this example but the importance can be seen more readily when the system does not know the answer. A response of:

None.

could mean:

There are no purple pens.

but it could also mean:

I don't have any information about "purple".

or:

I don't have any information about "pens".

or even:

I don't have any information about "purple" or "pens".

All of these latter answers would be more informative by telling

the user exactly what the system does or does not know.

Actually the process of answer generation is a far more complex one than this description might portray. Some work is being done in this area but it is not at all clear how one determines the correct words or phrases to use in the answer. However, the specialized answers generated for question answering systems and the required simplicity of them limits the task to an almost manageable one.

3.4 Summary

Most of the system components examined require little knowledge of the domain in which they are working in order to function. They do, however, all require a large amount of time to develop and this effort should not have to be repeated each time a new system is constructed. The handling of loose grammar, pronoun reference and verb phrase ellipsis should have little interaction with the domain specific information. The controlling portions of the user interaction, spelling correction and learning components need only use the domain dependent information as slot fillers. Likewise the answer generation component need only use the information returned from the database as slot fillers in the generated answer. All of these components can be combined together in one domain and database independent "linguistic core" (Figure 3.3). Since

these components are virtually domain independent, they should never have to be rewritten when the system is adapted to a new domain.

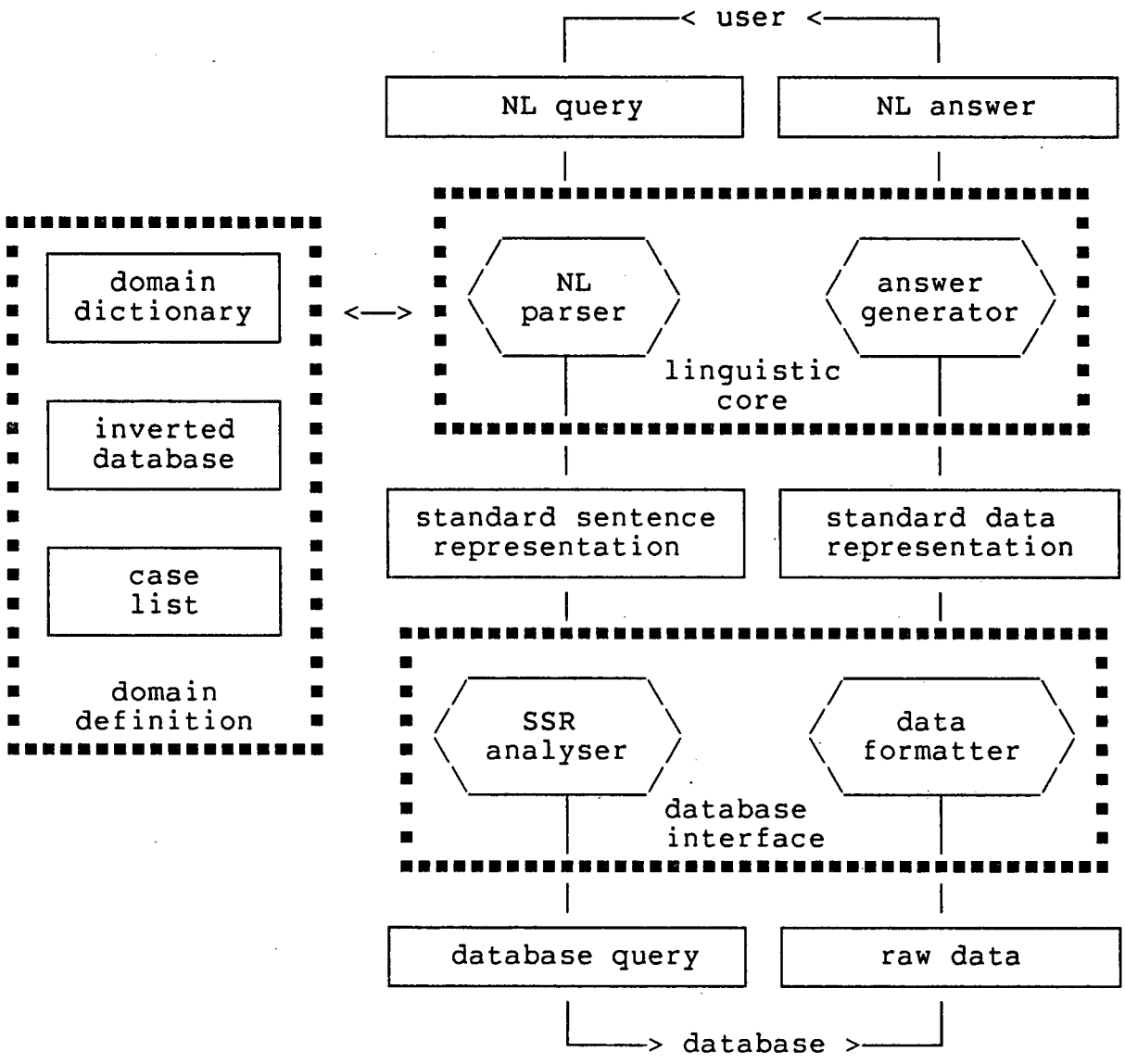


Figure 3.3: Proposed Natural Language System

While it may seem a little unconventional to suggest that the first phase of processing (parsing) and the last phase (answer generation) are combined in the same unit while an intermediate phase such as database retrieval is not, there are reasons to support such a structure. It is desirable to have the information structure which is passed from the parser to the retrieval routines be as well defined as possible. At the same time, in order to allow informative answer generation, a large amount of information both from the original sentence and from the previous dialogue must be accessable. To combine these two goals in a conventional system, the structure developed by the parser would have to be very complex indeed and the answer generator would have to be very complex to decypher it. Instead, by combining the natural language parser and the answer generator, these modules can communicate freely while a strictly defined interface between this component and the database retrieval portion is maintained.

Although parsing many universal constructs (such as quantifiers) requires little domain knowledge, it usually does require some. Furthermore, constructs such as simple idioms which definitely require semantic information can be processed much more easily if there is access to this information during the parse. The domain oriented information such as the general vocabulary, idioms, and specific jargon should also be kept in one unit. To make this module easily accessible as well as easily modifiable, a case structured declarative format is

suggested. An inverted index of the database would be most effective as a simple definition of all the world knowledge residing in the database.

To allow the system to be transferred with a minimal effort from one database management system (DBMS) to another it should be designed to query an "idealized database". An idealized database is one which has a good, basic set of functions and can be adapted easily to any "real" DBMS. This idealized database should contain only the essential functions, thereby reducing the effort needed to design the interface between the idealized database and the real database.

There are two separate functions which must be performed in the "database interface". Firstly, the output from the natural language parser must be translated into a legal database query. Secondly, the raw data returned by the database must be formatted into the structure expected by the answer generator.

Chapter 4

System Design: Part I - The Linguistic Core

In an attempt to lend credence to the concept of a domain and database independent natural language (NL) interface, a prototype question answering (Q/A) system has been constructed. It has been logically, if not physically, divided into three completely separate modules.

The user interface or linguistic core incorporates most of the features now seen in conventional Q/A systems. It is intended to form an application independent framework for database queries to which information concerning the current domain and database system can be attached. Components for NL parsing, knowledge acquisition, and answer generation are all included in this module. Whereas a structured interface exists between this linguistic core and the other modules, internal communication has been left fairly unstructured. This internal communication method, which basically consists of a number of "registers" and associated values, can be easily added to or modified to allow as much flexibility as possible.

The second logical unit is the domain definition. This module contains the definition of the particular domain in which we are working. It is a small, easily modifiable unit, smaller

in size than the linguistic core, but central to the ideas reflected in this thesis. The interface between the domain definition and the linguistic core must be strictly maintained since this definition will have to be changed and updated constantly, and without any alterations to the core itself.

The last module is also small in size compared with the linguistic core. This is the database interface module. Its purpose is to hide the real, physical database structure from the linguistic core and provide an idealized structure. Unlike the domain interface, which is based on a declarative format, this module does contain code. Fortunately though, this module should only have to be modified when adapting the system to a new database system, not when changing the domain. A formal structure has been defined which provides the basis for communication between the linguistic core and the database interface.

In this chapter we will concern ourselves with the the design of the linguistic core (Figure 4.1). The goal is to form a general purpose Q/A system which receives queries from the user, translates them into a standard sentence representation (SSR), consults the database (through the database interface) and formulates an appropriate answer, all with no notion of the domain in which it is working save for what information it can extract from the the domain interface. The general design of the core can be thought of as a three-phase process - NL

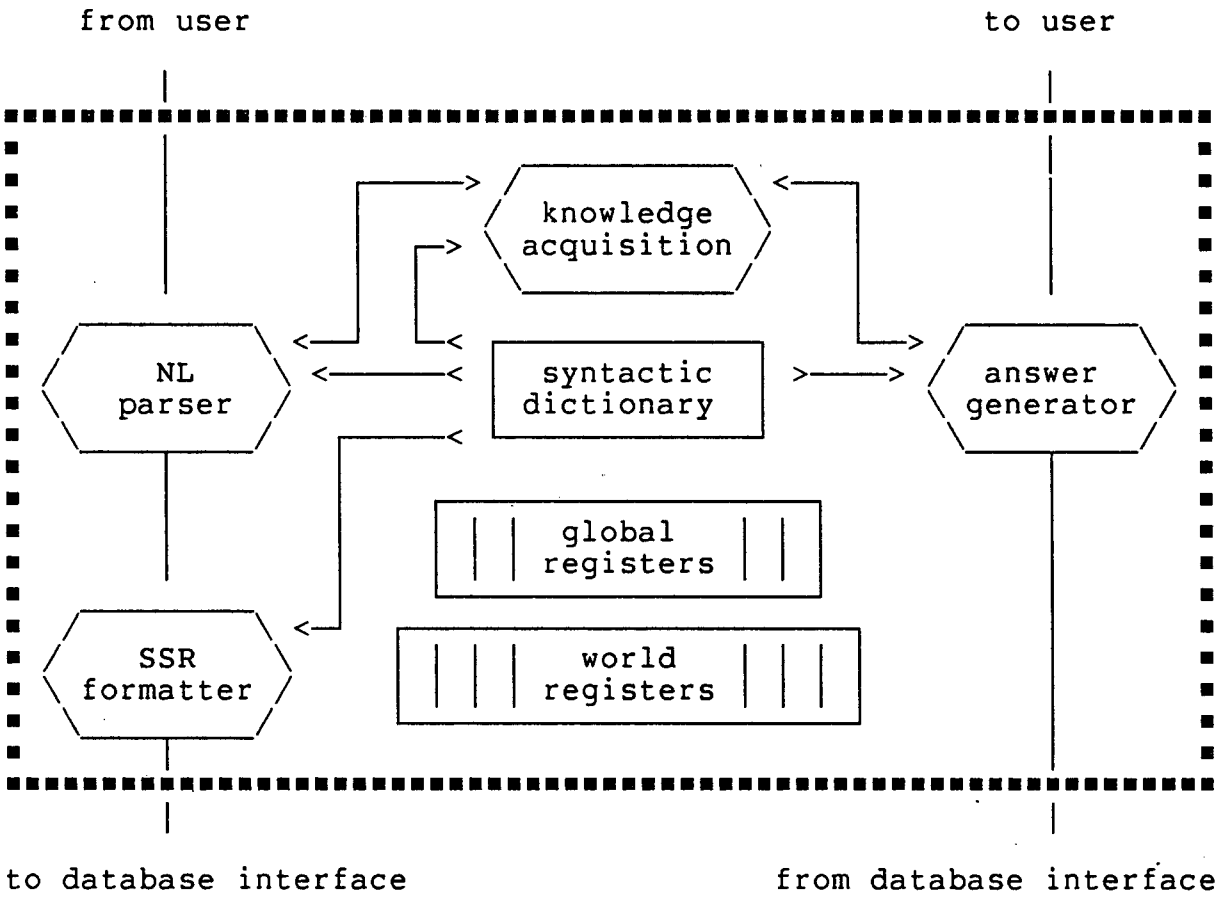


Figure 4.1: The Linguistic Core

parsing, SSR building and answer generation. In addition, there is a knowledge aquisition component and a syntactic dictionary which can be accessed from any of the other modules. A system of registers is used as an internal communication method.

4.1 The NL Parser

The job of the NL parser (Figure 4.2) is to convert an input sentence in natural language into some internal

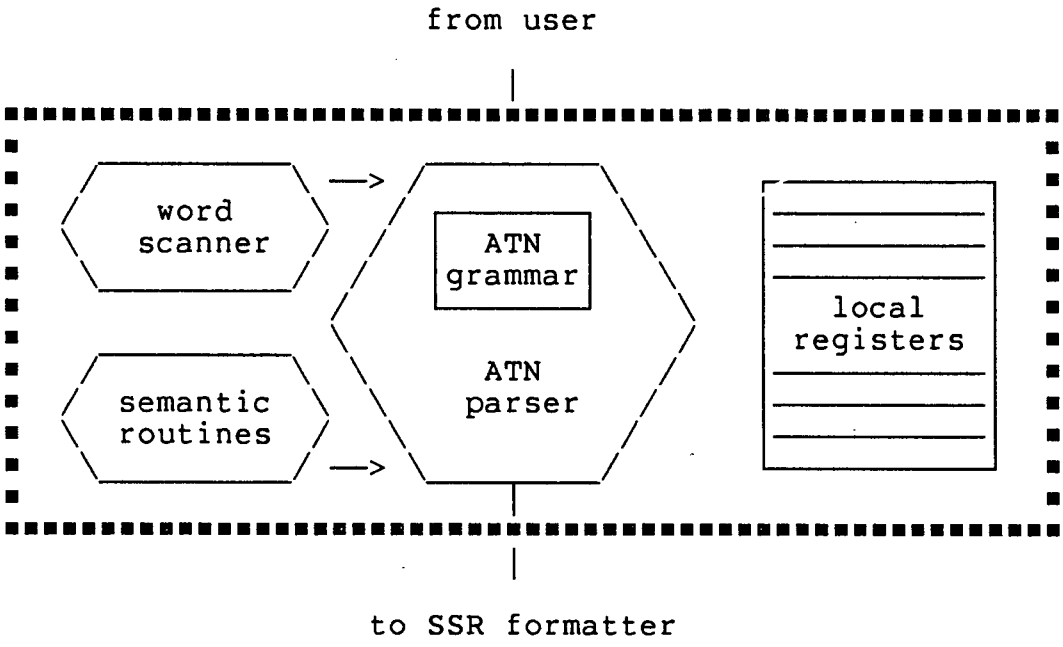


Figure 4.2: The Natural Language Parser

representation, while retaining as much of the original meaning as possible. The method of parsing used here can be described as "component parsing". A small, basic component such as a noun phrase or verb phrase is first combined together on a syntactic level and then added into the total internal sentence representation using the semantic interpretation of the component. In this way, the sentence representation is built up as the sentence is parsed. This may cause problems when it is found, part way through a parse, that a wrong decision has been made about the function of a component. However, it does alleviate the problem of having to keep all ambiguous versions around until the end of the parse. When the parser has finally finished with the sentence, there will be at most one parse generated. Another benefit of this parsing strategy is that

individual components, once they have been formed, will not be split up unless they fail some semantic test. They may, however, be switched with other components until an acceptable structure is found. This makes the parser more efficient because the components themselves will usually be correctly formed on the first attempt, even though their function in the sentence may not be known.

The NL parser is composed of a general augmented transition network (ATN) grammar parser (Woods 1970), the ATN grammar, and many small, specialized routines which handle tasks ranging from the lexical analysis of an input word to the modification of the current sentence representation to accomodate a new prepositional phrase. These components will now be examined in detail.

4.1.1 The ATN Parser

The function of the ATN parser is to produce a structural representation of the input sentence according to the ATN grammar. During this process, functions designated by the grammar are invoked to build this representation. The state of the parser is saved when a particular transition in the grammar is chosen, thus allowing for complete backup when an error is detected.

The ATN parser used was originally written in LISP by Dr.

R. Reiter (Reiter 1978) and has since been only slightly modified.

4.1.2 The ATN Grammar

The grammar used in this system is basically a syntactic grammar, augmented by calls to the semantic routines. It attempts to represent a sentence in terms of its component syntactic structures. Therefore, portions of the grammar are devoted to recognizing constructs such as noun phrases, verb phrases, determiners and quantifiers. The semantic routines are used both to verify certain semantic tests on the components as well as combine them together to form the internal sentence representation. Currently the grammar is a small, basic version; however, it should be able to be developed independently of the rest of the system to some degree. Development of the grammar is an ongoing process and whenever it is modified, since only linguistic knowledge is represented, the modifications should benefit all systems currently using it. Transition network diagrams for the grammar used here can be found in Appendix A.

