

799

AN NL INTERFACE TO RELATIONAL DATABASES

✓
694

Dissertation submitted to the Jawaharlal Nehru University
in partial fulfilment of the requirements
for the award of the Degree of
MASTER OF TECHNOLOGY

Thesis (M. Phil) — Jawaharlal Nehru University,
1989

Typescript (Photocopy)

By

VINOD KUMAR

102 p.

scss

Sup: P.C. Saxena



to the

SCHOOL OF COMPUTER AND SYSTEMS SCIENCES

JAWAHARLAL NEHRU UNIVERSITY JNU

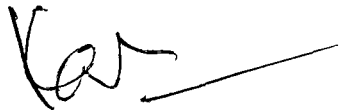
NEW DELHI - 110 067

1989

CERTIFICATE

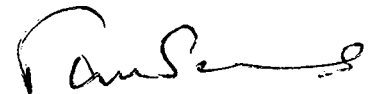
This is to certify that the research work embodied in this dissertation, entitled 'AN NL INTERFACE TO RELATIONAL DATABASES' by VINOD KUMAR has been carried out at the School of Computer and Systems Sciences, JNU New Delhi - 110 067, and has not been submitted elsewhere for a degree.

Vinod
VINOD KUMAR
(student)



PROF. KARMESHU

(Dean)



Dr. P.C. SAXENA

(Supervisor)

ACKNOWLEDGEMENTS

I express my sincere thanks and gratitude to Dr. P. C. Saxena for his continuous guidance and suggestion throughout the course of this work.

I would also like to extend my thanks to Mr. Fateh Singh, Administrative officer SCSS. JNU for his help throughout my M.Tech programme.

My thanks are also due to the faculty and other staff for their co-operation and help in many ways.

J. N. U., Delhi

July 1989



VINOD KUMAR

ABSTRACT

In this work, we present an approach towards building a portable natural language interface to relational databases. A representation scheme, called ^{"qframe" for} queries addressed in natural language is created from the output of a syntactic parse stage. An interface, using the above scheme has been implemented. This generates a formal query in relational algebra which operates on a simulated in-memory database to get results.

CONTENTS

| CHAPTER | PAGE |
|--|------|
| 1. Introduction | 1 |
| 2. Survey of related systems | 5 |
| 3. System overview | 10 |
| 4. Parsing | 19 |
| 5. Semantic Analysis | 38 |
| 6. Formal query generation & database access | 59 |
| 7. Conclusion | 64 |
| APPENDICES | |
| APPENDIX I | |
| PART I : Augmented Transition Networks | 70 |
| PART II : Program Listing | 78 |
| PART III : Examples with Parser output | 86 |
| PART IV : Examples of Parsing | 88 |
| APPENDIX II | |
| PART I : Relational Model | 90 |
| PART II : List of the Relations in the database & the simulated database | 94 |
| APPENDIX III LEXICON | 96 |
| APPENDIX IV Sample Runs | 99 |
| References | 101 |

CHAPTER 1

INTRODUCTION

Language is a process of communication between two intelligent active processors, in which both the producer and the comprehender perform complex cognitive operations. The producer begins with communicative goals, including effects to be achieved, information to be conveyed and attitudes to be expressed. In order to communicate, the producer must map this multi-dimensional collection of goals into sequence of words using the variety of information resources provided by the language.

1.1 Motivation

Ever since the evolution of computers as intelligent processors of numbers and symbols, there has been a growing interest among researchers in the area of Artificial Intelligence, to make computers understand natural language. By natural language, we mean a language spoken by humans.

In the last decade, there have been two different kinds of work in natural language on computers. One trend has been towards pursuing the deeper problems of meaning and conversation, concentrating on issues other than syntax.

The other trend has been towards systems that have a limited capacity to handle complex syntax and deduction, but can be used as practical "front ends" for data retrieval systems. generally, these systems make use of well known and understood techniques to cover a subset of natural language that is incomplete but habitable in that a person using the system will quickly learn what kinds of things are handled and which others to avoid.

In this endeavour, we present an approach towards a portable natural language interface for relational database systems.

1.2 Objectives

The system will accept queries in natural language (English) and convert them to formal queries which will be used to access the database and get the answers. The design of our system has been influenced by the following main objectives.

- (a) A reasonable subset of natural language must be accepted.
- (b) Linguistic phenomena like anaphora and ellipsis should be handled.

- (c) The system should be portable across different domains of database.
- (d) The system, while keeping the interaction with the user minimal, should give co-operative responses.

1.3 Contribution

In this work, we have presented an approach for the design of a natural language system which attempts to achieve the objectives. We have proposed a representation scheme for queries expressed in natural language and directed towards a database. We call this a "question frame". Our main idea is to capture the essential semantics of a query which is directed towards a database. We have also proposed a formal specification for database specific words in the lexicon.

We have implemented a system as an interface for a relational database, using the above ideas.

1.4 Organisation of the thesis

Chapter 2 is a brief survey of existing systems: Chapter 3 gives an overview of the various modules in the system; Chapters 4 to 6 discuss these modules in detail.

Chapter 7 summarises the work, and discusses, in brief, limitations of the system and scope for the future work.

Appendix 1 gives an introduction to ATNs and also contains the complete set of diagrams of the ATNs used in our system. Appendix 2 gives an introduction to Relational Algebra and lists the various relations that have been used in the system as an example. Appendix 3 contains the lexicon that is used and Appendix 4 contains sample runs of the system.

CHAPTER 2

SURVEY OF RELATED SYSTEMS

In this chapter we will discuss, in brief, a few of the natural language based systems that have been successful. The systems that we have chosen for discussion are **LUNAR** (Woods '78), **PLANES** (Waltz '75), **INTELLECT** (Harris '77) and **TEAM** (Grosz '82). We will discuss the design aspects of these systems in light of our objectives.

2.1 Lunar

LUNAR is a question-answering system that responds to questions based on data about the mineral samples brought back from the moon, using a large database provided by the NASA. It uses an ATN grammar that was motivated by the transformational grammar theory and therefore produces deep tree structures than register assignments. Registers are thought of as temporary holding places for use during the parsing of a constituent and so are used for holding features as well as constituents, but not in a systematic way.

Parsing is done left-to-right and top-down. Some versions of this system have used backtracking and others, a

parallel scheme. In both cases, use is made of a likelihood ordering on the arcs leaving any state. Given a choice, the system takes the most likely one first, in order to increase the probability of finding the right path, early in the process. The system includes some special heuristics to avoid large combinatorial searches.

The system is extremely good in handling queries idiosyncratic to the domain. However, the portability of the system is very low.

2.2 PLANES

PLANES is an English language question-answering system for a large database on Navy aircraft maintenance. Its limited domain results in a small vocabulary with no lexical ambiguity, and leads users to offer few complex sentences. Since the system is built with a good a priori idea of what the users will want to know, it can deal with some kinds of ellipsis and non-grammatical sentences.

The system consists of three stages: parsing, concept case frame generation and query generation.

In the first step, a pre-pass does spelling checking, removal of noise words and marks inflected words. The parser

does a phrase-by-phrase match and uses specially programmed heuristic techniques for locating the boundaries of noun groups and relative clauses.

PLANES is not designed to adapt to a new database; it uses many assumptions on the domain of discourse and hence it is heavily bound to the specific domain of aircraft database.

2.3 INTELLECT

The INTELLECT system uses an ATN grammar with backtracking. It generates all possible interpretations, then calls another component to distinguish those that are semantically possible, directly using the information in the database.

INTELLECT is designed to work with any database of the right form, without a special dictionary. It therefore uses the contents of the database itself as a way of determining what types of objects can go with what relations. It performs spelling correction and handles phrases (idioms) through special mechanisms. There is an explicit, separate ATN for sentence fragments, which is called only if the attempt to parse the input as a full sentence fails.

Conjunction is handled with explicit arcs in each place it is allowed to occur.

2.4 TEAM

This system is one of the earliest to have laid emphasis on 'portability' of the interface across different domains.

TEAM is designed to interact with two kinds of users: a database expert and an end-user. The database through a system-directed acquisition dialogue. As a result of this dialogue, the language processing and data access components are extended so that the end-user may query the new database in natural language.

The system has three major components:

- (1) An acquisition component ;
- (2) The DIALOGIC language system,
- (3) A data access component.

The translation of an English query into a database query takes place in two stages. First, the DIALOGIC system constructs a representation of the literal meaning or the logical form into a formal database query. Each of these

steps requires a combination of information that is dependent on the domain; and information that is not. To provide for transportability, TEAM carefully distinguishes between the two.

CHAPTER 3

SYSTEM OVERVIEW

The main objectives of our system, as outlined in the first chapter are,

- (1) A large subset of English should be accepted.
- (2) A high degree of portability across different domains should be achieved.
- (3) Linguistic phenomena like anaphora and ellipsis should be handled.
- (4) Co-operative responses should be provided.

These have influenced the design considerations of our system. The various modules of the system are discussed in brief in this chapter. Detailed discussions for these modules follow in later chapters. The scope and limitations of our system are outlined in the last section of this chapter.

3.1 Approach adopted

We have followed the syntactic grammar approach in our system. This approach becomes essential as we set "portability" as one of our goals. The disadvantage is that we might not be able to handle queries that are

idiosyncratic of some particular database. However, we can write a pre-processor to handle such queries, as a front end.

For semantic, we have proposed a frame structure, which we call the "question frame", in which we capture the essential semantics of the natural language query.

The formal query generated by the system is in "relational algebra" and therefore can be easily converted to any of the standard query languages supported by database systems. Presently our system is running on a database simulated in the memory itself, accepting relational algebra queries.

3.2 A brief description of the system

The block diagram given in the next page highlights the main components of the system. In the following sections we will discuss, in brief, the various modules in the system.

3.2.1 Parser

The active component of the parser is an ATN interpreter. It is a left-to-right top-down parser with a state-saving approach for backtracking. The implementation

BLOCK DIAGRAM OF THE SYSTEM

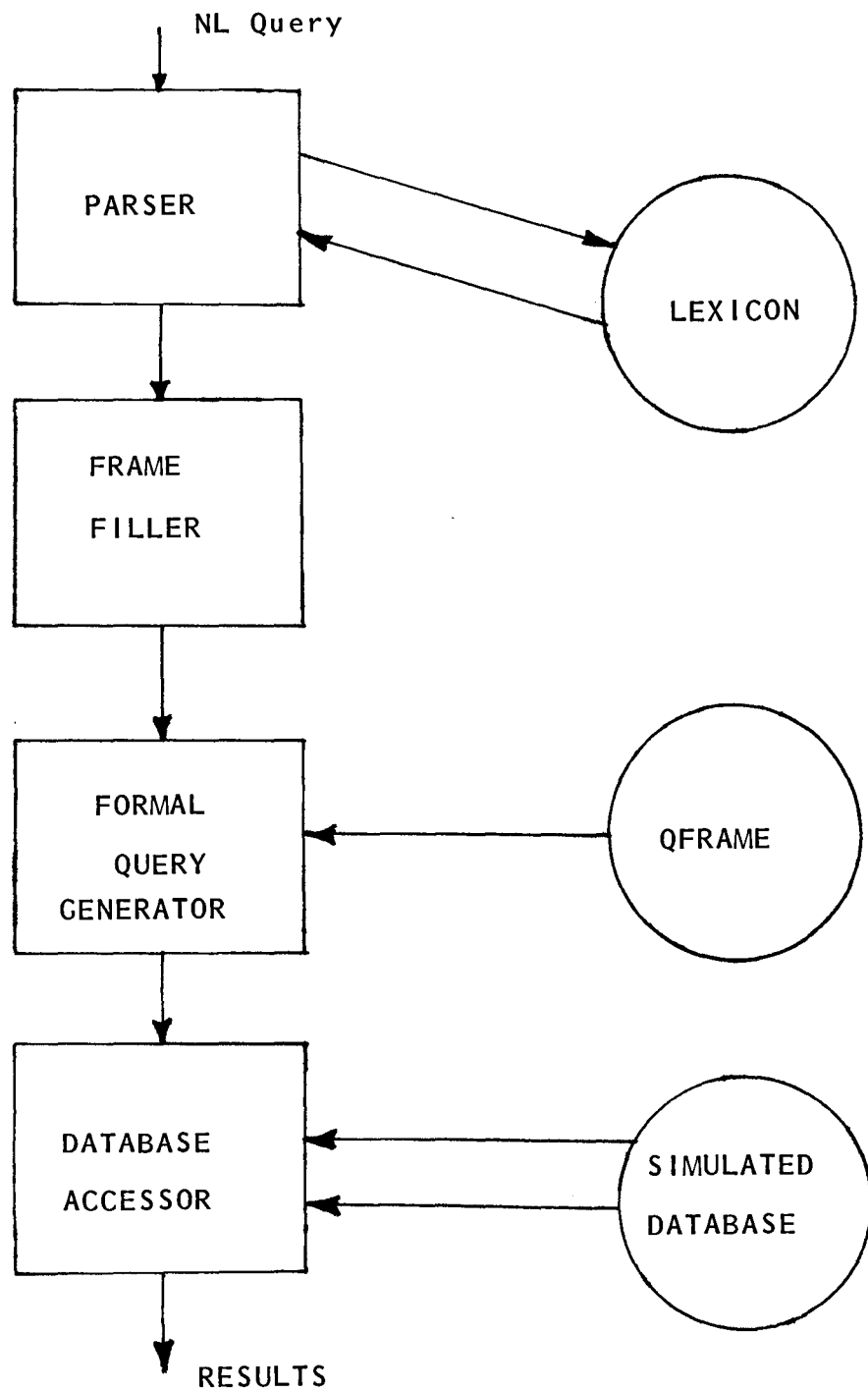


Fig. 3.1

also includes a grammar-specification language for giving the rules of the grammar.

3.2.2 Grammar

The grammar that is being used presently is tightly coupled with the system since the next stage (qframe filler) expects the parsed output to be in a predetermined manner. However, modifications to the grammar can be done provided there is no major change as to effect a modification in the frame filling module.

3.2.3 Lexicon

The lexicon serves two purposes : It provides the syntactic information required for the parsing of the sentence and also contains the data base specific information. It is logically divided into two parts; the core lexicon words and the database specific information that is stored depends on the lexical category of the words. The lexicon has been implemented as a " trie" structure.

3.2.4 Semantic analysis and the concept of "qframes"

To capture the essential semantics of queries in natural language that are directed towards a database, we

have proposed a frame structure which we call the "question frame" (or qframe for short). This representation is conceived as model for all that is involved in any such query.

The second module "qframe filler" takes the parsed output of the parser and uses a set of rules to form the question frame.

The structure of the frame and the rules that have been followed to arrive at the structure are discussed in detail in a later chapter.

3.2.5 Formal query generation

In this stage we generate a formal query corresponding to the natural language question. We consider that the question is a basically about the 'primary object of inquiry' and that the other information that are provided in the frame act as restrictive components.

3.2.6 Database simulation

The database is simulated and exist in the main memory. All the database access functions have been implemeted to interact with this "in-memory" database.

3.3 Scope and limitations of the system

In this section we will discuss, in brief, the class of questions accepted, linguistic phenomena that are tackled and the limitations of the system.

3.3.1 Types of questions accepted

In the present system, only "wh" questions are accepted. "wh" questions are those which start with one of: who, what, which, when and where. "Why" is not included in the list as it leads, invariably to a problem of reasoning which is outside the scope of this system.

Of the auxiliary verbs, modals are not included. (modals include: can, could, will, would, shall, must, may, might).

Any number of subclauses are allowed. Anaphoric references resulting thereof are resolved using.

- (1) Cohesion of idea
- (2) Dialogue with the user in case of ambiguities. In many cases it is found that using the first heuristic alone, we can successfully resolve anaphoric references.

Questions involving proper nouns can occur as they

would normally, without reference to the noun they are instantiating, as in

s1 who supplies ic8086 ? as against

s2 who supplies the part ic8086 ? or the more elaborate

s3 which company supplies the part ic8086 ?

