



National Library  
of Canada

Bibliothèque nationale  
du Canada

Canadian Theses Service

Service des thèses canadiennes

Ottawa, Canada  
K1A 0N4

## NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

## AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, S.R.C. 1970, c. C-30.

Issues in  
the Design and Implementation  
of a Portable Natural Language Interface System

Philip J. Vincent

A Thesis  
in  
The Department  
of  
Computer Science

Presented in Partial Fulfillment of the Requirements  
for the Degree of Master of Computer Science at  
Concordia University  
Montréal, Québec, Canada

February, 1988

© Philip J. Vincent, 1988

71  
Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-41599-1

## ABSTRACT

### Issues in the Design and Implementation of a Portable Natural Language Interface System

Philip J. Vincent

A natural language interface (NLI) is a computer program which allows interaction with computers via a language that is easy and natural for people to use. A portable interface is one which can be moved from one computer application to another with a minimum of effort. This thesis studies the issues involved in designing and implementing a portable NLI. A system is presented which divides the task of understanding questions on a database into three separate tasks. A "Heuristic Parser" is presented which parses natural language questions deterministically; a World Knowledge Base develops a domain-independent meaning representation of the user's query, and a Domain-specific Knowledge Base matches the objects and relationships of the domain with the concepts found in the world model. A prototype has been implemented on a microcomputer, and has been transported to two domains other than the one used in the design process. The capabilities and implementation details are discussed, and key issues identified.

## Table of Contents

Introduction	1
Chapter 1:	
Components of a Natural Language Understanding System	9
Chapter 2:	
Survey of Current Transportable Natural Language Interfaces	18
2.1. EUFID	19
2.2. CO-OP	22
2.3. INTELLECT: A Commercial NL Interface	23
2.4. Datalog	25
2.5. TQA	26
2.6. PRE	27
2.7. Ginsparg's System	28
2.8. ASK	30
2.9. TEAM	30
2.10. Guida and Tasso's System	31
2.11. LADDER	31
2.12. Using Semantic Primitives	32
2.13. KID	33
2.14. Summary	38

## Chapter 3:

Parsing	40
3.1 Basic Linguistics	40
3.2 Syntactic Theories of Language	44
3.3 Transition Network Grammars	47
3.4 Augmented Transition Networks	52
3.5 The Workings of an ATN	55
3.6 The Chart Parser	62
3.7 The Marcus Parser	70
3.8 Case Grammar	77
3.9 Semantic Grammar	80
3.10 Conceptual Dependency Theory	82
3.11 Procedural Parsers	84
3.12 QNL: The Heuristic Parser	86
3.12.1 First Attempts	87
3.12.2 Development of a Grammar	92
3.12.3 Design Criteria of the Heuristic Parser	93

## Chapter 4:

Knowledge Representation	95
4.1 What Kind of Knowledge is Needed?	98
4.2 Conceptual Modelling Tools:	102
4.2.1 The Entity-Relationship Model	103
4.2.2 The Semantic Data Model	105
4.2.3 TAXIS	108
4.2.4 Summary of the Conceptual Design Tools	110

## Chapter 4 (cont'd)

4.3 The AI Approach to Knowledge Representation	112
4.3.1 Logic	112
4.3.2 Procedural Representation of Knowledge	114
4.3.3 Frames	115
4.3.4 Semantic Primitives	116
4.3.5 Semantic Networks	118
4.4 Summary	121

## Chapter 5:

Implementation of a Prototype	123
5.1 The Lexicon	126
5.2 Parsing	129
5.3 Syntactic Ambiguity	132
5.4 Domain-Independent Semantic Analysis	137
5.5 Pronouns	145
5.5.1 Referential Pronouns	145
5.5.2 Relative Pronouns and Relative Clauses	149
5.6 Conjunctions and Disjunctions	153
5.7 Negation	155
5.8 Verbs	156
5.9 Anaphora	158
5.10 The Domain-Specific Knowledge Base	159
5.11 Linking the DKB to the WKB	162
5.12 Database Values	166
5.13 Transporting to a New Application	167

## Chapter 5 (cont'd)

5.14	Designing the Domain-Specific Knowledge Base	168
5.15	Capturing the Semantics of the New Domain	169
5.16	Implementation and Testing	176
5.17	Summary	182

## Chapter 6:

Concluding Remarks	185
6.1 Conclusion	185
6.2 Future Work	189
References	192
Appendix A: The Suppliers-Parts-Projects Database	208
Appendix B: The University Domain	212
Appendix C: The Airline Domain	216
Appendix D: Sample Queries Handled by QNL	220



## List of Figures

2.1	Definition of SUPPLIER in KID's Knowledge Base.	35
2.2	Definition of supplier_number.	35
2.3	Definition of City.	36
2.4	Definition of SPJ.	37
3.1	Syntax Tree for the sentence "The boy hit the ball."	43
3.2	Chomsky's Hierarchy of Languages.	45
3.3	A Finite State Transition Network for a noun phrase of a regular grammar.	47
3.4	Production Rules for a Context-Free Grammar of English.	49
3.5	Recursive Transition Networks for Embedded Noun Phrases and Prepositional Phrases.	49
3.6	Syntax Tree for the Sentence "Is the boy sleeping?"	51
3.7	Syntax Tree for the Sentence "The boy will sleep."	51
3.8	Phrase Structure Rules for an ATN.	56
3.9	An ATN for a Simple Grammar of English.	58
3.10	Ambiguous Sentences.	59
3.11	The same set of Phrase Structure Rules as Listed in Figure 3.8, Numbered for Convenience.	65
3.12	Initial Chart of the Chart Parser.	64
3.13	After a First Pass over the Initial Chart.	66
3.14	Final Chart for our Sample Sentence.	69
3.15	Production Rules for a Case Grammar.	77
3.16	Performance Comparison: Chart Parser vs. Heuristic Chart Parser.	91

4.1	Entity-Relationship Diagram for the Suppliers-Parts-Projects Database	103
5.1	Architecture of a Portable Natural Language Interface.	124
5.2	Sample Entries from the Lexicon.	128
5.3	The Parsing Rules Used by QNL.	131
5.4	A Sample Set of Heuristics.	134
5.5	A Sample Parse Tree as Produced by QNL.	137
5.6	A Small Part of the World Knowledge Base	138
5.7	Domain-Independent Meaning Representation passed to the DKB.	144
5.8	Nuances of Elliptic Reference with Conjunctions and Disjunctions	154
5.9	Domain-specific Meaning Representation	160
5.10	Mapping of Lexical Items to WKB Concepts, and WKB Concepts to DKB Concepts and Objects.	165
5.11	Entity-Relationship Diagram for a Simple University Database	170
5.12	Relationship between Number of occurrences of Entities, and the Number of Different Names.	173
5.13	Sample Queries in Imperative Form.	178
5.14	Results of Test Queries.	181

## Introduction

People communicate using language which has evolved into an easy and effective way for us to share knowledge and ideas. If we could interact with a computer in plain language, just explaining what we would like it to do rather than telling it how to accomplish the task in some rigid command language, we would attain what many refer to as an "ideal" method of man-machine interaction: computers would have to adapt to our way of thinking and of doing things, rather than the other way around. Currently, user interfaces to computer systems are biased in favour of the computer, and most of the learning and adapting is done by the human. If this burden could be shifted onto the computer, more people would have easier access to information.

Usually, the more complex the computer system, the more difficult it is for a person to use it, not just because the command sequences are complex, but also because the stored information may not have the same structure as the user perceives. People are distracted from the task at hand by having to understand these structures and express their requirements in an artificial, formal language the computer can understand.

There are several obstacles to natural language communication with machines: the extra processing to translate the user's expression into an expression the software understands incurs extensive memory and time overhead. Natural language is ambiguous, and capitalizes on things that humans do well, such as the sharing of common perceptions and experiences. Finally, understanding the intricacies of how language works is not a trivial task.

Within a narrow context, a computer can handle ambiguity if the amount of common knowledge is sufficiently small that it may be explicitly imparted to the machine's memory. A small subset of language is not difficult to implement on a machine. As we narrow the domain of discourse, we also move from a natural language to one which is a restricted, but still natural one, containing fewer words and fewer possible meanings per word or group of words. Although the language handled by such a restricted system is not completely natural and free-form, it can serve to make man-computer interaction easier. Two examples of this are the command language of dBase III+, which uses common English words (Castro, Hanson and Rettig, 1985), and HAL, a natural language interface to Lotus 1-2-3 (Miller, 1987).

A Natural Language Interface (NLI) is a computer program designed to bridge the gap between a formal computer language, and one which is relatively unconstrained and natural to the human. It receives a natural language expression and transforms it into an expression the computer can use.

An intelligent natural language interface to a database acts as a mediator between how the user thinks about data and how it is stored within the database. To accomplish this it should engage in a conceptually natural dialogue to free the user from having knowledge of an artificial language, and to avoid having to learn about the data model(s) and data organization. By conceptually natural, we mean it should know the vocabulary inherent to the domain, be able to handle ambiguity in a manner acceptable to the user, and tolerate minor errors.

Many existing NLI systems refer to a "professionally-oriented language" (Pavlov, Angelova, Paskaleva, 1985). The work environment, the habits of workers, and the nature of their information requirements naturally constrain the language they use to access a database through a NLI (Carbonell et al., 1983). This may mean it is possible to

build a fairly robust NLI without incorporating a substantial subset of natural language vocabulary, grammar and semantics.

A transportable NLI is one that is capable of being moved to a new database other than the one for which it was designed, without major modifications. The design of such a system requires that the language processing techniques be general enough to apply to all domains, and not specifically tailored to the application environment.

This thesis studies some of the issues involved in designing a transportable natural language interface to a database, and presents a prototype implementation called "Queries in Natural Language" (QNL). QNL is a question-answering (Q-A) system, as are many existing operational NLI. The implementation of a Q-A system was chosen for several reasons.

First, the linguistic knowledge required for a Q-A system is more easily modelled than for other types of discourse; the semantic interpretations of questions are limited by the knowledge a system has about the objects and relationships of a domain, and the ways it has of recognizing them. Second, question-answering situations are the most frequent with respect to databases. In general, users spend more

time extracting information than storing new information, although there are obvious exceptions like transaction-processing systems.

The structure of questions the user may ask should not be so restricted that an inexperienced user would require extensive training in how to use the system. For instance, a set of sample questions and an explanation of the type of information generally needed by a set of workers should suffice.

In a situation where the user is "non-hostile", that is, genuinely cooperating with the system in an effort to extract needed information, the semantic and syntactic structure of the discourse is reasonably predictable (Tufig and Cristea, 1985). Experiments have shown that users frequently restrict the dialogue to a small number of terms when extracting information from a computer system (Thompson, 1980). They find patterns which result in successful retrieval, they minimize the number of keystrokes and they try to reduce the mental effort required to formulate new, and perhaps complex, questions.

A transportable NLI is one that is independent of the application environment, and of the data base management system (DBMS). This means the system should exhibit "low structural semantic content" (Frost, 1985) so that it is not tied to the data organization of the underlying databases or their query languages.

To be considered transportable, the amount of effort required to transport the system from one environment to another should be minimal. The degree of portability is reflected in the amount of effort required to transport it to a domain other than the one for which it was designed.

The NLI developed as part of this thesis is meant to be run on a single-user workstation. A parser written in Lisp has been implemented to run on an IBM PC/XT with 640 K RAM. TLC Lisp, the implementation language, is an interpreted Lisp for microcomputers. Test results are encouraging and indicate this could be a viable NLI on a slightly larger and faster workstation.

The system was originally designed to handle the Suppliers-Parts-Projects database of Date (Date, 1982, pg. 114) (Appendix A). To determine the degree of portability, two new applications were introduced: A university database



(Appendix B), and the Airline database of Ullman (Ullman, 1982, pg. 19) (Appendix C). The Date database has been used by several researchers as an aid to developing a NLI (Boguraev and Sparck-Jones, 1982; Ginsparg, 1983). The university database was chosen because it is semantically different from the Date database, yet still possesses suitable ambiguities to test the natural language understanding abilities of the system. The airline database is more complex than either of the first two, and coping with the problems encountered with this domain demonstrated some interesting problems with the design of the NLI.

Chapter 1 of this thesis describes the components of natural language processing systems in general. Chapter 2 presents a survey of current NLI, including a brief discussion of their parsing mechanisms and knowledge representation schemes. Chapter 3 is a discussion of the theoretical aspects of parsing as may be applied to natural language, and presents some of the formalisms used in NLI. The parser used in QNL is also introduced. Chapter 4 studies the knowledge representation needs of a NLI and suggests the use of semantic networks as a suitable formalism. Chapter 5 presents the development and implementation of a prototype system, and includes a discussion of the design of a suitable parsing formalism, handling syntactic and semantic

ambiguity, conjunctions, pronouns, anaphora, etc. It concludes with a description of the process of moving the interface to a new domain, the problems encountered, and the solutions employed. Chapter 6 is the conclusion which discusses the results of this experiment and future work. The Appendices include the conceptual schemas of the data bases used in the study, and a list of the types of questions and sample dialogue which the system is capable of handling.

## Chapter 1

### Components of A Natural Language System

A computer system "understands" a sentence or question if it purposefully performs the appropriate actions to indicate that it has understood. This involves transforming the original expression into some internal representation which other components of the computer system can use. For instance, an English-language request may be transformed into a Lisp program which can be executed by a Lisp interpreter.

If a user asks a NLI "which suppliers sell red parts?", and the NLI displays a list of those suppliers, then the system may be said to "understand". However, if all the system is capable of doing is displaying that one list, then no understanding has occurred. There are degrees of understanding, and so a NLI must be judged on how broad a range of questions it can "understand". This is analogous to the case if you ask a person, "Parlez-vous francais?", and he responds "Mais, oui!", it does not necessarily mean that he understands the language, but perhaps only that he has issued a programmed response.

The understanding process can be broken into three broad components: recognizing the words and terms in the input stream and their structural relationships (syntax), understanding how the terms relate to each other (semantics), and finally some sort of "meta-knowledge" which would include knowledge about the way things normally happen within a specific context. Most NL systems rely more heavily on one of these aspects than the others.

Early work in NL understanding focused on templates, keyword matching, and ad hoc disambiguation rules. This approach assumed words are the basic unit of meaning, and that there could be a one-to-one mapping between the words in a sentence or question and the meaning representation. Later, as syntactic theories of language developed (Chomsky, 1957, 1965), parsers were developed to apply theories like Transformational Grammar (TG). Researchers were under the mistaken impression that an abstract machine to manipulate syntax rules as a means of understanding language would be relatively easy to construct (Barr and Feigenbaum, 1981, Vol. 1, pg. 281).

Early natural language systems include (Simmons et al., 1962; Craig et al., 1966; Weizenbaum, 1965; Bobrow, 1968.) These early systems were closely coupled to the database schema, and limited to narrow domains. Typically they had

low conceptual coverage; that is they did not understand the relationships between language and the contents of the domain, but rather provided a way of mapping the tokens of the natural language expression to concepts explicitly stored. They were generally not portable, and made no attempt at real understanding of language. See (Tennant, 1981) for a comprehensive review of a representative set of systems from that period.

As more research was done in the area of language comprehension, it became apparent that human language processing makes use of a "world model" of knowledge which acts as a framework for understanding, and so there developed an increased awareness of the differences between syntax and semantics. The functional role of a word is not necessarily indicative of the speaker's intent in using it (Katz and Fodor, 1964), and so some methods had to be developed to represent not just functional relationships between words, but semantic relationships between the concepts they represent. Two such formalisms are case frames (Fillmore, 1968) which represent relationships between verbs and their subjects and objects, and semantic networks (Quillian, 1968), which link concepts into

hierarchies. Both these methods have demonstrated that a finite set of relationships can describe a world sufficiently to differentiate between similar sentence constructions.

The 1970's saw an increased interest in natural language front ends for databases (Woods, 1970; Waltz, 1978.) This was partly due to increased funding of Artificial Intelligence from the U.S. military and partly to the increased use of computerised databases by business.

The systems of the era developed what has been referred to as the "classical, two-stage architecture" (Hafner and Godden, 1985): syntactic processing followed by the application of semantic knowledge. It was thought at the time that it was more efficient to refrain from applying semantics until all parsing was complete, since many ambiguous expressions could be solved by syntactic analysis alone (Woods, 1970; Earley, 1970). In any given sentence, there could be a large number of local ambiguities which would be magnified by the too-early application of semantic knowledge. For instance, a word may have multiple meanings, but many of these may be eliminated by the phrase or sentence structure. Typical of the two-stage method is parsing by an ATN parser or Chart parser, followed by domain-specific semantic processing (Waltz, 1982).

The problem with the two-stage approach is that the purely syntactic parsing module could generate multiple parse trees which would have to be processed by the semantic analyzer. If the syntactic analysis module has no access to semantic knowledge, then it could generate a large number of meaningless interpretations which would have to be studied and rejected by the semantic processing component.

For instance, without access to semantic knowledge, a parser would develop two parse trees for each of these sentences:

I hit the boy with the hammer.  
I hit the boy with the moustache.

To improve efficiency, there was a move to employ a "pragmatic grammar", which in effect guided the syntactic analysis with domain-specific semantic knowledge (Winograd, 1972; Hendrix et al., 1978): Semantic markers are stored with each word and are involved with the parsing process in place of syntactic information such as noun and verb indicators. The success of such systems demonstrated the importance of domain-specific knowledge to understanding, but to transport such a system to a new domain required considerable programmer effort because the pragmatic knowledge was so closely tied to the domain.

Another school of thought advocated the elimination of syntactic processing and the exclusive use of semantic knowledge. The use of "preference semantics" (Wilks, 1975) and Conceptual Dependency Theory (CD) (Schank and Colby, 1973; Schank and Abelson, 1977;) are based on the premise that humans ignore syntax and derive the meaning of a sentence by building a representation of the semantics of the concepts and the relationships in a sentence. CD has formed the basis for related work to include stereotypical knowledge about frequently occurring situations, the intentions of speakers and hearers, etc. Proponents of CD claim that syntactic understanding is completely unnecessary. However, as part of their understanding process, they must take into account the syntactic roles of at least some of the tokens in a sentence, specifically the nouns, proper names, pronouns and verbs.

All NL systems use the morpheme or word element as a basis for understanding, and either use an existing linguistic theory or develop one for sentence analysis. It is clear that syntactic knowledge is useful in some instances, and not in others. The same can be true for semantic knowledge. The question in designing a natural language system is when and how should these various types of knowledge be applied?



Syntax and semantics are not two different approaches to language understanding: they both attempt to relate a specific sentence pattern to the meaning of the words in the sentence. A system biased toward syntax analysis has certain advantages, such as recognizing the role of a word it has never seen before:

X broke the Y  
X sells Y to Z

But a syntax-biased system could have trouble with sequences of ill-formed sentences:

Which suppliers sell red parts to projects in London?  
Paris?

This demonstrates that syntax and semantics-based systems are not completely interchangeable, but rather each has a role to play in language understanding.

There has been considerable work done in syntax analysis of language, but semantic processing is still a new and controversial area because it demands a formal structure for representing human knowledge. There is no one "right" way to achieve language understanding, and so this makes it an interesting topic for study.

There are many real-world computer applications which could benefit from the use of natural language processing, and the one chosen for study here is a transportable natural language interface to a database. We have seen that the idea of designing a natural language interface based on a two-stage architecture was abandoned in favour of the computational efficiencies of systems employing a pragmatic or domain-specific grammar. But such systems are not portable. Is there a method of combining syntactic knowledge and semantic knowledge that will allow an efficient, transportable natural language interface? When should the various types of knowledge be brought to bear on the analysis? How do we represent this knowledge, and how will it be applied to the understanding process?

If we have two modules for understanding a sentence, syntax and semantics, it would be safe to assume the syntax module could remain relatively stable while the semantic module would change with each application. The structural relationships between words are more likely to remain constant than the semantic relationships. But how different is each application? There are concepts employed in database systems which are not unique, but apply to many environments; concepts like operations on data (counting, sorting, printing, and so on), and a large number of "closed-class" words (prepositions, numbers, and so on). In

databases, we typically deal with objects, their attributes or attribute-values, and relationships. It is possible to codify this general knowledge about databases so that it need not be recreated each time a new application is introduced. So what will emerge is a two-tiered semantic knowledge base, consisting of a general knowledge module containing "world-knowledge", a domain-specific knowledge module, and a means of merging the two. Finally, there must be a strategy of how and when to apply semantic knowledge, and syntactic knowledge to the understanding process.

## Chapter 2

### Survey of Existing Portable Natural Language Interfaces

Early NLI were experiments in developing formalisms for the understanding of language, but recently effort has shifted to the design of portable NLI which can be used for different subject areas and different DBMS. In all of the recent portable NLI, modularity is viewed as a key to portability, so that the domain-specific and domain-independent portions may be extended separately.

By domain-specific, we mean that part of the interface which has in-depth knowledge of the objects and relationships of the application environment. The domain-independent portion has knowledge about language in general.

One approach to transportability is to isolate domain-independent knowledge from the domain-specific part so that the introduction of a new application will have little impact on the overall system. Another strategy is to employ a sophisticated computer program to interact with a database designer or DBA to integrate the knowledge base for a new application with an existing system. Some systems combine both methods. This chapter describes recent NLI.

✓      }

A balanced interaction between syntactic and semantic processing can lead to an efficient implementation, but to achieve a degree of portability there must be a degree of domain-independence in these modules (Konolige, 1979; Ginsparg, 1983; Boguraev and Sparck-Jones, 1983; Hafner and Godden, 1985). If the syntactic processing is guided by semantic knowledge, the generation of partial parses which cannot contribute to understanding can be limited, therefore reducing the time and the amount of memory that must be used to store and manipulate multiple data structures. However, if the semantic knowledge is specific to the domain, then extensive effort will be required to transport the system.