Some portions of the grammar are modelled on the semantic grammar concept that constructs are parsed by looking at specific words and phrases rather than general syntactic categories. However, unlike true semantic grammars, the constructs being parsed here are linguistic in nature (e.g.

quantifer) rather than semantic (e.g. planetype (Waltz et al 1976)). The quantifier network in this system (Figure 4.3) can recognize such constructs as "at least four" and "more than three but less than 5".

```
(quant    (wrd at t (to q/super))
          (mem (not no) t (setr qneg t) (to q/comp))
          (jump q/comp t)
          (tst (qvalue *)
             (add-quant (getr qconj)
                        (qvalue *) nil (nvalue *))
             (to q/conj))
          (jump q/num t (setr q (qvalue 'exact))))

(q/comp   (cat adv (getf comparative)
             (setr q (qvalue *))
             (to q/than)))

(q/than   (wrd than t (to q/num)))

(q/super  (cat adv (getf superlative)
             (setr qneg t)
             (setr q (qvalue *))
             (to q/num)))

(q/num    (push number t
             (add-quant (getr qconj) (getr q) (getr qneg) *)
             (to q/conj)))

(q/conj   (cat conj t
             (setr qconj *)
             (setr q nil)
             (setr qneg nil)
             (to quant))
          (wrd of t (to q/reset))
          (jump q/reset t))

(q/reset  (jump q/acc t
             (setr quant (get-g (current-g 'quant)))
             (reinit-g 'quant)))

(q/acc    (pop (getr quant) (getr quant))))
```

Figure 4.3: The Quantifier ATN Network

Although the semantic grammar idea is useful in this situation, the database implementor should not be required to develop new code when he or she defines a new database. For this reason, semantic grammars have only been allowed in the linguistic core portion of the program. The non-programming techniques of setting up a new domain will be discussed later, but it is sufficient to say now that no modification of the grammar should be necessary when a new domain is defined.

4.1.3 Scanning

A large part of any NL system is devoted to the identification of the basic units (or words) of the input sentence. This component is concerned with identifying root words, compound words, abbreviations, synonyms and even database elements. In most cases, this is not a deterministic process - especially if it is done before the parse begins. In this system, the scanning is done during the parse in order to allow as much information as possible to be used in word identification. The various scanning procedures will now be examined.

4.1.3.1 The Morpher

The function of the morpher is to strip prefixes and suffixes from an input word in a systematic fashion to produce the root form. The rationale for including one in a NL parser

is to reduce the actual number of words needed in the dictionary. For any word, we should be able to determine its meaning from the combined meanings of its root and the prefixes and suffixes attached to it. Unfortunately, this means that the morphological information must somehow be included with the word in the dictionary. For most words this is quite a simple process but for some it can become rather complex (see Section 4.2).

The root-finding method used in this system is quite simple and straightforward. In turn, each possible suffix is removed from the candidate word. If the root is found to be in the dictionary and the word category agrees with that expected, the new word is entered into the dictionary. A table of some of the regular suffixes which are examined is in Figure 4.4.

This method works well for regularly inflected words whose morphological information is easy to store in the dictionary. For example, the morphological information needed for a verb in this system are the suffixes to add to form the present and past tenses. They are stored in the syntactic dictionary as:

(SERVE V S-D)

For a noun, the information required is the pluralizing suffix:

(DATUM N A)

and for an adjective, the suffixes required to form the

ending to remove	new ending to add	word category	features of new word
s		s-noun	plural
s		s-d-verb	present tense & 3rd person singular
s		s-ed-verb	present tense & 3rd person singular
es		es-noun	plural
es		es-ed-verb	present tense & 3rd person singular
ies	y	es-noun	plural
ies	y	es-ed-verb	present tense & 3rd person singular
's		s-noun	possessive
's		es-noun	possessive
's		proper-noun	possessive
's		pronoun	possessive
s'	s	proper-noun	possessive
s'	s	es-noun	possessive
s'	s	s-noun	possessive
ied	y	es-ed-verb	past participle & 2nd person plural
ed		es-ed-verb	past participle & singular-plural
ed		s-ed-verb	past participle & singular-plural
d		s-d-verb	past participle & singular-plural
ing		s-d-verb	present participle
ing		s-ed-verb	present participle
ing		es-ed-verb	present participle
ing		irr-verb	present participle
ing	e	s-d-verb	present participle
**ing	*	s-ed-verb	present participle
**ed	*	s-ed-verb	singular-plural past participle
est		er-est-adjective	superlative
**est	*	er-est-adjective	superlative
iest	y	er-est-adjective	superlative
st		r-st-adjective	superlative
er		er-est-adjective	comparative
**er	*	er-est-adjective	comparative
ier	y	er-est-adjective	comparative
r		r-st-adjective	comparative
ices	ex	es-noun	plural
a	um	a-noun	plural

Figure 4.4: The Suffix Table

comparative and superlative forms must be available:

(NEW ADJ ER-EST)

Since these words are regularly inflected, certain linguistic rules can also be applied. One such rule is to double the "n" in "run" before adding "ing" to form the participle. The root of an irregular word cannot usually be found with a word morpher. Therefore, these words must be initially stored in the dictionary along with all of their inflections.

4.1.3.2 Compound Words

Compound words are those which, although separate lexical items, function as a single unit. For these words it appears to be more beneficial to treat them as a single unit rather than as separate parts. However, most of the individual words have meanings of their own and so the system must allow for combination errors. The strategy adopted here to allow both compound words and the individual parts to exist simultaneously, is to first join the longest string which exists in the dictionary. If the parse subsequently fails, the scanning routines back up one level and attempt to use the next longest compound. For example, the name:

University of Illinois Chicago Circle

would first be parsed in the full form and then, if the parse fails, the successively smaller chunks:

1. University of Illinois Chicago
2. University of Illinois
3. University of

would be tried until finally the one word "university" would be attempted. In this example, "University of Illinois Chicago" and "University of" would probably not be found in the dictionary and so they would not be accepted as valid compound words.

4.1.3.3 Abbreviations and Synonyms

An abbreviation is considered to be a substitution of one word for another at the lexical level. Therefore, if the word "can't" is defined as an abbreviation of "can not", the substitution will occur before the word "can't" is ever morphed. A synonym, on the other hand, is considered to be a substitution at the root word level. If the verb "display" is defined as a synonym for "show", then "displaying" will be considered synonymous with "showing".

4.1.4 Semantic Routines

Specialized semantic routines are invoked by the grammar to build an internal representation of the original input sentence.

This internal representation is nothing more than a set of values for the global registers in the linguistic core. These values are subsequently used to format the standard sentence representation (SSR) (see Section 4.5). In addition to processing the sentence components such as noun and verb phrases, routines are included which handle conjuctions and find pronoun antecedents.

4.1.4.1 Adding a Noun Phrase

After the noun phrase (NP) has been syntactically determined, a semantic routine is called to integrate it into the existing sentence representation. Depending on the characteristics of the NP and the current representation, a number of things can happen. The first step is to determine whether it is the nominative, dative or accusative case. One of the rules used in this determination is that it will be assumed to be nominative if it is the first element of the sentence. Later, this assumption may have to be revoked owing to the influences of subsequent components.

In the sentence:

I can be served chicken at which restaurants.

the first NP found is composed of the single pronoun "I". After determining this, and having no information to the contrary, the NP will be assumed to be the agent of the sentence. The first

noun phrase in the sentence will also be saved for future pronoun antecedent determination (see Section 4.1.4.5).

Next, the verb phrase (VP) "can be served" will be constructed and the sentence will be found to be passive (see Section 4.1.4.2). At this point, the system will notice that it has made a judgement error about the role of the first NP and will have to modify the structure which it has built. The actual role of the NP "I" in the sentence is in the recipient case.

The next NP to be constructed will consist of the noun "chicken". In determining its role, the case filler restrictions of the verb will be taken into account. Since the verb "serve" can take a food type as the patient but not as the agent, "chicken" must fill the patient case. The NP constraints in the final sentence representation will be:

RECIPIENT: I

and:

PATIENT: (FOOD = CHICKEN)

The handling of the prepositional phrase:

at which restaurants

is discussed in Section 4.1.4.4.

4.1.4.2 Adding a Verb Phrase

When a verb is defined in a particular domain, the cases the verb allows and the relevant fields which can fill each case must be specified (see Section 4.2.2). When the parser tries to add a verb to the current sentence representation, the major task is to see that the noun and prepositional phrase units which have been found so far fit into the designated cases of the verb. This allows disambiguation of a noun element which may be found in more than one field. If the sentence turns out to be passive, the role of the initial noun phrase (NP) must be redetermined. For example, in the previous example:

I can be served chicken at which restaurants.

the NP "I" has been found before the VP "can be served". Because of this, the NP "I" was initially assumed to be the agent of the sentence. When the sentence is deemed to be passive, the system must find out what the real role of this NP is.

When adding the main verb to the sentence representation, the properties of each NP determined are checked to see that they can indeed fill the case slot of the action to which they have been designated. If one can not, a number of things may happen. Sometimes routines are called which will switch the case filler components until the structure is valid but, usually the possible roles of a given NP are severly limited and when

this happens, the parse will usually fail.

Auxiliary verbs contribute little to the overall sentence representation. Their main functions here are to designate a YES-NO question when they are found at the beginning of a sentence as in:

Does White Spot serve chicken?

and to make a sentence passive when found in conjunction with a main verb:

Chicken is served by which restaurants.

Relative clauses are sometimes introduced by a verb participle:

Find a restaurant serving chicken or steak.

When this happens, routines are called which suspend structure building at the current level and continue at a lower level. When this lower level processing is completed, the processing of the original structure is resumed.

Sometimes a situation will occur where the defined format of a verb should be overridden. Assume that the definition of the verb "serve", in the restaurant database, takes as a patient case the fields "food" or "meals". Then if the parser came across the unexpected sentence:

Who serves Hastings Street?

it should be able to override the definition of serve and generate the correct parse:

(AND (NAME = ?) (ADDRESS = HASTINGS STREET))

4.1.4.3 Adding Noun Phrase Modifiers

Many things fall into the category of noun phrase modifiers and they are all handled similarly by the system. Some of these are adjectives, ordinals and quantifiers. They are saved in local registers when found and added into the sentence representation when the head noun is determined.

For example, in the sentence:

Find all of Schank's recent books.

the possessive "Schank's" would be stored in the register NPMOD1 as:

NPMOD1: (AUTHOR = SCHANK)

Next, when "recent"† is found, the structure associated with NPMOD2 will be:

NPMOD2: (DATE > 1980)

---

† The definition of recent used here (later than 1980) is arbitrary. It would be defined by the database administrator and found in the domain dictionary (see section 5.1.1).

After the head noun "books" is finally found, all of the NP modifiers will be combined into one general modifier as:

NPMOD: (AND (AUTHOR = SCHANK) (DATE > 1980))

and this modifier will then be added into the global register sentence representation.

By creating a new modifier register for each NP modifier encountered, the system can handle a virtually infinite number of NP modifiers.

4.1.4.4 Prepositional Phrases

The prepositional phrase (PP) is very important in the Q/A paradigm. It is with these that many of the query constraints are determined. In a case-driven system such as this, the preposition is used to designate the possible cases which the associated NP can fill. Then, using this information along with the current sentence representation, the system can determine the actual function of the attached noun phrase. The definition of prepositional information will be discussed in the section on the syntactic dictionary (Section 4.2.5).

For example, if the system has defined the preposition "on" to handle the location and the time cases, then in the sentence:

Which restaurants are on Granville Street?

the system has the choice of either filling the location or the time case. When the NP "Granville Street" is found to designate a place, the disambiguation can be done. After the PP parse is finished, the constraint:

LOCATION: (ADDRESS = GRANVILLE STREET)

will be added into the sentence representation.

4.1.4.5 Finding Pronoun Antecedents

Only extremely simple pronoun reference is currently handled by the system. Specific pronouns referring to "it" and "them" are taken to refer to the last item retrieved by the database routines. Although this is an extremely naive view of pronoun reference, the methods used here can be expanded to include more complex cases. The reason for adding this component at all was that, even with only limited capabilities, it can help the user enormously. This simple solution can handle such constructions as:

How many restaurants serve chicken?
THERE ARE 2 REFERENCES.
Who are they?
THEY ARE "STEER AND STEIN" AND "WHITE SPOT".

The simplicity of this system is not inherent in its design, but rather is a function of the time and effort alloted

to the development of the individual components.

4.1.4.6 Conjunctions

Conjunctions cause some of the ambiguity of natural language. However, they can be used unambiguously and, at least in this form, must be allowed even for a simple NL system. For example, the conjunction "and" in:

Find some place which serves steak and lobster.

would cause no ambiguity, generating a query to satisfy the constraints:

(AND (FOOD = STEAK) (FOOD = LOBSTER))

On the other hand, the query:

How many people are coming from CMU and SRI?

is a little harder to process. Rather than generating the set intersection constraint:

(AND (INSTITUTION = CMU) (INSTITUTION = SRI))

which would try to find the people who come from both CMU and SRI, the user really wants to generate the set union constraint:

(OR (INSTITUTION = CMU) (INSTITUTION = SRI))

which should find people who are coming from either CMU or SRI. This subtle fact should somehow be recognized by the system. In

simple cases this can be handled by changing an "and" to an "or" if the field being processed can have only one value at a time. In the first case, the FOOD field could have more than one entry because a restaurant can obviously serve more than one type of food. However, in the second case, the INSTITUTION field would be single-valued because a person would (usually) come from only one institution.

Simple conjunctions are handled by combining all conjuncted components under one of the two categories AND or OR. These two "functions" are represented in the internal sentence representation (and also in the SSR) by *AND and *OR respectively and have a syntax of:

(*AND constraint1 constraint2 constraint3 . . .)
(*OR constraint1 constraint2 constraint3 . . .)

4.1.5 Local Communication

While building the internal sentence representation, any values which will be needed by another part of the parser are put into local registers. Later, the routine needing this information can easily retrieve the contents of the register. The use of these registers closely parallels that of the global and world registers used to communicate between various parts of the linguistic core. For further details refer to the section describing the function of these registers (Section 4.4).

4.2 Syntactic Dictionary

The syntactic, as compared to the semantic or domain, dictionary contains information relating to the syntactic and morphological properties of the words. Words relating specifically to one database will not be found here. Database values will probably be found in the inverted index (see Section 5.1.2) and domain specific verbs and nouns will be found in the domain dictionary (see Section 5.1.1). Most of the syntactic dictionary is taken up with common words such as determiners, pronouns, quantifiers and conjunctions. A large part of this dictionary is devoted to the definition of prepositions since they play an important role in most case-driven Q/A systems. The morphological information included varies with each word category but usually designates suffixes which might be added to the root word to form regular conjugations. The kind of syntactic information present also depends on the word category.

Irregularly inflected words pose quite a different problem. Any word which will be used often (such as "be") will be initially stored in the syntactic dictionary along with all of its conjugations. However, some words are not common enough to be initially put into the dictionary and some may simply be new to the system. This is a problem which one might think time would overcome. Surely, sooner or later all necessary words would have been entered in the dictionary. Unfortunately this is not the case and if our system expects this information, it

must be present. To aid in this task, a limited knowledge acquisition component (see Section 4.3) has been included. This component allows new words to be entered and spelling errors to be corrected by the user during the parse.

To see exactly what type of information is included in the syntactic dictionary, the definition of nouns, verbs and prepositions as well as synonyms, abbreviations and compound words will be discussed.

4.2.1 Noun Definition

Included in the category of nouns are common nouns, proper nouns and pronouns. The morphological information required for a common noun is the suffix which must be added to form the plural. Examples of these are:

```
(NUMBER       N    S)
(BOX          N    ES)
(INFORMATION  N    MASS)
(DATUM        N    A)
```

Proper nouns and pronouns are not commonly pluralized and so, no morphological information is stored with them. However, the morpher has been designed to allow reasonable proper noun pluralizations such as in:

How many McCarthys are coming to the conference?

The semantic information included depends upon the actual words. Domain specific words are discussed in Section 5.1.1. Any domain independent common nouns currently have no semantic information associated with them and so they are effectively ignored by the parser.

Pronouns such as he, she, everybody and anybody, have along with their morphological information, semantic information which includes both their category (general, question or relative) and any cases which they may designate. Examples of pronoun definitions are:

```
(ITS        PRO  (IT POSS))
(SOMEWHERE  PRO  *  PRO*  (GENERAL (CASES (LOCATION))))
(THAT       PRO  *  PRO*  (RELATIVE))
```

There are currently no domain independent proper nouns in the system.

4.2.2 Verb Definition

There are three classes of verbs in this system; auxiliaries, commands and actions. The definition of an auxiliary verb includes its root form and any semantic features such as the tense and modal characteristics. Some examples of auxiliary verb definitions are:

```
(AM   V (BE (TNS PRESENT) (PNCODE 3SG)))
(DONE V (DO (TNS PASTPART)))
(CAN  V *  V* ((TNS PRESENT)(PNCODE ANY)(AUX MODAL)))
```

Commands are used to designate possible database functions. When used in a sentence as an imperative, the verb takes the command definition. If we had a system routine DRAW-GRAPH which we wanted to invoke with the command "graph", it would be defined as:

```
(GRAPH COMMAND DRAW-GRAPH)
```

Actions are usually found only in the domain dictionary (Section 5.1.1), but some have been included here as examples. An action is defined by its morphological features as well as its semantic case frame. The morphological features are the endings to add to form the present and past tenses:

```
(SERVE V S-D)
(EAT   V IRR)
(ATE   V (EAT (TNS PAST)))
```

The case frame is implemented here as a list of possible cases of the verb. Not all possible cases need be included but, rather, only the ones which are important in the domain. For example, in the restaurant database "serve" and "eat" are

defined as:

```
(SERVE ACTION (AG NAME PA (FOOD MEALS) RE *HUMAN))
(EAT   ACTION (AG *HUMAN PA (FOOD MEALS))
```

The order in which the fields for each case are listed is used to determine a default priority ordering on them. For example, if the question was asked:

What does White Spot serve?

because of the ordering, the constraint which would be generated would be:

```
(AND (NAME = WHITE SPOT) (FOOD = ?))
```

If the type of meals was desired, the query would have to be:

What meals does White Spot serve?