All these forms are acceptable and they get mapped onto a frame which would have as much information as for s3 . This is by exploring the roles of the objects with respect to the verb in question. (This is in the case of questions where there is a verb describing an action. In other cases, other methods are followed and are described in a later chapter). In case there is an ambiguity because of the nature of the verb, allowing more cases of objects in the same sentence structure, as in

s4 who teaches Kumar ?

s5 who teaches cs605 ?

where "teaches" can take the dative case "student" and also the objective case "course", we enter into a dialogue with the user and resolve the ambiguity.

Questions involving comparators are allowed. (by comparators, we mean phrases of the form "more than", "less than", "equal to" etc. However, in the present version of

the grammar, these are restricted to be used in questions involving possessions or properties as in

s6 which company has a rating of more than 5 ?

s7 which parts have a price less than 500 ?

Questions with comparators acting on verbs as in

s8 who supplies more than 5 parts ?, are not included.

A limited number of adjectives are allowed, as long as they are properly defined in the lexicon (rules for specifying adjectives are discussed in the next chapter); specifying the restrictive role they play on the nouns. The scope of adjectives must be the noun immediately succeeding it.

Simple grammatical errors are not checked. These include

- (1) number agreement between noun and article.
- (2) number agreement between noun and verb.

Some non-grammatical sentences might be parsed by the system, but only if the structure suggests that the sentence is meaningful, does the system proceed to get an answer.

Ellipses are not handled by the system.

3.4 Conclusion

In this chapter we had an overview of the system, briefly discussing the various modules and also tried to classify the set of NL queries that are accepted by the system. The chapters that follow, discuss these modules in greater detail.

CHAPTER 4

PARSING

As mentioned in the last chapter, we have adopted a syntactic grammar approach, and therefore, the first step is to parse the input natural language query. The main components that are involved in the parsing stage are (1) The parsing mechanism, (2) The grammar and (3) The lexicon, of which the last two are passive components. In this chapter, we will discuss, in detail, the three components.

4.1 The parsing mechanism

The heart of the parsing mechanism is an ATN interpreter. ATNs are augmented transition networks [Woods '70], which are an extended form of transition networks and having the power of a Turing machine. Languages which are not context-free can be recognised by this machine. An introductory discussion of ATNs is given in Appendix 1.

We call this mechanism an interpreter because the grammar to be used by the parser is not built into the programs. It is specified separately and independently of the parser routines and therefore, may be modified easily.

The ATN used in our system has mem and num arcs : The mem arc is similar to the word arc, but accepts a list of

words. The arc will be traversed if the input word is a member of the list (and of course, if the condition on the arc is satisfied).

The num arc allows any number to be accepted and consumed to effect a transition to the next arc. Presently, only integers are allowed by the system.

The lexicon structure has been modified and is now based on 'trie' structure. A detailed discussion follows in a later section of this chapter.

4.1.1 The parsing strategy

A left-to-right, top-down parsing strategy with a state-saving approach for backtracking has been chosen. The truth of the conditions on an arc depends on the 'register-setting' actions that were taken in the arcs to the left of this arc and possibly the actions taken in this arc, but is independent of the actions to be taken in the arcs to the right of this arc. Since there may be several arcs going out from a state of the ATN, we have to make a choice of the next arc to try. For this, a simple depth-first strategy is used. The arcs going out of a state are statically ordered and tried in this order.

A state-saving approach for backtracking is used. The current frame with its registers and other environment called the 'configuration' is pushed into a special stack called 'history' at each step. This allows us to backtrack. Another stack is used a normal stack for SEEK and SEND operations.

An overview of the parsing process is given below.

- (1) Save configuration in 'history'.
- (2) Try next arc: It can be taken if the implicit condition is satisfied; then if the explicit conditions on the arc are also satisfied, the actions are performed.

If the arc cannot be taken or if the explicit conditions are not satisfied, then we have failed on this arc.

- (3) If there is no failure go to step 1
- (4) Failure: In this case we have to backtrack; Pop configurations until one in which atleast one untried arc is available is reached. If 'history' is emptied in the process, then parse is not possible; we exit.
- (5) go to step 1.

Success may be achieved in step 2 if the arc taken is a <send> arc which cannot be matched with a corresponding <seek>.



TH-2951

4.2 Grammar

The design of the grammar has been influenced by the fact that the parsed output is going to be used by the 'qframe filler' module, which expects the parsed output in a particular fashion.

In the last chapter, we discussed the kind of sentences that are accepted by the system. In this section, we will discuss, in detail, the ATN grammar structure.

4.2.1 ATN grammar structure

The top level ATN (for the query) is as follows.

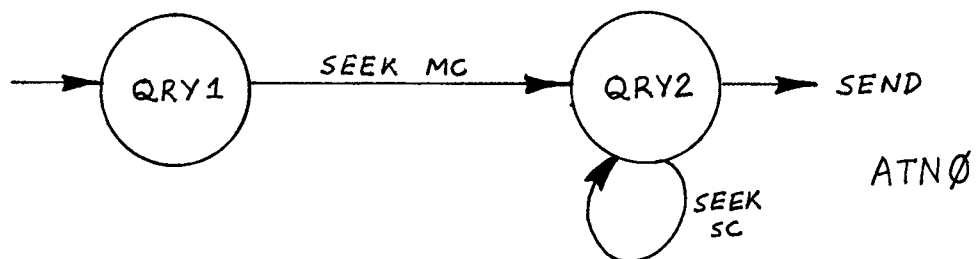


Fig. 4.1

Every query consists of

- (1) one main clause
- (2) zero or more sub clauses

In our grammar, we have slightly deviated from the regular grammatical definition of sub clauses. In our case, a sub clause is said to start with

(1) the occurrence of the second 'wh' word

or

(2) the occurrence of the second 'action verb'

and not otherwise. By action verb we mean any verb that is a non-auxiliary verb and signifying an action, like 'teach' etc.,. examples

s1: what is the address of the company supplying the part IC8086 ?

- no sub clause (note the deviation from regular grammar):

s2: what is the address of the company which is supplying the part IC8086 ?

- sub clause is underlined

s3: which company is supplying the parts having a cost of more than 500 ?

- sub clause is underlined.

4.2.1.1 Main clause ATN structure

This ATN is represented as a block diagram here, to give an idea of the kind of sentences accepted. The detailed ATN diagram is given in Appendix I.

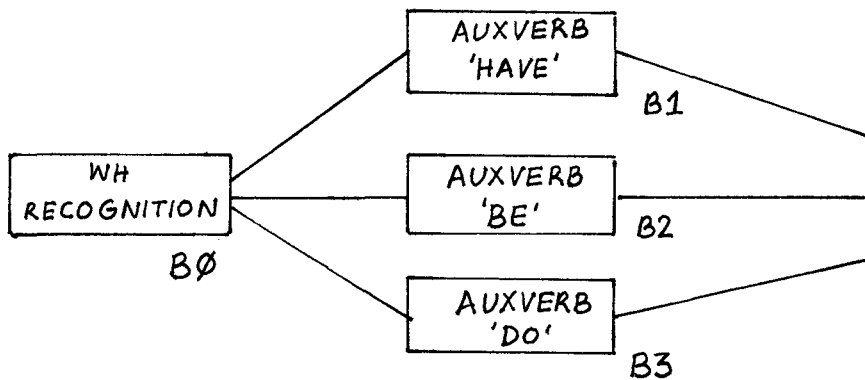


Fig. 4.2

As an illustration, the part of the network represented by B3 in the above figure is expanded and given below.

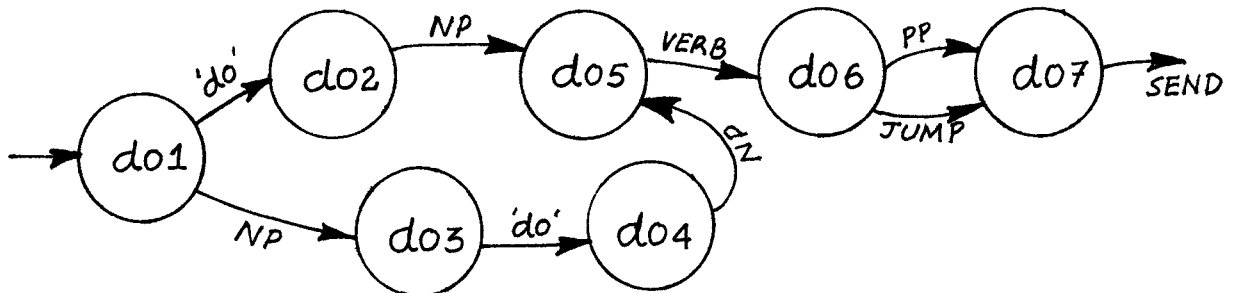


Fig. 4.3

The above network will accept queries of the form

s4—what does the company Intel supply ?

s5— which parts did the company Intel supply in the year 1988 ?

The main clause parsed output frame will include the following slots.

- (1) subject
- (2) direct object
- (3) indirect object
- (4) action verb
- (5) auxiliary verb
- (6) voice
- (7) question type

4.2.1.2 Sub clause structure

The structure is shown in the form of a block diagram below. The detailed diagram is shown in Appendix I.

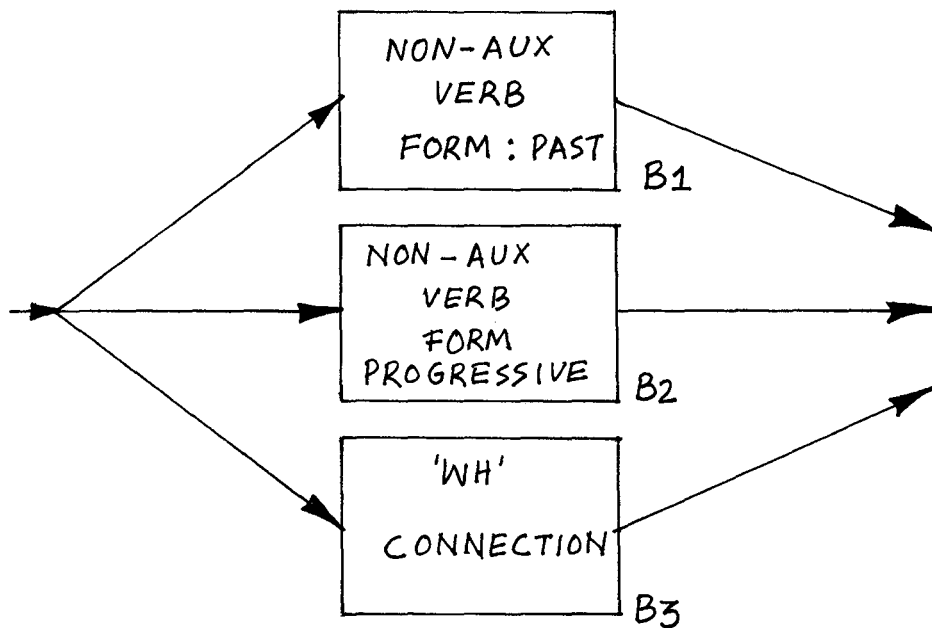


Fig. 4.4

As an illustration, part of the network represented by the block B3 of the figure above is shown in the following figure.

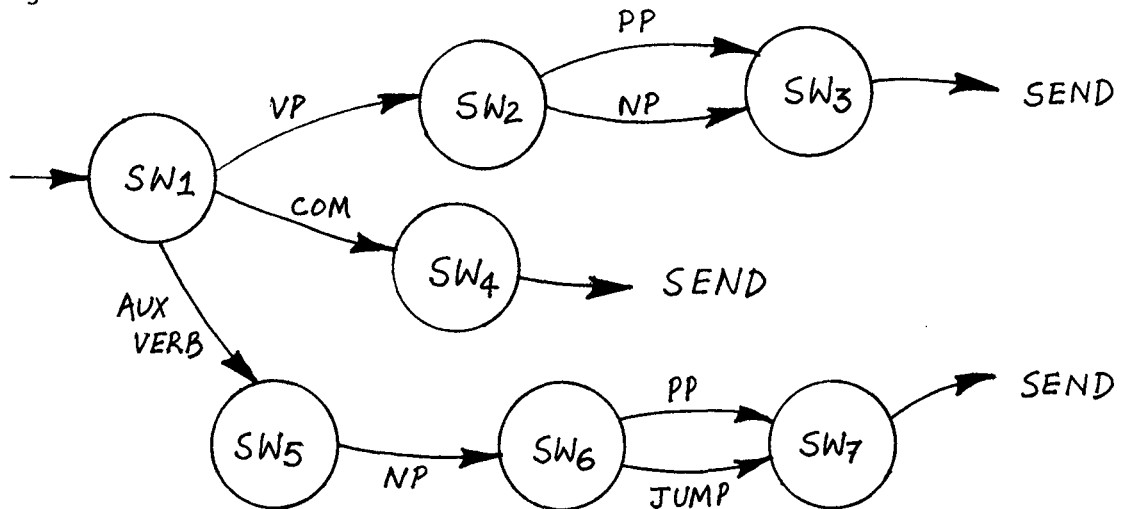


Fig. 4.5

The above network will accept queries of the type:

- s6 --- which is being supplied by Intel ?
- s7 --- which are having a cost of more than 100 ?
- s8 --- who is the supplier of IC8086 in the year 1988 ?

In all the above cases, only the sub clause portion has been shown as examples. The output frame for the sub clause parse includes the following slots.

- (1) connection word and type
- (2) mainverb
- (3) auxiliary verb
- (4) voice

- (5) subject
- (6) direct object
- (7) indirect objects

By 'voice' of a sub clause, we mean the voice that can be associated with the verb group occurring in the sub clause.

4.2.1.3 Support ATNs

Along with the two main ATNs described earlier, we have a lot of minor ATNs used to support them. A list of these ATNs along with their main functions is given below. The diagrams of all these are given in Appendix I.

- (1) WH: recognises the 'wh' element
- (2) PPP : prepositional phrase, seeking NPP in turn
- (3) NPP : noun phrase, allowing proper nouns and numbers
- (4) CNP : conjunctive noun phrase, allowing more than 1 noun as in the example (name address and rating of the best company)
- (5) NP : noun phrase, not allowing proper nouns or numbers
- (6) PP : prepositional phrase, seeking NP in turn
- (7) VP : verb phrase for recognising active and passive verb groups
- (8) COM : for comparators, as in 'rating more than 100.'

4.3 Design and implementation of a lexicon

In this section, we will look at the design and implementation details of the lexicon used by the system.

4.3.1 Introduction

A parser, for parsing an English query needs a dictionary for getting syntactic information about the words in the language, (the subset which is allowed in the system). The collection of words along with the syntactic information constitute the 'lexicon' of a parsing system.

The type and amount of information that is needed in the lexicon depends on the application. For natural language interface to a database, the lexicon would differ from a conventional one (which gives only syntactic information) in that it will provide additional information regarding those words which have a special or restricted meaning in the domain of the database.

4.3.2 Design of the lexicon

We have divided the lexicon, logically, into two portions, (a) core lexicon, and (b) database specific lexicon. In the following sections, we will discuss the two portions in detail.

4.3.2.1 Core-lexicon

This part of the lexicon contains those words whose usage hardly ever changes across different domains. Examples of such words are (i) pronouns like 'this' only the syntactic information, which includes (a) the lexical category and (b) feature dimensions.

The lexical category of a word is what we call 'part of speech' in English grammar, like for the word 'boy', the lexical category is 'noun'. It is possible that a word might have one of several lexical categories depending on the way it is used. For instance, the word 'play' in

'Let us play tennis', has the lexical category 'verb', and in the example

'It was a good play', it has the lexical category 'noun'. A feature is something that we associate with a lexical category. With 'noun' we can associate the features 'number' 'type' and 'gender'. The noun 'boy' will have the following features.

<number> singular

<gender> masculine

<type> common

The number of features required for parsing a sentence depends on the grammar for the language.