A rather simplistic approach to designing an interface is to use the database schema as the knowledge base. This essentially results in a program which maps words and synonyms to file objects and their attributes. Two recent systems which have employed this technique are EUFID (Templeton and Burger, 1983) and CO-OP (Kaplan, 1984).

## 2.1 EUFID

A major thrust of this project is that general methods of language processing will not work in real-world applications, and so the goal is to discover how to successfully acquire and integrate large amounts of database-specific knowledge.

The EUFID system employs a semantic grammar that is specific to each application, tailored to the database schema and the relational operations to be performed on the Database. The semantic categories of terms in the dictionary are either closed-class words, or terms which correspond to the objects, attributes and relationships in the database. The database itself is represented as a relational schema, even if it is not based on the relational data model.

The dictionary of application-specific words has pointers to the database concepts. Each concept is linked to others through case frames. A case frame describes a relationship, the allowed or expected participants in the relationship, and the roles they play. Case frames can be used to explicitly describe the relationships of the database and their associated objects. Also stored with each concept are one or more functions which map the concept to the files and fields of the database.

In EUFID, a natural language question is parsed according to the semantic grammar, with syntactic information used only in the case of semantic ambiguity. A typical strategy would be to identify the main verb of a sentence, and look-up the case frames associated with that verb. The case frame could then be used to guide the parser to generate a parse tree (a hierarchy representing the structure of the question). The

terminal nodes of the parse tree are the database concepts, and the mapping process matches these concepts with the semantic graph. The functions stored with the concepts in the graph then access a table of database-specific information such as file and field names, and the query translator generates the formal query.

EUFID attempts to achieve processing efficiency by using semantic information as early as possible in the understanding process. The problem with this as in other systems based on a pragmatic grammar is that portability is difficult to achieve since the entire grammar and most of the knowledge about the database must be re-acquired for every application. They try to attain portability by building tables to hold information about the database access, and about groupings of database-specific concepts.

The range of grammatical structures which EUFID can handle is limited, and the conceptual coverage depends largely on the efforts and expertise of the people involved in transporting the system to a new application or DBMS. Introducing a new application takes several months of effort (Grosz et al., 1987).

## 2.2 CO-OP

CO-OP is designed specifically to be a transportable NLI that handles ambiguity (more than one valid interpretation), vagueness (missing information in the question), and can recognize database values that are not explicitly stored in the lexicon. The system keeps domain-specific knowledge independent from the processes involved in transforming a natural language query into a database query. These processes (syntactic parsing and some heuristics to handle semantic ambiguity) access the lexicon and database schema, but treat these modules as declarative knowledge. Then, if the content of the declarative portions change, the functions of the procedures are not affected.

Three sources of declarative knowledge are used: the database schema, the database itself, and the user lexicon. Kaplan uses the lexicon to record a small set of closed-class words, and the content words of a new application are merged with this. A parser generates syntax trees representing the relationships between the objects in the query. In the case of semantic ambiguity some simple heuristics are brought to bear which draw on semantic rules regarding verbs and their objects and the predictive capabilities of prepositions. One set of heuristics evaluates the relative positions of objects or attributes in



a sentence to their positions in the database schema, and assigns a measure of their semantic relatedness. For such an approach to work, it is crucial that the database schema accurately reflects the semantics of the domain.

Database values are not stored in the lexicon since this could render the lexicon obsolete as the values change. Instead, values are recognized by the form of the query, using heuristics similar to the ones already mentioned, which rely heavily on the database schema.

The separation of procedural from declarative knowledge has resulted in a much more transportable system than EUFID. Experiments with moving CO-OP to a new domain required only a matter of hours rather than months (Kaplan, 1984).

### 2.3 INTELLECT: A Commercial NL Interface

One of the few successful commercial NLI systems is INTELLECT (Harris, 1978; 1979) (also known by its other commercial names such as OnLine English from Cullinet Software, ELI from Management Decision Systems and GRS EXEC. from InSci, and other obsolete names such as ROBOT) which provides a limited range of natural language structures, and has successfully interfaced with several large databases. This system is relatively easy to customize by a DBA without

intrinsic knowledge of the workings of the system, although the successful operation of the system depends on the goodness of the user-defined lexicon (Martin, 1985, pg. 219). Other sources of knowledge are the database itself and the database schema.

INTELLECT can be conceived of as a mapping between a user-defined lexicon and a collection of "processes" which expect certain parameters. Some of these processes access the database files, while others invoke graphics or report-formatting (The FOCUS language is the utility subset of INTELLECT used for these purposes). The user-defined lexicon contains a base collection of closed-class words and expressions ("how many", "between", "in", etc.) and identifiers for various processes, which is augmented by labels referring to the file structures, synonyms, and keywords (Martin, 1986, pp. 215-245).

INTELLECT expects the user to be familiar with the file structure to help solve ambiguities and handle database values and other words unknown to the system. To assist the user in this respect, it is able to show the file structures, thus giving both the program and the user a common knowledge base to work with. When it detects syntactic ambiguities, it invokes a menu-driven interactive session to remove the confusion. This may appear to be a

deviation from natural language, but if ambiguity arises in a conversation between two people, they might engage in a brief question-and-answer diversion to clear up the ambiguity. Using the menu approach is just an expedient.

## 2.4 Datalog

Datalog (Hafner and Godden, 1985) is a database query system which uses a "cascaded ATN grammar" allowing parallel syntactic and domain-independent semantic processing, rather than the two-stage approach described in chapter 1. The main objective of their study is semantic interpretation, and so they have only developed a simple grammar for question forms, which does not handle conjunctions and other difficult structures.

The domain-independent semantic processing is hierarchically arranged. Once a grammatical structure is recognized, procedures are called which recognize the semantic content of the phrase based on a domain-independent world model.

In Datalog, domain-independent knowledge is represented in the form of a semantic network and is connected to a base vocabulary. The terms in the lexicon are mapped to concepts in the world model, or the domain model. The domain-dependent knowledge base is a semantic network describing

the entities and relationships of the domain, and their attributes. The links in the domain semantic net are restricted so that the general semantic procedures, which can operate on the world knowledge base, and the domain semantic net, can process the domain knowledge according to a pre-determined, domain-independent set of rules. The rules are designed to manipulate concepts like entity and attribute, and so the domain objects must be specified in these terms. This means that the introduction of a new domain must conform to a structure expected by the NLI.

Transporting to a new domain requires a person familiar with the structure of the interface, and so far their experiments have only been with simple file structures.

## 2.5 TQA

The Transformational Question Answering System (TQA) (Damereau, 1985), is an SQL-oriented NLI. Damereau claims the grammar is a domain-independent semantic grammar, but it requires the terms of the domain-specific vocabulary to be defined as references to file attributes or synonyms thereof. Rather than being domain-independent, it includes a very simple set of domain-independent phrase-structure rules, but then resorts to matching entries in a table.

There is no semantic model of the domain, nor of the domain-independent knowledge. The system consists of a set of transformational rules which map English words to file names and attributes, so linguistic and conceptual coverage of this system is not extensive.

TQA uses an interactive program to acquire domain-specific knowledge for each new application. The program accesses information about the database structure and then requests a DBA or other knowledgeable individual to define the synonyms for the entities and relationships. It also requires that the SQL programs to accomplish various tasks be specified at this stage. What in effect happens during the processing of a query is that entities and relationships are recognized in the user input and a table lookup procedure determines which SQL procedures will answer the query.

## 2.6 PRE

Purposefully Restricted English (PRE) (Epstein, 1985) is a transportable NLI which restricts the range of natural language queries. Such restriction alleviates the need for sophisticated parsing and semantic processing, and allows a simple mapping from the tokens in an English language expression to the file names and attributes of the database.

PRE requires the user to be familiar with the structure of the database and the limited range of grammatical structures. The range of linguistic ability is not so restricted as to cause users to require a significant amount of training. In fact, the grammar accepts simple sentences with relative clauses and conjunctions, and a large set of pre-determined phrase structures.

Epstein describes PRE as being a "minimalist" approach to natural language processing, the goal of which is an easy-to-use interface that is also easily transportable. It has been tested on several large databases, and moving from one application to another has been achieved without much difficulty. Perhaps most importantly, transporting to a new application does not require system specialists or interactive programs, and can be accomplished in a matter of hours.

## 2.7 Ginsparg's System

Semantic networks are employed by (Ginsparg 1983), and combined with a case grammar in a system that is meant to be transportable to different domains, DBMS, and is also transportable linguistically. Conceptual knowledge is stored in a taxonomic hierarchy. The word phrases normally occurring within the domain are clustered based on similarity

co-efficients between terms. Domain-specific concepts are generalizations of terms which occur within the domain, and these are linked to domain-independent concepts. In this implementation, parsing is mainly kept separate from the domain, although some cues from the domain-specific knowledge base are used to guide the process. The linguistic coverage of this system is quite extensive, including noun-noun modification, quantification, conjunctions and disjunctions.

Transportability is achieved by persons knowledgeable in the workings of the system who compile the information needed by a new domain. This new information, including case information about file attributes, how to handle virtual relations, details of database concepts, and so on, is built into a semantic net and linked to an existing world model which is also a semantic net. Mapping to a new DBMS is achieved by writing a translator program between the relational algebra expression output by the system and the query language of the new DBMS.

Experiments have demonstrated the Ginsparg system may be transported to a new application in a matter of hours or days, depending on the semantic closeness of the new domain to existing domains.

## 2.8 ASK

The ASK system (Thompson and Thompson, 1983, 1985) is loosely designed on the E-R model and appears to be appropriate to databases with relatively simple conceptual schemas. Physically, a semantic net is used to store a taxonomic hierarchy of objects, events, and attributes of objects and events. A user-specific vocabulary holds the knowledge pertaining to user views necessary for accessing data, mainly in the form of scripts. A script will describe the type of information normally required, and the names of the procedures to obtain that information.

The linguistic coverage of ASK is extremely limited, and the system is basically a mapping of words to the contents of the database. The only verbs permitted in this system are forms of "to be".

## 2.9 TEAM

A taxonomic hierarchy depicting relationships between objects, constraints on arguments to predicates, and information about relationships is used by TEAM (Grosz, 1982, 1983, 1987). Parsing uses a general grammar as opposed to a semantic grammar, and transforms the input into a database-independent semantic meaning. This is then



applied against a dictionary of what the general terms mean with respect to the current database. Then the terms of the user input are mapped directly to the conceptual schema. There are separate rule sets for domain-independent syntactic analysis and domain-specific inferencing.

#### 2.10 Guida and Tasso's System

An interface to an information retrieval system (Guida and Tasso, 1983) uses a hierarchy of concepts drawn from the databases's hierarchy of concepts. It has a set of rules about how to complete a search for information, and a matching process between the rules and the input is used to understand the user's intent.

#### 2.11 LADDER

The LADDER system (Hendrix, Sacerdoti, Sagalowicz and Slocum, 1978), uses the functional data model (see for instance, Orman, 1984), and a semantic grammar. The functions are domain-specific and are used to fill templates.

A case grammar can itself be a form of knowledge representation which may contain either domain-independent or domain-specific knowledge (Hendrix and Lewis, 1981).

Their implementation, similar to the LADDER system employs a "pragmatic" grammar describing concepts that occur within any database, and a set of rules about how to access databases in general. Then the schema of the target database is mapped to these concepts. Thus, there is no meaning representation as such.

## 2.12 Using Semantic Primitives

Semantic primitives form the basis of a system by (Scha, 1982), wherein the concepts of the database are the most primitive concepts (i.e. filenames, attributes and data items) and all other concepts are defined in terms of these primitives. Scha's system uses translation rules to map directly between English words and the primitives.

Conceptual Dependency Theory is an extension of the idea of case grammars, which can themselves be thought of as primitive knowledge representations. A case grammar is procedural in nature: verbs have sets of expectations, so verbs occurring in a question invoke certain functions to be applied to the words and terms. A change of domains will require a change of the semantic grammar. It is possible, however, to use non-domain specific semantics in a front-end if those semantics represent semantic primitives. Then these primitives would be used to map to the database.

(Boguraev and Sparck-Jones, 1983) comment on this approach, and emphasize that the semantic processing should take place as early as possible, although this is a trade-off with respect to keeping domain-specific information separate from world knowledge.

A script-based system by (Pazzani and Engleman, 1983) uses the primitives of Conceptual Dependency Theory, and a dictionary of scripts. These scripts are domain-specific. Transportability is achieved by supplying a new set of scripts for each new domain.

### 2.13 KID

KID (Ishikawa et al., 1986), is a Japanese-language NLI which uses an object-oriented approach and a taxonomic hierarchy. Encoded in the knowledge base are explicit functions on how to access the database, and how data is stored. A case grammar is employed with cases specific to the database.

It will be useful to demonstrate the knowledge representation schema of KID, since it is specifically designed for a transportable NLI, and is one of the more recent developments.

A taxonomy is achieved by hierarchical arrangement of sub-classes and super-classes. Classes or objects have sub-structures called attributes which implement properties of and relationships between objects. Attributes have facets which describe all aspects of the attribute.

In KID, properties of objects are allowed to change over time. For instance, access information knowledge is explicit in the knowledge base. If the access routine must change, as long as the name remains the same, there is no effect on the knowledge base.

The knowledge base of KID is divided into two modules. The lexicon consists of definitions of terms and the relations between them. A case-grammar is used, with specific information about the database. For instance, the verb "to sell" would invoke the case frame

(SUPPLIER, PART, RECIPIENT, QUANTITY)

The parser would then be guided to look for terms in the input stream matching these concepts.

Figure 2.1 is the definition of Supplier in the Suppliers-Parts-Projects database in the language of KID:

SUPPLIER		
Super	value	ENTITY
Class	value	Non-primitive
Level	value	Classlevel
Attribute	value	s#, sname, status, city
Key	value	s#
Return	value	s#, sname, status, city
s#	class	supplier_number
storage		supplier_number_mapping
mandatory		yes
status	class	status_number
storage		status_mapping
city	class	city_class
storage		city_mapping

Figure 2.1 Definition of SUPPLIER in KID's Knowledge Base

The entity "SUPPLIER" of figure 2.1 has the attributes s#, sname, status and city, which are defined later. The identifier is specified as s#. The access path to the supplier relation is via a procedure called supplier-number-mapping, and the s# must be given as a parameter to this procedure, since it is the key. The other attributes also have specific procedures for accessing the relation. The user view is established by the RETURN value. The entities PARTS and PROJECTS are defined in a similar manner.

supplier_number		
Super	value	s#
storage		supplier_number_mapping
class	value	primitive
level	value	classlevel
Inverse	class	SUPPLIER
get_value_method		s#_get_value

Figure 2.2: Definition of supplier\_number

The attribute `supplier_number` of Figure 2.2 is described as subordinate to `SUPPLIER`, and is a primitive object. (The term "primitive" in this model describes low-level objects which cannot have any attributes.) Note that the access method is specified as well as a method for determining how to access the `S#` attribute.

The `city` attribute, shown in Figure 2.3 is slightly different because it is an attribute of `SUPPLIER`, `PART` and `PROJECT`.

<code>city</code>		
<code>super</code>	<code>value</code>	<code>s#,j#</code>
<code>storage</code>		<code>city_mapping</code>
<code>class</code>	<code>value</code>	<code>primitive</code>
<code>level</code>		<code>classlevel</code>
<code>Inverse</code>		<code>supplier,projects</code>
<code>get_value_method</code>		<code>city_get_value</code>
<code>normalize_value_method</code>		<code>city_normalize</code>

Figure 2.3: Definition of City

The `normalize_value_method` is a procedure for adjusting for abbreviations, codes, etc., depending on the type of attribute. For instance, cities may be stored by code number, but displayed by name.

Finally, the `SPJ` relation is defined in Figure 2.4 as a relation that joins other relations based on a commonality.

```

SPJ
  Super      value Entity
  class      value non-primitive
  level      value classlevel
  attribute  value s# p# j# quantity
  key        value s# p# j#
  return     s# p# j# quantity
  s#         class supplier
    storage  supplier_mapping
  Inverse    class supplier
  p#         class parts
    storage  parts_mapping
  Inverse    class parts
  j#         class projects
    storage  projects_mapping
  Inverse    class projects
  quantity   class integer
    get_value_method quantity_get_value
    normalize_value_method quantity_normalize

```

Figure 2.4: Definition of SPJ

The preceding description of KID demonstrates the variety and detail of information that must be held at each node of a knowledge representation scheme. In addition, there is the ability to describe derived data, define multiple views, etc.

KID collapses the database mapping and semantic model into one module, rather than maintaining two separate ones. Also, the lexicon incorporates database-specific cases, thus tailoring it to one database at a time. There is no information available at this time pertaining to how successful efforts to transport KID to a new domain have been, although the authors claim this has been accomplished for several different domain, and that the process was "easy" (Ishikawa et al., 1986).

## 2.14 Summary

This brief overview has demonstrated the current trends in the development of transportable natural language interfaces. The basic idea is to find some indirect means of linking natural language expressions to application-specific concepts, such that acquiring new domain knowledge requires a minimum of effort. A direct mapping between language and domain concepts is not portable, although it may be an improvement for the human in that it alleviates the need to learn the access language of the DBMS. To bridge the gap between how the user thinks about the objects and relationships of the domain, and how the information is actually stored, some higher-level intermediary is required. The record schema of the database is inappropriate because it reflects the needs of the DBMS and is too biased toward the domain, with not enough knowledge about language, the user, a world model, and so on. Alternatively, it may be necessary to modify the record structure to accomodate natural language, but this is impractical with large existing databases.

There are several higher-level conceptual models available, including case frames, Conceptual Dependency diagrams, scripts, and semantic networks. The more general the formalism employed, the less direct is the language-domain



transformation, and the more (potentially) portable the system becomes. However, these knowledge-representation models can become large and difficult to manipulate. Further, there is no one "right" way to process semantic knowledge, so the procedures involved are often tailored to research needs and implementation issues. Systems like ASK and TQA place artificial restrictions on the language, and so try to reduce the task of the semantic processor.

It is a goal of this thesis to investigate the implementation of a transportable NLI using a general knowledge representation formalism to free the user from having to know how the database is organized, and to find a means of significantly reducing the restrictions placed on the user's language. The approach to be employed is to reduce the work that has to be done in the semantic analysis process, and this will be achieved by designing a parser which does not pass inefficiencies to subsequent modules, but makes "good", early decisions based on all syntactic and semantic information available to it.

## Chapter 3

### Parsing

In this chapter we briefly discuss the theoretical foundations of parsing, and how they are applied to understanding natural language. Then we survey some of the predominant parsing methods and grammars used by natural language understanding systems, beginning with syntax-only parsers and moving through the spectrum to semantics-only methods. The chapter ends with a brief overview of the parser used in QNL, and explains how it developed.

#### 3.1 Basic Linguistics

A language is defined in terms of an alphabet and a grammar. The alphabet is a finite set of symbols or tokens which appear in the language, such as "a, b, c, d, e, ...z", or "the, boy, hit, ball." The grammar describes how these symbols may be combined and it enforces a structure on the language.

A grammar may be formally defined as a four-tuple

$$G = (A, N, P, S)$$

where

A = The set of symbols which may appear in the sentences of the language (the terminal symbols.)

N = A set of nonterminal symbols which make-up partially derived sentences but may not themselves appear in the sentences of the language.

S = A start symbol which is a specified member of the nonterminal alphabet.

P = A finite set of rules which describes how the nonterminals may be converted to terminals (production rules).

A production rule has the general form

$$x \rightarrow y$$

where  $x$  and  $y$  are strings in the terminal and nonterminal sets of the grammar, and  $x$  is not an empty string. Such rules allow us to derive one string from another, and so convert the input string into a form that may be manipulated by a machine.

Phrase structure rules for natural language which describe the constituent structure of a phrase can be seen as production rules. The production rule for a noun phrase could be:

$$NP \rightarrow det + adj + noun$$

where NP means "noun phrase", "det" means a determiner such as "the", "a", "an", and "adj" means an adjective. (It is assumed the reader is familiar with the basic syntax rules of English, and so these will not be explained in this paper. Where complex rules are involved, explanations will be given).

Parsing is a process which uses various forms of knowledge to transform an expression into a data structure from which the meaning of the expression may be determined. A parser is a formal procedure for sentence analysis with no experience of the real world, so it relies on a set of rules (the grammar), and restrictions on how to apply those rules.

Suppose we have a grammar

```
G = ( {the, boy, hit, ball},  
      {S, NP, VP},  
      {S},  
      {S --> NP VP,  
        NP --> the boy,  
        NP --> the ball,  
        VP --> hit NP})
```

and the input string

s1. "the boy hit the ball"

By applying the production rules of the grammar to the input string, we can derive a representation of the sentence which may be depicted as a tree:

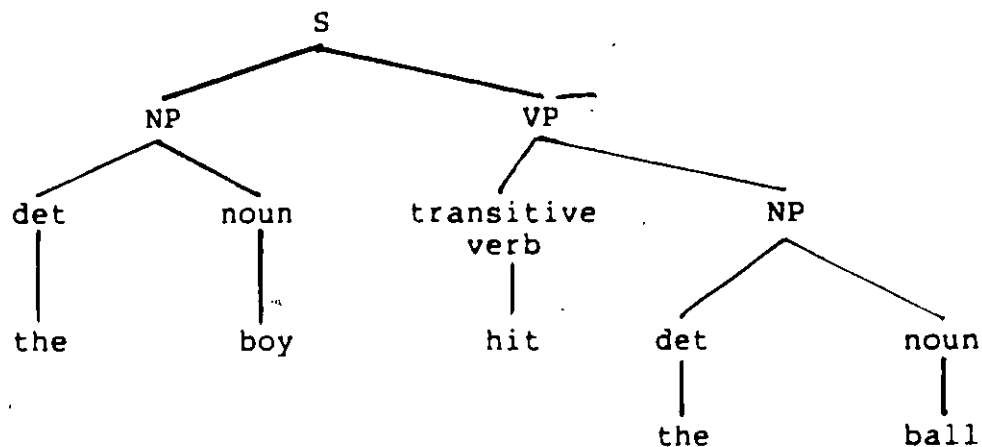


Figure 3.1: Syntax tree for the sentence "The boy hit the ball."