4.2.3 Adjective Definition

As is the case with both nouns and verbs, adjectives are usually domain specific. The morphological information which must be supplied is the suffixes required to form the comparative and superlative conjuncts. Many times the semantic information associated with an adjective is arbitrary. In the bibliography database, "recent" is defined as:

```
(DATE > 1980)
```

and in the restaurant database, "good" is defined as:

```
(STARS > 3)
```

4.2.4 Quantifier Definition

Quantifiers are sufficiently general to be found in the syntactic dictionary. They usually have a quantifier value (QVALUE) and/or a numeric value (NVALUE) associated with them. Some examples of QVALUEs are EXACT, MORE, and LESS. Examples of NVALUEs are 0, 1, 2, 3 and ALL. Some examples of quantifier definitions in this system are:

```
(COUPLE  QVALUE  *EXACT  NVALUE  2)
(FEW     QVALUE  *MORE   NVALUE  2)
(NONE    QVALUE  *EXACT  NVALUE  0)
```

4.2.5 Preposition Definition

Prepositions play an important role in this system. However, their definition is rather simple. The main part of their definition is a list of which cases they can refer to. For example the prepositions "at" and "to" are defined as:

```
(AT  PREP*  ((CASES  LOC TIME)))
(TO  PREP*  ((CASES  REC  DEST  BEN  PURP)))
```

These cases are used to fill slots in the definition of the main

verb in the sentence. See Appendix B for the list of cases supplied. Appendix C contains a sample syntactic dictionary with some prepositions and the cases they flag.

4.2.6 Synonyms and Abbreviations

Synonyms and abbreviations both perform the similar function of allowing the substitution of one word (or a group of words) in a sentence for another. The main distinction made between the two in this implementation is that a synonym substitution occurs at the root word level while an abbreviation substitution occurs at the lexical level. These concepts are very important because they allow a small, carefully defined core of information to be expanded simply into a large subset of natural language.

Synonyms are primarily used to inform the parser that two different words have the same meaning. For example, in most Q/A systems, the meanings of the commands "find", "show", "display", "print" and "list" would be the same. The definition of these can be made by defining only one (say "find") completely and then defining the others as synonyms:

```
(FIND     . . . complete definition)
(SHOW     SYNONYM  FIND)
(DISPLAY  SYNONYM  FIND)
(PRINT    SYNONYM  FIND)
(LIST     SYNONYM  FIND)
```

As well as allowing different verbs to appear the same, the synonym feature can also be used to allow different meanings of the same verb. For example, suppose that in this system, there exist three different meanings of the verb "take". These could all be defined by:

```
TAKE  SYNONYM (TAKE1 TAKE2 TAKE3)
TAKE1 . . . first meaning
TAKE2 . . . second meaning
TAKE3 . . . third meaning
```

Here the synonym feature is used to show that the verbs TAKE1, TAKE2 and TAKE3 are all really the verb "take". If the input is:

```
Who is taking reservations?
```

then, when the parser is trying to understand "taking", these steps will be followed. First the root word "take" will be found. Next, the system will discover that the word is a synonym of TAKE1, TAKE2 and TAKE3. The morpher will then return the definition of the word TAKE1 to the parser. It is not until the parse fails using this definition that TAKE2 will be considered. This means that the definitions of TAKE1 through TAKE3 should be sorted by plausibility so that the correct one will be found as soon as possible. The actual meaning definition of these verbs will be found in the section on verb definitions (Section 4.2.2).

Abbreviations, since they are processed before any normal parsing is initiated, can be used to define simple lexical idioms. But the most important use of abbreviations is to define jargon common to the domain (see Section 5.1.1).

4.2.7 Compound Word Definition

Each compound word is defined in the dictionary as a list of words forming the compound. For example, the restaurant name "White Spot" might be defined as:

```
((WHITE SPOT) NPR *)
```

The system prefers to manipulate the determiner "how many" as a single unit and so it has been defined as:

```
((HOW MANY) DET . . .)
```

4.3 Knowledge Acquisition

The major knowledge acquisition component in this system is involved with learning new words. There are several situations when this will happen. When a word is broken apart by the morphological routines and its properties are determined, this new word is then entered into the dictionary so that subsequent references to the word are found more efficiently. If the word cannot be analyzed by the system, then the user is asked to

clarify it. If this is successful, the new word is entered into the dictionary with this definition. The third way for the system to "learn" a new word is by querying the database. After finding the previously unknown term in the database, it will be entered into the domain definition for future reference.

This sample dialogue from the restaurants database will show the route taken to determine the meaning of an unknown word:

```
Who serves artichokes?

I CANNOT FIND ' ARTICHOKES ' IN THE DICTIONARY.
DO YOU WANT ME TO STOP PROCESSING THE QUERY?
no

DID YOU MISSPELL ' ARTICHOKES '?
no

WOULD I FIND ' ARTICHOKES ' IN THE DATABASE?
no

WOULD IT BE SAFE TO IGNORE THE WORD ' ARTICHOKES '?
no

DO YOU WISH TO ENTER ' ARTICHOKES ' INTO THE DICTIONARY?
no

ERROR >> ' ARTICHOKES ' CANNOT BE MORPHED.
```

An important benefit of keeping all of the user interaction in one unit, besides the obvious one that it is easier to modify, is that the user is facing a consistent interface and should know what response was expected from a particular question. Each of the attempts to learn a new word will now be examined in detail.

4.3.1 Spelling Correction

If, when asked:

DID YOU MISSPELL ' ARTICHOKES '?

the user had typed in "yes", he would have been prompted for a replacement. This replacement would have then been used throughout the rest of the parse.

In the current system, there is no attempt at automatic spelling correction, but this should certainly be a part of any real-world NL system.

4.3.2 Database Search

If there exists an inverted index of the database (Section 5.1.2), the possibility is that no intermediate database searches will be required. However, sometimes the inverted index has not been kept up-to-date. Then, if the word is not in the inverted index, it will be necessary to look in the database for it to make sure that it has not been just recently added. This can be done automatically by the NL system if there are some clues as to the field in which the unknown field might be found.

In this system, if the word can not be found in the inverted database, the system will ask:

WOULD I FIND ' ARTICHOKES ' IN THE DATABASE?

If the user responds with "yes" then the system will ask for the expected field and then search this field for the value.

4.3.3 Ignoring Words

When processing any natural language sentence, there occur many words which could be safely ignored without affecting the meaning of the entire sentence. This assumes that the word conveys no useful information to the processing of the sentence. Words like "please" and "thank you" can usually be ignored wherever they appear in the sentence. Others can only be ignored at certain points in the parse. It is important to remember, however, that no word can safely be ignored if it cannot be first identified by the system. Therefore, if the system does not know the meaning of a word, it must, if all else fails, ask the user for a definition or if the word can be safely ignored. The use of such a procedure can be shown when the user enters the query:

How many books did Noam Chomsky write?

Since the system has no information on first names of authors, it can not identify the word "Noam". It then asks the user:

WOULD IT BE SAFE TO IGNORE THE WORD ' NOAM ' ?

which, when the user agrees, will initiate a query satisfying

the constraint:

(AUTHOR = CHOMSKY)

Furthermore, the user is given the added information that the system has no information on "Noam". If it were the case that first names were in the database but that "Noam" was not, the initial query would have been effectively answered and the processing could be stopped. While this method requires more user interaction, it seems superior to simply producing an answer such as "none" which would not convey the same information.

4.3.4 Entering New Words

To enter a new word, one must currently give the complete dictionary definition (Section 4.2) for the new word. However, a complete NL system should allow for a smoother user interaction for the definition.

4.4 Internal Communication

The communication between parts of the parser and between the parser and other parts of the linguistic core is managed through three sets of registers. The local registers hold values for a short time and are used primarily within the NL parser to gather information about a particular construct (such

as a noun phrase or a verb phrase) before the component is added to the internal sentence representation. For example, when parsing the noun phrase:

a fast food place

after the determiner "a" has been found, the knowledge that the noun phrase is singular can be stored in the local register NUMBER by:

```
(SETR NUMBER 'SG)
```

The global registers are used to store information about the portion of the sentence which has already been parsed. For example, in the sentence:

Who is open for lunch and serves Chinese food?

the information that the sentence is in the present tense can be stored in a global register after the first verb phrase has been added to the internal sentence structure. This is done by:

```
(SETR-G TENSE 'PRESENT)
```

The stored information can be retrieved and verified when the second verb is being parsed by the function call:

```
(GETR-G TENSE)
```

which would return the value "present".

The third class of registers are the world registers. These represent the long term memory and contain information about the continuing dialogue. This information is currently only used for finding pronoun antecedents but could also be used for building a "model" of the user to aid in providing an answer more tailored to his needs. An example of what information might be stored in a world register is:

```
(SETR-W AGENT (GETR-G AGENT))
```

which would save the current agent for future reference by copying it from a global to a world register.

4.5 Building the Standard Sentence Representation (SSR)

The basic units upon which the standard sentence representation (SSR) is built are the cases. Each component is assigned a particular case to fill (or function to perform) in the current structure. Each filled case then becomes a constraint in the query to the database. The cases are designated when the verbs of the system are defined (see Section 4.2.2 for the definition of verbs).

After the parsing routines have developed an internal sentence representation of the query, the SSR is produced. This new structure provides a strictly defined (Figure 4.5) communication path between the linguistic core and the database

```
              SSR  ::=  ( STYPE CONSTRAINT )
            STYPE  ::=  whfind  |  yes-no
       CONSTRAINT  ::=  SIMPLECONSTRAINT      |
                        ( *and CONSTRAINT* )  |
                        ( *or CONSTRAINT* )   |
                        ( *not CONSTRAINT )
 SIMPLECONSTRAINT  ::=  ( FIELD RELATION ELEMENT )
            FIELD  ::=  fieldname  |  *number  |  *ref
         RELATION  ::=  =  |  =  |  <  |  <=  |  >  |  >=
          ELEMENT  ::=  elementvalue  |  ?  |  *
```

Figure 4.5: The Standard Sentence Representation

interface. The SSR attempts to capture the portion of a query's "meaning" which is relevant for extracting the answer from the database. Some information is lost in this structure because attempts are made to make it as simple as possible for the database interface to interpret and so the possibility always remains of an incomplete or erroneous answer. To build the SSR, the relevant portions of the current internal representation of the query are selected and formatted according to the definition. By using this two step method of internal representation and SSR, the internal representation can be modified simply without modification to the database interface routines. Additionally, all information needed for informative answer generation and finding a pronoun antecedent can be retained in the internal representation without cluttering the

SSR and without forcing the database routines to understand, or even simply ignore, this extra information. Some example SSRs are:

1) Who serves chicken?

```
(WHFIND
   (*AND (NAME = ?)
         (FOOD = CHICKEN)))
```

2) Where is the Empress of China?

```
(WHFIND
   (*AND (ADDRESS = ?)
         (NAME = EMPRESS OF CHINA)))
```

3) Find 4 restaurants that serve Japanese food.

```
(WHFIND
   (*AND (NAME = ?)
         (*NUMBER = 4)
         (FOOD = JAPANESE)))
```

4) What is on the menu at White Spot?

```
(WHFIND
   (*AND (FOOD = ?)
         (NAME = WHITE SPOT)))
```

The SSR formatting component retrieves its information from the registers of the short and long term memory (global and world registers). The "data" for the final SSR is taken from the case definition registers and the "control information" is taken from other, currently somewhat ad hoc, registers in memory. Only information which provides a constraint for the query is extracted from the registers and used in the SSR. For

example, in the query:

Which restaurants will serve me chicken?

the recipient case register will be filled by "me" (in fact it really contains the filler *HUMAN). Since this concept will supply no extra information to the query, it is ignored when creating the SSR:

```
(WHFIND
   (*AND (NAME = ?)
         (FOOD = CHICKEN)))
```

4.5.1 Reducing the SSR

After the SSR has been built, the formatting routines reduce it to a cannonical form, removing any unnecessary conjunctions. For example, for the query:

Who serves steak and lobster?

the initial SSR created will be:

```
(WHFIND
   (*AND (NAME = ?)
         (*AND (FOOD = STEAK)
               (FOOD = LOBSTER))))
```

and the reduced version will be:

```
(WHFIND
   (*AND (NAME = ?)
         (FOOD = STEAK)
         (FOOD = LOBSTER)))
```

4.5.2 Default Search Fields

During SSR formatting, a check is made to determine the default for any ambiguous field. In the query:

What does White Spot serve?

it is unclear from the definition of the verb "serve", which is:

(SERVE V S-D ACTION (AG NAME PA (FOOD MEALS) RE *HUMAN))

whether the field designating a type of food or the field designating a type of meal should be searched. Because of the ordering of the field list at definition time, the "food" field is taken as default. The SSR produced is:

```
(WHFIND
   (*AND (NAME = WHITE SPOT)
         (FOOD = ?)))
```

4.5.3 Counting Database Items - *NUMBER

Providing a count of items in a certain field is used so often in any database query language that it must be somehow be handled by the overall system. Because many database management systems will process this type of request faster than complete retrieval, it has been added as a part of the SSR definition, thereby allowing the DBMS to know that no actual retrieval of the records is necessary. The method used here to handle this feature was to use the imaginary field *NUMBER when an item

count is to be returned rather than the actual items themselves. For example, in:

How many foods does the Yangtzee have?

the SSR generated will be:

```
(WHFIND
   (*AND (FOOD = ?)
         (*NUMBER = ?)
         (NAME = YANGTZEE)))
```

The reason for handling the count as an imaginary field rather than as a separate sentence type (e.g. WHCOUNT) can be seen in the SSR for†:

What is the address and number of dishes of Yangtzee?

which would be:

```
(WHFIND
   (*AND (ADDRESS = ?)
         (*AND (FOOD = ?)
               (*NUMBER = ?))
         (NAME = YANGTZEE)))
```

which should return a count of the different foods available as well as the restaurant's location.

The *NUMBER field is also used to limit the number of answers printed. For example, the SSR for:

Find at least 4 and not more than 6 Greek restaurants.

---

† Sentences such as this cannot as yet be handled by the system even though the SSR allows for them.

will be:

```
(WHFIND
   (*AND (NAME = ?)
         (*NUMBER >= 4)
         (*NUMBER <= 6)
         (FOOD = GREEK)))
```

4.5.4 Using an Auxiliary Verb as a Main Verb

The check for use of an auxiliary verb as the main verb of a sentence is done here. At the end of the parse of the query:

Where is the Seven Seas?

the system is left expecting a main verb. The formatter determines whether the verb "be" is being used as a main or an auxiliary verb and generates the SSR:

```
(WHFIND
   (*AND (ADDRESS = ?)
         (NAME = SEVEN SEAS)))
```

4.5.5 Verb Phrase Ellipsis

Limited verb phrase ellipsis handling is done by the SSR formatter. The world registers are used to hold information from one query to the next. After a sentence such as:

Who serves steak?

the main verb "serve" is stored in one of the world registers.

If the next query entered was simply:

Steak?

then the system would infer the query to be:

Who serves steak?

and produce the SSR:

```
(WHFIND
   (*AND (NAME = ?)
         (FOOD = STEAK)))
```

4.5.6 Pronoun Reference - *REF

If the antecedent for any pronoun has not already been found before SSR formatting, it will be found at this time. For this, another imaginary field has been included in the SSR definition - *REF. The only pronoun reference currently handled by this system is in using the previous result from a search. If the user has asked:

How many restaurants serve Chinese food?

THERE ARE 14 REFERENCES.

then the next query might be:

What are their names and locations?

which would produce the SSR:

```
(WHFIND
   (*AND (*REF = *)
         (NAME = ?)
         (ADDRESS = ?)))
```

The "constraint":

```
(*REF = *)
```

informs the database interface to use the previous result in the current search.

4.5.7 Embedded Noun Phrases

A problem occurs when parsing sentences such as:

What is the White Spot on Granville's phone number.

Although the SSR generated is:

```
(WHFIND
   (*AND (NAME = WHITE SPOT)
         (*AND (ADDRESS = GRANVILLE STREET)
               (PHONE   = ?))))
```

we can see that the constraint:

```
(PHONE = ?)
```

has been attached to the SSR in the wrong place. Although the reduced form will be:

```
(WHFIND
   (*AND (NAME    = WHITE SPOT)
         (ADDRESS = GRANVILLE STREET)
         (PHONE   = ?)))
```

and this, when passed to the database, will generate the correct answer, the original attachment of a NP which is embedded within another is still rather limited in capability.

4.6 Answer Generation

The function of the answer generator is to build and return a meaningful answer to the original question. To do this, it cannot rely entirely upon the information stored in the SSR. The communication link between the NL parser and answer generator is buried deep within the registers which form the long and short term memories. This does not make the system any less flexible, however, because the entire linguistic core still remains separate from the application (domain and database) interfaces. An important difference which this system displays from previous NL systems is that the information used in forming the answer comes from the parser and not from the structure passed to the database. By this method the SSR can remain simple while the system, as a whole, can still provide informative answers.