4.3.2.2 Database-specific lexicon

This constitutes those words which have specific meaning with respect to the domain. For instance, the word 'offer' in the domain of a 'university database' would invariably mean the act of offering a course, by departments or the teachers in the departments. We see that the word has its meaning restricted. This information will not be used during the parsing stage, but will be pulled out of the lexicon and placed in the output frame of the parser (along with the word). It will be used by the subsequent module.

The 'qframe filling' stage has, to a good extent, influenced the design of the database-specific information part. The type of information that is associated with a word depends on its lexical category, as explained below, for the various cases.

4.3.2.2.1 Noun

If the word is a noun, it can be associated with the database in the following way. It is a synonym of a field

that occurs in one of the relations in the database. We further classify such nouns into the following groups.

entities : these are nouns which occupy a central position of importance, as in the examples : 'teacher', 'supplier', 'part', 'course' etc.,

entity-properties : these are nouns which usually occur as the property of a particular entity of the type described above, as in the examples : 'age', 'cost', etc.,

activity-entities : these are nouns which are typically engaged in some activity; these are basically formed out of verbs, as in the examples: 'teacher' 'supplier'

Note that the group <activity-entity> is a subset of the group <entity>. In our system, we would be primarily interested in differentiating between the second and the third of the groups described above.

Examples of entries for nouns :

```
(part (noun (number singular)) (db (field (part-details name))))
```

```
(teacher (noun (number singular)) (db (activity teach) (field (faculty name ))))
```

The advantage that is gained by storing this information is that questions which have proper nouns can be easily recognised, as in

s10 : what is the cost of UM368 ?

We can associate 'cost' with the entity 'part' and therefore interpret the proper noun 'UM368' as an instance of 'part'.

The specific information about the database is the 'field' specification. This gives the field that is commonly referred to by the word. It is specified as a list of two items, the first, specifying the relation in which it occurs, and the second, the field name itself. This specification is followed throughout the system.

4.3.2.2.2 Verbs

Verbs play an important role in our analysis of the parsed output to produce qframes. Accordingly, we require a lot of information regarding those verbs that have specific roles in the domain of the database. The following information is expected of any verb.

<who> the typically animate agent doing this act

<what> the object involved in the act

<whom> the beneficiary of the act (corresponding to the dative case)

<when> the time when the act was done

<where> the place where the act was done

When any slot is not applicable, it is simply left empty. This set of information, it can be noted, is the 'case' information, regarding the roles of various objects with respect to the verb. Our design of this part of lexicon has been influenced by the concept of 'case frames' [Fillmore '68]. An example of entries for a verb is given below.

```
(teach (verb (transitivity transitive intransitive))
```

```
(db
```

```
(who (field (faculty name)))
```

```
(what (field (course-details name)))
```

```
(whom (field (student name)))
```

```
(when (field (course-off year)))
```

```
))
```

4.3.2.2.3 Adjectives

We consider a subset of the adjectives which have specific meanings in the domain. we consider that adjectives are restrictive components on the nouns they qualify. The set of instances of the noun satisfying the

adjectival qualification will almost invariably be a subset of the set of instances of the noun. This helps us to view an adjective as a selection operator on a set of tuples. The meaning of the adjective decides what sort of a condition it imposes on the noun. We have tried to enlist a set of conditions, using which a good number of adjectives can be specified and also handled by a system. The following keywords can be specified.

- (1) Max <field-name>
- (2) Min <field-name>
- (3) Feq <field-name> constant
- (4) Count <field-name>
- (5) Sum <field-name>
- (6) Avg <field-name>

In the above list, the first three have been implemented.

4.3.2.3 Handling proper nouns

When a word is not found in the lexicon, the system will prompt the user by asking him whether it is a proper noun. If the user responds affirmatively, a temporary entry is made in the lexicon for this word, which will make it available for all subsequent queries in that LISP session. (as an option, the new entries can be saved also). If the

user responds with a 'no', then parse may or may not be possible; the system, whenever it encounters the word again, will assume that it is not a proper noun and will not ask the user.

4.3.3 A note on synonym handling

It is very likely that many words might refer semantically to the same entity within a specific domain of database. In such cases, we do not duplicate the database specific portion of the lexical entry. Instead, we have an indicator saying that it is a synonym of a word which occurs in the lexicon. For instance, the word 'agent', if it is a synonym of the word 'supplier', will have the following entry in the lexicon.

(agent (noun (number singular)) (db (syn supplier)))

4.3.4 A note on irregular forms

Irregular forms of verbs and nouns (like tense variants and plurals) are to be stored separately; no word transformation/generation is done by the system. However, information other than the 'form variance' need not be duplicated, as in the example

(taught (verb (form past) (irregular teach)))

Each node in the 'trie' consists of three elements.

- (1) the letter itself
- (2) any sons
- (3) entries (which will be non nil if a word is formed by the traversal upto this point).

4.3.5.1 Lexical access functions

These functions help to fetch the information for words and assemble them as required by the parser. These functions have been developed on top of the basic 'trie' implementation.

CHAPTER 5

SEMANTIC ANALYSIS

One of the key issues in the process of developing a natural language interface is that of capturing the semantics of the query being posed. When one is interested in designing a portable interface, this becomes all the more important.

There are basically two approaches that can be followed. One is to have a semantic grammar tied strongly to the domain and the other, to have a syntactic grammar followed by a semantic interpretation stage. WE have followed the latter. Syntactic parsing is followed by building of question frames (qframes for short) which capture the semantics of the query. In this chapter, we will discuss the motivation and the design and implementation of qframes.

5.1 Motivation

The need for a semantic structure other than a standard one of the many proposed by linguists arises because of the application. There are quite a number of differences in the outlook between theoretical linguists and computational linguists, some of them fundamental and others more superficial. One of the fundamental differences is that, in general, a theoretical linguist who has arrived at what he

believes to be a satisfactory underlying structure (or a deep structure or a meaning structure) for a set of sentences is usually contented with his results. A computational linguist, on the other hand, will normally not be satisfied until he has been able to use such a structure to carry out some computation and, usually, make a response based on the result of the computation. The means by which this is accomplished varies from person to person (or project to project) in large part because there is little in the way of the theoretical unity in the approaches used by the various groups of computational linguists.

Our methodology in proposing a semantic structure for questions has the following considerations and assumptions.

- (1) We consider a subset of the questions that is relevant in a data-base environment.
- (2) It is an attempt to answer the question "What are the things that generally constitute any such question?".

5.2 Concept of question frames

The following observations can be made of queries in general:

There is always an object, item, person or thing about which (or about whom) the question is asked, or on which the question is based. This, which we call the "primary object

of inquiry" (POI henceforth, for short), can appear in several forms.

-- It can be explicitly named, as in

s1) Does Intel supply the part IC8086?

s2) What is the address of Dr.Saxena?

-- It can appear as an entity name of a class of individuals, with or without instantiation, as in

s3 -What courses does Dr.Karmeshu offer?

s4 -Does the company Intel supply IC8086 ?

s5 -Which company supplies the part UM368?

-- It may not appear explicitly, but only be implied, as in

s6 -Who supplies the part UM368?------(company)

s7 -What does Intel supply ?------(parts.)

Naturally, the POI is the most important component of our semantic structure, the qframe. We consider that in a query, the other information provided contribute to the qframe in relation to the POI in the following way. The information can be classified as follows.

1. Detail :regarding some details of the POI which are required as in

s8 - What is the name, address and rating of the company supplying IC8086?

s9 - What is the cost of IC8086?

2. Action : regarding some action which is performed by the POI or in which the POI is involved either as an active or a passive participant, as in

s10 - Which company is supplying the part IC8086?

s11) - Which courses are taught by Dr. Saxena ?

3. Possession regarding some possession or attribute of the POI or its state of being, as in

s12 - which parts have a cost of more than 500 ?

S13 - which salesmen are in the Delhi region?

4. Adjectival regarding some adjectival qualification attributed to the POI, as in

s14 which imported parts were supplied in the year 1988 ?

Based on the above observations, we have the qframe structure, which, at the top level, looks as follows:

Qframe

1. primary object of inquiry
2. details of POI required
3. details of the action in which POI is involved
4. <possession> and <being> details
5. adjectival qualification

The type of details in each slot will depend on the slot-type. We shall see the details required in the slots 3, 4 and 5.

Action: The various entities involved in the given action have a definite relationship with the verb, called the case relationship. Fillmore identifies several cases including 'agent', 'instrument', 'object', 'dative', 'locative', 'time', instrument against etc., [Fillmore '68]. We have chosen a workable subset of the cases considering the nature of queries usual in database environment. Our action slot is a frame by itself and consists of the following slots:

act the act itself

agent the entity doing the act

object the object involved

dative the beneficiary of the act

active when the act is/was done

Possession The design of the subslots for this was influenced by the necessity to include sentences with comparators. Accordingly we have

feature1 an entity, usually a property of another entity:

feature2 a value of feature1, related to it by

relational-elem the relation between the above two, as = < >etc,.

locative where the entity is/was

time when the entity was around.

Adjectival This appears as a condition restricting the choice for P01.

Apart from the above, we include the type of question also, in the qframe, to help while answering different types

of question. So the frame structure in its expanded form will be as follows.

Qframe

1. question type
2. primary object of inquiry
3. details of POI required
4. action
 - a. act
 - b. agent
 - c. object
 - d. dative
 - e. locative
 - f. time
5. status
 - a. possess
 - i. feature1
 - ii. feature2
 - iii. rel-element
 - b. locative
 - c. time
6. adjectival qualification

5.3 Subframes

The qframe structure discussed in the last section is not sufficient, if we were to consider in our input queries, subclauses also.

For example, consider the query.

s15 What is the name and address of the company supplying the parts which have a cost of more than 100 ?

Clearly, the frame structure of the last section is not alone sufficient, since the semantic information of the subclause (underlined in the above question) cannot go into the 'possess' slot of the qframe (which is a qualifier for the POI only) and the subclause is an example of an anaphoric reference to 'parts' which is not the POI in the question s15 .

To take care of subclauses, we have another structure called 'subframe'. This information will be attached to the appropriate slot in the main qframe after resolving the entity of anaphoric reference. In the above example, the subframe corresponding to the subclause will get attached to the entry in which 'parts' figures, namely 'object' slot in the 'action' slot of the qframe.

Subframe structure:

1. act
2. agent
3. object
4. dative
5. locative
6. time
7. possess
8. adj

examples: The structure of the question frame is given for two example queries.

s16 ► What is the name address and rating of the company supplying the parts which have a cost of more than 500 ?

Only the conceptual information is given in the illustration, for the fillers of the slots, actual syntax follows a consistent pattern of field and value pairs.

Qframe

1. question type : wh
2. POI : company
3. details of POI : (name address rating)
4. action
5. act : supply
 - b. agent : company
 - c. object : part

subframe

- i. act : nil
- ii. agent -
- iii. object -
- iv. dative -
- v. locative -
- vi. time -
- vii. possess -
- a. feature1 : cost
 - b. feature2 : 500
 - c. rel-elem '>'

- viii. adj : nil
- d. dative : nil
- e. locative : nil
- f. time : nil
- 5. status : nil
- 6. adj-qual : nil

Following is another example, this time, without any subframe s17 Who are the suppliers of the part IC8086 ?

Qframe

- 1. question type : wh
- 2. POI : company
- 3. details of POI required nil (default:name)
- 4. action
 - a. act ; supply
 - b. agent : company
 - c. object ; part, IC8086:
 - d. dative : nil
 - e. locative : nil
 - f. time : nil
- 5. status ; nil
- 6. adj-qual ; nil

Actually, the entries for entities like agent or object will not just be the name, but a list of field and value, as in the case of object for the above example which will actually look as follows.

```
((field (part-details part-name)) (value IC8086))
```

The two entries in the list corresponding to the field are, respectively, the relation name and the field name.

5.4 Filling the qframes

In this section we will discuss the rules that have been formulated to arrive at the different components of the frame from the output of the parser. Since we know that syntactic information alone is not always sufficient, we look at the semantic information also. We had noted in the earlier chapter that the lexicon has semantic information in addition to syntactic information. This information will be available in the parsed sentences. The frame filler module looks into both of them to arrive at the qframe structure.

We can consider that there are two logical components and hence the block diagram for this module will look as follows.

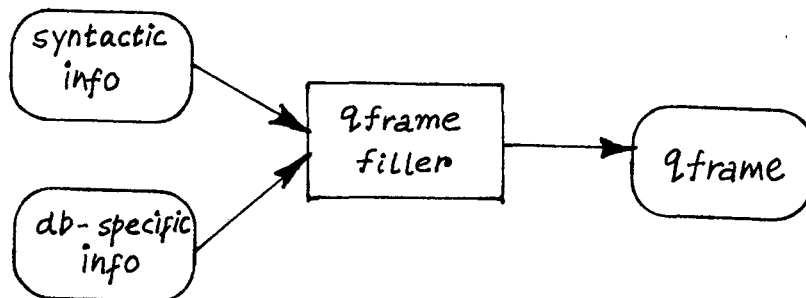


Fig. 5.1

5.4.1 Rules

Of the various slots in the qframe, the POI is the one that requires maximum attention. This is because, in several

cases, it doesn't occur explicitly. To arrive at the rules, we have tried to subdivide the class of input queries into some groups and find a pattern. The following set of rules are applicable to all 'wh' questions.

Before we proceed, we will clarify some terminology used.

The syntactic categories of 'subject', 'direct object', and 'indirect object' are used in the same sense as it is in regular grammar. However, we differ in the following ways for the following two terms.

Action An action verb is a typically non auxiliary verb signifying an action as teach, run, supply, manufacture, offer, etc.,.

Auxiliary An auxiliary verb is one which does not specify any action as in the examples is, was, are, were, etc.,.

Any query should have at least one of either the action verb or the auxiliary verb. A query cannot exist without either of them.

At this point we would like to note that while regular grammar would say that in the following query

s18 who is the supplier of the part IC8086 ?

that 'is' is the mainverb, we would say that it is still an auxiliary verb and note that the action verb is absent.

The way in which the rules are formed on groups of queries is represented as a tree below.

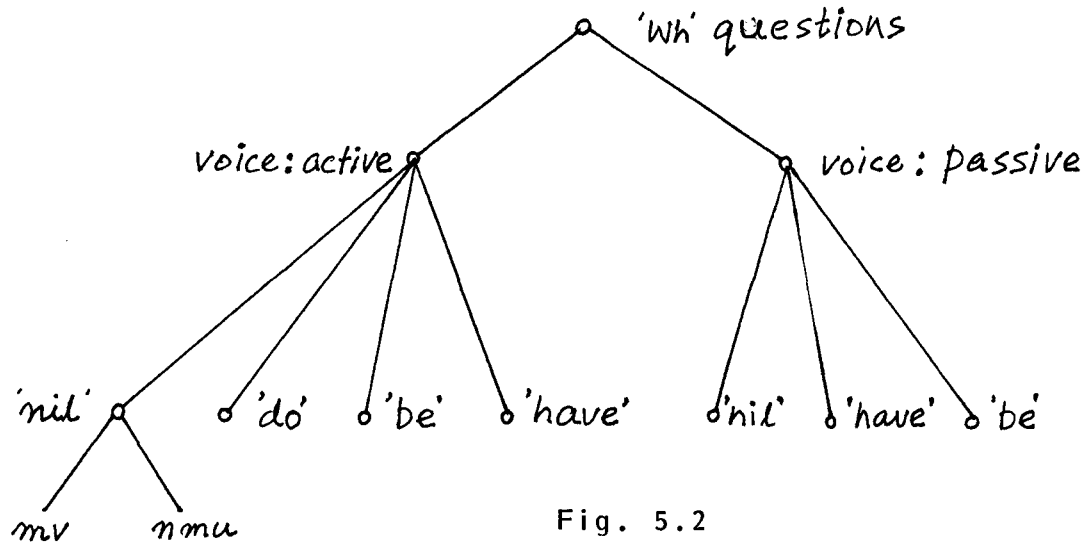


Fig. 5.2

In all the cases, at the leaves, the presence or absence of an action verb will determine, usually, different courses of action. A detailed explanation of the rules followed according to the tree shown above follows.

voice : Active

aux-verb nil

In this case, we wouldn't have been able to parse unless we have an action verb. Hence, if action verb is not present, it is an error condition (which normally shouldn't occur).

examples of queries

s19 who teaches the course 604 ?

s20 who supplies IC8086 ?

s21 which company manufactures the part IC8086 ?