The terminal nodes of the tree are the tokens of the language, and the nonterminal nodes are the partially derived sentences. Such a structure, called a parse tree or syntax tree, is useful as a notation because it expresses the relationships between the words in a sentence (the terminal symbols) hierarchically arranged as constituents of

non-terminal symbols like noun phrases (NP) and verb phrases (VP). A structure such as this allows the sentence to be re-examined once it has been syntactically processed.

### 3.2 Syntactic Theories of Language

We say that a grammar "accepts" a sentence in a language if it is possible to begin at some initial state, apply the rules of the grammar to the input string, and arrive at one of the end states of the grammar at precisely the same time as all of the terminal symbols of the sentence have been treated by production rules.

Grammarians have proposed many theories and grammars concerned with ways of describing natural language (Chomsky, 1957, 1965; Katz and Fodor, 1964; Fillmore, 1968). Some of the most important work on modern theories of language is based on the work of (Chomsky, 1965), who described a hierarchy of four classes of grammar which place restrictions on the forms the production rules may take. Each level of the hierarchy is a proper subset of the level(s) above it. A language fits a description in the hierarchy corresponding to the most restricted set of production rules which is adequate to describe it.

Type 0 languages

(recursively enumerable languages)

This is an unrestricted language in which there are no restrictions on production rules; production rules are not appropriate for describing this type of language.

Type 1 languages

(context-sensitive languages)

The length of left-hand side of each production must not be greater than the length of the right-hand side

Type 2 languages

(context-free languages)

The left-hand side of each production has exactly one nonterminal symbol

Type 3 languages

(regular languages)

Each production has only a single nonterminal symbol on its left-hand side, and either a single terminal or a single terminal followed by a non-terminal symbol on its right-hand side.

Figure 3.2: Chomsky's hierarchy of languages (Chomsky, 1965)

Type 0 languages are arbitrarily complex and so building a computer model to parse such a grammar could be an intractable problem. There are computer models which will efficiently parse regular grammars and context-free grammars (Hopcroft and Ullman, 1969, Earley, 1970, Aho and Ullman, 1972), and much work in the translation of computer languages has focused on context-free languages.

However, Chomsky determined that natural language is neither context-free nor regular. Consider the expression:

s2.                      Time flies like an arrow

which has several possible interpretations, depending on the context. It could be imperative or declarative; it could be a command to time the flight of flies as one would time the flight of an arrow, or it could be a command to assume the role of an arrow, and in this manner, time the flies as they do something, or it could mean that time moves swiftly, or that a certain breed of flies called "time flies" are somehow attracted to an arrow, or...

More work has been done on context-free parsing algorithms than context-sensitive parsing algorithms, and if exception cases like "time flies..." are ignored, context-free parsing algorithms can be adapted to natural language parsing. For a discussion of context-sensitive parsing, see (Woods, 1970).

}



### 3.3 Transition Network Grammars

Consider the following production rules of a regular grammar, which allows a noun phrase of the form "determiner adjective\* noun":

```
NP --> det + REST
REST --> adj* + NOUN
NOUN --> noun
det --> the
adj --> long, wooden, heavy...
noun --> hockey, stick...
```

(the symbol "\*" is the Kleene-star operator indicating repetition)

The path a parsing mechanism follows to apply the production rules may be described by a directed graph where the nodes represent states, and the arcs show how one state may change into another. The graph for a regular grammar may be modeled as a Finite State Machine, with one state marked as the start state, and transitions occurring from one state to another. The number of end states may be greater than one.

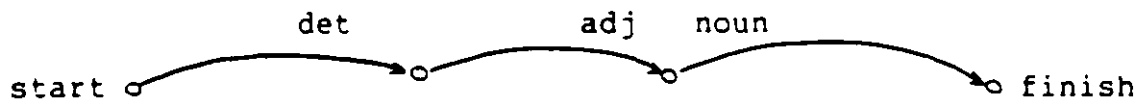



Figure 3.3: A Finite State Transition Network for a noun phrase of a regular grammar

The Finite State network of Figure 3.3 will accept the noun phrases



the long stick  
the long heavy wooden stick  
the heavy hockey

but it will not accept

the hockey stick  
heavy stick  
long heavy wooden hockey stick

To represent a natural language as a Finite State Machine would require a separate path through the network for every possible combination of constituents. This is impractical since the range of possibilities is infinite. Although a range of language expressions could be represented by a finite state network, such a language would not be "natural". A regular grammar does not allow embedding, so such natural language constructs as relative clauses, subordinate clauses, prepositional phrases, etc., would require a very large and complex network.

We can expand the capabilities of our grammar by allowing some states to be ignored, and by allowing some of the arcs of the Finite State Transition Network to be named as states, thus introducing the capability of recursion. Such a network is a Recursive Transition Network (RTN), and it has the power to recognize a context-free grammar (Chomsky, 1965).

$S \rightarrow NP$   
 $NP \rightarrow (det) + (adj^*) + noun^* + (PP^*)$   
 $PP \rightarrow prep + NP$   
 $det \rightarrow the...$   
 $adj \rightarrow long, heavy, wooden...$   
 $noun \rightarrow hockey, stick, cabinet...$   
 $prep \rightarrow in, with, for...$

Figure 3.4: Production rules for a context-free grammar of English.

An RTN that will accept relative clauses, embedded phrases, and so on is shown in Figure 3.5.

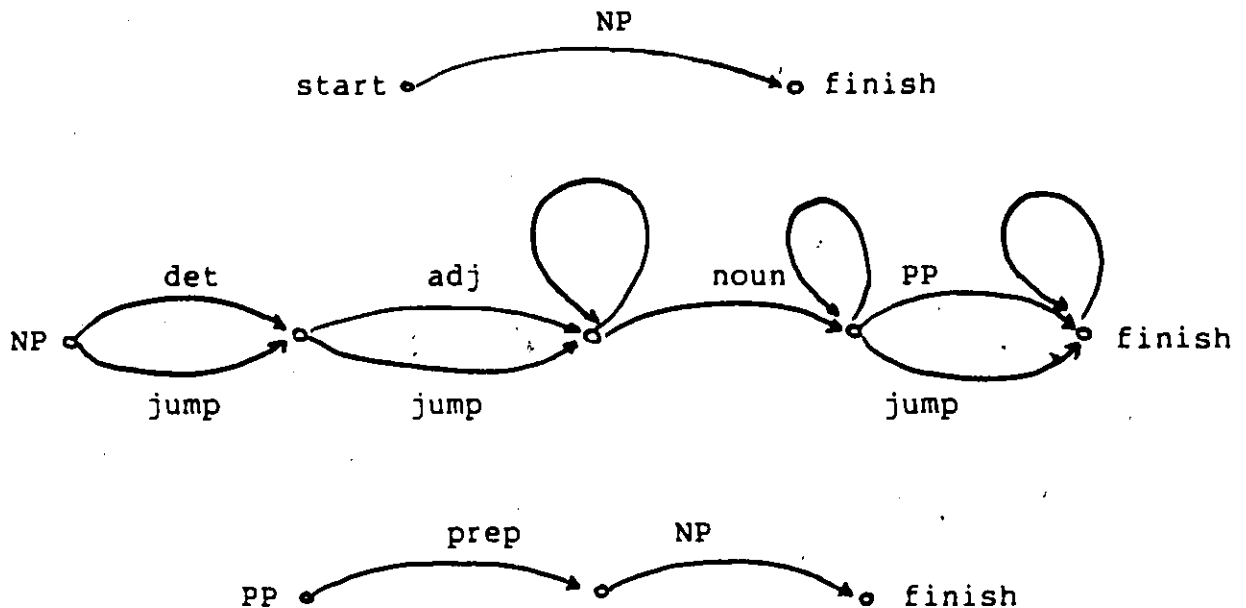


Figure 3.5: Recursive Transition Networks for embedded noun phrases and prepositional phrases

The following phrases are accepted by the context-free grammar of Figure 3.4:

the hockey stick in the wooden cabinet  
 long wooden hockey stick cabinet  
 the stick for hockey in the long wooden cabinet with...  
 heavy stick in the wooden hockey with the long cabinet...

Although these phrases are accepted by the grammar, the RTN only accepts or rejects a sentence based on a given grammar; it does not build an internal representation. Also, it is only sufficient to process a grammar based on phrase-structure rules, and natural language is much more complex.

The meaning of a sentence is not necessarily obvious from the words and the word order in a sentence. Chomsky, among others (Katz and Fodor, 1964), extended his early work on the theory of syntax to attack this problem, and the result was Transformational Grammar (TG) (Chomsky, 1965).

The thrust of TG is that the tokens actually appearing in a sentence represent a "surface structure", and that this surface structure may be manipulated by syntactic rules to achieve a meaning representation called a "deep structure". Two sentences which have the same meaning should have the same deep structure even if their surface structures differ. So, the sentences:

s3      Is the boy sleeping  
s4      The boy will sleep

will have the same deep structure, with markers explaining the differences in their surface structures.

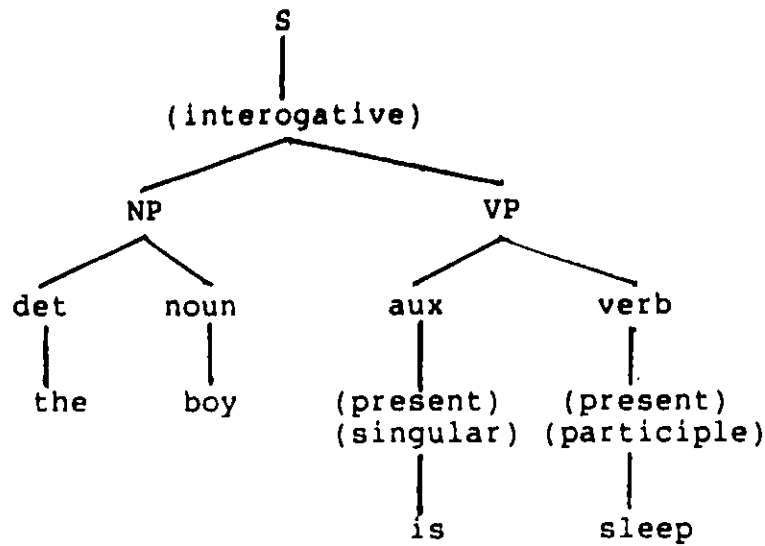


Figure 3.6: Syntax tree for sentence s3: "Is the boy sleeping?"

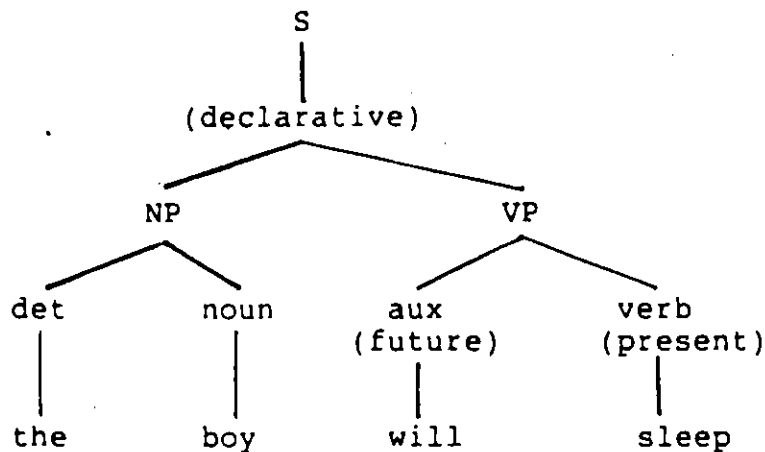


Figure 3.7: Syntax tree for sentence s4: "The boy will sleep."

We need the ability to remember those intermediate structures which have already been recognized, to recognize "features" of the sentence (such as the mood: interrogative, imperative, declarative; tense, noun-number, verb-number,

etc.) We need tests for noun-verb agreement, and elaborate rules for adding and deleting constituents (eg., implied "you" in imperative statements), inverting word order, etc. To accomplish this, we must add to our RTN the ability to perform these operations, store intermediate structures, and group them together in an appropriate fashion to produce a meaning representation.

### 3.4 Augmented Transition Networks

An Augmented Transition Network (ATN) (Woods, 1970) is an extension of an RTN that has a set of registers which hold parts of the sentence as it is being built. An ATN allows the contents of the registers to be swapped, transformed, and so on, and is able to accommodate the principles of Chomsky's Transformational Grammar. An ATN is a parsing formalism which is capable of transforming the surface structure of a sentence (the words as they actually appear) into a deep structure (a "meaning" representation).

The ATN is a "top-down" or "hypothesis-driven" parser; that is, it arranges its production rules based upon some a priori knowledge, then blindly hypothesizes that each rule in succession will succeed.

It is also a nondeterministic machine. At arbitrary points in any sub-network, there can be a choice as to which possible states the parser may move to next, whereas a deterministic parser would have a unique choice at each state. The "jump" arcs shown in Figure 3.5 are in effect empty moves which are made without scanning the next token. These jumps can be invoked when the entire sentence has been scanned, but the parser is not at a halt state. For instance, the production rules of Figure 3.4 allow an optional prepositional phrase as part of a noun phrase, and an optional noun phrase embedded in a noun phrase if the first noun phrase contains a pronoun (eg. "his red book"). If the optional phrase does not exist, the jump arcs allow the NP subnet to move to the next state to examine the current token.

The justification for nondeterminism is that decisions may be made based on very limited data. An ATN uses only what has been parsed so far, and a hypothesis about the next word or phrase. A deterministic parser, on the other hand, would make use of more information, requiring more processing at each node in the network, but with a goal of choosing only one path through the system. An ATN will have to generate several candidates if ambiguity is encountered, and so can produce all possible syntax trees for a given input.

One of two techniques are commonly used in a nondeterministic parser: backtracking and parallelism (or pseudo-parallelism). A backtracking strategy builds on one rule as far as possible in a depth-first manner, then backs-up to each alternative, and follows that in a depth-first manner, until all possibilities have been scanned. Parallelism implies a breadth-first search through the subnets. In this manner, all possible alternative interpretations are discovered more or less simultaneously. So if the choice at a given state is not unique, the parser will pursue all alternatives in parallel.

ATN's have become one of the standard natural language parsers in North America due to their conceptual clarity, efficiency, and success (Nenova, 1985). The grammar used by an ATN consists of rules detailing the allowable ordering of tokens in a sentence, and procedures which may execute when the rules succeed. These grammar rules can be separate from the processes which scan and manipulate the rules, so the grammar is independent of the parser (Friedman et al., 1971; Ritchie and Thompson, 1984). This allows the grammar to be tailored to meet the needs of a specific theory or application. The ATN formalism has been used extensively in NL systems, including LUNAR (Woods, 1970), PLANES (Waltz, 1976, 1978), INTELLECT (Harris, 1977, 1978), TEAM, (Grosz et al., 1982, 1983, 1987), DATALOG (Hafner and Godden, 1985).



In addition to the basic structure of the RTN, an ATN has a set of registers to hold constituents. The arcs of the network may be labeled with either specific words, "pushes" to other sub-networks, procedures to perform tests on the current token and the constituents already stored in registers, procedures to build structures, or any combination of these tests. It is this ability to define arbitrary tests on the arcs and so control its own flow that gives the ATN the power of a Turing machine (Rich, 1983, pg. 315).

### 3.5 The Workings of an ATN Parser

An ATN parser is an "all-paths parser" that produces all possible interpretations of a sentence.

Consider the question:

s5. Which suppliers sell parts to the project in London?

Using the production rules of Figure 3.8, depicted graphically in the RTNs of Figure 3.9, parsing of sentence s5 begins as the first rule in the grammar, "S --> NP + VP" is tried. The subnet to build a noun phrase is activated, and the first word in the sentence is subjected to a sequence of tests to see if it is either a determiner, an adjective, a noun or a pronoun.

```

S --> NP + VP
S --> VP
VP --> Verb* + (Adv)
VP --> (Aux) + Verb* + (NP)
NP --> (Det) + (Adj*) + noun* + (PP*)
NP --> (Adj*) + Pronoun + (noun*) + (PP*)
PP --> Prep + NP*
Det --> the, a, an,...
Aux --> can | must |...
Adv --> slowly, quickly,...
Verb --> supply, sell,...
Adj --> big, red, blue,...
Prep --> in, of, to, with,...
Pronoun --> she, it, which,...
Noun --> supplier, London, part, project,...

```

Figure 3.8: Phrase-structure rules for an ATN

The pronoun test succeeds so the next set of tests in the subnet is tried: first the presence of a noun is hypothesized, and this succeeds. A second test for a noun fails, and so the presence of a prepositional phrase is hypothesized.

For the PP test, we push to the PP subnet and test to see if the next token, "sell", is a preposition. It isn't, so the subnet fails, and control is returned to the NP subnet. The last jump arc is followed to exit the NP subnet. At this point, the contents of the noun phrase would be inserted into a temporary register.

At the termination of the current NP, the subnet is popped to return control to the previous subnet, which is popped to return to the S--> NP + VP rule.

Since we are using a depth-first strategy, the next subnet to be encountered will be the VP network according to the  $S \rightarrow NP + VP$  rule. A pseudo-parallel machine would evaluate the next S rule in the grammar,  $S \rightarrow VP$ , until all of the S rules were exhausted, then it would return to the first S rule which partially succeeded, and begin evaluating the as-yet unresolved subnets.

The next stage of the  $S \rightarrow NP + VP$  rule is tried. Again, a sequence of subnets is called in an attempt to build the phrase structure defined by each subnet. The first VP rule calls another VP rule, which looks for an auxiliary. If the current token is not an auxiliary then we jump to the next node of the subnet to test for a verb. "Sell" is a verb so it is stored in a verb register, and the next token is subjected to the next test in the VP network, which is a test for a NP.

The NP will succeed with the token "parts", and the PP subnet will be called. The PP subnet's first rule succeeds when the preposition "to" is found, and then another NP subnet is pushed. Processing continues as above.

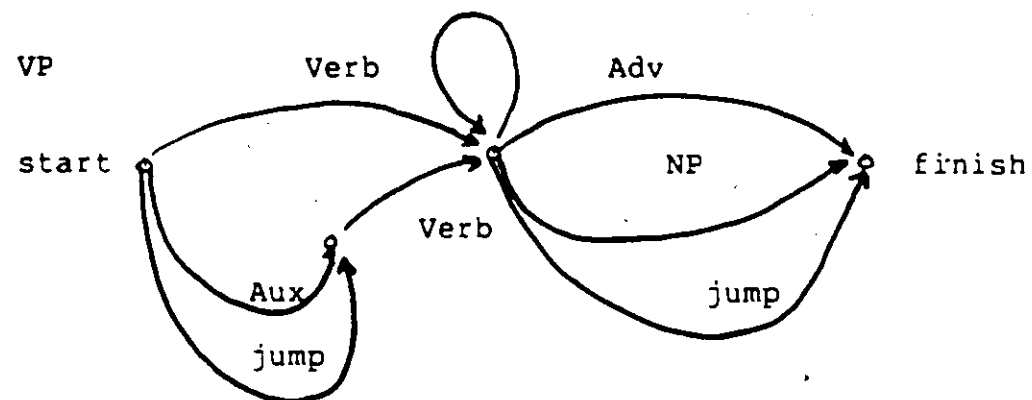
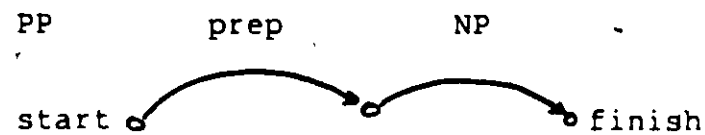
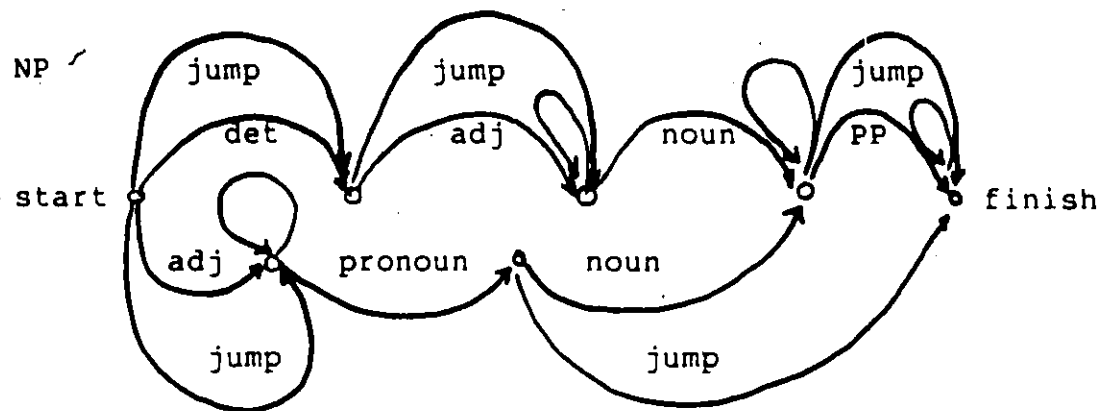


fig 3.9 An ATN for a simple grammar of English

Ambiguity is introduced with the start of the second prepositional phrase because there are two rules which could apply:

PP --> Prep + NP\*  
NP --> (Det) + (Adj\*) + noun\* + (PP\*)

According to these rules, the phrase "in London" of sentence s5 could be attached as a modifier of the PP "to the project", or as a modifier of the NP headed by "parts".