The first step of the answer generation mechanism is to extract the answer from the database. It does this by consulting the database interface. The answer generator passes the SSR to the database interface and receives a list of the appropriate answers (see Section 5.2). Information relating to

the type of question asked is extracted from the SSR and used to decypher the returned information. Next, the answer generator uses control information from the short term memory to limit or expand the answer to be returned to the user. Words that are to be included in the returned message are selected and the proper inflected form is determined. Finally, the answer is formed and returned to the user.

For example, after parsing the input:

Who serves chicken?

the NL parser will produce the SSR:

```
(WHFIND
   (*AND (NAME = ?)
         (FOOD = CHICKEN)))
```

which is then passed to the database interface (DBI). Returned by the DBI is the list of answers:

("WHITE SPOT" "STEER AND STEIN")

By using only the information returned by the DBI and that contained in the SSR, there exists only enough information to produce this same list of answers. However, by retrieving the verb from the short term memory (global registers), the system produces the response:

"WHITE SPOT" AND "STEER AND STEIN" SERVE CHICKEN.

4.7 Summary

The linguistic core forms the heart of the NL system. It has been designed to function as an independent unit with intermediate calls to the domain definition, database and possibly even the user to aid in processing the query. The core is made up of the NL parser, the answer generator and a communication path of registers between them. By using this structure, queries to the database are treated as simply one small step in the total process and the information structure passed to the database does not have to contain all of the information which the answer generator will eventually need. In addition, the parser can pose queries to the database interface whenever the need arises during the parse and not have to wait until the parse has been completed.

The NL parser processes a syntactic grammar to exploit the regularities of the English language while at the same time provides for intermediate calls to semantic verification and structure building routines to identify impossible interpretations early. In this way the general, domain independent portions of previous syntactically oriented systems can be captured without the drawback of generating large numbers of semantically unreasonable parses.

Definition of individual words in the linguistic section is primarily concerned with their morphological features. There

are few completely domain independent nouns and even fewer such verbs. Verbs are defined in a case frame structure to allow the greatest ease of both definition and use. The case frame definition for a verb is fairly straightforward, being simply a list of the cases which the verb takes and a list of possible fields which can fill each case. To take advantage of these case frame definitions, the NL parser attempts to fill the case slots with information extracted from the query.

The SSR definition is currently quite limited in scope; however, this is not a severe limitation at present as it still allows a reasonable variety of questions to be answered by the system. Further research should result in a more comprehensive representation.

Next we will look at the applications interface - the domain definition and the database interface.

Chapter 5

System Design: Part II - The Application Interfaces

(Domain Definition and Database Interface)

In Chapter 3 we saw that it would be beneficial to design a natural language question answering system so that the domain and database specific knowledge was distinctly separate from the linguistic core (Figure 5.1). In Chapter 4 we discussed the functions which were sufficiently domain and database independent to form a linguistic core. We will now turn our attention to the application interfaces - the domain definition and the database interface.

5.1 Domain Definition

The main thesis behind this work has been to attempt to remove as much domain specific information as possible from the system and isolate it in a "domain definition" (Figure 5.2). To make the changes to this definition simply and correctly, a declarative format has been used (see Appendix D for an annotated, sample domain definition). With this format it is hoped that any changes made to the domain will be reduced to the level of "slot filling" or "form filling". By removing the need for programming, the changes become understandable, even to a relative novice. The domain definition is broken up into three

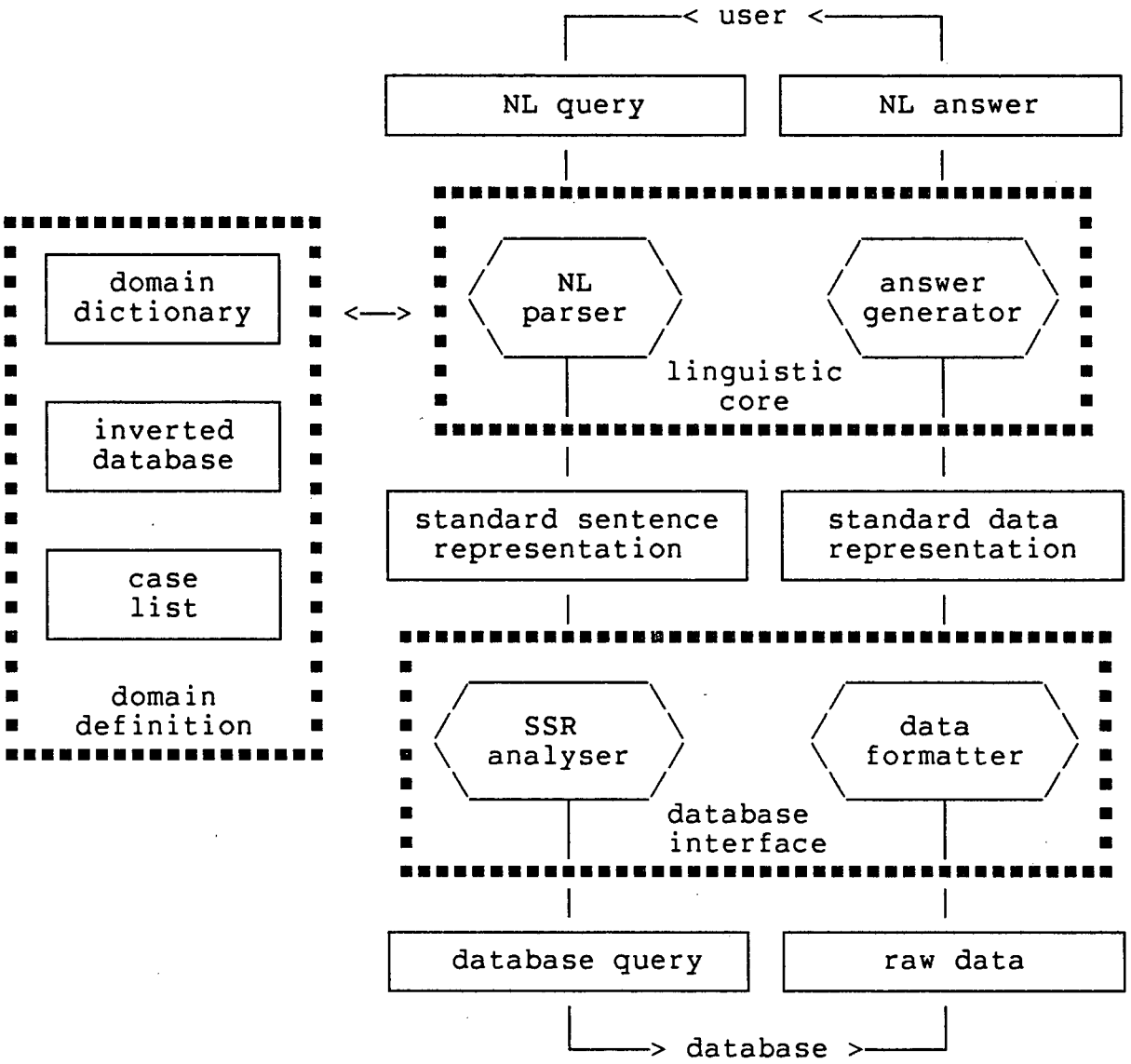


Figure 5.1: Proposed Natural Language System: A Review

logical sections. These are the domain dictionary, the case list, and the inverted index of the database.

It was initially hoped that the domain definition could have remained totally separate from both the linguistic core and the database interface. Keeping the definition separate from

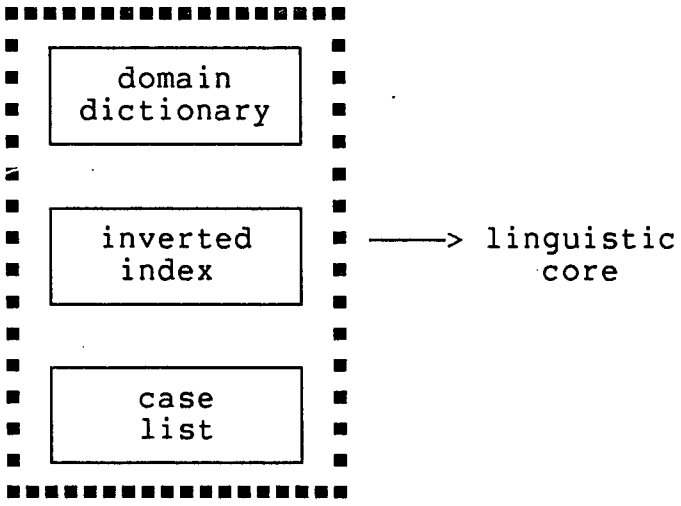


Figure 5.2: The Domain Definition Module

the linguistic core turned out to be a logical step because of its declarative structure as compared with the procedural structure of the linguistic core. However, it became obvious that any changes to the physical characteristics of the database could not help but be reflected, at least to some degree, in the domain definition. Consequently, separating the domain definition from the database interface became a more difficult task. What resulted was that the domain definition now contains a domain view of the database. This does not mean a definition of the entire database, but rather the parts of the database which will change when the domain changes. Such parts are the fields and field elements, but not the functions or data accessing methods.

5.1.1 Domain Dictionary

The domain dictionary contains the definitions of the actions, fields, jargon and even some field elements allowed within the particular domain. Much as in the syntactic dictionary, all individually defined terms in the domain dictionary must have morphological and syntactic information stored with them. Additionally, each category has associated with it some information which may change according to the domain. To simplify the definition of "meaning" of the domain specific terms, a case structure has been employed. The verbs of the system are defined over a range of cases and the nouns are placed into one of the case categories.

5.1.1.1 Actions

The verbs in the domain have an "action" definition which specifies a "conceptual pattern" to be interpreted by the grammar. Any relevent cases for the verb, usually at least agent (AG), patient (PA) and recipient (RE) cases, are defined by the fields which can fill them. For example:

```
(SERVE ACTION (AG NAME PA (FOOD MEALS)))
```

defines the verb "serve". The definition is taken from the restaurants database and means:

(a) that a restaurant can serve something

(b) that a type of food (e.g. Chinese food) or a type of meal (e.g. breakfast) can be served

5.1.1.2 The Fields

The fields in the domain describe the general category of things to look for. They specify where to look in the database but they do not supply special values of what to look for. For example, some of the fields in the restaurant domain are COST, FOOD, RESERVATIONS and ADDRESS.

The marker DBFIELD is used to identify the real, database name of a field. Since the name of the address field in the restaurant database is really LOC, it is defined in the domain dictionary as:

(ADDRESS DBFIELD LOC)

Another semantic marker (DBCAT) is used to specify the morphological properties of elements in the database. By specifying all entries in the COST field as adjectives by:

(COST DBCAT ADJ)

any of the database entries in that field can be inflected to the comparative or superlative. This "master field" method for

specifying all elements of a particular field has the benefit of allowing a simpler and smaller definition of the inverted database. It does, however, introduce some problems into the scanning and morphing routines. Usually all elements of a particular field would not have the same morphological features. Take for example the two elements of the FOOD field - "Chinese" and "chicken". Not only are the morphological entries for these words completely different, but the words do not even perform the same linguistic function. The word "Chinese", when referring to "Chinese food" is acting as an adjective while the word "chicken" is definitely a noun. A small change to the linguistic component, however, adds enough leniency that the system will now make allowances for these "master fields".

One optional marker which can be given to a field is one which designates ordering of a field. There are two general ordering types. The first is a simple numeric or lexical ordering and can be either ascending or descending. This has been used in the "date" field in the bibliography database and defined as:

```
(DATE ORDER *ASCENDING)
```

Subsequently, questions of "before" and "after" can be answered. As an example, assume that the dates in the database were B.C. dates. The only change which would have to be made to the domain definition would be:

```
(DATE ORDER *DESCENDING)
```

Questions involving both B.C. and A.D. dates in the same field (perhaps flagged by an entry in another field) have not been addressed here.

The other major type of ordering is not so easy to deal with. In the restaurants database the "cost" field is a finite-valued field containing only the values "expensive", "moderate" and "inexpensive". If the words could have been chosen differently then it could be left at a simple lexical ordering but this rarely occurs. In this field, the query:

Find a cheap Japanese restaurant.

would have no method of order reference. The word "cheap" would be defined to the system as:

(CHEAP INDF COST INDR *MORE INDE MODERATE)

but without some ordering on the field itself, this ordering would be of little use. If either *ASCENDING or *DESCENDING order were used then surely the system would return a faulty answer. For finite-valued fields (currently there is no way to handle infinite-valued fields), the definition would be:

(COST ORDER (INEXPENSIVE MODERATE EXPENSIVE))

Another optional marker defines the range of values allowed in a particular field. Simply because a word does not presently appear in a certain field in a database does not usually mean

that the word can never appear there. This is where the distinction between INFINITE-VALUED and FINITE-VALUED fields comes in to play. There are fields which allow only a limited number of different values to be present. Such a field is the STARS field in the restaurant database where the only possible values are 0, 1, 2, 3, 4 and 5. The main benefit in making this distinction is that when the inverted database handler has looked for a value and cannot find it, if processing a FINITE-VALUED field it can return immediately to the parser without invoking a futile database search.

5.1.1.3 Terms and Jargon

Many of the domain specific terms and jargon will indicate a specific field. Again from the restaurant domain, the noun "menu" would provide a reference to the FOOD field. There are usually many nouns which indicate the same field. For example, "cost", "expensive", "cheap" and "price" could all indicate a processing of the COST field. The field indicators are usually proper nouns, common nouns or adjectives. In addition to the necessary morphological definition of all words, the domain definition of these particular nouns and adjectives has three extra components (if relevent):

INDF - indicated field

INDR - relation between field and field element

INDE - indicated field element

For example, the definition:

(CHEAP INDF COST INDR *LESS INDE MODERATE)

means that any record with an entry in the COST field less than "moderate" will be considered to be "cheap".

An important feature that the system needs is the power to recognize non-database elements as database elements. There are many times when words which a user may use as jargon may not actually be in the database and therefore not in the index. However, to make the system usable, it must be able to identify these terms for what they are. The abbreviation mechanism is used to handle this problem. An example would be if we wanted to use MIT as an abbreviation for "Massachusetts Institute of Technology". The definition allowing this would be included in the domain dictionary or the inverted database simply as:

(MIT ABBREV "MASSACHUSETTS INSTITUTE OF TECHNOLOGY")

5.1.1.4 The Field Elements

Most field elements are defined simply by their presence in the inverted database. However, some provide more information to the query processing than simply a reference to their name and, therefore, would be found in the domain definition itself. For example, in the above definition of the adjective "cheap", a "cheap restaurant" would mean more than just a restaurant in the

database with the value "cheap" in the COST field. Actually it would mean any restaurant with a value in the cost field less than "moderate".

5.1.2 The Case List

The case list is used for internal manipulation of the case structures which form the basis of the internal semantic representation of the query. All prepositions and many of the general, domain independent adverbs are defined in terms of the case list. For example, "at" has been defined as relating to the "time" and "location" cases. The question adverbs "when" and "where" are also defined in terms of these cases and so, to make these adverbs functional in a new domain, only the definition of the "time" and "location" cases must be provided.

The definition of the case list simply requires the designation of which database fields fall into which case category. A list of possible cases has been provided to help guide the domain implementor when defining the case list but it is in no way meant to be exhaustive. The possibility exists for the domain implementor to add to the case list; however, since the prepositions have been defined in terms of this particular list, any changes to it would have to be reflected in the syntactic dictionary. The case list provided has been modified only slightly from the case list found in Taylor and Rosenberg (1975). The complete list of defined cases can be found in

Appendix B.

5.1.3 Inverted Database

As in the ROBOT system (Harris 1977a), the inverted index of the database is used to "define" all of the real world knowledge of the system. This is the set of terms found in the database itself. The use of an inverted database in this system is not absolutely necessary. The gains made by incorporating it into the design of the system are in query processing time. With the index, the system does not have to examine the database for the "meaning" of every database element.

There are times when it is useful to make a quick check in the database to see if an element is present. If there is an up to date inverted index it should only be necessary to search the index but, more often than not, if there is an index at all, it is probably out of date. In any large database system, updates to the database are made continually while updates to the inverted index would be done rarely.

There are other times when it would not help to search the database. If there is a precisely defined field such as COLOUR with "red", "green" or "blue" entries only, then no amount of database updates will change the fact that red, green and blue are the only colours allowed. Here we don't want to search the database (see Section 5.1.1.2). Another reason for not querying

the database, but rather querying the user instead, is if the database system is slow in responding. This has probably been the assumption made by most NL system designers until recently as they usually try to make only one call to the database. Actually, if the database system is reasonably fast, the natural language parser can retrieve an enormous amount of information from it through intermediate calls. Still another situation when the parser might not want to search the database is when there is no information to indicate the field to search. Clearly it would be ridiculous to search every field of the database to find the element.

The ideal situation would be if the database itself could be used at the base level of the inverted index. Indeed some database languages may provide this facility, but the system used here provides no such link. If the database language will not provide an index, it must be built by the database implementor. Fortunately, this is a task which can be readily automated. Sometimes, however, building an inverted database requires more space than the system is allowed to use. In this case, we must fall back on the database search method.