Rules

- (1) 'subject' will be made the POI
- (2) In case 'subject' is absent, (as in s20 and s21 above), the agent of the action verb will be obtained and made the POI.

aux-verb - of type 'do'

In this case, we have rules as follows:

examples of queries

- s22 which courses does Dr. Karmeshu offer ?
- s23 what does Intel supply ?

Rules

- (1) The direct object will be made the POI
- (2) In case the direct object is absent, the object case of the action verb will be made the POI

aux-verb - of type 'have'

In this case, we observe that the absence of the action verb invariably means that the verb of type 'have' is used in the sense of possession. We take action accordingly.

examples of queries

- s24 which company has a rating of more than 5 ?
- s25 who has been supplying the part UM368 ?
- s26 which students have A+ grade in the course 601 ?

Rules

(1) check for the action verb

(a) present

i. the subject of the sentences is made the POI

ii. If the subject is not explicit (as in s25), then the agent of the action verb is made the POI

(b) absent

i. make subject, the POI

ii In case the subject is absent, take the direct object and find, of which entity it is an attribute of ; (this information is available in the lexicon and therefore in the parsed output). Make this the POI. (In the example s26 , course is POI).

<aux-verb>- of type 'be'

In this case too, we will have several subcases, for which different courses of action should be taken. The following list of examples illustrates the variety of questions that are possible.

s27 who is the teacher of the course 604 ?

s28 who is the supplier of IC8086 ?

s29 what is the rating of the company Intel ?

s30 what is the same and address of the company supplying IC8086 ?

s31 who are supplying the imported parts ?

s32 who is teaching cs605 ?

We can observe that the first four queries are similar in their syntactic structure and different from the last two; the difference is that the action verb immediately follows the auxiliary verb 'be' in the last two cases. The information that a query belongs to one group and not the other is available to us, given by the parser which recognises the difference.

Rules

check which group the question falls into

group 1

Here we have two further types, as in the following examples

s33 who is the teacher of ~~cs~~604 ?

s34 what is the rating of Intel ?

In the first case, POI is the teacher, whereas in the second case, it is the 'company Intel', Clearly examination of the syntactic structure alone is not helping us resolve this apparent discrepancy. On closer scrutiny we find that the query s33 can be mapped to the equivalent query.

s35 who teaches the course ~~cs~~604 ?

Where unambiguously, the POI is the 'teacher' (agent of 'teach'). We contend that the presence of a noun like 'teacher' which basically is formed out of a verb, ('teach', in this case) causes this mapping to be allowed. (We had categorised these entities as activity entities, in the

earlier chapter). Such a mapping doesn't exist for the query s34 .

Rules

- (1) We make the noun (if it is an activity entity). the POI
- (2) In the case of this noun not being an activity entity, we check the noun occurring after the preposition 'of'; If it is present, we make it the POI. If only a proper noun is present, that is only an instantiating of the noun has occurred (as in s34), then we find the field of which 'rating' (or whatever is the noun given), is an attribute of and make it the POI.

group 2

In this group we have the action verb and hence this case becomes similar to the case of the auxiliary verb being nil.

Rules

- (1) If the subject is present, it is made the POI
- (2) If the subject is not present then the agent of the action verb is made the POI.

voice passive

In the case of queries addressed in passive voice, only two cases of auxiliary verbs are possible.

(i) have, (ii) be

examples of queries

s37 which parts have been supplied by Intel ?

s38 which courses are being offered in the year 1989 ?

in both the cases, the rules followed are

(1) The direct object is made the POI, if it is present

(2) If it is not present, then the object case of the
action verb is made the POI

5.4.2 Filling the other slots:

(1) question-element

This is straightforward; we just pick up the type of question and fill up, as

(2) details required

If different details of POI has been included in the question then this will be filled up else this slot will remain vacant,. During the query generation stage, if this slot is empty, then the default name for POI will be used.

(3) adjectival qualification

If the POI has an adjectival qualification, then it is entered in the main frame: If there is any adjective qualifying an entity other than the POI, then a 'subframe' is generated for the corresponding entry and the adjectival qualification is entered there.

(4) action slots

The appropriate slots in this category get filled up by referring to the occurrences of various entities in relation to the action verb. (information is got from the db-specific information to find out the entity name if only a proper noun is used). These can also get filled up for questions not having action verbs, but have activity entities as their nouns.

(5) possession slot

When the auxiliary verb is of type 'have' and the action verb is not present, we saw how the POI is formed. At the same time, we also fill the details that would go along with these slots.

5.5 Handling subclauses

As we saw earlier, the inclusion of subclauses might necessitate a 'subframe'. While filling up the subframe itself is very similar to the way in which we filled the main frame, it is important to find out, to which component of the main frame, does this subframe gets attached to. This is non-trivial, since the subclause might involve an reference. We proceed as follows to find the entity of anaphoric reference (EAR for short).

Rules

check for the presence of an action verb,

(a) present

check for the voice part of the group in subclause.

voice active

In this case, the EAR is the agent of the action verb in the subclause, occurring somewhere in the main clause. We get its identity and search it in the main frame slots and then plug in the subframe there. In the following example,

s38 what is the address of the company which is supplying
the part IC8086 ?

the EAR is the same as the POI, and so there is no need to create a separate subframe; the information in the subclause will go into the so far unfilled slots in the main frame itself.

In the following example, however, a new subframe is necessitated.

s39 who is the supplier of the parts which are having a
cost more than 100 ?

Here, the POI is 'supplier', while the EAR is 'part', We will have a 'subframe' attached to the entity 'part'.

voice passive

In this case, the EAR is the object of the action verb in subclause, occurring somewhere in the main clause. In the following example,

s40 what are the prerequisites of the course which is being offered by Dr. Saxena ?

the EAR is 'course'.

(b) absent

(aux-verb) -of type 'have'

We contend that 'have' appears in the form of possession and so we apply the following rule. We get the direct object which is being possessed and find out which entity could possess it. Then we loop into the main frame to find a match for the entity and make it the EAR, as in

s41 who supplies the parts which have a cost of more than 500 ?

here, 'parts' will be made the EAR.

(aux-verb)-of type 'be'

This is a case where the sub clause might be

- (1) a prepositional phrase alone
- (2) a regular clause

For the first case, we match the indirect object component with the slots in the main frame and get the EAR. In the second case, we contend that it must be of a type which can get mapped onto a clause with an action verb and

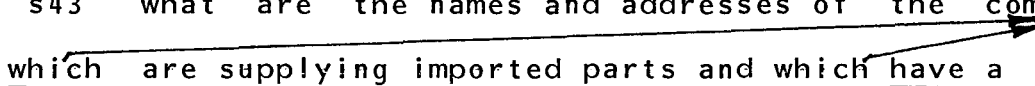
hence we follow the same rules as we would in the similar case in the main clause.

5.5.1 Handling multiple sub clause

Handling multiple sub clause is only an extension of the way in which we handle one sub clause. The main difference is that, for additional sub clause we have to look not only in the slots in the main frame, but also in the subframes generated thus far, to find out the entity of anaphoric reference. The following examples illustrate the idea.

s42 what is the name and address of the company which is supplying imported parts that have a cost of less than 400?

s43 what are the names and addresses of the companies which are supplying imported parts and which have a rating of more than 5 ?



The sub clause are underlined ; The anaphoric references are marked with arrows.

5.6 A note on the implementation

The frames discussed above have been implemented using the DEFSTRUCT macro. The rules for the conversion from the parsed output have been procedurally implemented, as LISP functions.

CHAPTER 6

FORMAL QUERY GENERATION AND DATABASE ACCESS

In this chapter, we will discuss the formal query generation module and the database access module. The former takes the information in the 'qframe' and generates a formal query in relational algebra. The latter evaluates the generated query to get the results.

6.1 Formal query generation

The main idea with which we operate on the 'qframe' is that, it gives a clear picture of what is wanted.

- (1) The entity about which the question is asked.
- (2) Supplementary information which restrict the choice of a value for the entity.

In the 'qframe', the primary object of inquiry is (1), and the other details lead to (2). The generation of a formal query has the above underlying ideas.

In terms of relational algebra, 'restrict' would mean using the 'select' operation, perhaps with the 'join' operation. In our system, we have restricted ourselves to

generate 'select-project-join' expressions. This kind of expressions appears with great frequency. Intuitively, many queries can be viewed as taking an entity (described by the selection clause), connecting it to an entity of another type, perhaps through many relationships (the natural join expresses the connection), and then printing of some attributes of the latter entity (the projection determines the attributes printed). Such expressions are called 'select - project-join' expressions (Ullman '85).

6.1.1 The generation module

We look into the 'qframe' for the three components that lead to restrictions in the choice of answer : action possession and adjectival qualification. We take each component and do the following. We check the various slots and find out for which slot there is an instantiation and generate a list of those field - value pairs. We also make a list of subframe information. The list of case of subframes, we go into the frame and find out the instantiated slots; this would form a restrictor on the component to which the subframe was attached and this would in turn, be a restrictor for the main component.

We generate 'selection' expressions corresponding to the restrictors. 'Join' will be generated whenever two relationships are involved. Thus, we arrive at a complete sub-expressions similarly for the other two.

The slot 'details of object required' in the 'qframe' basically gives us the fields that have to be projected. In the absence of this information, we take up the default information. We generate a 'project' sub-expression corresponding to this list.

Having got the four sub-expressions, we embed them to produce the final query. The embedding function takes care of removing any unnecessary or redundant 'join'. However, no optimisation is done by the system.

An example illustrating the conversion from a 'qframe' to a formal query is given in the following page.

6.2 Database access

We have simulated a relational database schema in memory by considering a relation as a set of tuples (or a list of lists). Adding more tuples to any relation can be done using a function.

The functions 'project', 'select' and 'join' operate on this database. The syntax for the calls of these functions is given below.

- (1) (Project (list of details) relation-name or an expression)
- (2) (select (condition list) relation-name or an expression)
- (3) (join relation-name or an expression relation-name or an expression)

The syntax in BNF is given below

```

<expression> ::= (project <plist> <expression>)
                : (select <clist> <expression>)
                : (join <expression> <expression>)
                : <relation-name>

```

```

<plist> ::= (<list-of-fields>)

```

```

<list-of-fields> ::= <field> : <field> <list-of-fields>

```

```

<clist> ::= (<rel-op> <field> <const>)

```

```

          : (<rel-op2> <field>)

```

```

<rel-op> ::= > : < : = : eq

```

```

<rel-op2> ::= max : min

```

```

<relation-name> ::= one of the valid relation names in
the schema.

```

```

<field> ::= one of the valid field names in any relation
in the schema

```

`<const> :: = a constant, either alphanumeric or numeric`

The join which is implemented is a natural join. Intermediate relations are created during the process of query evaluation. Identification of fields for natural join is achieved by having the first tuple of every relation, either the main relation or one that is created during the query evaluation, to have the various attribute names in the order in which their values appear in the subsequent tuples. However, these intermediate relations are not stored, and hence duplication of effort, if it so happens in a particular query may not be avoided.

6.3 Conclusions

In this chapter we discussed certain aspects of formal query generation and database access. In Appendix 2, along with the relations, a listing of the sample relational database is given.

CHAPTER 7

CONCLUSION

In this chapter, we will summarise the work done, and also detail limitations of the system and scope for the future work.

7.1 Summary

In this endeavour, we have proposed a semantic representation scheme for natural language queries directed towards database. We have also implemented an NL Interface system based on this representation.

Our system consists of three stages. In the first stage, we accept the natural language query and parse it using syntactic information alone. This increases the subset of NL accepted.

In the second stage, we convert the parsed output to the representation mentioned earlier (we call this 'qframe'), using the rules which we have developed. The essential semantics of the query are captured in this representation.

In the third stage, we generate a formal query in relational algebra, as select-project-join expressions, from

this 'qframe'. We have simulated an in-memory database operation functions like select, project and join. The formal query is then evaluated to get the results.

7.2 Limitations and scope for future work

We will look into these aspects with respect to four main areas.

- (1) syntax and linguistic phenomena
- (2) semantic interpretation
- (3) co-operative responses
- (4) automated porting

7.2.1 Syntax and linguistic phenomena

The system presently accepts only 'wh' queries. It can be extended easily to handle 'yes-no' questions and queries which are in the form of a command. Examples of such queries are

- s1 Is Dr. Saxena offering cs604 ?
- s2 Has Intel supplied IC8086 in the year 1988 ?
- s3 List the courses offered by the department of Computer science JNU
- s4 Give me the details of the course cs605

The 'command queries' can be easily mapped into 'wh' questions, and therefore the rules for the conversion to 'qframes' need not be extended very much. In the 'yes-no' type, we can have a marker saying that it is of that type, convert it into a 'wh' question and get the result, and finally compare it with the instance to give an appropriate answer. As an example, s1 can be mapped onto the 'wh' question

s1 who is offering cs604 ?

The 'instance-slot' can be filled with 'Dr. Saxena'. We then would proceed normally. After the result is got, we check with the slot to answer either 'yes' or 'no' depending on whether the answer is the same as in the instance-slot.

Multiple sub clauses and the anaphoric references resulting thereof are handled by the system, but ellipses are not handled. Ellipsis can occur non-grammatically as in

s6 Who taught cs604 in the year 1987 ?

followed by

s7 and cs605

or by

s8 in the year 1988 ?

or it can occur grammatically, as in

s9 who taught in 1988 ?

As a query following s6 (this is usually accepted as an elliptic form). In our system, we have the means to handle ellipsis to some extent. We can retain the 'qframe' corresponding to the old question, and if the system is able to detect an elliptic form, we can superimpose the new 'qframe', in which we have that detail which has varied, on the old 'qframe' and then use it to generate the query. Thus, the system can be extended to handle ellipsis in an elegant manner, though, for a limited variety.

7.2.2 Semantic Interpretation

We have attempted to make our formalism, the 'qframe' as general as possible. However, one can think of other possibilities by which the structure can be extended to include more coverage.

One example is the set of questions which ask for the 'count', 'sum', etc., of a field (which has numeric values). The 'qframe' can be extended to handle this. Another example is regarding questions about the structure of the database itself. The adaptability of 'qframes' for extensions to handle these and other such things can be investigated.

7.2.3 Co-operative responses

Co-operative responses [Kaplan '82] are required if we desire to give the user more information so that he is not misled. As an example, consider the question.

s10 Who got A+ grade in cs606 in 1988 ?

and the two answers that are given ;

(a) NIL

(b) cs606 was not offered in 1988

The answer (b) is an example of a co-operative response. Basically, a system providing co-operative response checks for presuppositions in the question and if they are false or incorrect, responds by pointing them out. For example, in question s10, the presupposition is that the course cs606 was offered in that year. We do not provide presupposition analysis in our system, as to give such a co-operative response.

7.2.4 Automated porting

One of our main objectives was to provide a high degree of portability. We have partly achieved it by restricting the domain - specific information to be present only as part of the lexicon and the schema description. However, we have

not automated the process completely. Though functions have been provided for schema description and lexicon generation, a person with knowledge of LISP is required.

The system can be extended by providing an acquisition module to gather database specific information.

APPENDIX I

PART I : INTRODUCTION TO ATNS

Augmented transition networks and their descendants are currently one of the most common methods of parsing natural language in computer systems. The formalism of ATNs evolves as follows.

Transition networks :

A transition network is a finite state machine which recognises regular expressions, as illustrated below.