The ambiguity inherent in natural language causes some input strings to have more than one syntactic interpretation, and thus it may be necessary for a parser to produce more than one parse tree so that subsequent semantic processing may resolve the ambiguities. Such a difficulty arises with nested relative clauses and prepositional phrases, conjunctions and disjunctions, pronoun references, etc. For example:

List parts available in Halifax or Toronto and Montreal.

Get me all suppliers and projects in London and their status.

Which projects use red parts which are not used by projects which use green parts or blue parts parts which are sold by supplier s1?

Figure 3.10: ambiguous sentences

In a grammar more complete than the phrase structure rules of fig 3.8, there may be several levels of nesting of the subnets, and so the top-down approach of an ATN could waste a lot of time trying to build structures which are not present. For instance, in sentences which contain some lexical or syntactic ambiguity, the ATN will not only have to test many possibilities, it may also build incorrect intermediate structures.

Consider the garden-path sentence:

s6.           Can the old man the boats?

The word "can" could be mistaken for a noun, the word "old" could be mistaken for an adjective, "man" could be mistaken for a noun. But these mistakes are not noticed until the ATN parser has already built a temporary structure for part of the sentence. When one hypothesis fails, a backtracking parser dismantles and discards the incorrect structures.

To achieve a reasonable level of efficiency, ATN's typically build a table of well-formed substrings to minimize the amount of backtracking. The idea is that the parser should not back up more than necessary if a subnet fails.

The arbitrary tests on the arcs of an ATN may include some semantic checks, but these are often related only to noun-verb number agreement, gender agreement, etc (Tanimoto, 1987, pg 349).

The justification for non-determinism is that it allows decisions to be made based on very limited data. An ATN uses only what has been parsed so far and a hypothesis about the next word or phrase. If a parser finds multiple interpretations of a sentence, then the semantic analyzer must manipulate these multiple interpretations. Therefore, an all-paths parser not only can waste time generating several incorrect parses, but the semantic component is made less efficient because it must manipulate these wrong interpretations.

An ATN parser can be very expensive to operate if a large amount of backtracking or pseudoparrallelism is required. If extensive semantic checks are incorporated into the tests on the arcs, then the grammar can become very complex, and the cost of processing those semantic checks may be comparable to that of backtracking without the semantic checks. Woods objected to semantic constraints too early, since it was inefficient (Woods, 1973). Ginsparg says most of the semantic choices can be eliminated more efficiently

by syntactic rules, and at less cost since there is less information involved in making the decision (Ginsparg, 1978).

There is no way to use heuristic functions in an ATN unless the grammar is itself a heuristic grammar. So the flow of control is blind and not flexible.

Finally, as with any transition network grammar, the parser must find an exact path through the network from the start state to a halt state. If any of the words or grammatical structures are unknown to the grammar or parser, as may happen for instance in the case of user errors, the parse will fail. There is no possibility for partial matching.

### 3.6 The Chart Parser

The Chart Parser is a bottom-up, non-deterministic parser which finds all possible parses of a sentence based on a given grammar. This is an extension of the work of (Earley, 1970), and it is Kay (Kay, 1973) who is credited with its original implementation. More recently, (Thompson, 1981) has contributed to the concept. The Hafner and Goddard system uses the Chart as a pre-processor, followed by a backtracking parser (Hafner and Goddard, 1985). The EUFID system also uses the Chart Parser (Templeton and



Burger, 1983). Their chart parser only implements a simple set of phrase structure rules, because their research concentrated on semantic understanding.

The Chart is a variation of a syntax tree which is able to represent multiple interpretations at once, as well as partially-built structures. It is a graph with the arcs representing both the terminal and non-terminal symbols, and the nodes representing the points between the tokens.

The bottom-up approach is akin to backward chaining, generating candidate structures based on the words and expressions present in the sentence. With such a method, the prepositional phrase "in London" will cause all rules, and only those rules, ending in a prepositional phrase, to fire. A top-down parser would blindly hypothesize the existence of, say a noun phrase or relative clause, depending on an a priori ordering of the rules.

A sentence is parsed by constructing arcs that span increasingly larger sections of the graph. As the arc advances through the chart, production rules are applied to the symbols encountered. If an ambiguity occurs, i.e. more than one production rule may be applied, then all possible

interpretations are constructed. If some of these arcs are later found to be incorrect, they are not destroyed, only abandoned.

When a sentence is input, a preprocessor matches the input tokens with a lexicon, or dictionary, and an initial chart is built. Figure 3.12 shows the initial chart for sentence s5.

1. S --> NP + VP
2. S --> VP
3. VP --> VP + (Adv)
4. VP --> (Aux) + Verb\* + (NP)
5. NP --> (Det) + (Adj\*) + noun\* + (PP\*)
6. NP --> (Adj\*) + Pronoun + (noun\*) + (PP\*)
7. PP --> Prep + NP\*
8. Det --> the, a, an,...
9. Aux --> can | must | ...
10. Adv --> slowly, quickly,...
11. Verb --> supply, sell,...
12. Adj --> big, red, blue,...
13. Prep --> in, of, to, with,...
14. Pronoun --> she, it, which,...
15. Noun --> supplier, London, part, project,...

Figure 3.11: The same set of phrase structure rules as listed in Figure 3.8, numbered for convenience.

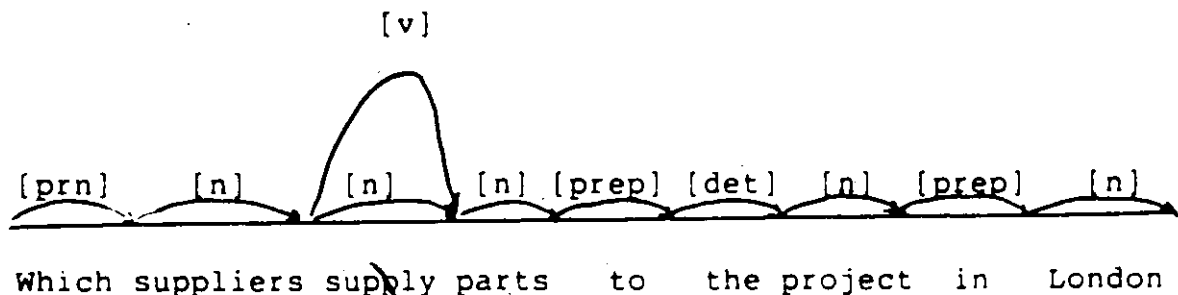


Figure 3.12: Initial chart of the Chart Parser. All arcs are inactive at the start

The arcs between the nodes are either active or inactive. An inactive arc is one which is not looking for more constituents in the chart; it is complete. Terminal symbols are represented by inactive arcs. The token "supply" has been recognized as potentially playing the role of either a verb or a noun, so an arc for both possibilities has been constructed.

In our diagrams, we will adopt the convention of indicating found constituents inside square brackets, and missing constituents before the brackets. In the initial Chart there are no missing constituents.

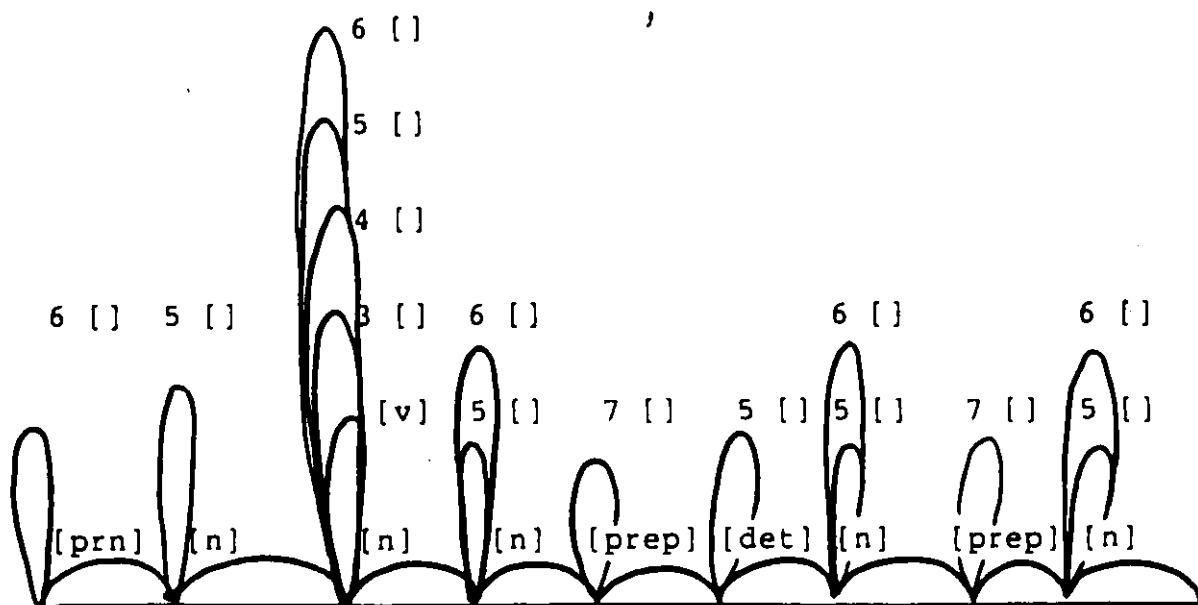
Once the initial chart is constructed, the parser begins constructing active arcs. An arc is created if, at any given node, there exists an inactive arc which could possibly allow a newly constructed arc to succeed.

So, each rule in the grammar is tried against the first token, "which", and any rule of the form

X --> ( ) + ...pronoun .....

will cause an arc to be built. The rule states that either a pronoun is the first constituent of the phrase, or there are optional constituents which could precede the pronoun. We expect this to be a NP rule. The next node will have a

NP arc built, the third node a NP and a VP node, etc. Note that if several NP rules exist, one active arc for each rule will be constructed. The parser has no way of knowing if the current node is the start of a phrase, or the middle. Rather than guessing, it assumes a worst case, and builds the structures accordingly.



which suppliers supply parts to the project in London

Figure 3.13. After a first pass over the initial chart, a set of active, empty edges is built. The numbers refer to the rule numbers in the sample grammar of fig 3.11.

Each active arc will attempt to stretch across the nodes to its right until it becomes inactive; that is, until it has found all of its constituent parts. It does this by

comparing those constituents in the rule remaining to be completed with the category of the inactive arcs starting at the current node. There could be many inactive arcs at each node, so the checking could become a large task.

For instance, the rule  $NP \rightarrow (Det) + (Adj^*) + noun^* + (PP^*)$  will test the nodes at its immediate right for optional determiner and adjective arcs which are inactive, and for a compulsory inactive noun arc. If the noun arc is found, then it will continue looking for an optional PP arc. If a compulsory inactive arc is not found, then the rule fails and the active arc is abandoned. Inactive arcs (in other words, those for which all compulsory constituents were found), are added to an agenda as they are built.

Manipulating this agenda as a LIFO queue produces a depth-first searching process; the most recently added inactive arc is the first that will be tested by the next active arc. If the agenda is used as a FIFO queue, the search strategy is breadth-first. The Chart parser is designed to find all possible parses, so the choice of depth-first vs breadth-first searching on the final outcome only matters if the Chart is to be modified, say, to produce only a small number of parses.

The heart of the Chart Parser is the Fundamental Rule (Thompson and Ritchie, 1984, p. 248), which is as follows:

Whenever the far end of an active arc A, and the near end of an inactive arc I meet for the first time, if I satisfies A's conditions for extension then a new arc is built as follows:

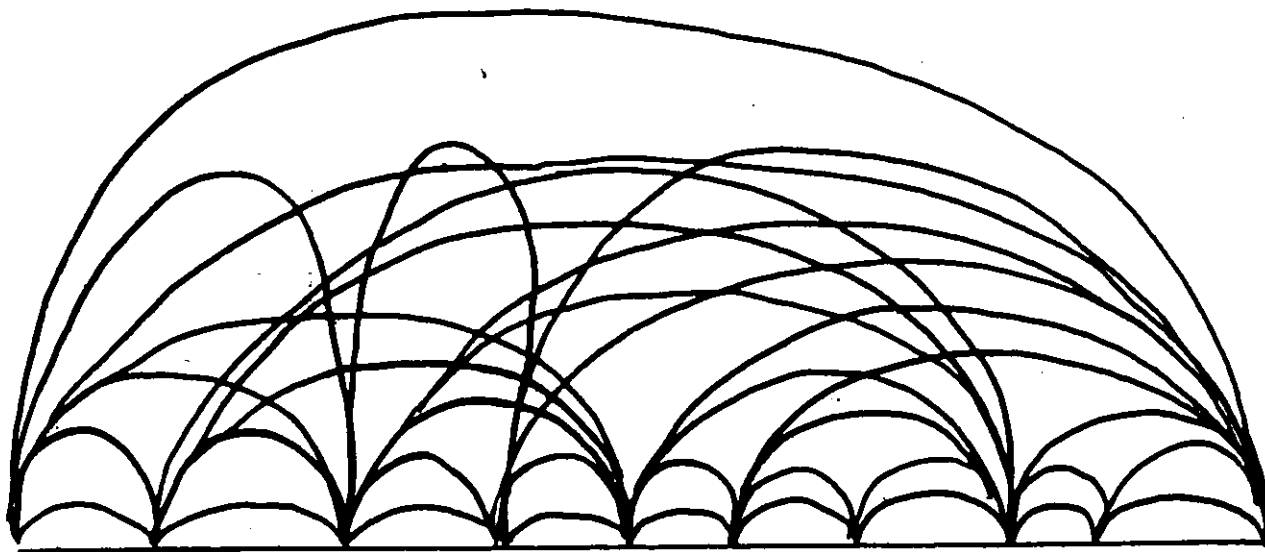
- Its near end is the near end of A.
- Its far end is the far end of I.
- Its category is the category of A.
- Its contents are a function (dependent on the grammar) of the contents of A and the contents of I.
- It is inactive or active depending on whether its extension completes A.

Algorithm 1: The fundamental Rule of the Chart Parser.

The grammar rules supplied to the Chart Parser are independent of the parsing mechanism, so this is a useful tool for developing a grammar. Since arcs are never destroyed, the parser can always come back to incomplete arcs to determine if recent actions will now allow them to be completed. It guarantees to find all possible parses with respect to a given grammar.

Even very ambiguous sentences such as "garden path sentences" may be correctly parsed. The success of the linguistic coverage of such a parser is totally dependent on the grammar rules, so to extend or alter the grammar, a linguist is required rather than a computer programmer.

The concept is simple, and the code to manipulate the graph structure is neither complex nor large. However, it is computationally expensive to operate. It builds all possible partial parses, even those which are redundant or which cannot contribute to an eventual solution. Since it never destroys any of the arcs, it tends to use a lot of memory. As the number of inactive arcs increases, the number of tests required at each node also increases. Further, as the size of the grammar increases, the running time increases exponentially. More intermediate structures are built, and so the number of tests increases, and the number of partial successes also increases.



which suppliers supply parts to the project in London

fig. 3.14 Final chart for our sample sentence

### 3.7 The Marcus Parser

← An LR parser is a type of parsing machine developed for context-free languages like programming languages. It is a table-driven parser which operates according to a set of rules which determine the unique action to be taken at any stage in the process. LR parsers have not often been used for natural languages because of the ambiguities of natural language which make it context-sensitive. Two approaches have been used to employ LR parsers in natural language understanding: one that allows multiple options and the production of more than one parse tree (Tomita, 1984), and the other which uses an attention-shifting strategy to determine a unique option at any given time (Marcus, 1980).

An LR(k) parser is one which processes a LR(k) grammar. A grammar,  $G$ , is LR(k) if, when examining a parse tree for  $G$ , we know which production rule is used at any interior node after seeing the boundary to the left of that node, what is derived from that node, and the next  $k$  terminal symbols (Aho and Ullman, 1972, pg. 379).

The Marcus Parser is a left-to right, bottom-up deterministic machine for parsing the syntax of a natural language. Marcus' theory was that any natural language could be parsed by a system which operates "strictly



deterministically", as opposed to a pseudo-parallel or a backtracking machine. The Marcus Parser never backtracks, and produces a single interpretation for any given input string.

It extends the concepts of a  $LR(k)$  parser in that it uses a lookahead buffer which can handle not only single words but also entire phrases. It is not as general as a  $LR(k)$  parser because it places restrictions on the way grammatical information is represented in state descriptions (Berwick, 1985, pg. 13).

An  $LR(k,t)$  parser is one which is  $LR(k)$  and where the lookahead string is allowed to contain non-terminals as well as terminal symbols; that is, incomplete sub-strings. The maximum size of the lookahead buffer in the Marcus parser is 3 elements (less 1 for the current token), so it is an  $LR(2,2)$  parser (Berwick, 1985, pg. 325). It is deterministic in that once a decision is made the machine remains committed to the decision. Also, at any point in the parsing process, at most one parsing rule may apply.

The output of the Marcus Parser is an annotated tree structure as proposed by Chomsky, but each node has a set of features associated with it, describing the function of that node with respect to other nodes.

The parser relies on three elements: a stack and a buffer which make up the interpreter, and the grammar. The grammar rules are independent of the interpreter. They are constructed as a set of if-then rules which are grouped into packets to control whether entire sets of grammar rules should be made available for matching against items in the buffer. The packets are in 1-to-1 correspondance to the non-terminal phrase structure rules. For now, we will use the same set of phrase structure rules as in the previous examples (see Figure. 3.11).

Rules in a packet are heuristically ordered according to expectations of the type of phrase structures which would normally occur given the current state. The selection of these rules may be based on statistical information, or more general "rules of thumb". Since this is a bottom-up parser, packets are invoked depending on the constituents found in the input string, rather than following a strategy of blind hypothesis.

The words in the input string are transferred one at a time to the buffer. Each rule contains a pattern and an action. If the pattern matches any item in the buffer, then the corresponding action is performed. These actions can create

new nodes and push them onto the stack, remove elements from the buffer and attach them to the stack, and pop the top of stack into the buffer.

The stack contains structural nodes in search of descendants, for instance a S node waiting for a VP, or a NP waiting for a noun. The stack will also contain those rule packets which are pushed when processing is suspended (attention is shifted to another part of the sentence). The packets at the top of the stack are active, while those lower in the stack are temporarily inactive.

A parse proceeds from left to right through the input sentence with the interpreter executing any grammar rule which matches the current environment of the parse. The environment is determined by features of the current tokens, portions of the already built tree, and the information from the lookahead buffer.

This lookahead facility is used to determine which rules within packets should be attempted first. In the event there is potential ambiguity, the parser moves to some other part of the sentence, builds other structures, then returns to the ambiguous point to choose a single path. This "attention-shifting" continues until sufficient information is available to make an irrevocable decision.

The decision is based on what Marcus calls "prediction". A top-down parser blindly guesses about the structure of the sentence, but the Marcus parser gathers information from several sources before committing itself to a decision.

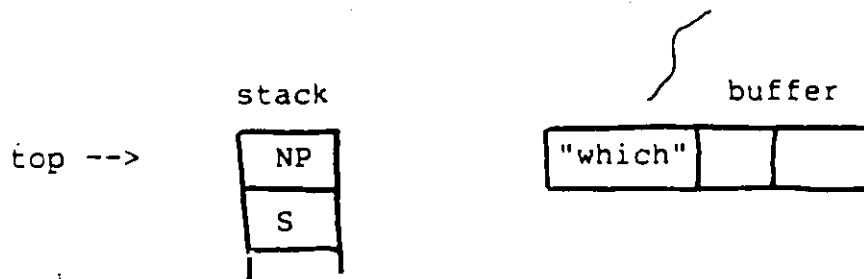
Consider the following question:

s7. "Which suppliers sell red parts?"

A Marcus Parser would process sentence s7 as follows: The stack and buffer start empty. An initial set of rules is active to recognize the first elements of the input.



The token "which" is recognized and brought into the buffer. "Which" is a special form indicating an interrogative statement, and it is the first element of a noun phrase. The NP node is pushed onto the stack, and the S rule is deactivated.



The Marcus Parser deviates from the standard concept of bottom-up parsers in several respects. In a bottom-up parser, all constituents of a phrase structure rule must be satisfied before the top level rules can be attached.

For instance, for the rule

S --> NP + VP

both the NP and VP rules must be satisfied before the S rule is activated. The Marcus Parser predicts what will happen, based on the current state and the lookahead buffer.

If a top-down parser like the ATN were to use a lookahead facility, it would be necessary to scan ahead through an arbitrary number of subnets. For each subnet, several levels of hypothesis may have to be constructed, (i.e. several subnets invoked) and later rescinded. There would not be any gain over backtracking.

If the lookahead operations were not restricted, then the Marcus Parser would be as non-deterministic as an ATN. The lookahead facility acts as a filter for the determinism hypothesis. Given the presence of certain words and phrases, a set of candidate rules is generated. The lookahead scans other words and phrases to determine which of the candidates should have priority.

The Marcus parser attempts to gain efficiency over the ATN and Chart by being sensitive to all the information available to it. It avoids building the arbitrary levels of hypothesis of an ATN parser, and only builds the structures which it predicts will succeed.

The concept of prediction is foreign to a strict bottom-up parser which would wait until a structure is built before recognizing it.

Like the ATN parser, the Marcus Parser cannot handle phenomena requiring extensive semantic processing, such as conjunction, disjunctions, etc.

Up to this point we have been assuming the same set of phrase structure rules is used by all natural language parsers, but such is not the case. In fact, many NL systems, including the one explained in this thesis, develop a unique set of production rules and transformational rules, chosen for reasons of efficiency, applicability to the domain and the purpose of the system. These sets of parsing rules may or may not incorporate semantic knowledge.

"Any parser based upon the determinism hypothesis must use semantic analysis to aid in decision-making to diagnose alternative structural possibilities" (Marcus, 1980, pg.

229). Marcus used a form of case grammar as originally proposed by (Fillmore, 1968) to assist in disambiguation.