The decision here becomes a classic one of space versus time and is usually based on machine limits. Since the machine underlying this particular system has few space problems, time was seen to be the crucial quantity.

Since the construction of an inverted index for a database is both a time consuming and menial process, a program was designed to generate an inverted index automatically. The program was written in the MTS Edit Procedure sublanguage (Hogg 1980) and is designed to take SPIRES database output and create a LISP dictionary with entries of the general form:

(element ELEMENT-OF field)

A sample inverted database can be found in Appendix D.

In addition to the dictionary of actual database elements, the inverted index requires the power to identify different elements in the database which are synonymous. Frequently the database will contain synonyms and it is only through identification of these synonyms that meaningful answers to queries can be produced. Take, for example, the case of the three database elements "burgers", "hamburgers" and "cheeseburgers". If the query was:

Who serves burgers?

all places with "FOOD = BURGER" as well as all places with "FOOD = HAMBURGER" and "FOOD = CHEESEBURGER" would be expected to be found. To handle this feature, a new semantic marker was created. It is simple to use, requiring a list of all synonyms found in the database, but must be entered by the domain implementor. The format to define the above case would be:

(BURGER FOOD+ (HAMBURGER CHEESEBURGER))

5.2 The Database Interface

The database interface (Figure 5.3) is designed to provide an idealized database to which the system can pose questions. Not only will there be the one query to find the data to answer the user query, but also possibly many intermediate queries to find information needed to continue processing at any time. All of the information relevant to a particular database query language must somehow be incorporated. Its purpose is to hide the actual physical characteristics of the particular database from the linguistic core. The database interface is composed of two completely separate sections - the database format routines, which handle input to the database and the data format routines which handle the database output. In changing the underlying database these routines would have to be rewritten but the rest of the system should not have to be modified.

The philosophy behind the entire database interface is simplicity. Since it is not clear which functions any database query language may or may not provide, assumptions have been kept to the bare minimum. In this way it should be simple to adapt this system to any and all database systems. The ideal method of communication with the database, for both the database

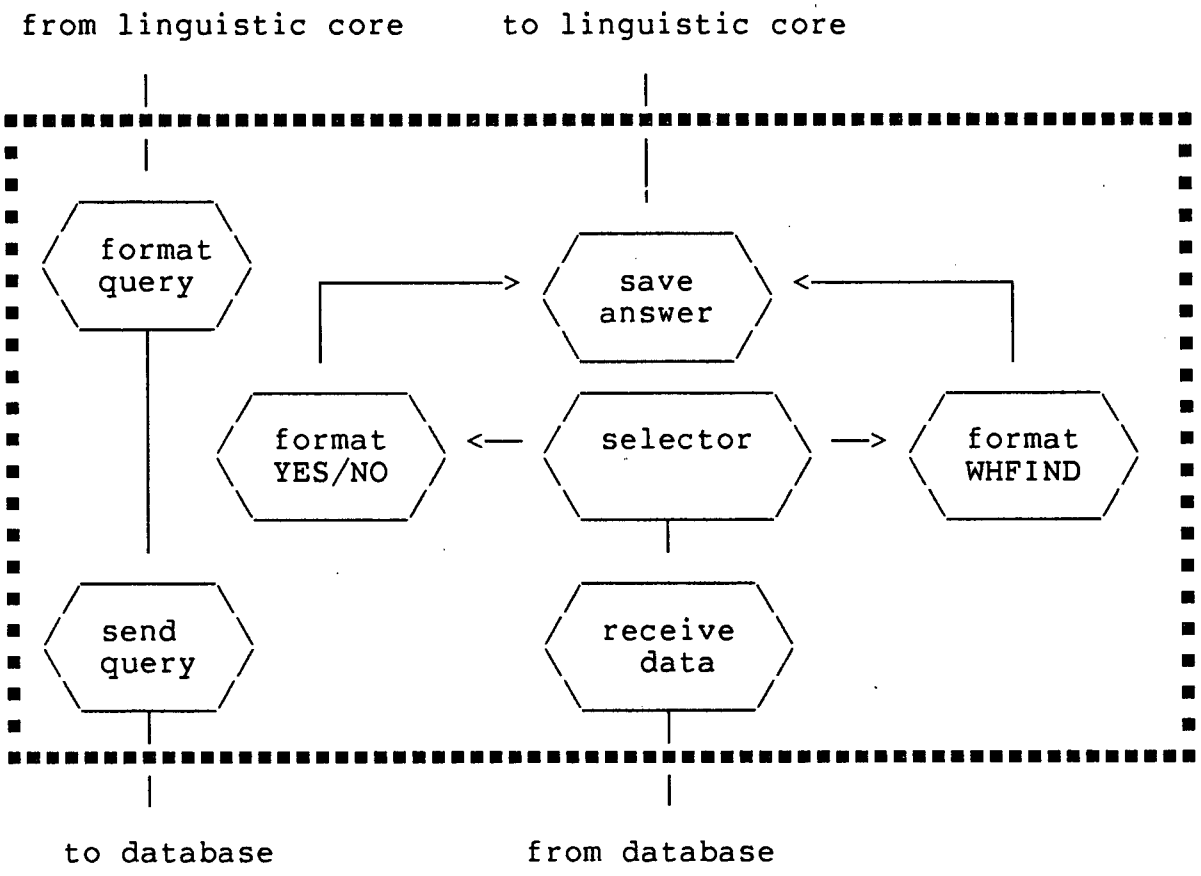


Figure 5.3: The Database Interface Module

format and data format routines, is to pass messages directly through low level function calls. Unfortunately the database system linked to in this system allows no such communication. Because of this, a rather roundabout route has to be taken. Two separate tasks have to be initiated, one running the NL interface and one running the database management system (DBMS). The two tasks communicate through a shared file with the database format routines generating "user" queries and the data format routines interpretting the DBMS responses. This method is extremely awkward and poses more problems than should

normally be expected but queries can still be handled in a reasonably short time.

5.2.1 Database Format Routines

The database format routines transform the standard sentence representation (SSR) into a query in the data base query language. As mentioned earlier, there are two obvious methods of approaching this problem. One is to communicate directly with the database through the low level function calls but, in this particular system, these functions were not available and the database format routines had to generate a "user" query to the database.

If a particular function is called for in the SSR, then the routines should find and call the appropriate database routine. Also among the responsibilities of these routines is the "faking" of any functions which the database should provide but doesn't. A typical example would be if the database were expected to provide a list of currently searchable fields upon request. Since this is a common function of many databases, it would not be an unwarranted expectation and if the database language we are communicating with does not provide this function, then these routines must.

Currently there are only three low level functions which the database language is expected to handle. These could be

expanded but it should be remembered that if an ideal interface is to be provided, one to which virtually any database language could adapt, then they should include only the very common functions. The three low level functions which would have to be implemented before a new database could be attached are:

DB-SELECT - which selects the appropriate database

DB-EXIST? - which returns the number of elements satisfying a certain query.

DB-FIND - which returns the elements specified by the constraints.

The database format routines for this system were written in LISP.

5.2.2 Data Format Routines

These routines work on the output of the database interface. Their function is to take the output data from the database as returned by the query language or low level database functions and return to the Q/A system the portion of the answer it requires. The standard format that this system expects is a list of the elements found. The basic structure is:

(FIELD = ELEMENT)

If more than one piece of information is to be returned, it will be returned as a list of lists:

```
((FIELD1 = ELEMENT1)
 (FIELD2 = ELEMENT2)
 (FIELD3 = ELEMENT3))
```

The data format routines in this system have been implemented partially in LISP, but mostly in the SPIRES Protocol sublanguage (Buckland 1981).

5.3 Summary

The database interface has been kept as small as possible. There have been no complex functions such as the identification and handling of metaquestions (see Section 3.3.1) included in it. Through this simple interface it should be straightforward to attach a new database to the NL system; however, as the questions from the NL system become more involved, the database interface will undoubtedly have to become more complex itself.

The domain definition contains three separate components: the domain dictionary, the case list and the inverted index for the database. Together they attempt to provide an information bank which the NL parser can query to retrieve domain dependent information. All parts of the domain definition have been structured in a declarative format to facilitate quick and easy modification.

To determine whether or not the domain definition process was both sufficient and simple, the NL system was transferred to a new domain of discourse. The next chapter provides a discussion of this process.

Chapter 6

A Change of Domains

In order to determine whether or not the natural language database interface created in Chapters 4 and 5 was indeed domain independent, a test was performed. The test was to adapt the interface to a new domain of discourse. After developing the interface to interact adequately in the initial restaurant domain, it was adapted to an AI bibliography domain. Then, after revisions based on the results of the domain change, the interface was transferred to a conference domain (see Appendix E for a sample session).

6.1 The Definition Process: A Guide to the Perplexed

The definition process is made up of a few tasks which must all be performed by someone familiar with the domain and database system being used. (e.g. the database administrator (DBA)). It would be helpful if this person had some knowledge of the NL system but hopefully it has been designed in such a way that this is not really necessary. The tasks to be performed are:

1) construct an inverted database (if none exists already)
2) define the database fields to be used
3) define the actions which will be allowed
4) define any abbreviations, synonyms and jargon to be used

Appendix D contains a sample domain definition.

6.1.1 Constructing an Inverted Database

Sometimes a database system will provide fast access to an inverted index of the database. More often it will be slow or non-existent. In these cases it is beneficial to build one external to the NL and database systems. As discussed in Chapter 5, the DBMS Spires to which the interface was attached contained no hooks for an inverted database. However, since the information required in the inverted database is only the field values and the name of the field(s) in which it is located, the construction was easy to automate. Building the inverted database for all of the domains tested was done automatically on the DBMS output by a procedure written in the MTS Edit Procedure sublanguage.

6.1.2 Database Field Definitions

All searchable and non-searchable fields in the database must be defined to the NL parser by the system administrator. This definition process is currently very simple as few general features have been implemented. However, the definition should be able to be extended when any new feature is desired. The field definitions inform the system what the properties of the particular field are; both mandatory and optional properties must be defined. Currently there are two field definitions which are mandatory (DBFIELD and DBCAT) and one optional property (ORDER) designating the ordering of the field (see Section 5.1.1.2).

6.1.3 Action Definitions

The actions are defined in a case frame structure as was discussed in Section 5.1.1.1. Deciding which actions need to be defined was done, for each domain, by generating a list of sample questions and then extracting from this list the domain specific verbs.

The cases to define for each action were taken from the case list provided - a copy of which can be found in Appendix B.

6.1.4 Abbreviations, Synonyms and Jargon

The definition of domain specific terms and jargon can be found in Section 5.1.1.3. Many of the terms defined were actually an extension of the inverted database. In some cases, common abbreviations (such as "UBC" for "University of British Columbia") may not be found in the database. To facilitate the use of these abbreviations, they must be added either to the domain dictionary or to the inverted database.

Sometimes non-standard abbreviations are used by the trade (even if a standard exists). Whereas "Comm. ACM" is the standard abbreviation for "Communications of the Association for Computing Machinery", "CACM" is also widely used. By making both of them abbreviations to the NL system, the user does not have to remember which is the standard.

6.2 The Restaurant Domain

The initial database around which the demonstration system was built holds information concerning restaurants. It is the type of database which might soon be found on a television information network (e.g. Telidon). Included here are data concerning the local eating establishments; the types of dishes they serve, their location, hours of operation, quality of food and relative prices. Both searchable and non-searchable fields

are included.

The restaurant database used here was developed by the UBC Computing Centre to demonstrate the SPIRES DBMS. During demonstrations, new users are encouraged to add data to the SPIRES subfile and so the resulting database is a little unreliable and inconsistent in naming conventions.

The fields of the restaurant subfile used are shown in Figure 6.1.

Fieldname	Description	Searchable
name	restaurant name	yes
location	address	yes
phone	phone number	no
cost	approx. cost of a meal for 2	yes
food	types of food served	yes
stars	quality of the restaurant	yes
meals	when is it open	yes
comments	anything else	no

Figure 6.1: The Fields in the Restaurants Database

Some example queries which prospective diners might have for such a system are:

What are some Italian restaurants?

Can you find me a cheap Japanese place?

Which are the best restaurants?

What is on the menu at White Spot?

How many Chinese food places are there?

When does the Yangtze open?

Is there a Turkish place which is open for lunch?

What is the White Spot on Granville's phone number?

Since this was the initial domain attached to the NL system, its design was tailored towards answering these questions.

6.3 Adaptation to the Bibliography Domain

After the system was able to function adequately in the restaurant domain system, it was time to turn to another. An Artificial Intelligence (AI) bibliography database was chosen. The vocabulary of this new domain was dissimilar enough to cause a potential portability problem even though the structure of the questions remained similar. Some of the fields involved in this database were the author, book, subject, publisher, date and abstract, again including both searchable and non-searchable fields.

The AI bibliography database has been developed by the UBC Department of Computer Science primarily as a research aid. The

additions to this database are made in a more uniform and controlled method than the restaurants database and it therefore presented a more reliable information base.

The actual fields in this database used are shown in Figure 6.2.

Fieldname	Description	Searchable
author	author	yes
title	title of the work	yes
date	date it was written	yes
type	what type of article	yes
abstract	abstract of the article	no
location	where the book is physically	yes
keywords	associated topics	yes
pub	publisher of the book	yes
inst	what institution put it out	yes

Figure 6.2: The Fields in the Bibliography Database

Some of the questions which were put to this system are:

Who wrote Aspects?

How many papers has Schank written?

How many vision books were written before 1978?

Find at least 4 papers by Minsky.

The major difference between the databases came in the area of vocabulary. There seemed to be no general way to define words and special terms so that they could apply to all databases simultaneously. For example, whereas in the bibliography database the mention of the word "name" brings about conflicts between the publisher, author and book title, in the restaurant database, the word is virtually unambiguous (signifying the restaurant name). Conversely, the question:

Where is Schank and Colby's book?

to the bibliography database involves no ambiguity (there was only one "location" field) whereas the query:

Where is a good steak place?

to the restaurant database does because it could refer to the name of the restaurant or the address. Other words play major roles in one database (such as "serve" in restaurants, "write" in bibliography and "register" in conference) but never appear in the others.

The database elements were defined by inverting the database. Although fully automated, the process did require a substantial amount of CPU time and disk space due to the size of the database. Additions to the database in terms of abbreviations and synonyms were made by a manual pass over the inverted database in an editor.

Definition of the fields, both in the domain dictionary and in the case list, was fairly straightforward and quick since the database had under 10 fields to define.

Next some sample sentences were generated to determine the domain specific actions and jargon to define. This was the most time consuming process since there was no formal procedure to follow. The actual definition of these domain specific terms (again using the pre-defined case list) was relatively quick.

One shortcoming which was uncovered during the domain changing exercise was the omission of an important universal feature from the initial version. Since there was no numeric ordering of elements in the restaurant system, it had no way to handle questions relating to "before" and "after". There was no possibility to simply add to the current definition; the parser itself had to be modified. The problem was solved by allowing the values *ASCENDING and *DESCENDING to appear on the marker ORDER. This turned out to be a simple extension to the field definition to define it and only a slight modification to the linguistic core to handle it. Problems caused by oversight are bound to happen in any system and no system will ever actually be complete; however, this oversight was due more to a severly limited testing stage of the restaurant system than to the design of the NL parser as a whole.

6.4 The Conference Domain

The third database hooked up to the natural language parser was a conference registration database. It was originally designed by the Computer Science Department at UBC to contain information on the participants at the 7th International Joint Conference on Artificial Intelligence (IJCAI) held at UBC in 1981.

The actual fields used in this database used are shown in Figure 6.3.

Fieldname	Description	Searchable
name	name of the participant	yes
institute	institution the person came from	yes
a-time	arrival time	yes
o-country	country the person came from	yes
type	how the person registered	yes

Figure 6.3: The Fields in the Conference Database

Some of the questions which were handled by this system are:

Who is coming from SRI?

Has John McCarthy registered?

Find all the people who have registered as an early-student.

When did Minsky register?

How many people are coming from MIT?

When is Schank coming?

As in the definition of the AI Bibliography domain, the creation of the inverted database was done automatically. Again a set of questions were generated in order to extract the domain dependent actions and jargon. Because there were no new concepts to handle for this domain, the entire definition process was completed within a few hours.

6.5 Summary

After the linguistic core had been brought up to a level of competence where it could handle the simple questions posed, the definition of a new domain became a straightforward and quick process. However, the different domains used in this test were all residing under the same database system and this undoubtedly played a role in limiting the structure of questions which could be asked or answered.

Chapter 7

Conclusions

The achievements of this system lie in the ease with which it can be adapted to a new domain of discourse. The structure of the domain dependent information allows a great deal of question answering capability to be defined easily and quickly. Of course, there remain issues of adequacy and extendability which have not been dealt with satisfactorily. It has never been expected that the methods and structures developed here could be transferred, as is, to a more complex world of general discourse; however, in the more limited question answering paradigm they do appear to be reasonably acceptable. The strategy of separating the knowledge base from the linguistic component does seem useful enough to be a necessary feature in many domains of discourse. With techniques such as these it should be possible to develop large natural language database interfaces which are general enough that the domain of discourse can be altered without requiring significant modifications of the entire system.