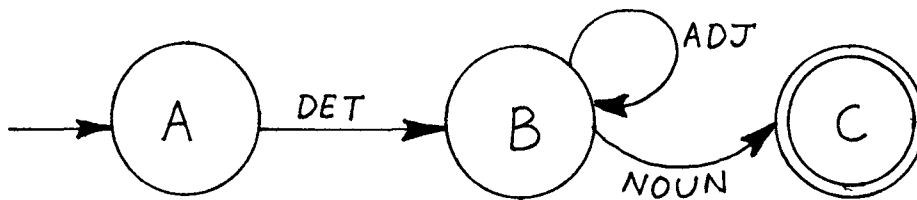


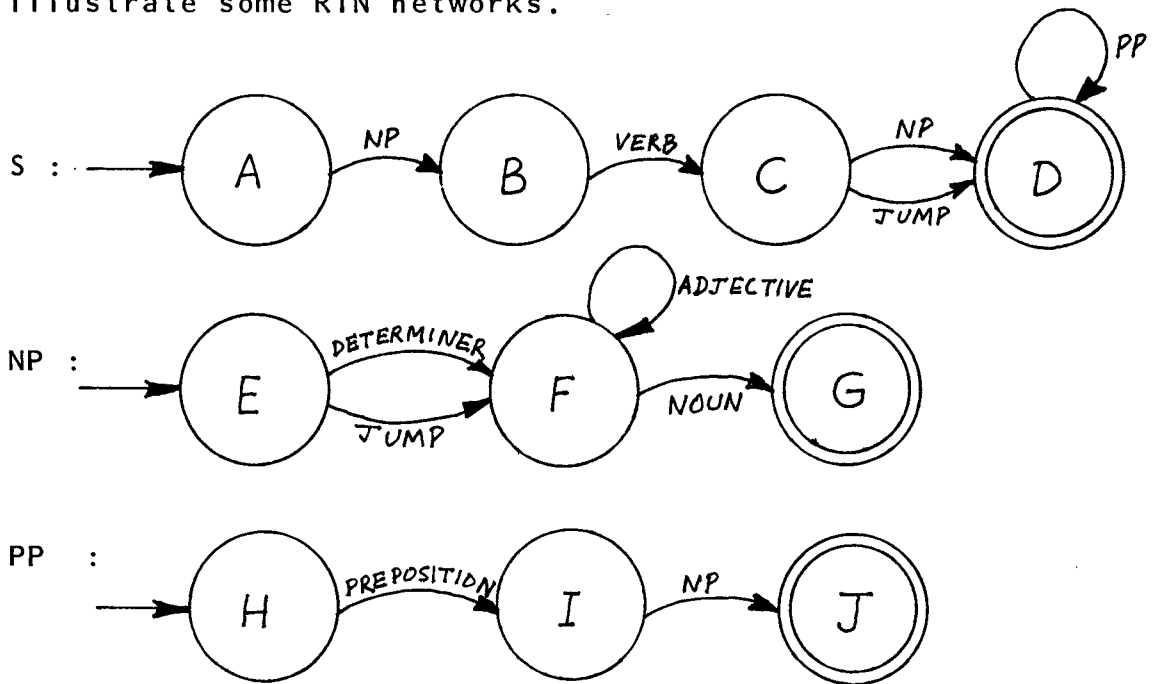
Fig. A-1

The above network will accept sentences of the type [determiner noun] [determiner adjective noun] [determiner adjective adjective noun] and so on.

RTNs :

A recursive transition network is like a simple transition network except that each network has a label and each arc is labelled with a word, a lexical category, or a

syntactic category that is the label of some network in the grammar. The arc containing a syntactic category as a label is traversed by matching a sequence of input symbols to this other network. This network is formally equivalent in power to a context-free grammar. The following diagrams illustrate some RTN networks.



ATNs :

An ATN is similar to an RTN with extensions (or augmentations) as follows.

1. Feature dimensions and role names can be associated with each network. When a phrase is parsed using a network, the register table of the resulting node will contain entries whose keys are the feature dimensions and role names for the network named.

2. Conditions and actions can be specified on the arcs. The conditions will have to be satisfied in addition to the implicit conditions on the arc. The condition will be usually regarding the value of some role register in the network.

3. Initialisation : When an arc that calls for parsing with a network is followed, some of its registers can be initialised to contents derived from the status of parsing so far.

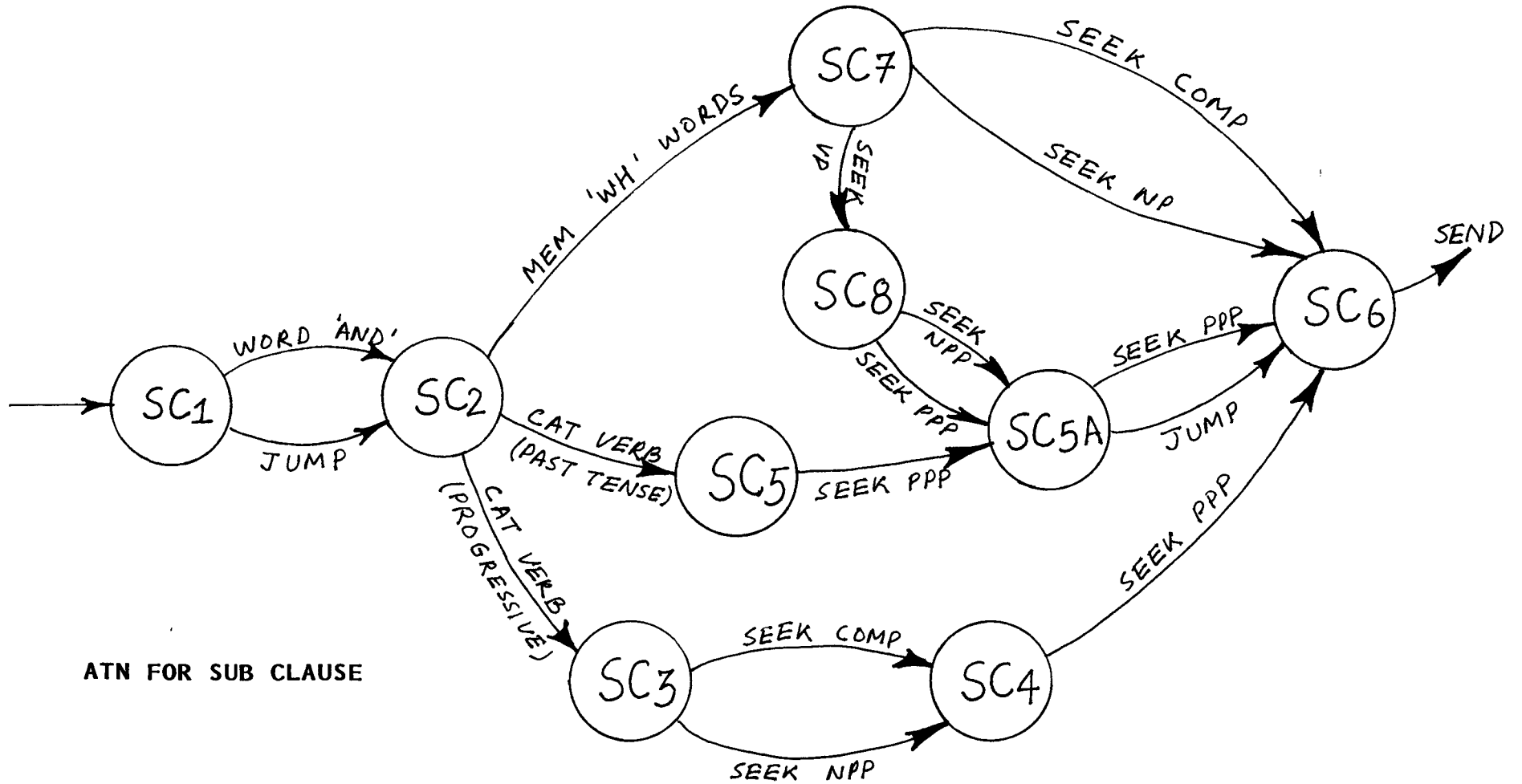
4. Classification of arcs.

<cat> category arc; matched against a single word. Its label is a lexical category.

<seek> a recursive call to a network. Its label is a syntactic category.

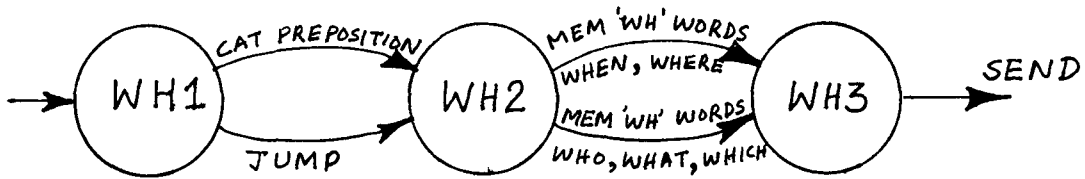
<send> this returns from the called network. It is the opposite of <seek> arc.

<jump> arcs are taken without parsing any element of the input.

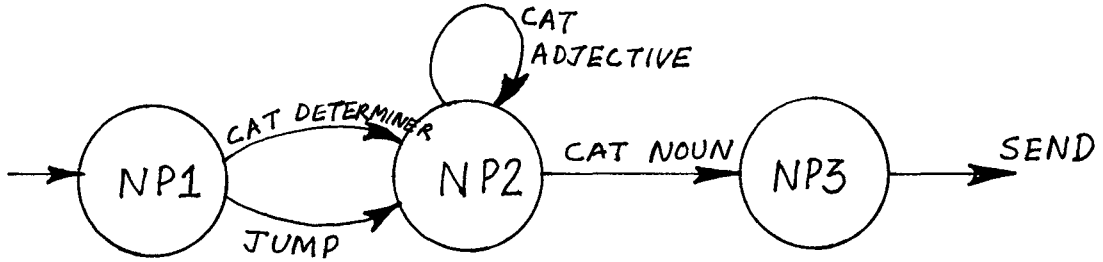


ATN FOR SUB CLAUSE

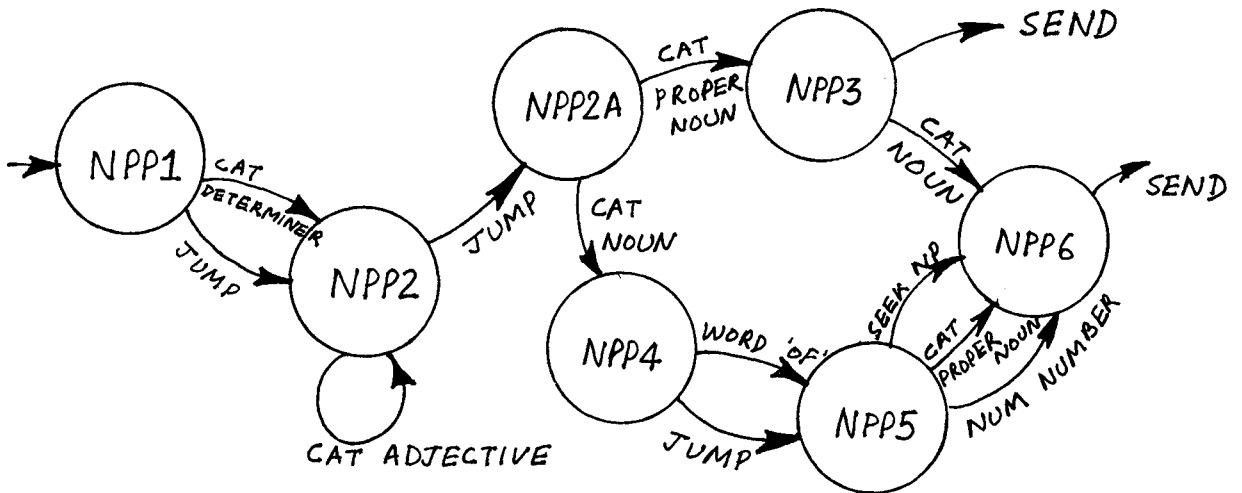
Fig. A-4



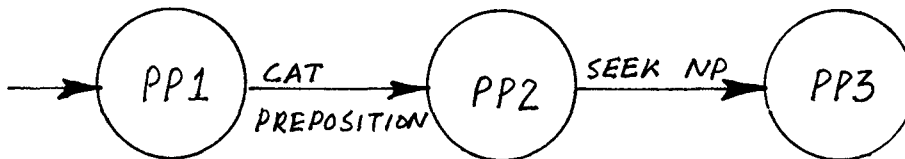
ATN 'WH' (WH - ELEMENT)
Fig. A-5



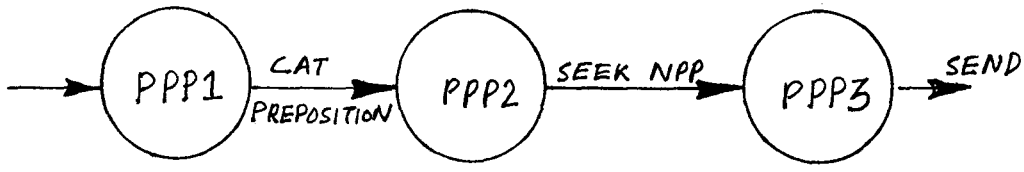
ATN 'NP' (NOUN PHRASE)
Fig. A-6



ATN 'NPP' (NOUN PHRASE including proper nouns & numbers)
Fig A-7

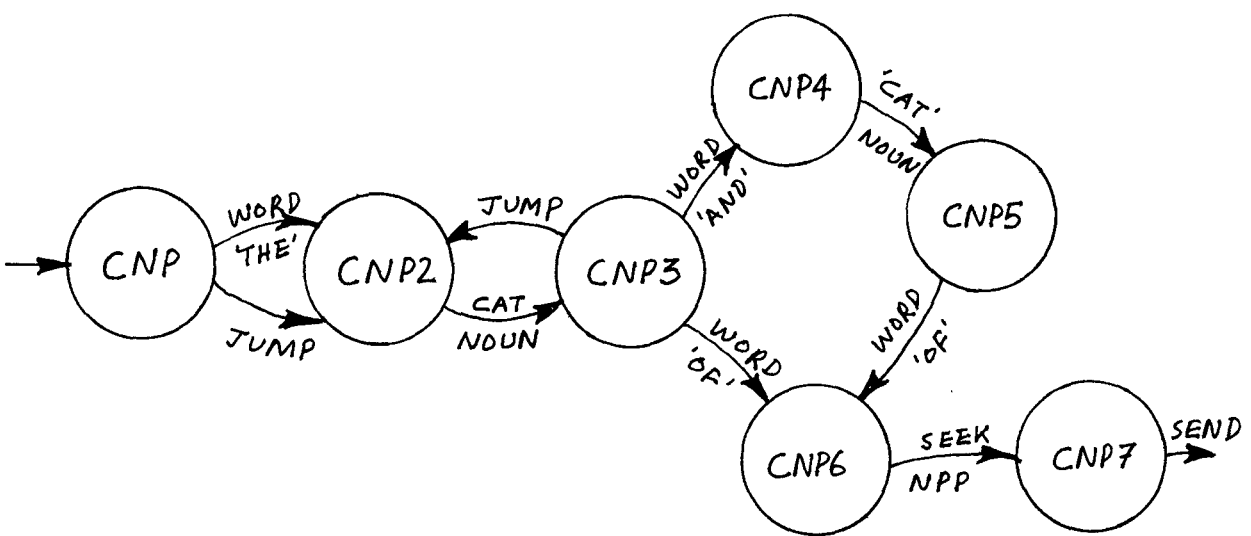


ATN 'PP' (PREPOSITIONAL PHRASE)
Fig A-8



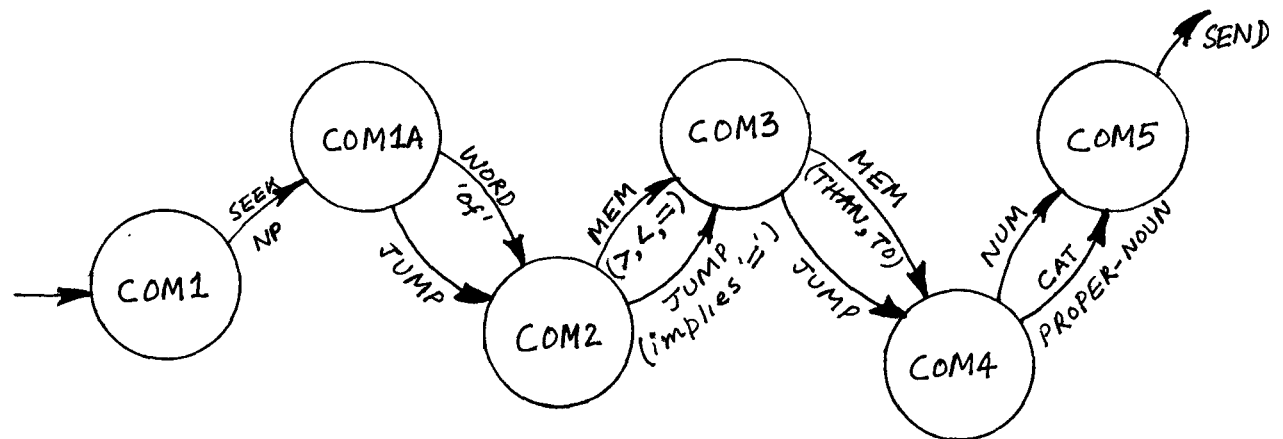
'PPP' (PREPOSITIONAL PHRASE)