### 3.8 Case Grammar

Case grammar is an attempt at developing a grammar which recognizes not only the structural relationships between words in a sentence, but the functional relationships as well. The idea is to make the relationships between a predicate and its arguments explicit, and so this approach is used to guide the parsing process.

A case grammar may be defined by the production rules:

```
S --> M + P
M --> tense, aspect, form, mood,
      time, essence, modal, manner,
P --> V + C1 +...+ Cn
V --> sell, supply, use,...
Ci --> K + NP
K --> null | prep
prep --> to, with, in, of,...
NP --> (det) + (adj)* + noun + (S | NP)*
```

Figure 3.15: Production rules for a Case Grammar  
source : (Harris, 1985, pg. 184)

A sentence consists of a series of terms making up the modality M describing the aspect of the sentence, and a proposition P. The proposition consists of a verb and an associated set of cases each of which would include the

agent and object(s) of the verb. Each proposition has a Kausus K which may be either null or a preposition, followed by a noun phrase.

The treatment of verbs is the real heart of case grammar as applied to computer-based parsing systems. Each verb has an associated list of case frames which describe the subject, object(s), etc, that may be associated with the verb. Any verb may have more than one associated case frame, and each case frame may be used by several verbs.

Marcus used the following set of cases (Marcus, 1981, pp 310-322):

- Agent : instigator of an action
- Instrumental : thing used to perform the action
- Locative : location of the action
- Dative : recipient of the action
- Neutral : object being acted upon

This is by no means a definitive set; other not dissimilar sets include those of (Fillmore, 1968, Simmons, 1973; Stočkwell, Schacter and Partee, 1973, and Bruce, 1975).

These cases are grouped together into a case frame. Each case frame has obligatory cases, optional cases, and



disallowed cases. For example the verb "sell" may have the case frame

```
sell (agent -required- seller
      (instrumental -not allowed-
      (locative -optional- place of transaction
      (dative -optional- buyer
      (neutral -required- object being sold)
```

The implementation of a Case Grammar involves the parser identifying the main verb, and then mapping the cases to their appropriate roles in the case frame associated with the verb. One set of syntactic rules to accomplish this mapping has been developed by (Stockwell, Schachter and Partee, 1973) and is used by Marcus.

Case grammar is not without its flaws, the most serious of which is that many of the "rules" are intuitive, and do not hold for all instances. Further, attempting to acquire semantic knowledge using only syntactic rules cannot possibly succeed in all instances.

Fillmore relied heavily on prepositions to provide clues as to the semantics of noun phrases. The words "in", "under", "on" indicate the locative case, "by" indicates an agent, "to" indicates a recipient, etc. But we cannot rely on such simplistic rules. "To" can also be used to indicate

location (to Toronto), "in" can be used for all kinds of abstract concepts (in trouble, in case, in the future, etc.), as can "on", (on time).

There are a large number of "content-empty" words in English which provide no semantic clues as to the case of their associated phrases, like "from" and "of". Finally, if we associate a number of case frames with each verb, how many do we associate with "to be", and "have"? These words occur so frequently in database access that they are the only verbs allowed by ASK (Thompson and Thompson, 1985).

Despite these objections, Case grammars have been used by several natural language interface systems, including KID (Ishikawa et al., 1986), TEAM (combined with an ATN (Grosz et al., 1987), DYPAR II (Carbonnell et al., 1983), and TQA (Damereau, 1985).

### 3.9 Semantic Grammar

"Syntax is that aspect of language that is so general and so basic as to reflect the most universal relational features of the kind of world it is designed to talk about" (Siklossy and Simon, 1972, pg. 49). To reduce the problem of syntactic ambiguity, some natural language interfaces have included semantics in the lexicon to build-in various

elements of meaning, i.e. constraints on agreement, expectations, etc. As a grammar becomes richer it becomes more "semantic" because as general rules become exhausted the semantics associated with the lexical entries incorporate more and more markers which describe what the term is about. This is what Case Grammar accomplishes.

A semantic grammar is a way of performing syntax and semantic analysis at the same time.

Formally, there is no difference between a semantic grammar and a context-free grammar. But in actuality, the non-terminal symbols of a semantic grammar represent more specific categories of words than those of an ordinary context-free grammar for natural language. Instead of having a general symbol "noun", a semantic grammar might have a more explicit symbol, like "tool", "pump", "screw", etc., and verbs like "tighten", "grasp", etc.

The parse of a sentence with a semantic grammar contains the information necessary to build a semantic representation of the sentence. A semantic grammar is useful for accessing a database, because the non-terminal symbols can be used to represent the objects, attributes and relationships within the domain.

There are several potential problems: The number of phrase structure rules could become quite large and therefore compromise the efficiency of the interface. More importantly, a system built on a semantic grammar may be completely non-portable: to move to a new domain would require a redefinition of the lexicon, the phrase structure rules, and the semantic analysis module, as well as the DB mapping.

Semantic grammar has been used by several natural language systems, including LADDER (Hendrix et al., 1978), EUFID (Templeton and Burger, 1983), and PLANES (Waltz, 1977, 1978).

### 3.10 Conceptual Dependency Theory

The other extreme of the syntax-semantics continuum is represented by Conceptual Dependency Theory (CD) (Schank, 1973, Schank and Abelson, 1977, Schank and Colby, 1973), and other related theories (Wilks, 1975). The basic idea is that syntactic knowledge is unnecessary to the understanding of language. This would mean that grammatical inconsistencies in a statement can be ignored by the system, as opposed to the approach taken by syntactic parsers which must build a means of handling them.

CD was designed to model the way people understand natural language, and is based on "semantic primitives", a small set of meaningful elements into which the meanings of words and phrases can be broken down. It is highly intuitive, but has achieved a high degree of success in certain restricted domains, including question-answering.

Semantic primitives have meanings defined by their function within a language, and are combined to form higher-level concepts. Typical primitives as described by (Schank 1973) include "propel", "speak", "ptrans" (movement of a physical object), etc., but there is no one set of correct primitives. Understanding real-world situations requires a huge amount of knowledge, and to represent this as primitives requires a very large memory and excessive processing time. To get around this problem, the idea of "scripts" is introduced (Schank and Abelson, 1977). These data structures formalize stereotypical knowledge about a situation to control inferencing and to guide the building of conceptual structures. This approach must of necessity be restricted to a narrow domain.

Scripts can be a useful way of modelling user views in a database environment, since they provide a means of defining what the user is expected to ask questions about. This

basic approach is used in (Cullingford and Selfridge, 1983; Scha, 1982). The Wilks parser, also based on semantic primitives, has been employed by Boguraev and Sparck-Jones, (in addition to the Chart Parser for some pre-processing) (Boguraev and Sparck-Jones, 1982).

### 3.11 Procedural Parsers

The choice of a parsing formalism depends on many factors: the purpose of the system (linguistics research, man-machine dialogue or the implementation of a real-world system); the resources available to the developers (hardware and software resources, manpower, time, financial limitations), and other constraints. It is possible to write a procedural parser for a relatively narrow domain that will accept a suitable range of grammar for that narrow domain. Several natural language systems have combined the philosophies of the formalisms above to create unique, workable parsers. For instance, the LADDER system (Hendrix et al., 1978) has a language processing package called LIFER which is designed to build special-purpose natural language front ends.

Natural language interfaces often cannot afford the luxury of a theoretically sound parser because such formalisms often require considerable computer resources (a sizable knowledge base, a large amount of list processing, and so

on). It is unlikely an organization would want its computer tied up trying to correctly parse natural language utterances instead of, and perhaps interfering with other information processing activities; it would be more efficient to have the parser on a microcomputer or workstation, where the parsing process only delays one user rather than all. But to run on a personal computer, the program and data must be relatively compact and efficient.

Any system based on a finite grammar will run into trouble with people who cannot or will not use expressions which conform to the grammar. In a real-world environment, users may be frustrated and dissatisfied with a system if they deem it uncooperative or unintelligent (Järke et al., 1985).

For these reasons, several of the systems noted above have used a combination of approaches. Often these are combined with heuristics to guide the parsing process, to compensate for failings in the grammar, and so on. Such hybrid parsers may include combinations of top-down and bottom-up parsing, depth-first and breadth-first search, determinism and non-determinism, single parse or all-paths parsing, guided by semantics or totally ignoring semantics, etc. The challenge then in developing a natural language interface is to find an appropriate mix of these techniques.

### 3.12 QNL: A Heuristic Parser

The choice of a parsing mechanism and grammar is crucial to an implementation of a NLI. If the parser is not robust, then it is impossible for complex concepts to be passed to the knowledge base for processing. An inefficient parser holds up all processing, and its inefficiencies can ripple through the entire system.

A simple set of phrase structure rules is not a sufficient grammar to express complex concepts, but a complex set of phrase structure rules may contain subtle inconsistencies. A complete grammar of English is a complex affair (Sager, 1981) which requires a specific parsing mechanism. Often such parsers and grammars are designed to develop and demonstrate theories, are the results of years of effort, and are impractical for implementation on a personal workstation.

Several parsers were implemented and tested as part of this thesis. Eventually, the Heuristic Parser emerged as a suitable vehicle for a transportable NLI. What follows is a description of the other parsers, and the evolution of the Heuristic Parser.



### 3.12.1 First Attempts

The first parser was a variation of an ATN using a semantic grammar which mapped directly to the file names and attributes. This was a top-down parser which accepted the first successful path through the network. This was deemed an acceptable compromise to finding all parses because it is generally conceded that ambiguity is not a severe problem in database access (Grishman, 1984). However, such an approach limits the conceptual coverage of the system; users can only use specific phrase structures and there is a limited number of ways of referring to the database. If a question is asked which does not conform to the phrase structure rules, the parse fails, and the question has to be re-asked. If the parser were extended to become an all-paths parser, we would encounter the problem of multiple syntactic interpretations to be handled by the semantic analyzer.

The next approach tried was a parser operating on a case grammar. The rationale was that it would incorporate into the parsing stage, some semantic analysis to constrain the number of possible interpretations. Using a case grammar, the meaning of a sentence is determined by analysing the case relationships between the terms in a sentence. This was also a top-down, non-deterministic parser, able to parse many queries very quickly. However, there are certain

severe problems with case grammars as described earlier which make it unsuitable for use in a database front end if it is the only grammar. As the range of acceptable language grew, the problems caused by ambiguity and especially content-empty prepositions also increased.

The next formalism studied was the Chart parser (Thompson, 1981, 1982, Thompson and Ritchie, 1983). A major problem with the Chart Parser is that as the size of the grammar grows, the execution time increases exponentially. Even with relatively simple sentences, the number of arcs in the graph could grow to more than a hundred. However, it was appealing because the grammar could be separate from the parser, and the coding for the Chart parser is not too difficult.

In order to improve the performance of the Chart parser, several heuristics were implemented to reduce the number of arcs generated. These included reducing the number of rules by specifying the grammar in terms of options and repeat factors, as in Figure 3.11. The inactive arcs at each node must be examined by every rule, and the more rules there are, the more successful matches there will be. For instance, the start of a noun phrase ((Det) + (Adj).) may occur in several noun phrase rules, and so a large number of

rules could fire on a phrase like "the red.." or "red...". The approach used here allowed these structures to be built only once.

The problem of left recursion must be faced whenever production rules are recursively defined, or if their sub-rules refer back, perhaps very indirectly, to the calling rule. Again, the amount of checking necessary at each node grows exponentially with the number of inactive arcs, so methods of reducing the necessary checking were devised. These included hierarchically ordering the inactive arcs at each node, and maintaining a "shortlist" of active rules.

Incorporating some semantic knowledge into the grammar allowed the clustering of some rules so that when there was no possibility of them succeeding, they could not fire. This was employed with relative clauses which can only follow noun phrases, and with other, less frequent phrase structures. All rules were prohibited from firing unless all of their required sub-rules had been previously satisfied.

Tests comparing the Heuristic Chart Parser with the Chart Parser demonstrated reductions in the number of arcs built of between 8.3% and 80.2%, as shown in Figure 3.16, but the performance was still unacceptably slow due to the overhead

of checking so many hierarchies at each node in the graph.  
The Chart parser was eventually abandoned as a parser for  
an interactive system.

Sample Questions	Chart Parser time edges	Heuristic Chart Parser time edges	Improvement time edges %
list all parts.	16	19	25
Get status of suppliers in London.	19	27	27
List all red parts.	23	37	37
Is p1 red?	27	51	37
Tell me about the Paris project.	28	48	31
List all red or blue parts.	32	55	37
Tell me about each part.	51	52	44
Get any supplier in London who sells p1.	69	89	51
Which suppliers sell blue parts?	80	91	49
List all suppliers who sell red parts.	85	125	73
List all projects which use blue parts.	127	137	79
Which suppliers are in London?	130	139	44
What colour is p1?	134	143	95
Which parts are red?	136	149	97
Is there a supplier who supplies red or blue parts?	152	192	86
Does any London supplier sell p1?	213	251	57
List all parts which are red.	339	267	99
Are there any suppliers in Paris?	375	276	58
Are there any red parts?	393	280	58
Which suppliers are in London and sell blue parts	480	378	118
Are there any suppliers in paris who sell red parts?	521	439	131
Which red parts are available in London?	900	590	117
Are there any suppliers who sell blue parts	1143	693	121
		137	137
			89.4
			80.2

fig 3.16 Performance Comparison: Chart Parser  
versus Heuristic Chart Parser.  
Time is measured in seconds.

### 3.12.2 Developing a Grammar

The Chart Parser did prove useful in developing and fine-tuning a set of phrase-structure rules that would process the type of interaction which is expected to occur in a database environment, specifically interrogatives, imperatives and declaratives.

Three distinct sets of phrase structure rules were examined: The first was Chomsky's set of rules (Chomsky, 1965), which was eventually rejected because it did not cover the full range of sentence fragments which occur frequently. The second set of phrase structure rules was a merger of those defined by Naomi Sager in the Linguistic String Project (LSP) (Sager, 1981), and a set put forth by Harris (Harris, 1985). This was altered to compensate for the fact that the LSP relies very heavily on syntax analysis, and to include relative clauses, pronoun-deletions, substitutions, adjectival subordinate clauses, and other structures.

The grammar is a modification of the second set of rules to conform more closely with the needs of a database environment, and it can be considered a "domain-independent semantic grammar". Although the words in the lexicon are described in terms like "noun", "verb", etc., they also have semantic definitions which are closely tied to their

syntactic roles. Once it was determined what kind of phrase structure rules would be needed, and what semantic analysis could be intermixed with the parsing process, the Heuristic Parser was created to implement this grammar.

### 3.12.3 Design Criteria of the Heuristic Parser

Any multi-path parser must follow some false paths and generate unreasonable parses if it relies solely on syntactic analysis. As noted earlier, several researchers have stated that the sooner semantic knowledge is brought to bear, the fewer the ambiguities. Fewer ambiguities means faster syntactic and semantic processing, and therefore more efficient implementation; an important factor when considering implementation on a workstation. A single parse is easier to maintain and faster to produce.

Bottom-up parsing builds fewer intermediate structures than top-down parsing, and constrains the search space to the likely solutions. A deterministic machine can be modified to deal with ambiguities by shifting attention to another part of the input string, and later incorporating more knowledge to be used in disambiguation. Finally, it is desirable to model a parsing mechanism on an existing formalism so that it is not too closely allied with the application.

The syntactic processing of the Heuristic Parser incorporates many of the ideas of Marcus (Marcus, 1980), but also uses a domain-independent case grammar to assist in the semantic processing of a sentence. The syntactic rules are procedurally encoded at this time to ease the development of the grammar and the parser, but the case grammar is declarative. This is a one-path, deterministic parser which uses syntactic and semantic knowledge to determine the functional roles of words and their semantic attachments. Once a decision is made to assign a role to a word or phrase, the decision is not reversed. If, at any stage, there is insufficient information to make a certain decision, the decision is postponed until more information can be accumulated. Therefore, only totally unambiguous attachments are made during syntactic processing, and others are left to a combination of syntactic and semantic rules.

The implementation details of the Heuristic Parser are described at length in Chapter 5.



## Chapter 4

### Knowledge Representation

There are many knowledge representation schemes that have been employed by AI researchers. Each formalism has certain features which make it appropriate for certain applications, and inappropriate for others. The most important consideration when examining and comparing the various formalisms is the eventual use of the knowledge (Barr and Feigenbaum, Vol I, Pg, 145, 1981).

From a philosophical standpoint, knowledge has been defined to be "justifiable true belief", but there has been no well-formed and precise theory of knowledge (Addis, 1985, pg. 24). Philosophers have been concerned with developing and providing the criteria that establishes certainty and what could be true. Attempts have been made to define absolute primitives that exist and behave according to well-defined laws, and although a degree of success is achievable, no claim may be made that these formalisms approach a complete and sound theory of knowledge. Few of these concepts have been demonstrated to be useful to computers, partly because computers require specifics, and philosophical definitions are often too fuzzy to implement.

If knowledge is considered to be the logical closure of a set of axioms, an approach chosen by philosophers, a representation formalism based on an arbitrary set may be neither correct nor useful. Determining all of the potential implications is exceedingly difficult, and so a computer representation of knowledge should be based on pragmatic rather than philosophical concepts (Frost, 1985).

A computer science theory of knowledge representation is explained by Newell where he defines knowledge to be "whatever can be ascribed to an agent, such that its behaviour can be computed according to the principle of rationality" (Newell, 1982), a principle that governs the system and predicts its behaviour. If a system has a goal, and has the means to achieve that goal, and knows which actions will achieve the goal, then it will invoke those actions.

This definition concedes that knowledge is a very abstract entity that has no physical form. But knowledge must be characterized by what it does, not how it is structured. Therefore Newell divides knowledge as used by computer systems into two components: the "knowledge level", and the "program level". The knowledge level is declarative: the

facts that exist within a closed world. The program level, sometimes called the symbol level, is the data structures which arrange and allow access to the knowledge.

With respect to natural language processing, the term "knowledge representation" has a specific meaning. An input string is mapped from a surface structure of knowledge to an internal form. There is normally no "meaning" output from the syntax analysis stage. Rather, the output of the semantic analysis module is to be considered an internal representation of knowledge (Harris, 1985, pg. 283). The knowledge representation formalism is something that is applied to the output of the parser to discover the meaning of the user input.

To achieve independence from particular environments, we must use robust and general methods which do not rely on individual applications. The NLI should accept the structure of existing databases, be able to encode knowledge about a new environment in a systematic way, and then manipulate that knowledge correctly. Such independence can be achieved by modularizing the knowledge so that there exists a clear division between world knowledge, application knowledge, and DBMS knowledge (Hendrix and Lewis, 1981).

The representation formalism established for the database-specific part of the interface must be such that the form is

applicable to all databases, and that the semantic content of all databases is readily acquirable. Then a set of

generalized procedures may operate on the domain-specific knowledge in standard ways independent of the knowledge contained therein.

#### 4.1 What Kind of Knowledge is Needed?

A natural language interface must of course have knowledge about language, and this must include domain-specific knowledge and "real-world" knowledge. There must be a means of both syntactic and semantic disambiguation, understanding anaphora and ellipsis, as well as a means of understanding user intent.

A problem introduced by a running dialogue is that the dialogue itself may become a subject of the conversation, requiring the interface to refer back to the results of previous queries..

We need to know about what is in the database. Rather than have information only about the file names and attributes, we want to have knowledge about the entities and relationships within the domain so we can recognize them by their form and their roles in relationships. We need to

know how they relate to one another, and how the user perceives of them. We need to know how to do things within the domain, like computing derived data or employing intermediate relations to achieve required results.

Finally, we need knowledge about what is known (meta-knowledge).

An intermediary between the language knowledge and the domain-specific knowledge is pragmatic "real-world" knowledge, which is applied to the natural language utterance to determine its meaning with respect to a world model. This semantic meaning will then be mapped to the database. We might think of the world model as "that which could be", and the domain-specific knowledge as "that which exists at this time".

Entities may be defined in terms of other entities, and relationships also may be defined in terms of entities and other relationships. There must be a facility to represent these hierarchies of relationships, such as class, subclass, and super-class relationships.

What roles do entities play in a relationship? Sowa notes that these roles should be explicit enough that they may be directly transcribable into a natural language (Sowa, 1980).

There should be a set of default roles for each entity and concept in the database. This is a description over and above the type description, and provides background information about how entities function within relationships.

There must be a means of aggregation and generalization such that entities may be appropriately clustered, and so that entities may be recognized by their attributes. There must be a means of accomodating different views of data, so that requests for specific information result in precisely the same response, no matter how the request is originally phrased. Different views of the data should be represented internally in a single fashion. This will require the definition of some low-level entities, which may be combined in various fashions to form higher-level abstractions.

Some knowledge about how to do things within the domain must be included. This may include functions to compute derived data, and must include some meta-knowledge about when and how to invoke various functions. User views may not have a one-to-one relationship with the relations in the database, so there must be knowledge about this extra level of mapping.

An inference engine is a crucial part of the domain-specific component, allowing it to determine how to respond to requests for information that are not explicit in the database, where to look for intermediate information that can be used to determine other information, etc. Users may have varied views of the domain and it may be necessary to do an arbitrary amount of inferencing to connect the user's question with the database. For instance, multiple relations may have to be accessed to accomodate a request, but the user need not be aware of this.

Rules of inference elevate the system from being a passive retrieval language to that of an expert assistant. These rules will manipulate knowledge to determine the implications of data within the database, and can be used to enforce semantic constraints.

Knowledge about semantic constraints is necessary to "capture the knowledge and reasoning strategies of a designer, programmer, or database analyst" (King, 1980). The database analyst is especially important for your purposes as someone who mediates between a user with a problem to solve, and the database which has the data to be used in the solution.

## 4.2 Conceptual Modeling Tools

There has been considerable effort directed at capturing the semantic content of databases. The conceptual schema of a database is generally inappropriate as a knowledge representation formalism because of its low semantic content. It is too closely tied to a data model and reduces the real world to that model.