During the development of this particular system there have been a number of issues raised which, for some reason or another, could not be adequately addressed in the current context. Frequently these problems were set aside because of

time constraints but others were just beyond the scope of this thesis. Next we will briefly consider some of these issues.

7.1 Open Issues

Some of the issues which have not been resolved in this system are the handling of text, value judgements, multi-field answers, complex conjunctions, pronoun reference, clarification dialogue and sample sentence generation.

7.1.1 Text Retrieval by Content

The whole subject of retrieving text by content is much too difficult for the current system. This became an issue in the restaurant domain while attempting to process the "comments" field and again in the AI bibliography domain when processing the field "abstract". However, this is a problem which has not yet been adequately addressed by researchers in general. There are few, if any, current systems which can properly process text.

7.1.2 Value Judgements

An added benefit of a natural language database query system would be its ability to make some types of value judgements. An example of what is meant here is the following:

Which is the best restaurant in town?

Of course, methods of answering this will be different in each database system. In some, the following steps might have to be performed:

1. Select STARS
2. Sort into descending sequence
3. Return the first record

while in another it might be done more simply. In this example, our semantic (retrieval) component must be able to handle multi-level commands to the database and this adds complexity.

In this system, "best" has been defined as any entry with the highest number of stars. The structure passed to the retrieval component will be

```
(FIND (NAME = ?) (STARS >= *ANY))
```

Currently, while this structure can be defined and processed by the linguistic core, it can not be handled by the database interface.

7.1.3 Multi-Field Answers

In some databases an answer may involve entries in more than one field. An example of this might be found in a telephone directory system. Assume that the area code was not

explicitly stored in the database but could be determined by the province and city fields together. A query such as:

What is John Smith's phone number?

would have to do some reasonably complex calculations to determine the answer.

Another type of multi-field answer would arise when the values of one field depended upon the values of another. This might happen in a accounting database where one field is an absolute amount and another is a code signifying a debit or a credit.

7.1.4 Complex Conjunctions

The processing of simple conjunctions was discussed in Section 4.1.4.6. When many conjunctions are strung together it becomes difficult to give any general rules to process them. For example, in:

Have you seen a dog and a bone or a cat?

the tendency is to join "the dog" and "the bone", while in:

Have you seen a lady and a boy or a girl?

the grouping is not quite as obvious. Humans use both context and semantics to decide the grouping and we cannot expect a program to handle these types of conjuncted phrases until it can

deal competently with these concepts.

7.1.5 Pronoun Reference

Only simple pronoun reference has been dealt with. This was only partially because of time constraints. The question of complete pronoun reference (at the human level) is far beyond the ability of most current systems. Fortunately the design of the syntactic-semantic interface (general registers) allows for a great deal of flexibility.

7.1.6 Clarification Dialogue

When the system fails in some part of its processing it can either give up or enter into a clarification dialogue with the user. This problem has been addressed superficially in Chapters 3 and 4 however it is an important issue which must be explored more fully (Codd et al 1978).

7.1.7 Sample Sentence Generation

The problem here is how to find out which words should be defined in a new domain and it is a problem which has been glossed over during the development of, not only this system, but also of most previous systems. The problem is not a totally trivial and unimportant one if we are to adapt a system to a new domain quickly. In the domain change undergone to test this

system, sample sentence generation was one of the more time consuming portions of the process. In a real world application, professionals in the field would be called upon to generate the sample sentences and then the system would be adjusted to handle these particular questions.

7.2 Problems for Future Work

There are many problems on which more work must be done. Some of these are extensions to both the syntactic and semantic portions of the system, adaptation of the system to a new database system, and computational optimization.

7.2.1 Extensions to the Syntactic Component

The syntactic component of this system has been left incomplete, for obvious reasons. Many additional features of natural language should be able to be implemented as part of the current syntactic grammar. Expansion of the syntactic structure building can be done with little or no modification to the parser.

7.2.2 Extensions to the Semantic Component

Some of the open issues discussed previously could probably be resolved with an extension to the semantic component. The

internal register structure allows for a great deal of information to be stored and retrieved at any point of the parse. Because of this, a new feature can be added or modified without affecting the entire system.

7.2.3 Adaptation to a New Database System

Although discussed briefly in Chapter 5 and although hooks for this have been implemented, a change of database systems was never implemented. This stemmed from the fact that there were no other database systems available on the MTS system at UBC. However, we expect that it should be relatively easy to carry out such an implementation. The major advantage of the design of this system with respect to a database system change are in the modularity of the system as a whole and of the database interface in particular. For example, to define a new database interface would require the coding of only 3 functions.

7.2.4 Computational Optimization

This system has been built with little regard for either time or space efficiency - not surprising in an experimental system. Consequently there are many areas of the program which could be optimized.

For example, the use of an inverted index reduces the amount of CPU time required. It does this by cutting down on

the database searches (which are costly) but increases the space requirements if not implemented as part of the original database.

7.3 Summary

The NL system developed has been split into 3 separate parts. The linguistic core contains all of the domain independent components seen in recent natural language question answering systems. It parses queries, consults the database interface for the data and formulates the appropriate reponse.

The domain definition is a collection of all of the domain dependent terms and database values. It has been designed in such a way as to facilitate definition and modification. The linguistic core consults the domain definition during a parse to retrieve the domain dependent information it needs to process the query.

The database interface provides an idealized, well-defined interface to the real database. Because of the simplicity of the functions required in this interface, it should be able to be rewritten for a new database with a minimum of effort.

Bibliography

Ball, Eugene and Hayes, Phil (1980), Representation of Task-Specific Knowledge in a Gracefully Interacting User Interface, Technical Report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, P.A.

Bobrow, Robert J., and Webber, Bonnie L. (1980), PSI-KLONE: Parsing and Semantic Interpretation in the BBN Natural Language Understanding System, Proceedings 3rd National CSCSI/SCEIO Conference, Victoria, B.C., pp. 131-142.

Brown, J. S., Burton, R. R. and Bell, A. G. (1974), SOPHIE: A Sophisticated Instructional Environment for Teaching Electronic Troubleshooting, Bolt Beranek and Newman, Report No. 2790, Cambridge, Mass.

Buckland, Tony (1981), An Introduction to SPIRES, University of British Columbia Computing Centre, Vancouver, B.C.

Celce-Murcia, M. (1979), Paradigms for Sentence Recognition, System Development Corp., Final Report No. HRT-15092/7907.

Chomsky, Noam (1965), Aspects of the Theory of Syntax, MIT Press, Cambridge Mass.

Codd, E. F., Arnold, R. S., Cadiou, J-M., Chang, C. L., and Roussopoulos, N. (1978), RENDEZVOUS Version 1: An Experimental English-Language Query Formulation System for Casual Users of Relational Databases IBM Technical Report RJ2144, IBM Research Laboratory, San Jose, Ca.

Fillmore, C. J. (1968), The case for case. Universals in Linguistic Theory, N.Y., Holt, Rinehart and Winston, pp. 1-90.

Harris, Larry R. (1977a), ROBOT: A High Performance Language Interface for Database Query, Technical Report TR77-1, Mathematics Department, Dartmouth College, N.H.

Harris, Larry R. (1977b), Natural Language Data Base Query: Using the data base itself as the definition of world knowledge and as an extension of the dictionary, Technical Report TR77-2, Mathematics Department, Dartmouth College,

N.H.

Hayes, Phil, and Reddy, Raj (1979), An Anatomy of Graceful Interaction in Spoken and Written Man-Machine Communication, Technical Report, Computer Science Department, Carnegie-Mellon University, Pittsburgh, P.A.

Hendrix, G. G. (1977), Human engineering for applied natural language processing, Fifth Int. Jt. Conf. on Artificial Intelligence, MIT., pp. 183-191.

Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D., and Slocum, J. (1978), Developing a natural language interface to complex data, ACM Transactions on Database Systems, 3(2), pp. 105-147.

Hirst, Graeme (1979), Anaphora in natural language understanding: A survey, Technical Report 79-2, Department of Computer Science, University of British Columbia.

Hogg, John (1980), UBC Edit: The Line File Editor, University of British Columbia Computing Centre, Vancouver, B.C.

Johnson, Jan (1981), Intellect on Demand, Datamation, 27(12), pp. 73-78.

Johnson, S. C., and Ritchie, D. M. (1978), Portability of C Programs and the UNIX System, The Bell System Technical Journal, 57(6), pp. 2021-2048.

Katz, J. J., and Postal, P. (1964), An Integrated Theory of Linguistic Description, MIT Press, Cambridge Mass.

Marcus, Mitchell P. (1979), A Theory of Syntactic Recognition for Natural Language, Artificial Intelligence: An MIT Perspective, Vol 1, eds. P. H. Winston and R. H. Brown MIT Press, Cambridge Mass.

Minsky, M. (1975), A Framework for Representing Knowledge, The Psychology of Computer Vision, ed. P. H. Winston, Mcgraw Hill, pp. 211-277.

Rosenberg, Richard S. (1980), Approaching Discourse Computationally: A Review, Representation and Processing of Natural Language, Carl Hanser Verlag, pp. 10-83.

Reiter, Ray (1978), The Woods Augmented Transition Network Parser, Technical Note 78-3, Department of Computer Science, University of British Columbia.

Richards, M. (1969), BCPL: A Tool For Compiler Writing and System Programming, Proc. Spring Joint Computer Conf., pp. 557-566.

Sacerdoti, E. D. (1977), Language access to distributed data with error recovery, Fifth Int. Jt. Conf. on Artificial Intelligence, MIT., pp. 196-202.

Schank, Roger C. (1972), Conceptual Dependency: A Theory of Natural Language Understanding, Cognitive Psychology, 3(4), pp. 552-631.

Schank, Roger C. (1973), Conceptualizations Underlying Natural Language, Computer Models of Thought and Language, eds. R. C. Schank and K. M. Colby, San Francisco, Freeman and Co.

Taylor, B. H. and Rosenberg, R. S. (1975), A case-driven parser for natural language, American Journal for Computational Linguistics, AJCL Microfiche 31.

Waltz, D. L., Finin, T., Green, F., Conrad, F., Goodman, B., and Hadden, G. (1976), The PLANES system: natural language access to a large data base, Technical Report T-34, Coordinated Science Lab., University of Illinois, Urbana.

Waltz, David L. (1978), An English Language Question Answering System for a Large Relational Database, Comm. ACM, 21(7), pp. 526-539.

Woods, W. A. (1967), Semantics for a Question Answering System, Ph.D. thesis., Report NSF-19, Aiken Computational Lab., Harvard University, Cambridge, Mass.

Woods, W. A. (1970), Transition Network Grammars for Natural Language Analysis, Comm. ACM, 13(10), pp. 591-606.

Woods, W. A., Kaplan, R. M. and Nash-Webber, B. (1972), The Lunar Sciences Natural Language Information System: Final Report, Bolt Beranek and Newman, Report No. 2378, Cambridge, Mass.

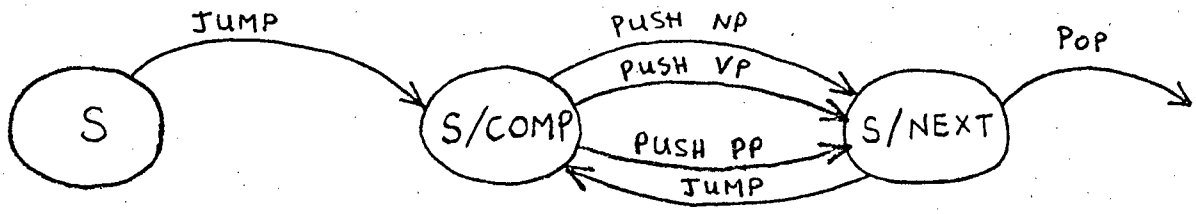
Woods, W. A. (1980), Cascaded ATN Grammars, American Journal for Computational Linguistics, 6(1), pp. 1-12.

Appendix A

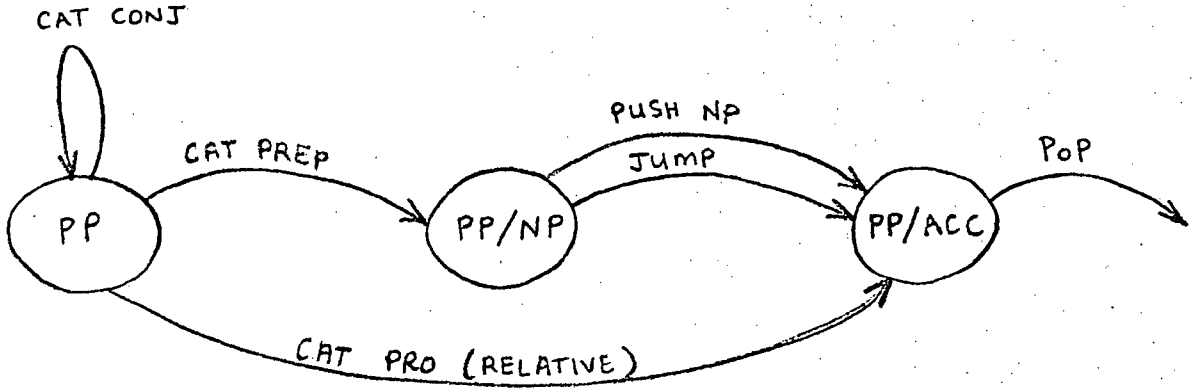
Transition Network Grammar

This appendix consists of transition network diagrams for the grammar described in Section 4.1.2.