Fig A-9



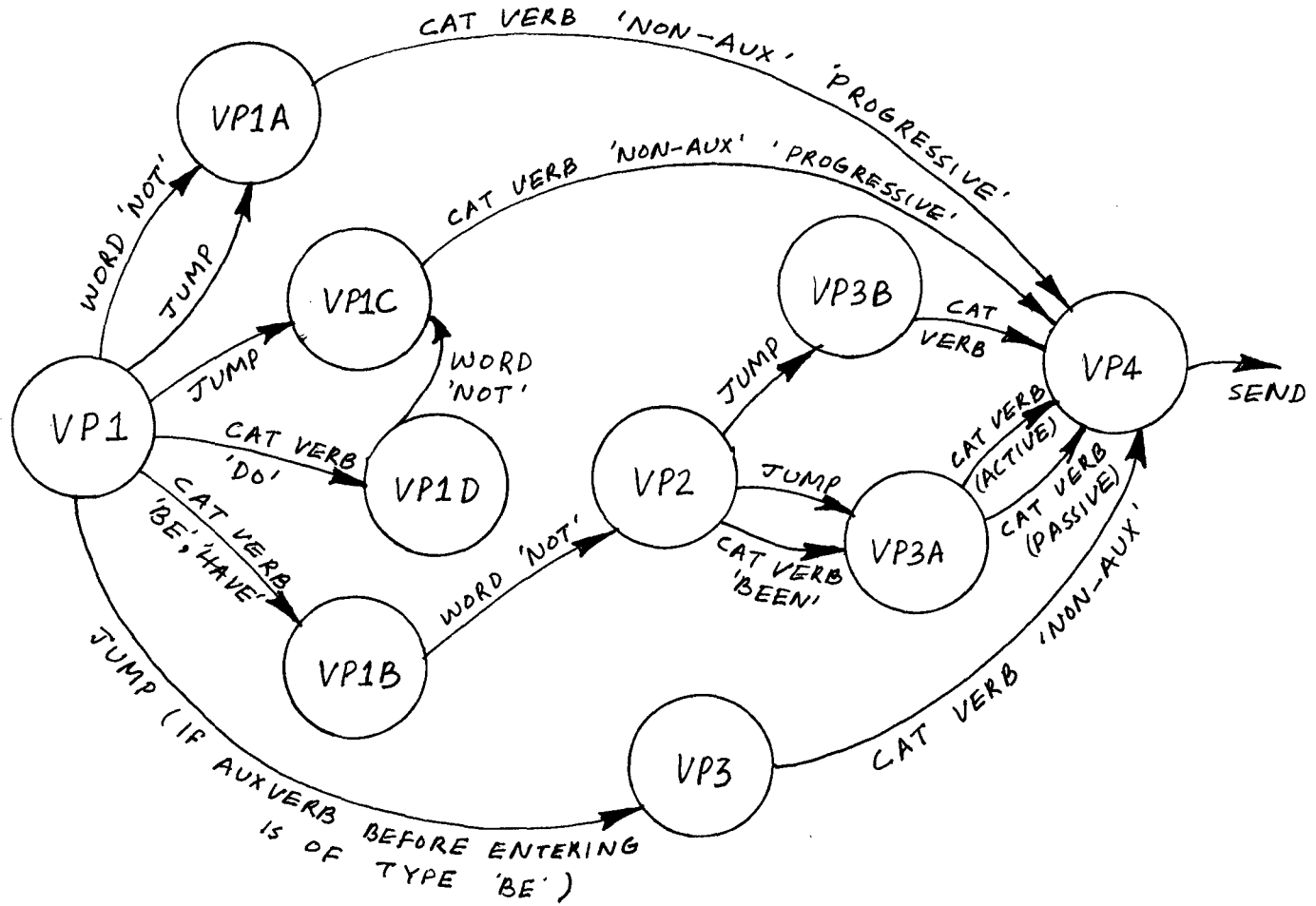
(CONJUNCTIVE NOUN PHRASE)

Fig A-10



(COMPARATORS)

Fig. A-11



'VP' (VERB PHRASES)

Fig. A-12

APPENDIX I

PART 2: PROGRAM LISTING

```

1: (SV ATN1
2:   (MC ((QUESTELEMENT)
3:     (QUESTWORD)
4:     (MAINVERB)
5:     (AUXVERB)
6:     (NEG-TAG)
7:     (VOICE)
8:     (SUBJECT)
9:     (DIRECTOBJECT)
10:    (COMPARATORS)
11:    (INDIRECTOBJECT)
12:    (CHDTRANSITIVITY)
13:    (THERESSET)
14:    (LOCTIME)
15:    (DONE)
16:    (INITPREP)
17:    (CONJFP)
18:    (CONJNP)
19:    (A (SEEK T
20:      (PROG NIL
21:        (SETRV QUESTELEMENT WH)
22:        (SETG 'INITPREP *)
23:        (SETR QUESTWORD *-WORD)
24:        (SETG 'LOCTIME *)
25:      B (WH WH1)
26:      NIL))
27:    (B (CAT (EGRV *-TYPE NONAUX)
28:      (PROG NIL (SETR MAINVERB *) (SETRV VOICE ACTIVE))
29:      D VERB)
30:    (SEEK (NOT (EGRV LOCTIME T))
31:      (SETR DIRECTOBJECT *)
32:      J (NP NP1)
33:      NIL)
34:    (CAT (EGRV *-TYPE BE HAVE )
35:      (PROG NIL (SETRV VOICE ACTIVE) (APPENDR AUXVERB *))
36:      F VERB)
37:    (CAT (EGRV *-TYPE DO)
38:      (PROG NIL (SETRV VOICE ACTIVE) (APPENDR AUXVERB *))
39:      F1 VERB)
40:    (SEEK T (SETR SUBJECT *) C (NP NP1) NIL)
41:    (JUMP T T C))
42:  (F1 (SEEK T (SETR SUBJECT *) L (NPP NPPI) NIL))
43:  (C (SEEK T (SETR AUXVERB *-AUX) P (HAV HAV1) NIL)
44:    (SEEK T
45:      (PROG NIL
46:        (SETG 'MAINVERB *)
47:        (SETG 'CHTRANSITIVITY *)
48:        (SETG 'AUXVERB *)
49:        (SETG 'VOICE *)
50:        (SETG 'NEG-TAG *)

```

```

51:          (COND [(EGRV VOICE PASSIVE)
52:                (SETR DIRECTOBJECT SUBJECT)
53:                (SETRV SUBJECT NIL)]))
54:      D (NP VP1)
55:      (INITR AUXVERB AUXVERB)))
56:  (P (SEEK T (PROG NIL (SETR COMPARATORS *) (SETRV VOICE ACTIVE))
57:      D (COM COM1)
58:      NIL)
59:  (D (JUMP T T E))
60:  (D (SEND (EGRV CNDTRANSITIVITY INTRANSITIVE) NIL)
61:      (SEEK (EGRV VOICE ACTIVE) (SETR DIRECTOBJECT *) E (NPP NPP1) NIL)
62:      (SEEK (EGRV VOICE PASSIVE)
63:          (PROG NIL
64:              (COND [(EGRV *-PREPOSITION BY) (SETR SUBJECT *)]
65:                    [T (SETR INDIRECTOBJECT *)]))
66:          E (PPP PPP1)
67:          NIL)
68:  (JUMP T T E))
69:  (E (SEEK T
70:      (PROG NIL
71:          (COND [(EGRV *-PREPOSITION BY) (SETR SUBJECT *)]
72:                [T (SETR INDIRECTOBJECT *)]))
73:      E (PPP PPP1)
74:      NIL)
75:  (JUMP T T E1))
76:  (E1 (SEND T
77:      (PROG NIL
78:          (COND [(AND (NOT (IS-EMPTY DIRECTOBJECT))
79:                    (NOT (IS-EMPTY SUBJECT)))
80:                (SETRV DONE T)]
81:                [T (SETRV DONE NIL)]))))
82:  (F (SEEK (AND (NOT (EGRV LOCTIME T))
83:              (NOT (EGRV2 AUXVERB-TYPE DO)))
84:      (PROG NIL (SETR SUBJECT *) (SETRV CONJNP T))
85:      H (CNP CNP1))
86:  (SEEK (NOT (EGRV2 AUXVERB-TYPE DO))
87:      (SETR SUBJECT *)
88:  C (NPP NPP1)
89:  NIL)
90:  (CAT (EGRV *-FORM PROGRESSIVE) (SETR MAINVERB *) D VERB)
91:  (SEEK (NOT (EGRV2 AUXVERB-TYPE DO))
92:      (SETR SUBJECT *)
93:      G (NP NP1)
94:      NIL)
95:  (G (SEEK (NOT (AND (EGRV QUESTELEMNT YESNO)
96:                    (EGRV2 AUXVERB-TYPE DO HAVE)))
97:      (SETR DIRECTOBJECT *)
98:      H (CPP CPP1)
99:      NIL)
100:*  (JUMP (OR (NOT (IS-EMPTY MAINVERB)) (EGRV THERESET T))

```

*

```

101:          (PRG NIL
102:            (COND [(AND [IS-EMPTY DIRECTOBJECT]
103:                      [(IS-EMPTY INDIRECTOBJECT)]
104:                      [SETRV DONE NIL])
105:                  (T (SETRV DONE T))] T ))
106:      (H (SEND T (SETRV DONE T)))
107:      (I (SEND T
108:          (PRG NIL
109:            (COND [(AND [IS-EMPTY DIRECTOBJECT]
110:                      [IS-EMPTY INDIRECTOBJECT]]
111:                  (SETRV DONE NIL))
112:                  (T (SETRV DONE T))))))
113:      (J (CAT (EGRV *-TYPE DO)
114:            (PRG NIL (SETRV VOICE ACTIVE) (SETR AUXVERB *)
115:                  K VERB))
116:      (K (SEEK T (SETR SUBJECT *) L (NPP NPP1) NIL))
117:      (L (CAT (EGRV *-TYPE MONAUX) (SETR MAINVERB *) M VERB))
118:      (M (SEEK T (SETR INDIRECTOBJECT *) N (PPP PPP1) NIL)
119:      (JUMP T T N))
120:      (N (SEND T (SETRV DONE T))))))
121: (DV ATN2
122:   (WH (INITPREP) (LOCTIME) (WORD) (PREPOSITION))
123:   (WH1 (CAT T (PRG NIL (SETRV INITPREP T) (SETR PREPOSITION *)
124:              WH2 PREPOSITION))
125:   (JUMP T (SETRV INITPREP NIL) WH2))
126:   (WH2 (MEM (IS-EMPTY INITPREP)
127:            (SETR WORD *-WORD)
128:            WH3 (WHAT WHO WHICH WHEN WHERE WHOM))
129:   (MEM T (PRG NIL (SETR WORD *) (SETRV LOCTIME T))
130:   WH3 (WHEN WHERE WHOM)))
131:   (WH3 (SEND T T))))
132: (DV ATN11
133:   (PPP (PREPOSITION) (PREP-OBJECT))
134:   (PPP1 (CAT T (SETR PREPOSITION *-WORD) PPP2 PREPOSITION))
135:   (PPP2 (SEEK T (SETR PREP-OBJECT *) PPP3 (NPP NPP1) NIL))
136:   (PPP3 (SEND T T))))
137: (DV ATN4
138:   (CNP ((HEAD) (DETERMINER) (SUBDETAILS) (HOLDINGPREP))
139:   (CNP1 (WORD T (SETR DETERMINER *) CNP2 THE) (JUMP T T CNP2))
140:   (CNP2 (CAT T (APPENDR SUBDETAILS *) CNP3 NOUN))
141:   (CNP3 (JUMP T T CNP2)
142:   (WORD T T CNP4 AND)
143:   (WORD T (SETR HOLDINGPREP *) CNP6 OF))
144:   (CNP4 (CAT T (APPENDR SUBDETAILS *) CNP5 NOUN))
145:   (CNP5 (MEM T (SETR HOLDINGPREP *) CNP6 (OF IN)))
146:   (CNP6 (SEEK T (SETR HEAD *) CNP7 (NPP NPP1) NIL))
147:   (CNP7 (SEND T T))))
148: (DV ATN5
149:   (NPP ((DETERMINER)
150:        (HEAD)

```

```

151:      (DETAIL)
152:      (INSTANCE)
153:      (DESCRIBER)
154:      (HOLDINGPREP))
155:      (NPP1 (CAT T (SETR DETERMINER *) NPP2 DETERMINER)
156:           (JUMP T T NPP2))
157:      (NPP2 (CAT T (APPENDR DESCRIBER *) NPP2 ADJECTIVE)
158:           (JUMP T T NPP2A))
159:      (NPP2A (CAT T (SETR HEAD *) NPP4 NOUN)
160:            (NUM T (SETR INSTANCE *) NPP3)
161:            (CAT T (SETR INSTANCE *) NPP3 PROPER-NOUN))
162:      (NPP3 (CAT T (SETR HEAD *) NPP6 NOUN) (SEND T T))
163:      (NPP4 (MEM T (SETR HOLDINGPREP *-WORD) NPP5 (OF IN))
164:            (JUMP T T NPP5)
165:            (SEND T T))
166:      (NPP5 (SEEK T (PROG NIL (SETR DETAIL HEAD) (SETR HEAD *)))
167:            NPP5 (NP NP1)
168:            NIL)
169:            (NUM T (SETR INSTANCE *) NPP6)
170:            (CAT T (SETR INSTANCE *) NPP6 PROPER-NOUN))
171:      (NPP6 (SEND T T)))
172: (DV ATN6
173:      (NP ((DETERMINER) (HEAD) (DESCRIBER) (NUMBER)))
174:      (NP1 (CAT T (PROG NIL (SETR DETERMINER *)
175:                    (SETR NUMBER *-NUMBER))
176:            NP2 DETERMINER)
177:            (JUMP T T NP2))
178:      (NP2 (CAT T (APPENDR DESCRIBER *) NP2 ADJECTIVE)
179:            (CAT T (PROG NIL (SETR NUMBER *-NUMBER)
180:                    (SETR HEAD *))) NP3 NOUN))
181:      (NP3 (SEND T T )))
182: (DV ATN7
183:      (PP ((PREPOSITION) (PREP-OBJECT)))
184:      (PP1 (CAT T (SETR PREPOSITION *) PP2 PREPOSITION))
185:      (PP2 (SEEK T (SETR PREP-OBJECT *) PP3 (NP NP1) NIL))
186:      (PP3 (SEND T T )))
187: (DV ATN8
188:      (COM ((FEATURE1) (COMPSIGN) (FEATURE2)))
189:      (COM1 (SEEK T (SETR FEATURE1 *) COM1A (NP NP1) NIL))
190:      (COM1A (WORD T T COM2 OF) (JUMP T T COM2))
191:      (COM2 (MEM T (SETR COMPSIGN *)
192:                 COM3 (EQUAL GREATER LESSER MORE LESS))
193:            (JUMP T (SETRV COMSIGN EQUALS) COM4))
194:      (COM3 (MEM (OR (NOT (EQRV COMSIGN EQUAL)) (EQRV *-WORD TO))
195:              T COM4 (TS THAN)))
196:      (COM4 (CAT T (SETR FEATURE2 *) COMPS PROPER-NOUN)
197:            (NUM T (SETR FEATURE2 *) COMPS))
198:      (COM5 (SEND T T )))
199: (DV ATN10
200: *      (VP ((MAINVERB) (AUXVERB) (NEG-TAG) (VOICE) (CMDTRANSITIVITY))