Although formalisms which seek to extend the semantic content of the relational model are principally design tools, they must be considered here because they do provide a high-level representation of the contents of a database. We will provide an overview of three systems: the Semantic Data Model of Hammer and McLeod (Hammer and McLeod, 1981), which claims to be the basis of a robust user interface, the Entity-Relationship Model (Chen, 1976, 1977), because it is such an easy to understand formalism and TAXIS (Mylopoulos, Bernstein and Wong, 1980), because it specifically organizes knowledge using a formalism called a semantic network, which will be studied in-depth later in this chapter. For the purposes of this discussion, we will use Date's Suppliers-Parts-Projects database (Date, 1982, pg. 114).



#### 4.2.1 The Entity-Relationship Model

One of the earliest attempts at overcoming some of the limitations of record-based data models is the Entity-Relationship (E-R) model (Chen, 1976, 1977). The E-R model is independent of the constraints introduced by data structures and storage and access considerations, and is thus a more natural view of the application environment. It is used to define how users perceive the data and the relationships that exist within the organization. It is possible to map from the E-R model to the three classical models: the hierarchical, the network and the relational models. Changes in the underlying DB or DBMS will not necessarily affect the model, and vice-versa.

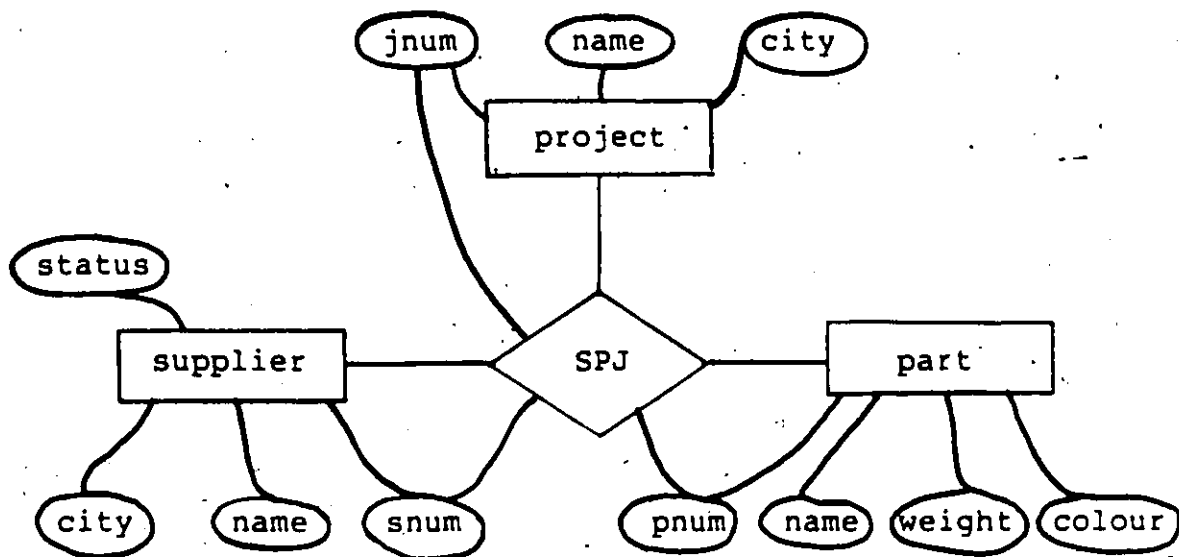


Figure 4.1: Entity-Relationship Diagram for the Suppliers-Parts-Projects Database

In the E-R model, the procedure for the selection of entities and relations can not be precisely defined, and this is a drawback. What one user perceives as a relation may be seen by another as an entity. Further, attributes (like "city") may also be perceived as entities. Chen has provided mechanisms by which entities, relations, and even attributes may change over time, but this does nothing to improve the expressive power of the model.

It is not possible for any "thing" to be both an entity and a relation at the same time. The decision is an arbitrary one made by the designer.

A user wanting to question the database shown in Figure 4.1 will perceive of events that normally occur within the domain. Suppliers sell parts, projects use parts, and so on. The E-R model can not specifically represent each of these events as distinct. The only way to represent them would be to embed in the linguistic knowledge of the system, semantic clues as to how to map the user's perception of the event to the schema at the E-R level.

For instance, given the queries

s8. "Who sells part number 2"

s9. "Who uses part number 2"

both will access the SPJ relationship to determine the answer. The E-R model has only one internal representation for two events, so either the lexicon must have the relationship explicit, or there must be some intermediate mapping. Since we are attempting to find a formalism that will be the intermediate mapping as well as capture the semantics of the database, we must reject the E-R model as a knowledge-representation formalism for a natural language interface.

#### 4.2.2 The Semantic Data Model

The Semantic Data Model (SDM) is basically a semantic network that is specific to a narrow domain. One of the principles of its design is that rigid distinctions between concepts such as entity, relationship and attribute should not be made, since these may change over time, or different users may have different perceptions.

SDM is a more powerful conceptual design tool than the E-R model. First, it supports a relativist view of the database: alternative ways of looking at the same information. Other features are that it is logically redundant, and integrated.

By logically redundant, we mean that derived information is

explicitly embedded in the schema. This is an important point which is noted as being vital for a natural language interface in order to control inferencing (Ginsparg, 1983). It is basically an encoding of procedural knowledge in the knowledge base.

By integrated, we mean that relationships between multiple ways of viewing the same data are explicit. Such integration is needed to control redundancy.

Entities are hierarchically arranged in terms of classes, which are homogeneous collections of entity-types. The model is defined in terms of two kinds of classes: Base classes are defined independent of all other classes, and are mutually disjoint. Every entity is a member of exactly one base class. Non-base classes may be collections of entities or other classes.

Inter-relationships between classes and members of classes are explicit, and so the concepts of generalization and aggregation are supported. Each class has a collection of attributes describing either the class as a whole, or the members of the class. Member attributes describe a member by linking the member to one or more related entities in its

own class, or another class. Class attributes describe the properties of the class as a whole. Sub-classes and groupings inherit the properties of their super-classes.

It is possible to expand the semantics of the database by defining some classes which do not map 1:1 to the relations in the database. For instance, Suppliers (S#, Name, Status, City) can be grouped by the City attribute.

SDM was designed as a conceptual modelling tool of a narrow domain, and the only world knowledge which it has is either procedurally encoded, such as the concept of "size" or the calculation of quantity, or is explicitly defined as the class of Supplier-City. There is no means of defining entities or relationships other than those of the database, so the frame of reference for the user is a very restricted world. If objects are referred to which are not explicit in the representation, they cannot be derived. Finally, some words may refer to an object or objects which may belong to more than one class, and this cannot be handled. For these reasons, SDM is not appropriate as the knowledge representation formalism for a transportable NLI.

#### 4.2.3 TAXIS

TAXIS is a database design language for highly interactive transaction-oriented applications that is based on semantic networks. It is meant to be customized to each domain by programmers who will write procedures specific to the application, and so does not have a high degree of portability by our definition. It is also specifically for narrow, well-defined problems.

In TAXIS, the semantic net is used for its generalization and abstraction ability. By building a language around this formalism, the user will conceptually manipulate hierarchies rather than individual files and records. A semantic net is used so that a hierarchy can be extracted from the concepts of the network, depending on the user's needs. This means two different operations can manipulate the same concepts, but in different manners, using the same data structure.

TAXIS does more than use the semantic net as a declarative store of knowledge, it also encodes domain-specific functional knowledge so that procedures can be written which reflect the semantics of the domain.

There are three kinds of object in TAXIS: tokens (or constants), classes, and meta-classes. Tokens having common

properties are grouped into classes, which are grouped into meta-classes. A meta-class is similar to a class, except its members are classes rather than tokens. Classes and tokens have properties which relate them to other classes and tokens. Token properties represent specific facts, whereas class properties are determined by abstract rules. So, SUPPLIER is a class, described by a collection of tokens as follows: Each SUPPLIER has a unique supplier number, a name, a status, and is located in a city.

The individual occurrences of SUPPLIER are the tokens, and each of the properties of the token is represented as a triplet:

```
SUPPLIER s# supplier.s#  
SUPPLIER sname supplier.sname  
SUPPLIER status supplier.status  
SUPPLIER city supplier.city
```

SUPPLIER could be grouped into a meta-class SPJ, with a common property s#. This would allow access to SUPPLIER through SPJ, and vice-versa.

Also embedded in the semantic network are the relationships and operations which are associated with the tokens and classes. So the relationship SELL will be a procedure with parameters SUPPLIER, PART, PROJECT, and the method of determining these sub-class values will be created by the programmers.

One of the major design goals of TAXIS is to be easy to customize to new domains by applications programmers, but this is not true portability. To achieve their goal, they blend procedural knowledge with factual knowledge, and this renders the formalism too specific to each domain for our purposes. There must be a clear separation between procedural and declarative knowledge so that the peculiarities of one domain are not accidentally transported to others.

#### 4.2.4 Summary of the Conceptual Design Tools

Each of the database modelling tools described above attempts to capture the semantic meaning of stored data by organizing the objects and relationships hierarchically.

While the E-R model only provides a single way of viewing data, the other two use a network structure and allow extensive relationships between the objects and relationships as a means of supporting aggregation and generalization. Functional dependencies and domain roles are also explicit. SDM allows multiple ways of viewing data, but such is not the case for the E-R model, and in TAXIS the different ways of manipulating information must be specifically encoded by programmers beforehand.



SDM and TAXIS encode procedural knowledge into the model, primarily for efficiency considerations. Inferencing in all of the models is achieved by following links in the graphical representations.

The main problem with all of these approaches is that they build a large amount of domain-specific knowledge into the system, much of it procedural. Although they purport to support useful user interfaces, such an interface cannot be a natural language interface because there is no facility for representing knowledge about natural language. This is not to say that a separate linguistic knowledge base and a mapping mechanism could not be added as a front end, but the purpose of this study is to determine a knowledge representation formalism for the entire interface, not just the database part.

There is also no way of representing information which is not present. Suppose a user asks about the price of an object, but the database does not store price data. Does the system know that it does not have this information? Does it know anything about price? Could it be computed? The advantage of having a higher-level world model in addition to a domain model is that there is a framework in which to represent knowledge outside of the domain. This

world model can be used to explain to the user the discrepancy between the user's perception, and what actually exists within the domain.

What is needed then is a formalism that can be used to capture the semantic content of any database, no matter what the design model. The knowledge representation formalism chosen by this thesis must be independent of the underlying DB, DBMS, and the design model.

#### 4.3 The AI Approach to Knowledge Representation

The Artificial Intelligence approach to knowledge representation in a natural language front end has focused on the following formalisms:

- 1) Logic
- 2) Procedural Representations
- 3) Frames
- 4) Semantic Primitives (including scripts)
- 5) Semantic Networks.

##### 4.3.1 Logic

The logic most frequently used in AI is mathematical logic or the predicate calculus (Barr and Feigenbaum, Vol I, 1981, pp 160-172), although other logics including non-monotonic reasoning, fuzzy logic, and statistical reasoning have been employed (Rich, 1983, pp. 173-198).

The chief advantage of mathematical logic is that it is sound and complete, and has a well-defined set of inference rules guaranteed to produce correct deductions. It is also a natural way to express many concepts, and often corresponds to intuitive understanding of a domain. Logic is also precise, and is subject to well-understood rules.

Logic determines the validity of situations by syntactic manipulation, and gleans nothing from semantic information which may be useful in confining the search space. Another basic problem with the predicate calculus is that there is no proof procedure, only a disproof procedure. That is, the validity of a conclusion is determined by deriving a contradiction to a contradiction of the conclusion. The standard method of deduction is the resolution method, (Charniak and McDermott, 1985, pg. 382) and a major problem with this technique is that it is slow, subject to combinatorial explosion and even heuristics to restrain it are not very helpful.

Another disadvantage is that translation from natural language is a tedious process, and a logic representation of an utterance often bears no resemblance to the natural language expression. Finally, only entire sentences can be negated, not just individual terms in a sentence.

#### 4.3.2 Procedural Representation of Knowledge

Rather than representing facts as syntactic structures of formal logic, procedural knowledge takes advantage of the domain to guide searching and determine which rules to apply, and when. This approach, pioneered in SHRDLU (Winograd, 1972) makes no claims to soundness nor completeness, but rather relies on heuristics to limit the generation of inferences to those which may plausibly contribute to acceptable decisions. This permits efficient execution at the price of the possible introduction of inconsistencies. It relies on the user to help compensate for those inconsistencies through an interactive dialogue.

In a procedural representation, data is stored in a database, and a set of functions and procedures is built to operate on the data in an "intelligent" manner. Inferencing is controlled by the ways the facts are stated, or by interacting with the user. Although blind reasoning is eliminated, the embedding of knowledge in the code results in a loss of modularity.

Procedural knowledge is useful in directing the search for solutions, but there is no generally accepted means of implementing it (Barr and Feigenbaum, Vol. I, pg. 179, 1981).

#### 4.3.3 Frames

Combining declarative and procedural knowledge into one data structure, a frame typically contains knowledge about stereotypical situations. That is, a frame will hold expected values, default values, and procedures about what to do if certain conditions hold (Minsky, 1975).

Humans don't always solve problems by reasoning from first principles. Frames encapsulate "chunks" of knowledge which are called into use in certain situations. This is appealing when we consider the routine work that is often associated with databases, such as the generation of periodic reports. Frames are a useful way of partitioning the knowledge base into chunks that will be useful for a given problem, and thus may improve performance over semantic nets and logic, with respect to searching and the depth of inferencing needed (Charniak, 1981). However, frames are also restrictive in that new ways of looking at data must be made explicit to the system. Also, much of the knowledge in a frame is procedural, and this is contrary to our idea of separating declarative knowledge from procedural.

#### 4.3.4 Semantic Primitives

Semantic primitives are data items which are not much different from natural language words. These primitives are combined according to a set of rules to form conceptualizations, of which there are two basic types: objects and events.

Each semantic primitive has its meaning defined by its function within the overall language, and does not refer to any specific "thing". A natural language question is translated into a network of primitives and relationships between the primitives. Although there is no single correct vocabulary, two sets of primitives have emerged as dominant in the field.

The first set of eighty (80) primitives was developed for a natural language translation project (Wilks, 1975). These were derived in a rather ad-hoc manner, but did achieve a fair degree of success.

The second set of 14 primitives forms the basis of Conceptual Dependency (CD) Theory (Schank and Colby, 1973). The basic axiom of this theory is that if two sentences have

the same meaning, their internal representation should be the same. This implies that implicit information should be made explicit in the internal form.

CD Theory has a strict set of allowable relationships that may connect the primitives, and syntactic and semantic rules determining how conceptualizations may be built. CD is in fact, an extension of case grammars, and the claim is made that syntactic parsing is almost completely eliminated. By having such a small set of rules and primitives, all higher level concepts may be represented in a canonical form.

The major drawbacks of the use of semantic primitives include the tremendous complexity of the parsing process, the large memory requirements, and the fact that everything must be represented at a primitive level, from first principles, as it were.

Extensions to CD include the concept of scripts (Schank and Abelson, 1977). These are sets of expectations built of CD primitives, and are similar to the declarative knowledge stored in frames. One major difference is in the inferencing ability. Using scripts, inferred information is explicit before the script is invoked. For instance, if a person enters a restaurant, an appropriate script is

triggered to infer that the person is going to have something to eat, that he will sit at a table, be served by a waiter or waitress, pay the bill, leave a tip, etc.

The use of CD primitives may not be appropriate for use in an interface to databases because most of the primitives relate to human acts, and the expressive power with reference to non-human objects is very limited.

#### 4.3.5 Semantic Networks

A semantic network is a directed graph with labelled nodes representing entities, and named arcs between the nodes representing relationships between the entities. Each entity is described by a set of attributes, and the entity is considered by the network to be no more and no less than that set of attributes. In a semantic net there is a fixed set of possible relationships, and each has an associated inverse relationship. These two-way relationships convey the same meaning, but from different perspectives. There is no one set of semantic relations; rather, relationships are usually tailored to the type of application environment.

Three basic types of relationships can be represented: superset-subset, attribute-object, and participation in higher-level relationships. This allows the system to



extract information about an object, test for membership in a class, and identify an object by its attributes.

Deductive reasoning with a semantic net is simply a matter of following arcs through the graph, examining and selecting the relationships that fan out from each of the nodes as they are visited. Heuristics are implemented by determining which arcs to follow from any given node. Thus, reasoning is a data-driven process.

Semantic Nets emphasize the roles objects play in relationships and are especially important in natural language understanding systems (Quillian, 1968; Findlar, 1979; and others). They are one of the most popular forms of knowledge representation formalism in AI as a means of representing commonsense knowledge.

The "IS-A" relationship is one of the most commonly used associations to implement the inheritance hierarchy or taxonomy of a semantic net. The attributes of objects can be represented by a "HAS-PART" relationship. The variety of relationship labels is extensive, and thus a key feature of this formalism is its flexibility. Other types of links are possible, and are employed based on the needs and purpose of the system.

While facts and relationships are represented explicitly the procedures which govern manipulation of the network are usually distinctly separated from the knowledge contained in the network, although such is not the case with systems like TAXIS.

Work by Simmons (Simmons, 1973), used semantic networks to represent concepts defined in a case grammar. He showed that a semantic net can be used to express concepts expressed in natural language as nodes connected to other concepts by particular sets of arcs called semantic relations. The most primitive concepts in the net are word-sense meanings. Primitive semantic relations are those the verb has with its subject, object and prepositional phrases.

This demonstrated an easy transformation from case grammar to semantic nets. We can use case frames to determine a relationship among objects, and to infer objects based on the role-players in a relationship. Objects may be referred to by their attributes, and so the attributes are connected to objects by specific arcs. Case frames provide a means of determining the participants in higher-level relationships.

Problems associated with semantic networks are subtle, and include such topics as the goodness of ways of representing ideas, the uniqueness or degree of redundancy involved with

representations, and the fact that a node only represents a collection of attributes, and not an abstract entity (Griffith, 1982).

Experience with semantic nets has shown that computational problems emerge as the network becomes large and non-trivial. This is because the number of nodes and arcs may become exceedingly large, and so the time spent at each node can become unwieldy. However, some of these problems may be caused by hardware limitations. Also, a database schema consists of many occurrences of a few record types, so the number of nodes in our semantic network may be relatively small.

#### 4.4 Summary

Of the formalisms presented here, the conceptual modelling tools were found to be too restricted to their narrow domains, and/or they include procedural knowledge in the model. Such procedural knowledge ties the knowledge base closely to the application environment, and this is something we are trying to avoid.

The semantic network formalism is attractive as a representation scheme for several reasons: There is a clear separation between procedural and declarative knowledge, and this will make it easy to transport to new domains; it is relatively easy to implement; deductive reasoning is constrained to useful or potentially successful lines of reasoning; it is useful as a tool in the experimental design of a NLI because it is so flexible and so general.

## Chapter 5

### Implementation of a Prototype

QNL consists of three procedural modules to accomplish syntax analysis, domain-independent semantic analysis and domain-specific semantic analysis. Each module accesses an appropriate declarative knowledge base. A user enters a question or command which is parsed by the syntactic analysis module. A single parse tree is passed to the World Knowledge Base (WKB) which attempts to gain a general understanding of the question, and modifies the original parse tree appropriately. This domain-independent meaning representation is passed to the Domain Knowledge Base (DKB) which interprets the question with respect to the particular database.

Feedback demonstrates to the user, at each stage of the process, what the system has understood, how it has processed the input, and alerts the user to potential problems. The user can take appropriate action to continue, abort, or modify the processing. There must exist an ability to demonstrate to the user the system's final understanding of the question or command before it attempts execution.

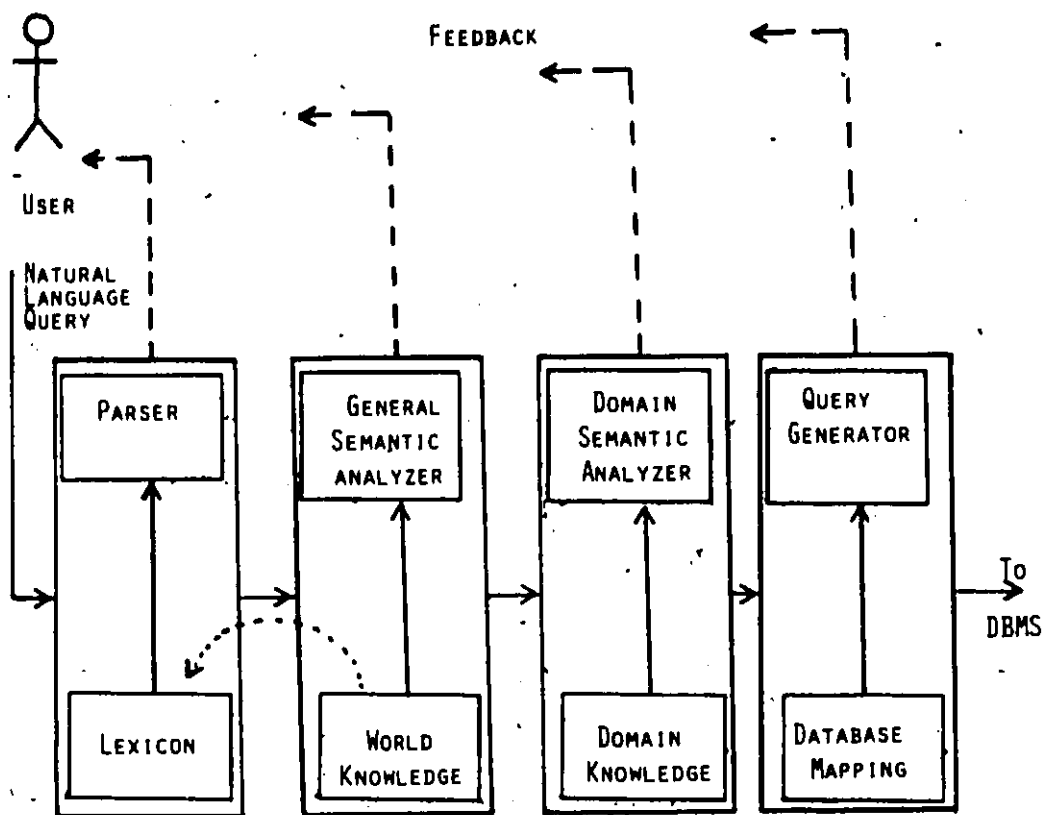


fig 5.1 Architecture of a portable natural language interface.