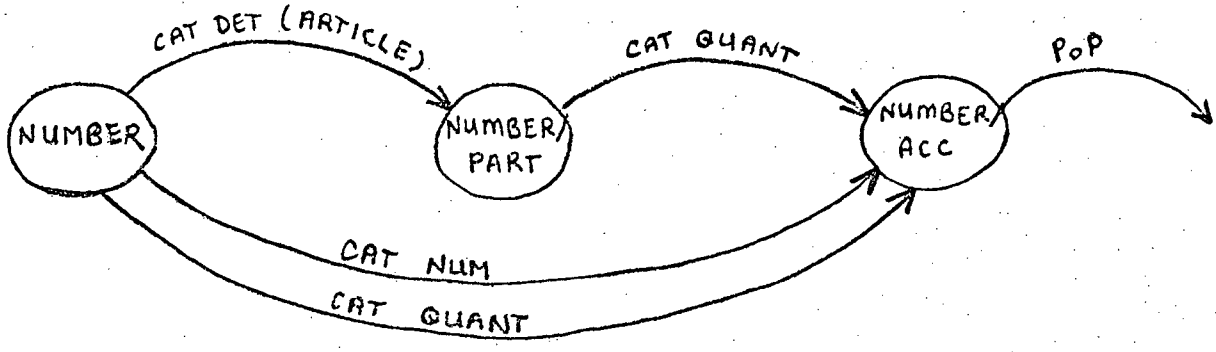
SENTENCE



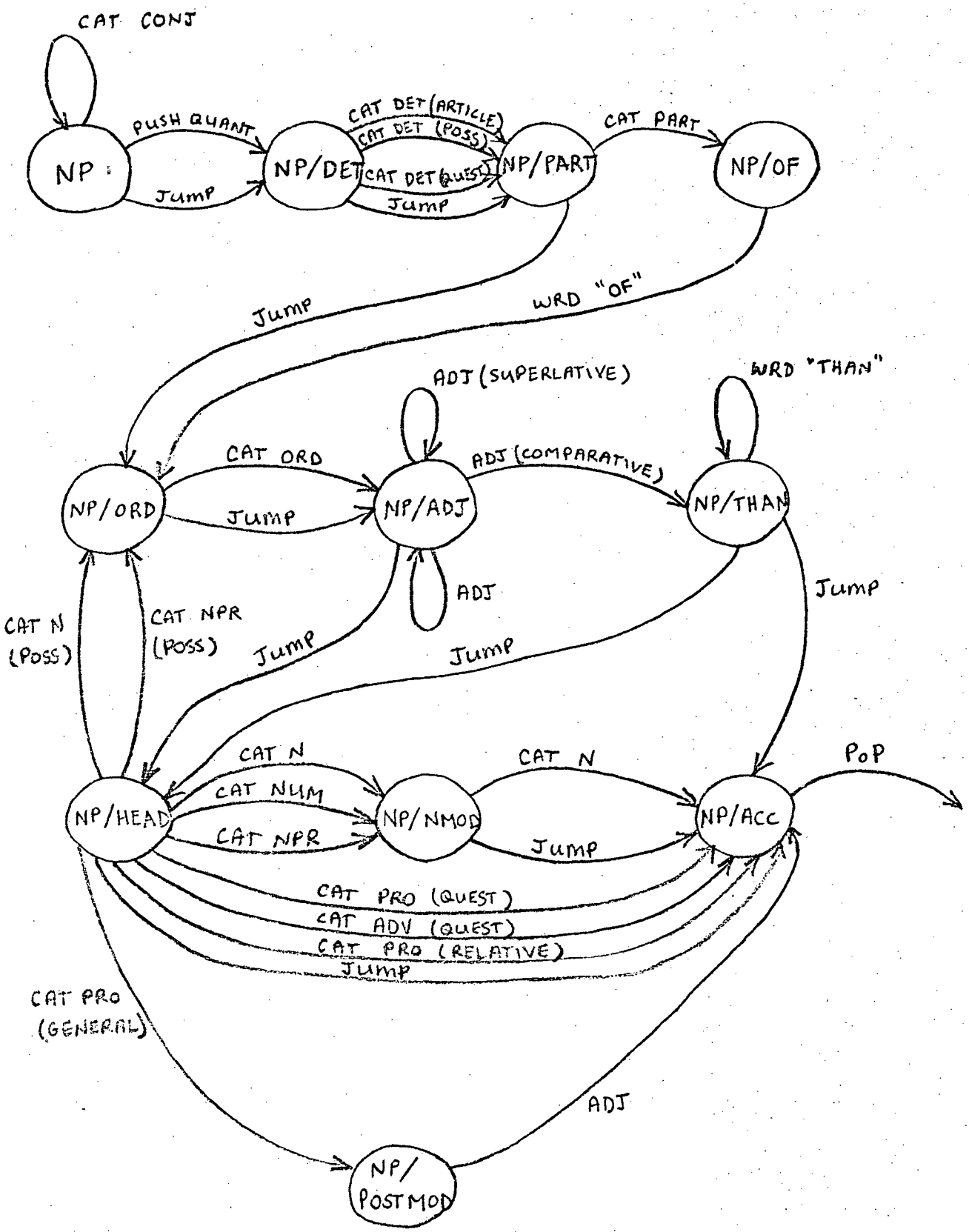
PREPOSITIONAL PHRASE



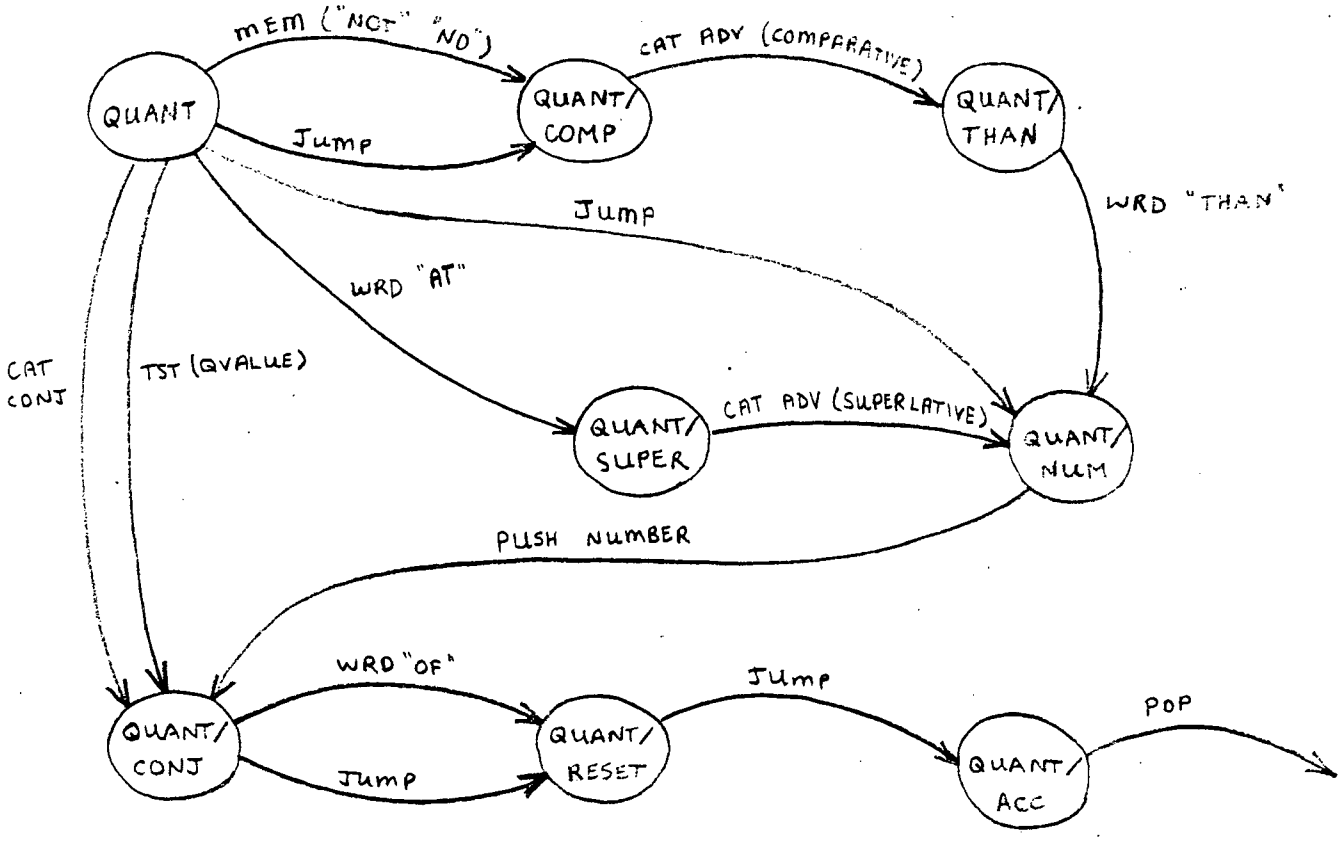
NUMBER



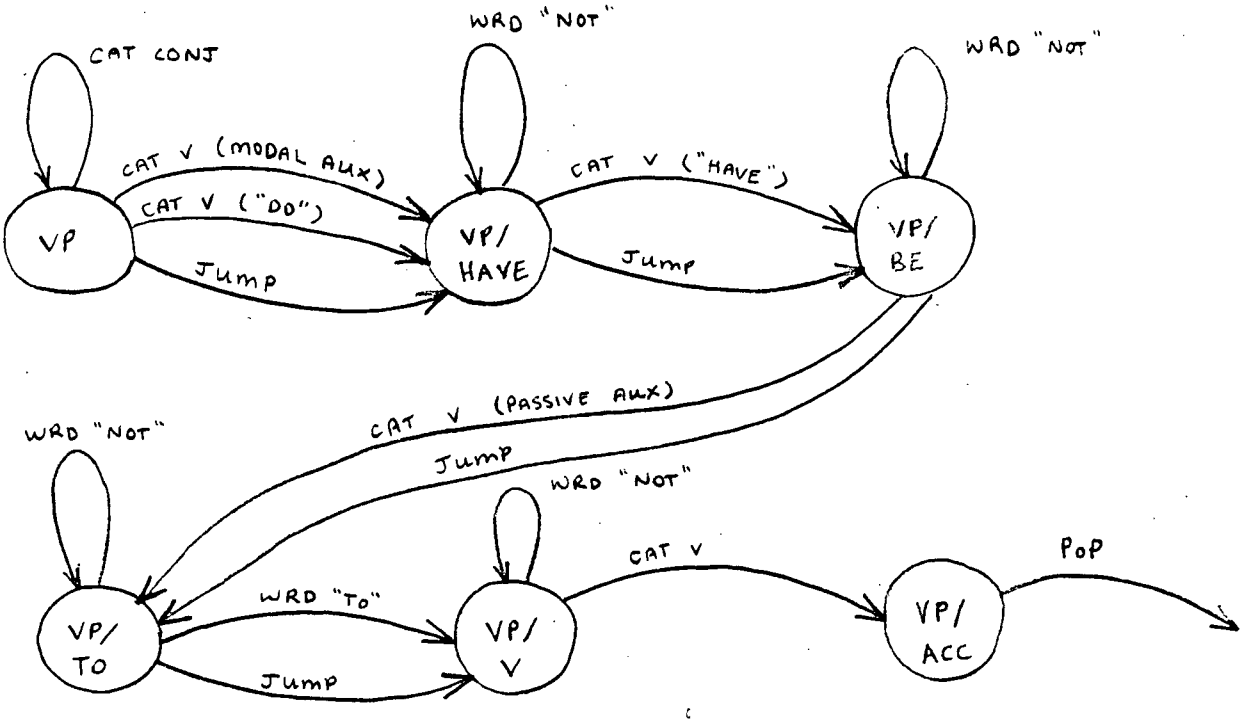
NOUN PHRASE


CAT CONJ
NP
PUSH QUANT
Jump
NP/DET
CAT DET (ARTICLE)
CAT DET (POSS)
CAT DET (QUEST)
Jump
NP/PART
CAT PART
NP/OF
Jump
WRD "OF"
ADJ (SUPERLATIVE)
WRD "THAN"
CAT ORD
NP/ORD
Jump
NP/ADJ
ADJ (COMPARATIVE)
NP/THAN
ADJ
Jump
CAT N (POSS)
CAT NPR (POSS)
Jump
Jump
CAT N
CAT NUM
CAT NPR
NP/HEAD
NP/NMOD
CAT N
Jump
NP/ACC
POP
CAT PRO (QUEST)
CAT ADV (QUEST)
CAT PRO (RELATIVE)
Jump
CAT PRO (GENERAL)
NP/POSTMOD
ADJ

QUANTIFIER



VERB PHRASE



Appendix B

Case List

This appendix consists of a list of cases supplied to the NL system to simplify the definition process. It is neither an exhaustive nor completely defined list. The case list used here is a slightly modified version of the one found in Taylor and Rosenberg (1975).

AG - the AGENT of an action
- the one who acts

BEN - the BENEFICIARY of an action
- the one who receives an advantage

CAUS - the CAUSE
- the agent which produces an effect or result

COAG - the COAGENT of an action
- the one who acts with the agent

DEST - the DESTINATION
- where something is directed

EN - to ENABLE
- to make possible

EX - to EXCHANGE
- to give and receive

INST - the INSTRUMENT of an action
- what it was done with

LOC - the LOCATION of an action
- where it took place

MAN - the MANNER of an action
- how it was done

MOT - the MOTIVE behind an action
- why it was done

PATH - the PATH
- the course of action

PA - the PATIENT of an action
- the one which is acted upon

PURP - the PURPOSE
- the reason for carrying out the action

QUAN - the QUANTITY
- the amount

RE - the RECIPIENT of an action
- the one who receives something from the action

SOU - the SOURCE of the action
- the origin

TIME - the TIME of the action
- when it took place

TOP - the TOPIC of the action
- what it is about

Appendix C

Partial Definition of the Syntactic Dictionary

This appendix attempts to give a flavor of the entries in the syntactic dictionary (see Section 4.2). Included here are the common, domain independent words: the determiners, quantifiers, prepositions and even some adjectives and adverbs. Along with the entries is a brief explanation† of the semantic markers used in their definition.

```
(","
     ; the comma is treated as a conjunction to stop parts
     ; of two different compound words being joined together
   CONJ *)
     ; the '*' just means that the conjunction will be
     ; treated the same as the next non-* conjunction found;
     ; for example, in "Bill, John or Mary" the comma is
     ; treated as an "or" while in "Bill, John and Mary" it
     ; is treated as an "and"
(A
   DET *
     ; signifies that "a" is a determiner
   DET* ((NUMBER SG) (ARTICLE INDEF))
     ; the properties that this determiner give to the NP
     ; following it are "singular" and "indefinite"
```

† Any line beginning with ";" is a comment and not part of the word definition.

```
(AFTER
   PREP *
     ; this word is a preposition
   PREP* ((CASES (TIME)))
     ; it indicates the "time" case
   INDR *MORE)
     ; it indicates the relation ">";
     ; for example, "after 1970" means "> 1970"
(ALSO
   CONJ AND)
     ; this conjunction is the same as "and"
     ; the ABBREV marker could also have been used here
(AM
   V (BE (TNS PRESENT) (PNCODE 1SG)))
     ; this definition says that "am" is a verb whose root
     ; is "be"
     ; TNS PRESENT informs the parser that the verb is in the
     ; present tense and
     ; PNCODE 1SG says that it is first person singular
(AN
   DET (A))
(AND
   CONJ *)
(ANY
   QUANT *
   NVALUE 0
   QVALUE *MORE)
```

```
      ; the NVALUE and QVALUE markers define "any" to
      ; mean "> 0"
(ANYTHING
   PRO *
   PRO* (GENERAL)
      ; defines it to be a general pronoun
   QVALUE *ONE)
(ARE
   V (BE (TNS PRESENT) (PNCODE X13SG)))
(AREN'T
   ABBREV (ARE NOT))
      ; this is how abbreviations are added to the dictionary
(AS
   ADV *)
(BE
   V *
      ; the '*' signifies that the verb is irregular and so
      ; all of it's conjugations must be prestored in the
      ; dictionary
   V* (COPULA (AUX PASSIVE)))
(BEEN
   V (BE (TNS PASTPART)))
(BEFORE
   PREP *
   PREP* ((CASES (LOC TIME BEN)))
      ; the cases indicated by this preposition are "location",
      ; "time" and "beneficiary"
```

```
   INDR *LESS)
(BEING
   V (BE (TNS PRESPART)))
(BEST
   ADJ (GOOD SUPERLATIVE))
(BETTER
   ADJ (GOOD COMPARATIVE))
(BOTH
   QUANT *
   QVALUE *ALL)
(BUT
   CONJ (AND))
(CAN
   V *
   V* ((TNS PRESENT) (PNCODE ANY) (AUX MODAL)))
(COUPLE
   QUANT *
   NVALUE 2)
(DATUM
   N A)
      ; morphological information to derive the root from
      ; the plural
(DO
   V *
   V* ((AUX TNS)))
(EACH
   DET *
```

```
   QVALUE *ALL)
(EARLY
   ADJ ER-EST)
(FOR
   PREP *
   PREP* ((CASES (EX BEN))))
     ; the cases indicated by this preposition are the
     ; "exchange" and "beneficiary" cases
(FROM
   PREP *
   PREP* ((CASES (SOU METH))))
     ; this indicates the cases "source" and "method"
(GOOD
   ADJ *)
     ; the actual definition of "good" must be supplied in
     ; the domain definition since it will change from domain
     ; to domain
(HANDFUL
   QUANT *
   NVALUE 3)
(HOW
   ADV *
   ADV* (QUEST (CASES (MAN))))
((HOW MANY)
     ; the words are in parenthesis to inform the parser to
     ; to treat them as a single entry
   DET *
```

```
   DET* (QUEST)
   PRO *
   PRO* (QUEST)
   INDF *NUMBER)
(IN
   PREP *
   PREP* ((CASES (LOC TIME MAN DESC))))
     ; this indicates the cases "location", "time", "manner"
     ; and "description"
(LEAST
   ADJ (LITTLE SUPERLATIVE)
(LITTLE
   ADJ *
   ADV *
   QVALUE *LESS)
(NONE
   QUANT *
   NVALUE 0)
(ON
   PREP *
   PREP* ((CASES (LOC TIME))))
     ; the cases indicated by this preposition are "location"
     ; and "time"
(SECOND
   ORDINAL *
   NVALUE 2)
(SOME
```

```
   QUANT *
   NVALUE 3)
   QVALUE *MORE)
(THE
   DET *
   DET* ((NUMBER SG-PL) (ARTICLE DEFINITE)))
(THEY
   PRO *
   PRO* (SUBJ (NUMBER PL) (PNCODE 3PL)))
(WHAT
   DET *
   DET* (QUEST)
   PRO *
   PRO* (QUEST))
(WHEN
   ADV *
   ADV* (QUEST (CASES (TIME))))
     ; this definition allows "when" to indicate any word
     ; filling the "time" case
(WHERE
   ADV *
   ADV* (QUEST (CASES (LOC))))
     ; this allows "where" to indicate any word filling the
     ; "location" case
   PRO *
   PRO* (QUEST RELATIVE))
```

Appendix D

Partial Definition of the Restaurant Domain

Each domain definition (see Section 5.1) is composed of a domain dictionary, case list and an inverted database. Samples of these are given here along with a brief explanation of their use† in the NL system.