```

281: (VP1 (WORD T (SETRV NEG-TAG ON) VP1A NOT)
282: (CAT (EGRV *-TYPE DO) (SETR AUXVERB *) VP1D VERB)
283: (JUMP T T VP1A)
284: (CAT (EGRV *-TYPE BE HAVE) (SETR AUXVERB *) VP1B VERB)
285: (JUMP (EGRV2 AUXVERB-TYPE BE) T VP3)
286: (JUMP T T VP1C))
287: (VP1B (WORD T (SETRV NEG-TAG ON) VP2 NOT) (JUMP T T VP2))
288: (VP1A (CAT (AND [EGRV *-TYPE NONAUX] [EGRV *-TYPE PROGRESSIVE])
289: (PRG NIL
290: (SETRV VOICE ACTIVE)
291: (SETR MAINVERB *)
292: (SETR CMDTRANSITIVITY *-TRANSITIVITY))
293: VP4 VERB))
294: (VP1D (WORD T (SETRV NEG-TAG ON) VP1C NOT))
295: (VP1C (CAT (EGRV *-TYPE NONAUX)
296: (PRG NIL
297: (SETRV VOICE ACTIVE)
298: (SETR MAINVERB *)
299: (SETR CMDTRANSITIVITY *-TRANSITIVITY))
300: VP4 VERB))
301: (VP2 (WORD (EGRV2 AUXVERB-TYPE BE) T VP3 BEING)
302: (WORD (EGRV2 AUXVERB-TYPE HAVE) T VP3A BEEN)
303: (JUMP (EGRV2 AUXVERB-TYPE HAVE) T VP3B)
304: (JUMP (EGRV2 AUXVERB-TYPE BE) T VP3A))
305: (VP3 (CAT (AND [EGRV *-TYPE NONAUX] [EGRV *-FORM PAST])
306: (PRG NIL
307: (SETRV VOICE PASSIVE)
308: (SETR MAINVERB *)
309: (SETR CMDTRANSITIVITY *-TRANSITIVITY))
310: VP4 VERB))
311: (VP3A (CAT (EGRV *-FORM PAST)
312: (PRG NIL
313: (SETR MAINVERB *)
314: (SETR CMDTRANSITIVITY *-TRANSITIVITY)
315: (SETRV VOICE PASSIVE))
316: VP4 VERB))
317: (CAT (EGRV *-FORM PROGRESSIVE)
318: (PRG NIL
319: (SETR MAINVERB *)
320: (SETR CMDTRANSITIVITY *-TRANSITIVITY))
321: (SETRV VOICE ACTIVE))
322: VP4 VERB))
323: (VP3B (CAT (EGRV *-FORM PAST)
324: (PRG NIL
325: (SETR MAINVERB *)
326: (SETRV VOICE ACTIVE)
327: (SETR CMDTRANSITIVITY *-TRANSITIVITY))
328: VP4 VERB))
329: (VP4 (SEND T T)))
250: *(DV ATH9

251: (CPP (CONJPP) (FIRSTPREP) (OBJECT) (HOLDINGPREP) (INSTANCES))
252: (CPP1 (CAT T (SETR FIRSTPREP *) CPP2 PREPOSITION))
253: (CPP2 (WORD T T CPP3 THE) (JUMP T T CPP3) (JUMP T T CPP5))
254: (CPP3 (CAT T (SETR OBJECT *) CPP4 NOUN))
255: (CPP4 (JUMP T T CPP5))
256: (CAT T (SETR HOLDINGPREP *) CPP5 PREPOSITION)
257: (SEND T T))
258: (CPP5 (CAT T (APPENDR INSTANCE *) CPP6 PROPER-NOUN))
259: (CPP6 (JUMP T T CPP5) (SEND T (SETRV CONJPP T))))
260: (DV ATN12
261: (SC ((TYPE)
262: (CONNECTWORD)
263: (COMPNP)
264: (MAINVERB)
265: (AUXVERB)
266: (VOICE)
267: (SUBJECT)
268: (DIRECTOBJECT)
269: (INDIRECTOBJECT)
270: (SC1 (WORD T T SC1A AND) (JUMP T T SC1A))
271: (SC1A (WORD T T SC2A ARE) (JUMP T T SC2))
272: (SC2A (CAT (AND (EGRV *-TYPE HAVE) (EGRV *-FORM PROGRESSIVE)
273: (SETR AUXVERB *)
274: SC7A VERB))
275: (SC2 (CAT (AND (EGRV *-TYPE NONAUX) (EGRV *-FORM PROGRESSIVE))
276: (PRG NIL
277: (SETR MAINVERB *)
278: (SETRV TYPE VERB)
279: (SETR CONNECTWORD *-WORD)
280: (SETRV VOICE ACTIVE))
281: SC3 VERB)
282: (CAT (AND (EGRV *-TYPE NONAUX) (EGRV *-FORM PAST))
283: (PRG NIL
284: (SETR MAINVERB *)
285: (SETRV TYPE VERB)
286: (SETR CONNECTWORD *-WORD)
287: (SETRV VOICE PASSIVE))
288: SC5 VERB)
289: (MEN T (PRG NIL (SETRV TYPE WHQS)
290: (SETR CONNECTWORD *-WORD))
291: SC7 (WHICH WHO WHOSE WHERE))
292: (JUMP T T SC7)
293: (SC3 (SEEK T (SETR COMPNP *) SC4 (COMP COMPI) NIL))
294: (SEEK T (SETR DIRECTOBJECT *) SC5A (MPP MPP1) NIL))
295: (SC5 (SEEK T
296: (PRG NIL
297: (COND [(EGRV *-PREPOSITION BY) (SETR SUBJECT *)]
298: IT (SETR INDIRECTOBJECT *)))
299: SC5A (PPP PPP1))
300: NIL))

E


```

301:      (SC5A (SEEK T
302:          (PROG NIL
303:              (COND ((EGRV *-PREPOSITION BY)
304:                  (SETR SUBJECT *)))
305:                  (T APPENDR INDIRECTOBJECT *)))
306:          SCS (PPP PPP1)
307:              NIL)
308:      (JUMP T T SC6))
309:      (SC4 (SEEK T (PROG NIL (SETR DIRECTORJECT SUBJECT)
310:                          (SETR SUBJECT *)))
311:          SC6 (PPP PPP1)
312:              NIL))
313:      (SC7 (SEEK T (SETR AUXVERB *-AUX) SC7A (HAV HAV1) NIL)
314:          (SEEK T (PROG NIL (SETG 'MAINVERB *) (SETG AUXVERB *)
315:                          (SETG 'VOICE *)))
316:          SC8 (VP VP1)
317:              NIL)
318:          (CAT (EGRV *-TYPE BE DO HAVE) SC6 (NP NP1) NIL))
319:      (SC8 (SEEK (EGRV VOICE ACTIVE)
320:          (SETR DIRECTORJECT *)
321:          SC5A (NPP NPP1)
322:              NIL)
323:          (SEEK T
324:              (PROG NIL
325:                  (COND ((EGRV VOICE PASSIVE)
326:                      (SETR DIRECTORJECT SUBJECT)
327:                      (SETR SUBJECT *))))
328:              SC5A (PPP PPP1)
329:                  NIL)
330:          (SC7A (SEEK T (SETR COMPNF *) SC6 (COM COM1) NIL))
331:          (SC6 (SEND T T)))
332: (DV ATN3
333: (NMP NIL (NMP1 (CAT T T NMP2 DETERMINER) (JUMP T T NMP2))
334: (NMP2 (MEM T T NMP3 (LIST DETAILS)))
335: (NMP3 (WORD T T NMP4 OF))
336: (NMP4 (SEND T T)))
337: (DV ATN13
338: (COMP ((FEATURE1) (AUX) (COMPSIGN) (FEATURE2))
339: (COMP1 (CAT (EGRV *-TYPE BE) T COMP1Z VERB))
340: (COMP1Z (CAT (EGRV *-TYPE HAVE) (SETR AUX *) COMP1A VERB))
341: (COMP1A (CAT T T COMP2 DETERMINER) (JUMP T T COMP2))
342: (COMP2 (CAT T (SETR FEATURE1 *) COMP2A NOUN))
343: (COMP2A (WORD T T OF) (JUMP T T COMP3))
344: (COMP3 (MEM T (SETR COMPSIGN *) COMP4 (EQUAL GREATER LESSER MORE LESS))
345: (JUMP T SETRV COMPSIGN EQUALS) COMP5))
346: (COMP4 (MEM (OR (NOT (EGRV COMPSIGN EQUALS)) (EGRV2 *-WORD TO))
347: T COMP5 (TO THAN)))
348: (COMP5 (CAT T (SETR FEATURE2 *) COMP6 PROPER-NOUN)
349: (NUM T (SETR FEATURE2 *) COMP6))
350: (COMP6 (SEND T T)))

```

```

351: (DV ATN6
352:   (QRY ((MAINCLAUSE) (SUBCLAUSES)))
353:   (QRY1 (SEEK T (SETR MAINCLAUSE *) QRY2 (MC A) NIL))
354:   (QRY2 (SEEK T (APPENDR SUBCLAUSES *) QRY2 (SC SC1) NIL)
355:         (SEND T T)))
356: (DV ATN14
357:   (HAV ((AUX)))
358:   (HAV1 (CAT (EQRV *-TYPE BE) T HAV2 VERB)
359:         (CAT (AND (EQRV *-TYPE HAVE) (NOT (EQRV *-FORM PROGRESSIVE)))
360:              (SETR AUX *)
361:              HAV3 VERB))
362:   (HAV2 (CAT (AND (EQRV *-TYPE HAVE) (EQRV *-FORM PROGRESSIVE))
363:          (SETR AUX *)
364:          HAV3 VERB))
365:   (HAV3 (SEND T T)))

```

4

APPENDIX I

PART III

EXAMPLES WITH PARSER OUTPUT

(WHICH PART IS HAVING A COST OF MORE THAN 100)

```
(MAINCLAUSE ((DONE NIL)
  (VOICE ACTIVE)
  (COMPARATORS ((FEATURE2 ((WORD 100)))
    (COMPSIGN ((WORD MORE)))
    (FEATURE1 ((HEAD ((NUMBER SINGULAR)
      (DB (FIELD (PART-DETAILS COST))
        (ENTITY-PROPERTY PART))
        (WORD COST)))
      (NUMBER SINGULAR)
      (DETERMINER ((NUMBER SINGULAR)
        (WORD A))
        (DESCRIBER NIL))))))
  (AUXVERB ((TYPE HAVE)
    (NUMBER SINGULAR PLURAL)
    (FORM PROGRESSIVE)
    (WORD HAVING)))
  (SUBJECT ((HEAD ((NUMBER SINGULAR)
    (DB (FIELD (PART-DETAILS PART-NAME))
      (ENTITY-PROPERTY SUPPLIER))
      (WORD PART)))
    (NUMBER SINGULAR)
    (DETERMINER NIL)))
    (DESCRIBER NIL)))
  (LOCTIME NIL)
  (QUESTWORD WHICH)
  (INITPREP NIL)
  (QUESTELEMENT WH)
  (MAINVERB NIL)
  (NEG-TAG NIL)
  (DIRECTOBJECT NIL)
  (CNDTRANSITIVITY NIL)
  (THERESET NIL)
  (CONJPP NIL)
  (CONJNP NIL)))
(SUBCLAUSE NIL)
```

(WHO IS THE SUPPLIER OF IC8086)

```
(MAINCLAUSE ((DONE T)
  (CONJNP T)
  (SUBJECT ((HEAD ((INSTANCE ((PROPER-NOUN NIL)
    (WORD IC8086)))
    (DETERMINER NIL)
    (HEAD NIL)
    (DETAIL NIL)
```

(DESCRIPTOR NIL)
(HOLDINGPREP NIL)))
(HOLDINGPREP ((WORD OF)))
(SUBDETAILS ((NUMBER SINGULAR)
(DB (FIELD (SUPPLIER SUPPLIER-NAME)))
(WORD SUPPLIER)))
(DETERMINER ((WORD THE))))))
(AUXVERB ((TYPE BE)
(NUMBER SINGULAR)
(FORM PRESENT)
(WORD IS)))
(VOICE ACTIVE)
(LOC TIME NIL)
(QUESTWORD WHO)
(INITPREP NIL)
(QUESTELEMENT WH)
(MAINVERB NIL)
(NEG-TAG NIL)
(DIRECTOBJECT NIL)
(COMPARATORS NIL)
(INDIRECTOBJECT NIL)
(CNDTRANSITIVITY NIL)
(THERESET NIL)
(CONJPP NIL)))
(SUBCLAUSES NIL)

A>

PART IV

Examples of parsing

The nodes referred to here are as given in the ATN diagrams in the previous sections.

Mainclauses (refer to the ATN of figure A-3)

- 1) Who supplies IC 8086 ?
 <path >(mainclause A B D (noun phrase) E E1); succeeds.
- (2) Who is the supplier of the part IC8086 ?
 path (mainclause A B F (conjunctive noun phrase) H
 (jump) I); succeeds
- (3) What is the name address and rating of the agent Uptron?
 < path > (mainclause A B F (conjunctive noun phrase) H
 (jump) I); succeeds
- (4) Which agent is having a rating of more than 8 ?
 < path > (mainclause A B (verb phrase with 'have') P
 (comparators) Q (jump) E (jump) E1); succeeds.
- (5) Who is having shares more than 50 ?
 < path > (mainclause A B (verb phrase with 'have') P
 (comparators) Q (jump) E (jump) E1); succeeds.

For complex sentences : (refer to the ATNs of figures A- 3 and A-4)

(6) Who is supplying the parts which are being manufactured by Intel?

< path > (mainclause A B F D (noun phrase) E (jump) E1 (subclause SC1 (jump) SC2 SC7 SC8 (prepositional phrase) SC5A (jump) SC6)); succeeds.

APPENDIX II

PART I

Relational model

The mathematical concept underlying the relational model is the set-theoretic relation, which is a subset of a cartesian product of a list of domains. A domain is simply a set of values. The cartesian product of domains D_1, D_2, \dots, D_k , written as $D_1 \times D_2 \times \dots \times D_k$, is the set of all k -tuples (v_1, v_2, \dots, v_k) such that v_1 is in D_1 , v_2 is in D_2 and so on.

A relation is any subset of the cartesian product of one or more domains. As far as databases are concerned, we shall assume that a relation is finite unless stated otherwise.

The members of a relation are called tuples. Each relation that is a subset of $D_1 \times D_2 \times \dots \times D_k$ is said to have 'arity' k ; It helps to view a relation as a table, where each row is a tuple and each column corresponds to one component. The columns are often given names, called 'attributes'. The set of attribute names for a relation is called 'relation scheme'.

The collection of relation schemes used to represent information is called a relational database scheme and the current values of the corresponding relations is called the

'relational database'. We are, of course, at a liberty to create relations with any set of attributes as a relation scheme, and we can place any interpretation we wish on the tuples.