The three procedural modules operate in series. Upon receiving the user's question, a preprocessor identifies the tokens in the input string and generates a normalized expression containing the possible syntactic and semantic roles of each of the tokens. The parser receives this expression and builds a single parse tree to represent the syntactic structure of the sentence. The domain-independent semantic analyzer assigns attachments of modifying phrases, resolves some pronoun references, handles conjunctions, disjunctions and negation, and resolves anaphora and ellipsis to generate a domain-independent meaning representation in the form of a tree.

Next, in the domain semantic analysis process, the objects in the tree are transformed into the relations and attributes of the database. Implied relationships are made specific, objects are identified by their attributes, interrogative pronouns which could only be resolved in the domain are handled here, intermediate relations are specified, and so on.

The output of this module is a non-procedural domain-specific expression which may be converted into a relational calculus or relational algebra expression. This expression can be transformed into a data manipulation language (DML) like QUEL or SQL. It will be up to the query generator to

determine how to schedule the order of operations, and how to generate the appropriate expressions in the DML of the database.

### 5.1 The Lexicon

The lexicon is a list of words and their syntactic roles. Each syntactic role has an associated list of semantic markers describing the term in that role. These semantic markers are drawn from the WKB, so the words in the lexicon are defined in terms of the semantic net. The semantic markers are symbols like "physobj", "animate", "number", etc, while syntactic roles are "noun", "verb", "pronoun", etc.

The lexicon must have access to all of the words used to describe the objects and events of the domain, as well as expressions which can be used to manipulate data in a database ("sort", "count"), and sufficient "content-empty" words ("of", "please", "such", and so on) to carry on a natural language conversation.

Terms which have a general meaning in the WKB but have a special meaning in a domain are defined in both the WKB and DKB, and accessed by the preprocessor. This way, both definitions are available to the NL processor. If the NLI is



transported to a new domain, the peculiarities of the previous domain may be ignored, and those of the new application easily acquired. The set of procedures which looks for domain-specific interpretations of terms remain constant; only the data they manipulate changes from application to application.

Such a need arises with terms like numeric or alpha-numeric keys ("S234" may refer to a supplier, "P564" refers to a part, etc.), and numbers which may be used to refer to domain-specific concepts (007 is a secret agent, 747 is an aircraft, .357 is either a very good batting average or a handgun, and 872-3239 is a telephone number).

If the lexicon itself were augmented with these special meanings, then it would have to be reduced for each new domain, otherwise the special terms would be inadvertantly applied to other domains. The chosen solution is to define these terms in the DKB, and instruct the pre-processor to look in the DKB for this set. When attempting to determine the meaning of a term, the domain interpretation is always given priority. For each of these special terms, there is a pointer directly to the domain concepts to which they refer.

Dictionary entries take the form of

```
<word> (<syntactic category>
        (semantic markers | case frame) | synonym | nil
)*
```

Each word in the dictionary has a list of syntactic categories. Each category has either a list of semantic markers (sm) or case frames (cf), or it is defined in terms of a synonym. Figure 5.2 demonstrates some sample dictionary entries.

```
(able (adj (sm capable)))
(ability (n (lex able)))
(aircraft (n (sm physobj)))
(airplane (n (lex aircraft)))
(city (n (sm location)))
(its (pronoun (sm animate physobj abstractobj)))
(many (adj (sm number))
      (pronoun (sm abstractobj physobj animate))
      (n (sm number)))
(sell (vt (cf agent {rcpt} (object) (price) (qty))))
(supply (vt (lex sell))
        (n (sm physobj)))
(which (pronoun (sm physobj abstractobj animate))
       (relpn)
       (qword (sm physobj abstractobj animate)))
```

Figure 5.2: Sample entries from the lexicon showing how words are defined in terms of semantic markers, case frames, and synonyms.

The current implementation of the lexicon does not include information about morphemes: the parts of words which include the root of the word, and suffixes and prefixes which alter the meaning. Such analysis is quite complex, time consuming, and does not contribute to the issue of

portability. In a real-world implementation, morphological analysis could prove useful to help reduce the size of the dictionary and derive subtle nuances in an utterance, but for this prototype it is ignored. For a Lisp implementation of a program for morphological analysis, see (Oppacher, 1981).

## 5.2 Parsing

Within the constrained environment of a question-answering system it is frequently possible to predict the occurrence of an interrogative or imperative statement based on the first few words of a sentence. We begin syntactic processing by hypothesizing the existence of an interrogative question, and if that fails we resort to a default interpretation of imperative. The range of sentence types is further constrained in that the NLI expects questions and commands dealing only with objects, attributes and relationships, and assumes these refer to the contents of a database about which it has appropriate knowledge.

Following recognition of the leading tokens of an input string, processing proceeds with the rest of the phrases and sub-phrases within a sentence using a combination top-down and bottom-up strategy.

The bottom-up approach predicts which phrase structures could exist given the syntactic categories of the next token in the input string, and so acts like a filter. Some tokens like conjunctions, disjunctions and adverbs are simply pushed onto a stack to be processed at the semantic stage. Verbs require special attention because they may be complex structures, but this is also postponed to the general semantic analysis. Noun phrases are the most frequent occurrences, representing objects and their attributes.

Some phrase structures can be expected to occur after other phrase structures, such as noun phrases after prepositions and relative clauses after noun phrases. Using the approach of Waltz (Waltz, 1978), the parser checks for frequently-occurring patterns of phrases. However, instead of building a tentative structure and then dismantling it, as would a hypothesis-driven parser, QNL looks ahead to make sure the phrase can be built, and only proceeds if the lookahead confirms the hypothesis. Once confirmed, a representation is built and stored in a temporary buffer.

```

S --> interrogative | imperative

interrogative --> Qword + WHQ
interrogative --> Vi + YesNoQ
interrogative --> Prep + Rest
interrogative --> modal + YesNoQ
imperative --> (adverb) + Vi | Vt + (Vt) + Rest

WHQ --> (modal) + (Vi) + YesNoQ
WHQ --> (modal) + Rest
YesNoQ --> (particle) + Rest

Rest --> Adverb
Rest --> NP + Relclause
Rest --> infinitive
Rest --> PP
Rest --> modal
Rest --> Aux + (Vi | Vt)
Rest --> Vi + (particle)
Rest --> participle
Rest --> conj | disj
Rest --> nil

VP --> Vi | Vt | participle
VP --> Aux + VP

NP --> Modifier + Pronoun + (Noun)*
NP --> Modifier + Noun*

Modifier --> (det) + (adj)*

PP --> Prep + nil
PP --> Prep + NP
PP --> Prep + Rel

Relclause --> Relpn + (participle) + NP
Relcaluse --> participle + NP

```

Figure 5.3: The parsing rules used by QNL

### 5.3 Syntactic Ambiguity

Syntactic ambiguity arises when the form of a sentence may be represented by more than one parse tree. This could be caused by words having multiple syntactic roles, or in the case of a combination of conjunctions and disjunctions. An ATN parser would follow each possibility, backtracking to the choice point, trying to generate a complete parse tree for each combination possible. The Chart parser would generate all candidate structures which could also result in multiple parse trees. Even if either of these do not cause several parses to succeed, there could be a significant waste of effort if local ambiguities are encountered (i.e. only one word in a phrase might have multiple syntactic roles). The parser developed as part of this thesis follows the approach of the Marcus parser and uses the lookahead facility to choose a unique syntactic structure for each phrase.

Marcus restricted the size of his lookahead buffer to three elements on the grounds that if it were not constrained, no advantage would be gained over backtracking (Marcus, 1980). The implementation presented here uses a heuristic lookahead that looks as far as needed to determine what to do next. If it is confronted by a potential ambiguity, it combines lookahead and the current environment to see what would be

the "best" structure to build. We use the term "best" here to mean a structure which conforms to certain expectations of the grammar, and minimizes the work to be done by the parser and other components of the system.

The parser only recognizes a few types of syntactic ambiguity, the most frequent of which will be in the case of a verb mistaken for a noun ("supply", "list", etc.) while the parser is building a noun phrase. In this instance, a set of heuristics can be applied to resolve the problem or defer it until the semantic analysis stage. Figure 5.4 demonstrates a sample of the heuristics to be applied to noun phrases.

Suppose we are processing a noun phrase, and encounter the word "supply", which could be either a noun or a verb. If it is a verb, it could be the main verb of the sentence (or one of them). If we do not yet have a main verb, "supply" might be a good choice. If there is a noun phrase immediately following the ambiguous token, there is a good chance "supply" is a verb connecting two noun phrases. Are there any other nouns in this noun phrase before the word "supply"? If so, the noun phrase could be complete without "supply". However, if the noun phrase begins with a determiner, and there are no other nouns, then "supply" must be a noun.

Another type of problem could occur when trying to build a noun phrase as part of a prepositional phrase, but the noun-phrase is found to be non-existent:

s10. "Where does it come from?"

In the case of sentence s10, the preposition "from" would be stored in the parse tree, but then abandoned during the semantic analysis.

```
If a token is encountered which could be either a verb
    or a noun then
    If there are already nouns in the phrase, then
        phrase is complete without the token in question
        so make the token a verb
    else
        If the noun phrase is incomplete and
            there are any adjectives in the phrase, then
                phrase is a predicate adjective clause, which the
                semantic analyzer will resolve and
                token is a verb
        else
            If there is only a pronoun and no noun, then
                the pronoun will be resolved by semantic analysis
                and token is a verb
        else
            if there is a determiner but no modifiers, pronouns
                nor nouns, then
                    only the modifier is pushed onto the stack,
                    and it will be discarded in the
                    semantic analysis and token is a verb
            else the token is a noun.
```

Figure 5.4: A Sample set of heuristics to be applied to a noun phrase if a potential verb is encountered.



If a phrase cannot be parsed, QNL attaches a special phrase marker "INC" (incomplete) to the phrase, and attempts to handle it at the semantic stage. Such a problem could occur with relative clauses like those in sentences s11 and s12.

s11. "list suppliers in London which sell a red part"

s12. "list suppliers of parts in London which have a status of 30"

The relative clauses "which sell...", "which have...", must be marked as incomplete because we cannot attach the following noun phrase without semantic knowledge. As will be explained shortly, the semantic analysis modules of QNL can correctly attach such phrases, using a combination of domain-independent and domain-specific knowledge.

There are a few other types of complex decisions that must be made by the parser, and all of them make use of the lookahead and attention-shifting strategy: If there is insufficient information to make a unique decision, defer making the decision until more information, or different processes, or both, can be employed.

The syntax analysis module does not attach modifying phrases, and this is a major contribution to its ability to produce only one parse tree.

Consider the question

s13. "Get suppliers selling red parts in London to projects which are in Paris which use green parts."

The prepositional phrase "in London" in sentence s13 could be attached to either parts or suppliers. The relative clause "which are in Paris" could be syntactically attached to either suppliers or red parts or london, and the relative clause "which use green parts" could be attached to parts, projects, suppliers, Paris or London. The growth of the number of parse trees is exponential because only syntactic knowledge is brought to bear. Rather than have the semantic analysis module manipulate and verify or discard so many structures, it was decided to produce only one tree from the syntactic analysis module, and allow the semantic analyzer to use its knowledge to assign attachments.

The output of QNL's parser for sentence s13 is shown in Figure 5.5.

```

(mood is imperative)

(vt (lex retrieve))
(np (n (sm animate)) (var-113 supplier)))
(rel ((n (sm physobj)) (var-114 part))
      ((adj (sm color)) (value red))
      ((participle
        (cf agent (rcpt) (object) (qty) (price)) (lex sell))))
(pp ((n (sm location) (value London))
     ((prep (lex in))))
(pp ((n (sm abstractobj physobj)) (var-115 project))
     ((prep (lex to))))
(inc ((vi (lex be)))
      (relpn (lex which)))
(pp ((n (sm location) (value paris))
     ((prep (lex in))))
(rel ((n (sm physobj)) (var 116 part))
      ((adj (sm color)) (value green))
      ((vt
        (cf agent (object)) (lex use))
       (relpn (lex which)))

```

Figure 5.5: Sample parse tree as produced by QNL on sentence sl3.

#### 5.4 Domain-Independent Semantic Analysis

All of the processing done in the general semantic analyzer is independent of the domain and of the DBMS. The World Knowledge Base is a semantic network that describes, in general terms, all of the objects and relationships which are expected to occur with respect to database access, and also has general knowledge about the domain. If a human were to join a new organization, even without knowing a lot about the database, he would be expected to have general



The lexicon may have the following definitions:

```
project: (noun (abstractobj/ physobj))
supplier (noun (animate))
supply (noun (physobj))
        (vt (cf agent
              (object)
              (rcpt)
              (location)
              (price)
              (qty)))
```

Each of the semantic markers is defined in the semantic net as having a collection of attributes, of being attributes of other objects, and of playing roles in certain relationships.

There is currently a small set of primitives which defines everything in the WKB. Eventually, this set could grow to be quite large.

The current set of primitives includes

```
animate
abstractobj
physobj
content-empty
```

The last primitive is a default for all content-empty words.

Each primitive object has a set of attributes which describes it in this world model, and these attributes may be shared with other primitives. A problem with any

implementation of a semantic net is that a primitive is only equal to the sum of its parts, and nothing else. To be truly robust, there must be a large number of attributes, or a large set of primitives, and an efficient way of following the links through the graph.

Any object or entity is linked to exactly one primitive, but may be linked to any other number of entities. Entities are related to their attributes by symmetrical binary relationships.

For instance, a physical object may have the following definition:

```
physobj: (is-a object)
          (has-a identifier)
          (has-a location)
          (has-a size)
          (has-a colour)
          (has-a use)
```

The IS-A link indicates the attributes of an entity are inherited from some other entity (or a primitive). HAS-A indicates the entity has a specific attribute. PART-OF indicates the entity is an attribute of something else.

Each of the attributes could also be described in terms of its characteristics and so an entity inherits the properties of its attributes.

```

(size (is-a measure)
      (has-a number)
      (has-a size-attribute))
(length (is-a size-attribute))
(metre (is-a length))
(weight (is-a size-attribute))
(kilogram (is-a weight))

```

Currently the choice of these markers is arbitrary, created based on those generally used in the literature (Sowa, 1980; Harris, 1985). This set is expected to grow quite large as the size of the world model grows, and as the databases grow larger. While the number of semantic markers in CD Theory has been kept small, there is a trade-off with respect to processing overhead. As the number of semantic markers grows, the entries in the lexicon become more semantic, and so we tend toward a semantic grammar. For instance, a briefcase might be described not just as "physobj", but as "physobj-27", with all of the attributes of a "physobj", in addition to those of some other entity, "physobj-15", which is a hand-held bag used to hold things, and "physobj-27" also has certain attributes of its own. One goal of this research is to keep this semantic grammar domain-independent, so that portability is not compromised.

When the semantic analyzer receives the parse tree, it traverses the parse tree from right to left, transforming each branch according to appropriate rules which depend on the semantic focus of the phrase. Objects and their

attributes are handled separately from relationships. For each branch of the tree, the redundant and content-empty terms are removed, and the semantic focus is brought to the head of the phrase. For instance, continuing with the processing of sentence s13:

```
(PP (in (prep)) (london (n sm location)))  
becomes (location (value london))
```

and

```
(np (red (adj (colour)) (n (part physobj))))  
becomes (physobj (part) (colour (value red))))
```

The fundamental rule of attaching modifiers to a concept is:

Attach a modifier to the most recently referenced object for which it "makes sense" to attach that modifier.

Each semantically defined object is stored on an agenda. However, before placing it on the agenda, the current item is compared to each item already on the agenda. If an item on the agenda is an attribute of the current item, the agenda item is removed and attached to the current item. Then, using a "first-fit" strategy, if the current item is an attribute of any agenda item, it is attached to that agenda item, otherwise the current item is placed at the top of the agenda. Once one item is attached to another, it



cannot be assigned to any more items.

This process suffices for regular noun phrases and clauses which contain noun phrases, like relative clauses and prepositional phrases. Other types of noun phrases (like noun-noun modification), require different strategies, and these are only handled superficially in QNL at this time by the following rule:

Rule 1.

If two nouns are adjacent to one another then  
  If one of the nouns is a number then  
    it modifies the other  
  else the first noun is a modifier of the second.

This rule is sufficient to handle phrases like "London Suppliers" and "Flight 57". Marcus describes an algorithm for parsing noun-noun modifiers which is more complex than this, but still is unsuccessful in many cases, and relies to a large extent on intuitive judgement (Marcus 1980, pg. 251). It was meant to parse phrases like "water meter cover adjustment screw" (Marcus, 1980, pg. 253).

Basically, the Marcus algorithm attempts to assign one noun to another on the basis of their "semantic goodness", (which could be interpreted to mean "if noun1 is an attribute of noun2, then assign noun1 to noun2, otherwise assign noun2 to noun1. While this works for a phrase like "London flight",

it is not capable of handling "London flight 57", "Tuesday's  
London flight 57 passenger list", and neither is QNL.

```
((for all)
  ((animate (var-113 supplier))
    (((location (value london)))))
  ((physobj (var-116 part))
    (((color (value green)))))
  ((abstractobj physobj (var-115 project))
    (((location (value paris)))))
  ((physobj (var-114 part)) (((color (value red)))))
  ((relationship is)
    (participle (cf agent (rcpt) (object) (qty) (price))
      (lex sell))
    (vt (cf agent (object))
      (lex use)))
  ((command is) (retrieve))
```

Figure 5.7 Domain-independent meaning representation  
which is passed to the DKB

## 5.5 Pronouns

QNL provides a fairly satisfactory way of dealing with relative pronouns, interrogative pronouns and referential pronouns that succeeds in a large number of cases. Distributive pronouns like "each" and "both" are better handled as distributive adjectives. Personal pronouns are basically ignored, although to a user, it would appear the pronouns "I", "me" and "you" are understood. Interrogative pronouns serve as a replacement for the subject of a verb, and are processed by the domain-specific semantic analyzer, as will be explained later.

### 5.5.1 Referential Pronouns

It is very convenient to use a simple term to refer to a concept or event which requires a complex linguistic structure to express:

"a monolithic CMOS technology universal counter circuit evaluation kit" (Tennant, 1980, pg. 115).

The pronoun "it" provides an abbreviated reference for this concept. Consider the alternative to

"Take it and put it in the drawer until it is needed"

Referential pronouns like "it", "them", "they", "ones", etc. make the expression of ideas simpler for a speaker, but they impose a burden of understanding on the receiver of a sentence. In order to understand what the pronoun refers to the receiver has to associate the pronoun with the same concept as the speaker intended. This is formalized by both the sender and receiver maintaining a list of candidate concepts that were either mentioned or implied by preceding discourse. Although referential pronouns usually refer to concepts that have come before, there are exceptions:

"Although it was expensive, he bought the windsurfer anyway."

Pronouns refer to entire concepts, not just individual words:

- sl4. "Are there suppliers in London with status 25?"  
sl5. "Which ones sell red parts?"

The pronoun "ones" is assumed to refer to "suppliers in London with status 25", a complex concept. QNL maintains recently-referred-to concepts in a queue with the most recent concepts at the head and uses the fundamental rule of attachment expressed earlier to replace a pronoun with an appropriate concept. Although this is not a foolproof strategy, it succeeds "about 90% of the time" (Charniak and

McDermott, 1985, pg. 596). In database access where users are requesting information about a small set of different items at a time, this appears to be a good strategy.

When a referential pronoun is encountered, we compare it against each concept in the list. If the pronoun has associated attributes (eg. "red ones"), then a potential match is subjected to a further test: the pronoun's attributes must be allowable attributes of the candidate concept. When a concept is found which matches the semantics of the pronoun, it is removed from the concept list, and replaces the pronomial phrase in the parse tree. A little "jiggling" is required if the concept and the pronoun have the same attributes with different values. For instance, if the candidate concept is "blue parts" and the pronoun is "red ones", the replacement phrase must be "red parts".

sl6. "Which suppliers sell blue parts"  
sl7. "red ones"

This strategy does not always work the way a user might want it to. Consider the sequence:

sl8. "List red parts made by projects in London"  
sl9. "list blue ones"

When parsing sentence s19, QNL would find "projects in london", from s18, to be the most recent concept; "project" is defined as an abstract object, and the WKB defines abstract objects as being allowed to have a colour attribute. So the pronoun phrase "blue ones" is replaced by "blue projects in London". This would be rejected by the domain-specific analyzer since the domain knowledge base has no colour attribute for projects.

Another example of potential error:

s20. "list suppliers and projects in London"  
s21. "which ones are in Paris"

In s21, "ones" will refer to the most recent concept, but because a LIFO queue is used and the processing of the preceding sentence moved from right to left, the pronoun "ones" is taken to refer to "projects in Paris", and no consideration is given to possible ambiguity.

In the current implementation, with processing moving from right to left, this strategy does not find references to concepts in the same sentence. To achieve this, it would be necessary to postpone the pronoun processing until all concepts in the sentence have been recognized. Then it would be possible to discover the referent by analysing the relationship between a verb and a pronoun:

s22. "Which ones sell red ones to the ones in London?"