D.1 The Domain Dictionary

The domain dictionary contains a definition of the fields in the database as well as all of the jargon common to the domain. Items which will be found in the database itself will not usually be found here.

```
(ADDRESS
   N ES
     ; morphological information
   DBFIELD LOC
     ; signifies that the name of the address field in the
       database is LOC
   DBCAT N)
     ; directs the system to treat entries in the datbase
     ; field LOC as nouns
```

† Any line beginning with ";" is a comment and not part of the domain definition.

```
(BAD
     ; morphological information is already in the common
     ; dictionary so need not be repeated here
   INDF STARS  INDR *LESS  INDE 2)
     ; the 3 tags INDF, INDR and INDE define a restaurant to
     ; be "bad" if the condition "STARS < 2" holds
(CHEAP
   ADJ *
   INDF COST  INDR *LESS  INDE MODERATE)
     ; the condition for a "cheap" restaurant is
     ; "COST < MODERATE"
(COST
   N S
   DBFIELD COST
   DBCAT ADJ
     ; elements of this field are treated as adjectives
   ORDER (INEXPENSIVE MODERATE EXPENSIVE))
     ; an ordering is placed on the COST field where
     ; INEXPENSIVE is lower than MODERATE which is lower than
     ; EXPENSIVE
     ; the orderings are used in answering comparative
     ; questions - in this case questions relating to
     ; "cheaper" and "more expensive"
(DRINK
   N S
   INDF WINE-LIST
   V IRR
```

```
   ACTION (AG *HUMAN PA WINE-LIST))
     ; this ACTION definition says that "humans drink what is
     ; on the wine list"
(EAT
   V IRR
   ACTION (AG *HUMAN PA (FOOD MEALS) RE *HUMAN))
     ; humans eat both food (e.g. chicken) and meals
     ; (e.g. lunch)
(FOOD
   N S
   DBFIELD FOOD
   DBCAT N)

(GET
   SYNONYM EAT)
     ; the SYNONYM feature allows us to quickly define many
     ; words which mean the same thing
(GOOD
   INDF STARS  INDR *MORE  INDE 3)
     ; definition of "STARS > 3" as being "good" is subjective
     ; as are all of the adjective definitions in the domain
     ; dictionary
(HAVE
   SYNONYM SERVE)
(LIQUOR
   N MASS
   INDF WINE-LIST)
(LOCATION
```

```
   N S
   SYNONYM ADDRESS)
(MEAL
   N S
   DBFIELD MEALS
   DBCAT N)
      ; since there are only a few MEALS, the order could be
      ; defined here (as in COST) to facilitate answering
      ; questions concerning "earlier" and "later"
(MENU
   N S
   INDF FOOD)
(NAME
   N S
   DBFIELD NAME
   DBCAT NPR)
(NUMBER
   N S
   INDF PHONE)
(OPEN
   V S-ED
   ACTION (AG NAME PA MEALS))
      ; this ACTION definition means that restaurants are open
      ; for meals (e.g. lunch)
(ORDER
   V S-ED
   SYNONYM EAT)
```

```
(PHONE
   N S
   DBFIELD PHONE
   DBCAT N
   V S-D
   ACTION (AG *HUMAN PA PHONE))
((PHONE NUMBER)
      ; the parenthesis around the dictionary entry mean that
      ; the two words "phone" and "number" are to be treated as
      ; a single entry
   N S
   INDF PHONE)
(PLACE
   N S
   INDF NAME)
(PROVIDE
   V S-D
   SYNONYM SERVE)
(QUALITY
   N S
   INDF STARS)
(RATE
   V S-D
   ACTION (AG *HUMAN PA STARS RE NAME))
(RATING
   N S
   INDF STARS)
```

```
(RESERVATION
   N S
   DBFIELD RESERVATIONS
   DBCAT N)
(RESTAURANT
   N S
   INDF NAME)
(SERVE
   V S-D
   ACTION (AG NAME PA (FOOD MEALS) RE *HUMAN))
(STAR
   N S
   DBFIELD STARS
   DBCAT N)
(STREET
   N S
   INDF ADDRESS)
(WHO
   INDF NAME)
((WINE LIST)
   N S
   SYNONYM WINE-LIST)
(WINE-LIST
   N S
   DBFIELD WINE-LIST
   DBCAT N)
```

D.2 The Case List

There are only a few cases defined for the restaurant domain. The cases are the basis for determining the function of a prepositional phrase as well as general averbial questions such as "when" and "where". The cases defined are:

```
(LOCATION        (NAME ADDRESS)
(TIME            MEALS)
```

D.3 The Inverted Database

Most of this inverted database was produced automatically by a program written in the MTS Edit Sublanguage (Hogg 1980). Additions were then made to it to include abbreviations and synonyms.

```
- Restaurant Names -
(ACROPOL
   ELEMENT-OF NAME)
     ; means that ACROPOL can be found in the NAME field
((AKI JAPANESE RESTAURANT NO 2)
   ELEMENT-OF NAME)
(AKI
   SYNONYM "AKI JAPANESE RESTAURANT NO 2")
     ; means that "Aki" is a synonym for "Aki Japanese
```

```
     ; Restaurant No 2"
((CANYON GARDENS) ELEMENT-OF NAME)
(GAZEBO ELEMENT-OF NAME)
((IL GIARDINO) ELEMENT-OF NAME)
((LAS TAPAS) ELEMENT-OF NAME)
((SALMON HOUSE ON THE HILL) ELEMENT-OF NAME)
((SEVEN SEAS) ELEMENT-OF NAME)
((WHITE SPOT) ELEMENT-OF NAME)
((WILLIAM TELL) ELEMENT-OF NAME)
((YANGTZE KITCHEN) ELEMENT-OF NAME)
(YANGTZE SYNONYM YANGTZE KITCHEN)

- Types of Food -
(AFGHAN ELEMENT-OF FOOD)
(AMERICAN ELEMENT-OF FOOD)
(AMERICAN
   FOOD+ (BURGER "HOT DOG" CHICKEN STEAK))
     ; the FOOD+ designator means that any search for American
     ; food will also look for "burger", "hot dog", "chicken"
     ; and "steak"
(BURGER SYNONYM (HAMBURGER CHEESEBURGER))
     ; directs the database interface to search for "hamburger"
     ; and "cheeseburger" whenever "burger" is requested
     ; the combined definitions of "American" and "burger" will
     ; cause any request for "American food" to produce a query
     ; looking for any of "American", "burger", "hamburger",
     ; "cheeseburger", "hot dog", "chicken" or "steak"
```

```
(CHINESE ELEMENT-OF FOOD)
(CURRY ELEMENT-OF FOOD)
(HAMBURGER ELEMENT-OF FOOD)
(HAMBURGER FOOD+ BURGER))
(ITALIAN ELEMENT-OF FOOD)
(JAPANESE ELEMENT-OF FOOD)
(LASAGNE ELEMENT-OF FOOD)
(LOBSTER ELEMENT-OF FOOD)
(SCHNITZEL ELEMENT-OF FOOD)
(SEAFOOD ELEMENT-OF FOOD)
(STEAK ELEMENT-OF FOOD)

- Types of Meals -
(BREAKFAST ELEMENT-OF MEALS)
(DINNER ELEMENT-OF MEALS)
(DINNER MEALS+ SUPPER)
((LATE NIGHT)
   ABBREV LATE-NIGHT)
     ; ABBREV is different from SYNONYM in that an abbreviation
     ; will occur at the lexical level and a synonym will occur
     ; at the root word level
(LATE-NIGHT ELEMENT-OF MEALS)
(LUNCH ELEMENT-OF MEALS)
(TEA ELEMENT-OF MEALS)
(SUPPER SYNONYM DINNER)

- Types of Costs -
(CHEAP SYNONYM INEXPENSIVE)
```

```
(EXPENSIVE ELEMENT-OF COST)
(EXPENSIVE COST+ PROHIBITIVE)
(INEXPENSIVE ELEMENT-OF COST)
(INEXPENSIVE COST+ (CHEAP REASONABLE))
(MODERATE ELEMENT-OF COST)
(REASONABLE SYNONYM INEXPENSIVE)

- Locations -
(BC ABBREV BRITISH COLUMBIA)
(BOULEVARD ELEMENT-OF LOC)
((BRITISH COLUMBIA) ELEMENT-OF LOC)
(GASTOWN ELEMENT-OF LOC)
((PACIFIC CENTRE) ELEMENT-OF LOC)
((PARK ROYAL) ELEMENT-OF LOC)
(VANCOUVER ELEMENT-OF LOC)

- Possible Stars -
(0
   ELEMENT-OF STARS)
     ; the rating goes from a 0 star restaurant to a 5 star
     ; restaurant
(1 ELEMENT-OF STARS)
(2 ELEMENT-OF STARS)
(3 ELEMENT-OF STARS)
(4 ELEMENT-OF STARS)
(5 ELEMENT-OF STARS)
```

Appendix E

Sample Session

This is a sample session of the NL system. The questions here are indicative of ones which can be answered by the system and are not intended to show the system limits. Lines prefixed with the character ">" were output by the system and lines prefixed with "*" were input by the user.

```
>
>    Which database do you want to look at?
>
>    1. Restaurants
>    2. AI Bibliography
>    3. Conference
* 1
>
>      You are currently examining the RESTAURANTS database.
>
>      All of your queries should be entered in upper case and
>    should be terminated by either a ' ? ', ' . ' or ' ! '.
>    Compound names (e.g. the title of a book) should be
>    entered with surrounding double quotes ' " ' and
>    apostrophes may be used to signify possession. No other
>    punctuation should be used.
>
>      To leave the system type STOP., to get more information
>                               -----
>    type HELP. and to get this message again type INFO.
>         -----                                   -----
>
>
>    HELLO THERE
>
* WHO SERVES CHICKEN?
>
>    PARSE:
>    WHFIND
>        *AND
>            NAME = ?
>            FOOD = CHICKEN
>
>    "STEER AND STEIN" AND "WHITE SPOT" SERVE CHICKEN.
```

```
>
* WHAT ARE SOME ITALIAN RESTAURANTS?
>
>     PARSE:
>     WHFIND
>         *AND
>             *NUMBER = 3
>             *OR
>                 FOOD = ITALIAN
>
>     THEY ARE "CAFFE DE MEDICI", "FRANCESCO ALONGI" AND
>     "CORSI TRATTORIA".
>
* CAN YOU FIND ME A CHEAP JAPANESE PLACE?
>
>     PARSE:
>     WHFIND
>         *AND
>             *AND
>                 *OR
>                     COST = INEXPENSIVE
>                     COST = CHEAP
>                     COST = REASONABLE
>                 *NUMBER = 1
>             FOOD = JAPANESE
>
>     NONE CAN BE FOUND.
>
* WHICH IS THE MOST EXPENSIVE RESTAURANT IN VANCOUVER?
>
>     PARSE:
>     WHFIND
>         *AND
>             *AND
>                 *OR
>                     COST = EXPENSIVE
>                     COST = IMMODERATE
>                     COST = PROHIBITIVE
>                 NAME = ?
>             *OR
>                 LOC = VANCOUVER
>                 LOC = VANC
>                 LOC = VAN
>
>     THERE ARE 19 REFERENCES.
>     DO YOU WANT THEM PRINTED?
* NO
>
```

```
* WHAT IS ON THE MENU AT STEER AND STEIN?
>
>     PARSE:
>     WHFIND
>         *AND
>             NAME = WHITE SPOT
>             FOOD = ?
>
>     THEY ARE STEAK, SALADS, CHICKEN AND FISH.
>
* IS THERE A SPANISH PLACE WHICH IS OPEN FOR LUNCH?
>
>     PARSE:
>     WHFIND
>         *AND
>             *AND
>                 *NUMBER = 1
>                 FOOD = SPANISH
>             MEALS = LUNCH
>
>     IT IS "LAS TAPAS RESTAURANTE".
>
* WHO SERVES CHINESE FOOD?
>
>     PARSE:
>     WHFIND
>         *AND
>             NAME = ?
>             *OR
>                 FOOD = CHINESE
>                 FOOD = INDO-CHINESE
>
>     THERE ARE 21 REFERENCES.
>     DO YOU WANT THEM PRINTED?
* OK
>     "THE JASMINE INN", "DAI KEE",
>     "SHANGHAI PALACE RESTAURANT", "NEW DIAMOND RESTAURANT",
>     "HO INN CHOP SUEY", "SNOW GARDEN", "KAM YUEN RESTAURANT",
>     "PINK PEARL", "VARSITY GRILL", PENINSULA,
>     "KEN WONG VILLAGE", "YANGTZE KITCHEN",
>     "MY TAN RESTAURANT", "DRAGON INN", "CHINA KITCHEN",
>     "CANTON KITCHEN", "SHANGHAI HOUSE, MANDARIN RESTAURANT",
>     "HO HO", "THE CATHAYAN", "MY LAI GARDEN RESTAURANT", AND
>     "NEW WORLD CHOP SUEY" SERVE CHINESE.
>
* DO THEY SERVE CHICKEN?
>
>     PARSE:
>     YES-NO
>         *AND
>             *REF = *
>             FOOD = CHICKEN
>
>     NO, THERE ARE NO REFERENCES.
```

```
>
* IS THERE A FRENCH PLACE IN RICHMOND WHICH IS OPEN FOR LUNCH?
>
>     PARSE:
>     WHFIND
>         *AND
>             *AND
>                 *NUMBER = 1
>                 FOOD = FRENCH
>             *OR
>                 LOC = RICHMOND
>             MEALS = LUNCH
>
>     IT IS "LE CERCLE".
>
* WHO SERVES DINNER?
>
>     PARSE:
>     WHFIND
>         *AND
>             NAME = ?
>             *OR
>                 MEALS = DINNER
>                 MEALS = SUPPER
>
>     THERE ARE 109 REFERENCES.
>     DO YOU WANT THEM PRINTED?
* NO
>
* WHERE IS KEN WONG VILLAGE?
>
>     PARSE:
>     WHFIND
>         *AND
>             NAME = KEN WONG VILLAGE
>             LOC = ?
>
>     IT IS "1262 BURRARD STREET AND 555 W HASTINGS, VAN. B.C.".
>
* WHEN IS IT OPEN?
>
>     PARSE:
>     WHFIND
>         *AND
>             *REF = *
>             MEALS = ?
>
>     IT IS "LUNCH AND DINNER".
>
```

```
* WHAT DOES WHITE SPOT SERVE?
>
>     PARSE:
>     WHFIND
>         *AND
>             NAME = WHITE SPOT
>             FOOD = ?
>
>     WHITE SPOT SERVES AMERICAN, HAMBURGERS AND CHICKEN.
>
* WHAT MEALS DOES WHITE SPOT SERVE?
>
>     PARSE:
>     WHFIND
>         *AND
>             NAME = WHITE SPOT
>             MEALS = ?
>
>     WHITE SPOT SERVES BREAKFAST, LUNCH AND DINNER.
>
* FIND AT LEAST 4 RESTAURANTS THAT HAVE SWISS FOOD.
>
>     PARSE:
>     WHFIND
>         *AND
>             *NUMBER = 4
>             NAME = ?
>             FOOD = SWISS
>
>     THERE ARE ONLY 3 RESTAURANTS THAT FIT THE CONSTRAINTS.
>     "WILLIAM TELL", "LA RACLETTE" AND "GIZELLA SWISS CHALET"
>     SERVE SWISS.
>
* STOP.
```

```
>    Which database do you want to look at?
>
>    1. Restaurants
>    2. AI Bibliography
>    3. Conference
* 2
>
>       You are currently examining the AI BIBLIOGRAPHY database.
>
>       To leave the system type STOP., to get more information
>                                -----
>     type HELP. and to get this message again type INFO.
>          -----                                   -----
>
>    HELLO THERE
>
* WHO WROTE APSECTS?
>
>    PARSE:
>    WHFIND
>        *AND
>            TITLE = ASPECTS
>            AUTHOR = ?
>
>    "CHOMSKY N" WROTE APSECTS.
>
* HOW MANY BOOKS HAS MCCARTHY WRITTEN?
>
>    PARSE:
>    WHFIND
>        *AND
>            AUTHOR = MCCARTHY
>            *AND
>                *NUMBER = ?
>                TITLE = ?
>
>    THERE ARE 9 BOOKS.
>
* FIND A BOOK WRITTEN BY MINSKY BEFORE 1978.
>
>    PARSE:
>    WHFIND
>        *AND
>            *AND
>                *NUMBER = 1
>                TITLE = ?
>            *AND
>                AUTHOR = MINSKY
>                *AND
>                    DATE < 1978
>
>    IT IS PERCEPTRONS.
>
* STOP.
```

```
>    Which database do you want to look at?
>
>    1. Restaurants
>    2. AI Bibliography
>    3. Conference
* 3
>
>       You are currently examining the IJCAI-81 database.
>
>       To leave the system type STOP., to get more information
>                                -----
>     type HELP. and to get this message again type INFO.
>          -----                                   -----
>
>    HELLO THERE
>
*
* WHO IS COMING FROM MIT?
>
>    PARSE:
>    WHFIND
>        *AND
>            NAME = ?
>            INST = MIT
>
>    THERE ARE 13 REFERENCES.
>    DO YOU WANT THEM PRINTED?
* YES
>    THEY ARE "GLASS, BRIAN", "MCALLESTER, DAVID",
>    "SUSSMAN, GERALD J.", "WHITE, BARBARA", "DAVIS, RANDALL",
>    "HEWITT, CARL", "OGILVIE, WILLIAM", "HAWKINSON, LOWELL B",
>    "HAMSCHER, WALTER", "PITMAN, KENT", "FRY, CHRISTOPHER",
>    "WATERS, RICHARD C" AND "LESCANE, PIERRE".
>
* WHEN DID "LAM, MONICA" REGISTER?
>
>    PARSE:
>    WHFIND
>        *AND
>            NAME = "LAM, MONICA"
>            TYPE = ?
>
>    LAM, MONICA REGISTERS EARLY-STUDENT.
>
```

```
* HAS ROSENBERG REGISTERED YET?
>    I cannot find ' YET ' in the dictionary.
>    Do you wish to stop processing this query?
* NO
>    Did you misspell ' YET ' ?
* NO
>    Would I find ' YET ' in the database ?
* NO
>    Would it be safe to ignore the word ' YET '?
* YES
>
>    PARSE:
>    YES-NO
>        NAME = ROSENBERG
>
>    YES, THERE ARE 2 REFERENCES.
>
* WHAT ARE THEIR NAMES?
>
>    PARSE:
>    WHFIND
>        *AND
>            *REF = *
>            NAME = ?
>
>    THEY ARE "ROSENBERG, RICHARD" AND "ROSENBERG, STEVEN".
>
* STOP.
```