Relation Algebra

The notation for expressing queries is usually the most significant part of a data manipulation language. The non-query aspects of a relational data manipulation language, or a query language, are often straightforward, being concerned with the insertion, deletion and modification of tuples. On the other hand, queries, which in the most general case are arbitrary functions applied to a relation, often use a rich, high-level language for their expression.

The relational algebra language was proposed by Codd (1972) to represent the minimum capability of any reasonable query language using the relational data model.

Operations in relational algebra.

- (1) Union: The union of relations R or S , denoted $R \cup S$, is the set of tuples that are in R or S both. (The two relations should be of the same arity).
- (2) Set difference: The difference of relations R denoted by $R - S$, is the set of tuples in R and not in S .

- (3) Cartesian product: Let R and S be relations of arity K_1 and K_2 . Then, $R \times S$, the cartesian product of R and S is the set of $(K_1 + K_2)$ tuples, where first K_1 components form a tuple in R and whose last K_2 components form a tuple in S .
- (4) Projection: If R is a relation of arity K , we let $P_{i_1, i_2, \dots, i_K}(R)$, where the i_j 's are distinct integers in the range 1 to k , denote the projection of R onto the components $i_1 i_2 \dots i_k$. If R has attribute names, then we may substitute attribute names for component numbers.
- (5) Selection: The selection $S_f(R)$ is the set of tuples t in R such that the condition in 'f' becomes true. 'f' is a formula involving operands that are constants or component numbers (or attribute names), arithmetic expressions, and comparisons, and comparators and logical operators. The most important of which is the natural join.
- (6) Natural Join: The Natural join, written as $R \bowtie S$, is applicable only when both R and S have columns that are named attributes. To compute $R \bowtie S$, we

- i) Compute $R \times S$
- ii) For each attribute A that names both a column in R and a column in S , select from $R \times S$, those tuples whose values agree in columns $R.A$ and $S.A$.
- iii) For each attribute A as above, project out the column $S.A$.

APPENDIX II : PART II
 LIST OF THE RELATIONS IN THE DATABASE
 AND
 THE SIMULATED DATABASE

<26>PF SCHEMA1

(DV SCHEMA1

(DB1 (PART-DETAILS (PART# PART-NAME INIM CLASS COST STOCK)
 (KEY PART#)
 (DEFAULT PART-NAME))
 (SUPPLIER (SUPPLIER# SUPPLIER-NAME ADDRESS RATING)
 (KEY SUPPLIER#)
 (DEFAULT SUPPLIER-NAME))
 (SUPPLY (PART# SUPPLIER# YEAR) (KEY PART#) (DEFAULT NIL))
 (MANUFACTURER (MANUFACTURER# MANUFACTURER-NAME MANUFACTURER-ADDRESS)
 (KEY MANUFACTURER#)
 (DEFAULT MANUFACTURER-NAME))
 (OWN (MANUFACTURER# OWNER-NAME PERCENTAGE-SHARE))))

NIL

<27>PF PART-DETAILS

(DV PART-DETAILS

((PART# PART-NAME CLASS INIM COST STOCK)
 (123 IC6709 A IMP 90 100)
 (222 IC8086 B IMP 120 625)
 (295 IC8080 B IMP 70 375)
 (1237 IC8088 A IMP 140 225)
 (378 UM328 C IND 55 245)))

NIL

<28>PF SUPPLIER

((SUPPLIER# SUPPLIER-NAME ADDRESS RATING)
 (S001 HCL DELHI 12)
 (S002 UPTRON LUCKNOW 10)
 (S003 WIPRO BANGALORE 5)
 (S004 ECIL HYDERABAD 4)))

NIL

<29>PF SUPPLY

(DV SUPPLY

((PART# SUPPLIER# YEAR)
 (123 S001 1988)
 (222 S002 1988)
 (295 S001 1987)
 (1237 S003 1988)
 (378 S004 1989)))

NIL

<30>PF MANUFACTURE

(DV MANUFACTURE

((PART# MANUFACTURER# LOCATION)
 (123 M001 TAIWAN)
 (222 M123 USA)
 (295 M123 USA)
 (1237 M123 USA)
 (378 M009 INDIA)))

NIL

<31>PF MANUFACTURER

(DV MANUFACTURER
((MANUFACTURER# MANUFACTURER-NAME)
(M001 MOTOROLA)
(M123 INTEL)
(M009 IND-CRAY)
(M115 BEL)))

NIL
<32>PF OWN

(DV OWN
((MANUFACTURER# OWNER-NAME PERCENTAGE-SHARE)
(M001 WILLIAMS 80)
(M123 VINOD 45)
(M009 KUMAR 25)
(M115 SANJAY 55)))

NIL

A>

APPENDIX III
LEXICON : CORE LEXICON

(WHAT (PRONOUN))
(WHICH (PRONOUN))
(WHERE (PRONOUN))
(WHO (PRONOUN))
(WHEN (PRONOUN))
(WHY (PRONOUN))
(THAT (PRONOUN))
(THIS (PRONOUN))
(THOSE (PRONOUN))
(THESE (PRONOUN))
(THERE (PRONOUN))
(ON (PREPOSITION))
(OF (PREPOSITION))
(IN (PREPOSITION))
(INTO (PREPOSITION))
(AT (PREPOSITION))
(FROM (PREPOSITION))
(ABOVE (PREPOSITION))
(BELOW (PREPOSITION))
(A (DETERMINER (NUMBER SINGULAR)))
(AN (DETERMINER (NUMBER SINGULAR)))
(THE (DETERMINER (NUMBER SINGULAR PLURAL)))
(IS (VERB (TYPE BE) (NUMBER SINGULAR) (FORM PRESENT)))
(ARE (VERB (TYPE BE) (NUMBER PLURAL) (FORM PRESENT)))
(WAS (VERB (TYPE BE) (NUMBER SINGULAR) (FORM PAST)))
(WERE (VERB (TYPE BE) (NUMBER PLURAL) (FORM PAST)))
(BE (VERB (TYPE BE) (NUMBER SINGULAR PLURAL) (FORM PRESENT)))
(BEING (VERB (TYPE BE) (NUMBER SINGULAR PLURAL) (FORM PROGRESSIVE)))
(DO (VERB (TYPE DO) (NUMBER PLURAL) (FORM PRESENT)))
(DOES (VERB (TYPE DO) (NUMBER SINGULAR) (FORM PRESENT)))
(DID (VERB (TYPE DO) (NUMBER SINGULAR PLURAL) (FORM PAST)))
(DONE (VERB (TYPE DO) (NUMBER SINGULAR PLURAL) (FORM PAST)))
(DOING (VERB (TYPE DO) (NUMBER SINGULAR PLURAL) (FORM PROGRESSIVE)))
(HAS (VERB (TYPE HAVE) (NUMBER SINGULAR) (FORM PRESENT)))
(HAVE (VERB (TYPE HAVE) (NUMBER PLURAL) (FORM PRESENT)))
(HAD (VERB (TYPE HAVE) (NUMBER SINGULAR PLURAL) (FORM PAST)))
(HAVING (VERB (TYPE HAVE) (NUMBER SINGULAR PLURAL) (FORM PROGRESSIVE)))
(LESS (COMPARATOR))
(MORE (COMPARATOR))
(LESSER (COMPARATOR))
(GREATER (COMPARATOR))
(HIGHER (COMPARATOR))
(LOWER (COMPARATOR))
(EQUAL (COMPARATOR))
(MUCH (DETERMINER))
(MANY (DETERMINER))
(THAN (DETERMINER))
(AND (CONJUNCTION))
(BUT (CONJUNCTION))

A)

DATABASE SPECIFIC LEXICON

{SUPPLY (VERB (TYPE NONAUX)
(FORM PRESENT)
(TRANSITIVITY TRANSITIVE)
(DB (WHO (SUPPLIER SUPPLIER-NAME))
(WHAT (PART-DETAILS PART-NAME))
(WHEN (SUPPLY YEAR))))})
{SUPPLIED (VERB (FORM PAST) (IRREGULAR SUPPLY))}
{SUPPLYING (VERB (FORM PROGRESSIVE) (IRREGULAR SUPPLY))}
{PART (NOUN (NUMBER SINGULAR)
(DB (FIELD (PART-DETAILS PART-NAME)) (ENTITY-PROPERTY SUPPLIER))))}
{PARTS (NOUN (NUMBER PLURAL) (IRREGULAR PART))}
{RATING (NOUN (NUMBER SINGULAR) (DB (FIELD (SUPPLIER RATING))))}
{INDIGENOUS (ADJECTIVE (DB (FEG (PART-DETAILS ININ) (CONST IND))))}
{IMPORTED (ADJECTIVE (DB (FEG (PART-DETAILS INIM) (CONST IMP))))}
{SUPPLIER (NOUN (NUMBER SINGULAR) (DB (FIELD (SUPPLIER SUPPLIER-NAME))))}
{SUPPLIERS (NOUN (NUMBER PLURAL) (IRREGULAR SUPPLIER))}
{MANUFACTURER (NOUN (NUMBER SINGULAR)
(DB (ENTITY MANUFACTURER)
(FIELD MANUFACTURER MANUFACTURER-NAME)
(ACTIVITY MANUFACTURER))))}
{MANUFACTURERS (NOUN (NUMBER PLURAL) (SYN MANUFACTURER))}
{COMPANY (NOUN (NUMBER SINGULAR) (SYN SUPPLIER))}
{COMPANIES (NOUN (NUMBER PLURAL) (SYN SUPPLIER))}
{ITEM (NOUN (NUMBER SINGULAR) (SYN PART))}
{ITEMS (NOUN (NUMBER PLURAL) (SYN PART))}
{COST (NOUN (NUMBER SINGULAR)
(DB (FIELD (PART-DETAILS COST)) (ENTITY-PROPERTY PART))))}
{STOCK (NOUN (NUMBER SINGULAR)
(DB (FIELD (PART-DETAILS STOCK)) (ENTITY-PROPERTY PART))))}
{PRICE (NOUN (NUMBER SINGULAR)
(DB (FIELD (PART-ORDER PRICE)) (ENTITY-PROPERTY PART))))}
{COSTLIEST (ADJECTIVE (DB (MAX (PART-DETAILS COST))))}
{CHEAPEST (ADJECTIVE (DB (MIN (PART-DETAILS COST))))}
{HIGHEST (ADJECTIVE (DB (MAX FOLLOW))))}
{ADDRESS (NOUN (NUMBER SINGULAR) (DB (FIELD (SUPPLIER ADDRESS))))}
{ADDRESSES (NOUN (NUMBER PLURAL) (IRREGULAR ADDRESS))}
{SUPPLIES (VERB (FORM PRESENT) (IRREGULAR SUPPLY))}
{YEAR (NOUN (NUMBER SINGULAR) (DB (FIELD (SUPPLY YEAR))))}
{MANUFACTURE (VERB (TYPE NONAUX)
(TRANSITIVITY TRANSITIVE)
(FORM PRESENT)
(DB (WHO (MANUFACTURER MANUFACTURER-NAME))
(WHAT (PART-DETAILS PART-NAME))
(WHERE (MANUFACTURE LOCATION))))})
{MANUFACTURERS (VERB (IRREGULAR MANUFACTURE) (FORM PRESENT))}
{MANUFACTURED (VERB (IRREGULAR MANUFACTURE) (FORM PAST))}
{MANUFACTURING (VERB (IRREGULAR MANUFACTURE) (FORM PROGRESSIVE))}
{DOWN (VERB (TYPE NONAUX)
(TRANSITIVITY TRANSITIVE)
(FORM PRESENT))}

(DB (WHO (OWN OWNER-NAME)
(WHAT (MANUFACTURER MANUFACTURER-NAME))))
(OWNS (VERB (IRREGULAR OWN) (FORM PRESENT))))
(OWNING (VERB (IRREGULAR OWN) (FORM PROGRESSIVE)))
(OWNED (VERB (IRREGULAR OWN) (FORM PAST)))
(NAME (NOUN (NUMBER SINGULAR)
(DB (FIELD (MANUFACTURER MANUFACTURER-NAME)
(SUPPLIER SUPPLIER-NAME)
(PART-DETAILS PART-NAME))))
(SHARE (NOUN (NUMBER SINGULAR) (DB (FIELD (OWN PERCENTAGE-SHARE))))))

A)

APPENDIX IV

SAMPLE RUNS

<53>DB-INTERFACE

KEY IN THE QUERY

>WHO IS THE SUPPLIER OF THE PART IC8886

PARSING....

THE WORD -- IC8886

IS NOT IN THE LEXICON;PLEASE KEY IN YES IF IT IS A
PROPER-NOUN

>YES

...SUCCESSFUL

FILLING QFRAME.... ...SUCCESSFUL

FORMAL QUERY BEING GENERATED ...SUCCESSFUL

DATABASE BEING ACCESSED...

(SUPPLIER-NAME)

(UPTRON)

>WHO SUPPLIES IC8886

PARSING.... ...SUCCESSFUL

FILLING QFRAME.... ...SUCCESSFUL

FORMAL QUERY BEING GENERATED ...SUCCESSFUL

DATABASE BEING ACCESSED...

(SUPPLIER-NAME)

(UPTRON)

>WHICH AGENT SUPPLIES THE PART IC8886

PARSING.... ...SUCCESSFUL

FILLING QFRAME.... ...SUCCESSFUL

FORMAL QUERY BEING GENERATED ...SUCCESSFUL

DATABASE BEING ACCESSED...

(SUPPLIER-NAME)

(UPTRON)

WHICH AGENTS ARE HAVING A RATING OF LESS THAN 6

PARSING.... ...SUCCESSFUL
FILLING @FRAME.... ...SUCCESSFUL
FORMAL QUERY BEING GENERATED ...SUCCESSFUL
DATABASE BEING ACCESSED...

(SUPPLIER-NAME)

(WIPRO)

(ECIL)

WHICH PARTS WERE SUPPLIED BY UPTRON IN THE YEAR 1988

PARSING.... ...SUCCESSFUL
FILLING @FRAME.... ...SUCCESSFUL
FORMAL QUERY BEING GENERATED ...SUCCESSFUL
DATABASE BEING ACCESSED...

(PART-NAME)

(IC8886)

REFERENCES

1. [Fillmore '68] Fillmore, Charles, J., 'The case for case' in 'Universals in linguistic theory' ed Bach and Harms. Holt Rinehart and Winston 1968.
2. [Grosz '82] Grosz, Barbara 'TEAM ': An extended abstract. Stanford Research Institute, 1982.
3. [Halliday '76] Halliday , M.A.K, et al. 'Cohesion in English' Longman group ltd. London 1976.
4. [Harris '77] Harris, Larry, R.I 'User-oriented database query with "ROBOT" natural language query system. 'International journal of Machine studies' 1977.
5. [Hendrix '77] Hendrix, G. 'Human engineering for applied natural language processing'. AI Centre, Stanford Research Institute. California, 1977.
6. [Kaplan '82] Kaplan, S J. 'Co-operative responses from a portable natural language query system'. Artificial Intelligence' vol 19 Oct.1982.
7. [Nilsson '83] Nilsson, N.J. 'Principles of Artificial Intelligence' Springer Verlag. 1983.
8. [Tennant '80] Tennant, H. 'Natural language processing' Petrocelli, Princeton, 1980.

9. [Ullman '83] Ullman, J. 'Principles of database systems' 1983.
10. [Waltz '75] Waltz, D. 'Natural language access to a large database: An engineering approach'. Proceedings of the 4th International Joint Conference on Artificial Intelligence. Tbilisi USSR. 1975.
11. [Winograd '83] Winograd, T. 'Language as a cognitive process: syntax'. Addison Wesley 1983.
12. [Woods '70] Woods, W.A. 'A transition network grammar for natural language analysis' CACM vol 13. Oct. 1970.
13. [Woods '72] Woods, W. A. et al 'The Lunar sciences natural language interface system' Final report. BBN report 2378 Cambridge, (Mass) 1972.