Although this has not yet been implemented in QNL, the process is understood, and requires a considerable amount of domain-specific knowledge. The semantic analyzer will recognize the case frame associated with the verb "sell", of s22, as

SELL (agent,object,rcpt,quantity,price)

and the domain-specific analyser will recognize that the agent of the verb has not been specified, so will create a variable called "supplier". (This is precisely how QNL currently handles interrogative pronouns). The next role in the case frame refers to an object which has a colour attribute, and so the domain-specific analyser would again be required to generate an appropriate variable. Finally, the preposition "to" usually indicates a destination or recipient, and since the case frame has a recipient role associated with it, the appropriate variable could again be generated by the DKB module.

#### 5.5.2 Relative Pronouns and Relative Clauses

Relative clauses fall into three broad types, and are introduced by either a relative pronoun (eg. "who", "which",

"that"), or a participle, or an ellipsed relative pronoun.  
A "defining relative clause" is essential to the understanding of a noun phrase as it differentiates the noun from other noun phrases of the same class.

"The man whom I saw in Toronto said he was happy."  
as opposed to

"The man said he was happy."

A non-defining relative clause modifies a noun phrase:

"Get suppliers who sell red parts."  
"...the parts which are used ..."

Finally, connective relative pronouns are those which continue the content of a sentence. These are often replaced by "and" or "but", although some transformation rules may also be required.

"I asked Bob who said he didn't know."

Relative clauses need not begin with a relative pronoun, but rather a participle or an ellipsed relative pronoun:

"List suppliers who sell red parts."  
"List suppliers selling red parts."  
"He said (that) he was happy."



A Relative clause is expected to contain a verb and a noun phrase, and refers to a recent noun (although not necessarily the most recent).

s23: "Get suppliers who are in London which sell red parts."

QNL's strategy for processing a relative clause is shown in Algorithm 2:

```
Set a pointer to the start of the phrase.
If the first token is a relative pronoun, discard it
  and set the pointer to the next token.
If the current token is an adverb (eg. negation),
  push it onto a temporary stack and advance the
  pointer.
Process the current token as a verb, which may or may
  not be the main verb of the sentence. Advance the
  pointer.
If what remains is a noun phrase, process accordingly,
  else mark the relative clause as incomplete.
```

#### Algorithm 2: Strategy to process a Relative Clause

In sentence s13, the parser tries to build a relative clause immediately after the noun phrase "suppliers". The parser recognizes the participle "selling", and so sets a pointer to that term. The first term is neither a relative pronoun nor an adverb, so the third line of Algorithm 2 is invoked. "Selling" is processed as a verb, and the pointer is advanced to the next token of the sentence, which is "red". The phrase beginning with this adjective is processed as a

noun phrase, and the relative clause is completed at the end of this noun phrase. The case frame for "selling" is kept separate from the rest of the clause, and the NP "red parts" is stored on the stack to be processed by the semantic analyzer. It will be picked up by the case frame later.

The next relative clause, "which are in Paris", will have the term "which" discarded, "are" is processed as a verb, (but not the main verb, since by default all verbs of a form of "to be" are subsidiary to all other verbs), and then it is marked as incomplete since what follows is a prepositional phrase and could modify any of the preceding noun phrases. The phrase "in Paris" is added to the stack in the same manner as "red parts", and each will be attached to the most recently referred-to object for which they may be an attribute, according to the fundamental rule of modifier attachment, Rule 1. "In Paris" will be attached to "projects" since city can be an attribute of projects, but "red parts" is not an attribute of anything, so it will be left as an aobject on the stack, to be attached to the case frame "SELL".

## 5.6 Conjunctions and Disjunctions

When a conjunction or disjunction is recognized, it is stored in a LIFO queue in a normal form. When the next concept is recognized and added to the agenda, a rule fires which connects the top two items on the agenda by the front element of the LIFO queue.

The first step in this connection process is to verify the existence of a noun in both conjuncts/ disjuncts. If there is no noun in both parts, it will be necessary to generate both a noun and a variable having the appropriate semantic features. A collaboration between semantic and syntactic knowledge is required. Figure 5.8 demonstrates some of the difficulties involved in this collaboration.

If a noun and variable are to be generated, the noun must be semantically appropriate. The entity of the half of the conjunction/disjunction is used as a guide to generate a pseudo-noun, and the semantic analyzer verifies that the allowable attributes of this pseudo-noun are consistent with the attributes explicit in the sentence. So, for the phrase number 6 in Figure 5.8, "red or blue parts", "part" is a physical object, "red" is a colour attribute, physical objects can have colour attributes, the other disjunct has a colour attribute, so it is reasonable to generate a

"physobj" pseudo-noun called "part". However, if the phrase had been "red and blue parts", as in number 3 in Figure 5.8, a rule would simply copy the description of "parts" to the other conjunct, unless of course there were other influences, such as in number 4 which includes a distributive adjective "both", etc.

"A supplier who sells...

"...red parts and blue parts."

"...red parts and blue."

$$\exists x \exists y \exists z (Sx \cdot (Py \cdot Cy, \text{red}) \cdot (Pz \cdot Cz, \text{blue}) \\ \cdot (Vx, y \cdot Vx, z))$$

"...red and blue parts."

$$\exists x \exists y (Sx \cdot (Px \cdot (Cy, \text{red} \cdot Cy, \text{blue})) \cdot Vx, y)$$

"...both red and blue parts."

$$\exists x \exists y \exists z (Sx \cdot (Py \cdot Cy, \text{red}) \cdot (Pz \cdot Cz, \text{blue}) \\ \cdot Vx, y \cdot Vx, z)$$

"...red parts or blue parts."

"...red or blue parts."

$$\exists x \exists y (Sx \cdot Py \cdot (Cy, \text{red} \vee Cy, \text{blue}) \cdot Vx, y)$$

"...only red parts or blue parts."

$$\exists w \exists x \exists y \exists z (Sx \cdot (Py \cdot Cy, \text{red}) \vee (Pz \cdot Cz, \text{blue}) \\ \cdot (Pw \cdot (Cw, \text{red} \vee Cw, \text{blue})) \\ \cdot (Vx, y \vee Vx, z) \cdot Vx, w)$$

Figure 5.8. Nuances of elliptic reference with conjunctions and disjunctions. S = Supplier, P = Part, C = Colour, V = Sells.

Unfortunately, there is no small set of rules which conveniently handles a large collection of conjunctions and disjunctions like the referential pronoun rule does. Although several rules have been implemented, including those described in Figure 5.8, there is a much larger group of conjunctions and disjunctions which are not handled. As the combinations become more complex, the number of tokens in a sentence which must be considered rapidly increases.

#### 5.7 Negation

Negatives themselves can be a little tricky. It is possible to negate attributes of objects, parts of sentences or entire sentences:

"Get names of suppliers who are not in London."  
"Get names of suppliers not in London."  
"Get suppliers not in London who sell red parts."  
"Which suppliers in London do not sell red parts."  
"Which suppliers who sell red parts do not sell  
blue parts."

The attachment of a negation to an object or relationship is determined by a lookahead (from right to left) which attaches the negation to either the top of the agenda or the most recent relationship. At this time, only explicit negation is handled, and double negatives are left as two negations. The negation is stored in a stack while the rest

of the phrase is processed normally. Then the negation is popped from the stack and attached to the front of the phrase.

Conjunction and negation can also be combined in other interesting ways:

"Who sells red parts and not blue?"  
"who sells red parts but not blue"  
"Who sells all colours but blue?"

## 5.8 Verbs

Verbs in this system either are content-empty, or they have one or more case frames, or they are database commands like "sort", "find", "list", etc.

When a database command is recognized, it is stored in a command register and passed straight through to the DKB. Accepting content-empty verbs like "have" and "are", requires considerable effort on the part of the NLI. QNL will hypothesize a relationship based on the objects in the sentence, as opposed to forcing a system designer to anticipate all verbs and their arguments. A discussion of how the DKB resolves such implied relationships appears in section 15 of this chapter.

1

2

Verbs with a case frame refer to a relationship between some domain objects. The case frame makes specific those cases which play obligatory and optional roles in the relationship. In the early stages of development of this prototype, the general semantic analyzer assigned the objects in the sentence to roles in the relationship, but this approach has been put aside for now. The domain semantic analyzer verifies these role assignments, so doing it in the WKB duplicated the effort. The current method results in the output of the general semantic analyzer not really being a meaning representation, since it has left some of the work undone; however, all of the information required to generate the meaning of the sentence with respect to the WKB is still there in the form of a list of concepts, commands in a command register, and optionally, one or more relationships involving the objects.

This implementation ignores the concept of time, so all verbs are in either the present simple tense, the present continuous (auxilliary and participle), infinitive or imperative. Auxilliaries such as "might", "can", etc., are not handled.

Many NLI do not handle verbs at all. ASK (Thompson and Thompson, 1983, 1985) allows only content-empty verbs of a form of "to be" or "to have". Other systems consider

meaningful verbs so crucial that "if the range of acceptable English is so small as to exclude verbs, then the user is better off learning a formal query language; natural language will not be very useful" (Grosz et al., 1987). QNL accepts content-empty verbs as well as domain predicates. Examples of such predicates are "supply" and "use" in the Supplier-Parts-Projects domain; "are assigned", "working", "are booked", "are certified", and others in the Airline domain. There are many verbs which have special meanings within one domain, but are content-empty in other applications: "available", "able", "get", etc.).

#### 5.9 Anaphora

People frequently speak in grammatically incorrect sentences, and often they leave out information which is understood or implied between them. QNL has a limited ability to accept ill-formed input, and to carry on a running dialogue. It can correctly handle a sequence similar to the following:

"Which suppliers sell red parts in Toronto?"  
"blue ones"  
"yellow"  
"to projects in Paris"  
"what about green?"



Many anaphoric references omit a verb, and this is how QNL recognizes its occurrence. The omission of a verb triggers a set of rules which look at the previous context, and match the current context with it. If an object in the current context matches an object in the previous context, those two are scrutinized for a potential match. If the current context refers to objects not previously mentioned, the new objects are accepted, as is. If an object was referred to before, its attributes are compared with the previous context, and if the previous context had more or different attributes, the different attributes from the previous context are merged with the current context. It is also possible to add modifiers to an object in the previous context. Once all possible matches are made, those choices not eliminated are merged with the current context to generate a new context. For the user's next question, this current context will become the previous context.

#### 5.10 The Domain-specific Knowledge Base

The objects and relationships occurring in the database are described in the Domain Knowledge Base (DKB) using the same semantic primitives as define the WKB. The DKB is also a semantic network. So if we have a relation "part", which has the attributes "size", "colour", "identifier", it would

be defined in the DKB as a "physobj", and the attributes which it actually possesses would be explicitly defined.

The domain-specific analyzer receives a hierarchical meaning representation from the domain-independent analyzer. It traverses this tree, using a heuristic lookahead to determine which domain objects are being referenced. We will continue using sentence s13 and describe how the DKB transforms the data structure passed to it from the WKB into a domain-specific meaning representation, as shown in Figure 5.9.

```
(for all)
  ((var-113 dsupplier (animate)) ((location (london))))
  ((var-116 dpart) (physobj)) ((colour (green)))
  ((var-115 dproject) (abstractobj)) ((location (paris)))
  ((var-114 dpart) (physobj)) ((colour (red)))
  ((and)
    ((dsupply ((var-113 dsupplier (animate))
               ((location (london))))
      ((var-114 dpart) (physobj))
      ((colour (red))))
      ((var-115 dproject) (abstractobj))
      ((location (paris))))
    ((duse ((var-116 dpart) (physobj))
      ((colour (green)))
      ((var-115 dproject) (abstractobj))
      ((location (paris))))))
```

Figure 5.9 Domain-specific Meaning Representation

For each object in the tree, QNL determines if the object is defined in the domain. If it does not find the object, it hypothesizes that it could be an attribute of some other

object. If this hypothesis returns a unique object, then we assume a successful match and attempt to process it. If unsuccessful, we postpone further processing of that object until the entire tree has been traversed.

QNL uses already-recognized objects and relationships to help guide its search. In a narrow domain this is an effective way of resolving ambiguities, since there is a small number of objects and relationships, but in a large application, some user-interaction may be necessary.

When it is determined that an object does exist within the domain, the attributes referred to by the user, if any, are compared with the attributes actually appearing in the domain. If the attributes match, then the DKB recognizes which object the user is referencing.

The user may refer to a domain concept which has more than one object, and the process of matching attributes may eliminate some of the objects from consideration. If more than one candidate object still remains, then it will be up to the DBA when specifying the user views, to determine which domain objects should be returned. This approach is used in (Waltz, 1978). QNL uses a priority list with priorities established by the needs of the users and the characteristics of the domain.

### 5.11 Linking the DKB to the WKB

If an object or relationship exists in a domain which is not defined in the WKB, the WKB must be augmented to describe this new type of entity. In this respect the DKB is always a subset of the WKB. All objects in the domain must be described in terms of the WKB: either in the primitives of the WKB, or in terms of objects which are described in terms of the primitives. The only exception to this is the attribute-values which may map directly between a token in the input string to an attribute in the DKB.

A domain-object may have attributes not generally associated with those objects in the world model. The domain-specific definition of "part" may also have attributes which would not normally be ascribed to parts in general, but are necessary in the domain. For instance, expiry date, replacement part, etc.

If an attribute is added to an object which is not normally part of that object in the real world, one of two things may happen: the DBA may choose to augment the WKB definition of the object, or he may elect to add that attribute only in the DKB.

If the attribute is added in the WKB, potential ambiguity may be introduced. If the definition is only in the DKB, extra processing is required, but these procedures are already built-in to the system.

In the SPJ database, supplier, an animate object, has an attribute called "status". This is not normally an attribute of an animate object, so if the attribute were added to the primitive "animate" in the WKB, then every animate object could have a status.

If a user asks about the status of parts, the WKB would reply that parts cannot have a status. But this is only true in this domain, because parts may very well have a status in some other domain, and so we are compromising the WKB. We should allow a user to refer to a concept of status, and then have the DKB be responsible for explaining that parts do not have an attribute of that type.

Status only refers to suppliers with respect to a particular domain. So it would seem more reasonable to define status only in the DKB.

The preprocessor finds the token "status" in the dictionary, but its semantic marker will not allow it to be attached to anything in the WKB. The WKB accepts that "status" may have special meaning in the domain, and so attaches it as an


independant object to the tree. At the domain-specific semantic analysis stage, the DKB will match status and supplier.

The WKB may need to be augmented to expand its concept of the world. For instance, when the current system was moved from the Suppliers-Parts-Projects domain to the Airline domain, it was necessary to add the concepts of day-of-week, source-location and destination, phone number etc. to the WKB. The lexicon also had to be expanded to include nouns, verbs, and adjectives which describe objects in the domain, such as flights, aircraft, passengers, etc.

Some of these concepts should be permanently added to the WKB, others may only be application-specific additions. A program could be created to access existing WKBs and use the knowledge of similar domains to guide the process of augmenting an existing WKB for a new domain. People accumulate a large amount of knowledge over time, but occassionally we have to be reminded about things which we may forget. We could conceive of the WKB storing knowledge which is not very useful in a specific application in a secondary memory, which could be called in when appropriate. The knowledge would always be there, just not used in certain situations.



Each WKB concept may refer to 0, 1 or n DKB concept, and a DKB concept may be referred to by 1 or more WKB concepts.




Each concept in the DKB will refer to objects or relationships, or attributes of them. Again, the mapping could be m:n, but note there must not be a DKB concept which does not map to some relation in the DKB.

#### 5.12 Database Values

In any database environment, users could be expected to refer to objects by their attribute values. If all of these values are stored in the lexicon, the lexicon will become extremely large, and it will be duplicating information in the database.

If the user is asked to define each value as it appears in a question, there is the potential for user frustration and also for the user to introduce errors by incorrectly describing the semantics of a value. A trade-off then is to store value sets and frequently occurring values in the DKB where they can be accessed by the lexical analysis stage. Further, there are certain WKB concepts which could be considered as values in a database, like day-of week and colour, which could apply to any domain. These concepts are explicitly stored in the WKB.





### 5.13 Transporting to a New Domain

The degree of portability of a NLI depends on how the system designers view the problem of accepting new domains, new DBMS, new grammars and languages, different computer systems, etc. For the purposes of this study, we only consider the introduction of a new domain.

There appears to be a trade-off between robustness and efficiency of the natural language understanding component on the one hand, and ease of portability on the other. Systems such as PRE (Epstein, 1985), CO-OP (Kaplan, 1984) do not have strong NL understanding capabilities, but transporting to a new domain is a matter of a few hours. TEAM (Grosz et al., 1985, 1987) and ASK (Thompson and Thompson, 1985) employ extensive software aids to acquire a new application, and the Ginsparg system (Ginsparg, 1983) requires a person knowledgeable of the NLI to assist the process. The degree of portability is gauged by the time and effort required to complete the task: the knowledge and skills required for the task, and the degree of success of the understanding of questions in the new domain.

The PRE system (Epstein, 1985) does not require a computer program nor a person familiar with the NLI to perform the operation. As with other NLI which are basically a mapping

between attributes, objects, values and a dictionary, like ASK (Thompson and Thompson, 1985) and CO-OP (Kaplan, 1984), the process is relatively straight forward, requiring a matter of hours even for a non-trivial database. As the NL processing knowledge base evolves from schema-mapping or table lookup to a more robust conceptualization, portability becomes more difficult, and so extensive tools or expertise is necessary.

#### 5.14 Designing the domain-specific Knowledge Base

The design process is not unlike the designing of the database itself, in that the objects, attributes, value sets and relationships must be specified. Since we are employing semantic nets as the formalism, it is a straightforward matter of transferring the concepts from a data design model like SDM or the E-R model to the semantic net. The relationships are kept intact, but also new connections are recognized and implemented to reflect the different ways of viewing the domain.

A major difference between QNL and conceptual design models is that primitives in QNL are high-level objects like physobj, animate, etc., while in database design models the primitives are typically low-level objects like string, char, integer, and so on.

The value sets particular to a given application must be specified in the DKB, so that the semantic analyzer may use them to make inferences about attaching those values to objects and attributes. But rather than specifying them in terms of char and integer, we are interested either in the range of values (S0001 to S9999 for a range of identifiers), or the form of the values (999-9999 for a phone number) (Damereau, 1985).

#### 5.15 Capturing the Semantics of the New Domain

To demonstrate the portability of QNL, the front end was attached to a new database other than the ones tested during the design of the system.

The University conceptual model is structurally different from the Suppliers-Parts-Projects database (SPJ) in that there are two relationships instead of one, and each relationship has only two participants. The number of attributes of each entity has been kept small for demonstration purposes, but the attributes were chosen to show how ambiguity is handled ("location" in SPJ and "name" here). From start to finish, the introduction of this new domain to QNL took four hours. Figure 5.11 shows an entity-relationship diagram for the University database.

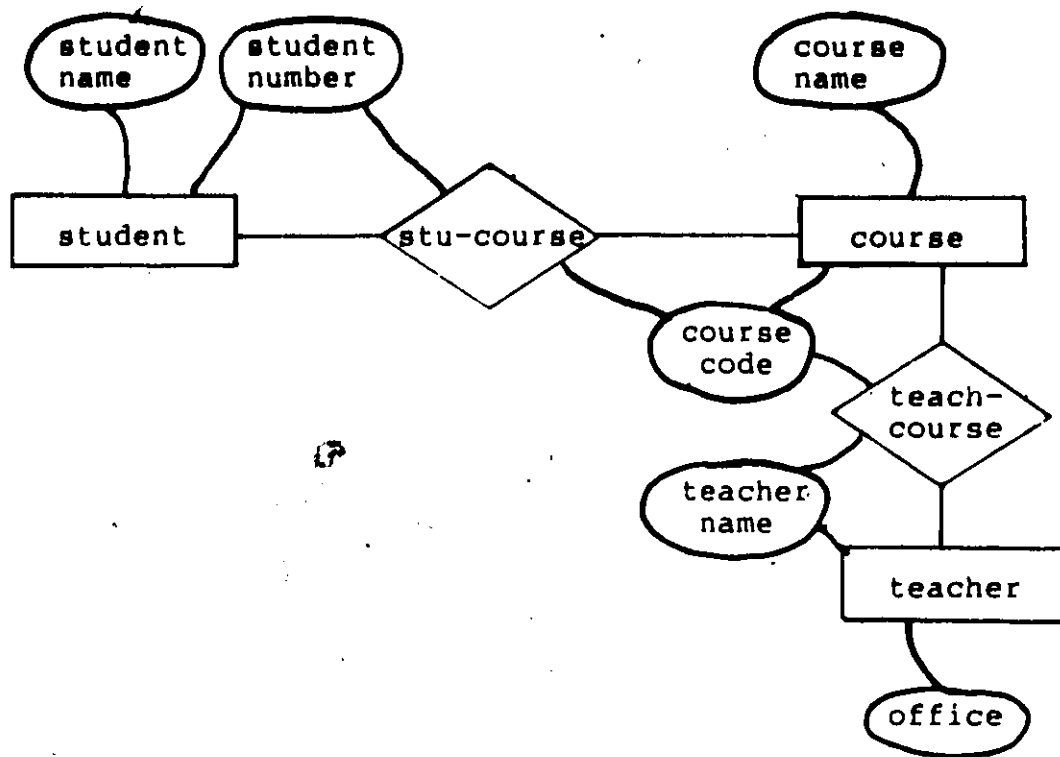


Figure 5.11: Entity-Relationship diagram for a simple University database

The use of the E-R diagram here is for demonstration purposes only, and not because it is necessarily the "best" data design model to be used in this process.

The first step is to construct a graph with the objects as nodes and the relationships as labelled arcs between them. The names of the nodes and arcs are inconsequential, as long as they are unique identifiers. Each node is defined in

terms of some object in the WKB, and has some specific attributes which are defined in the DKB. The IS-A link connects the object to the WKB, and the HAS-A links connect the attributes to the object. The inverse of HAS-A is ATTRIB-OF, and all attributes are stored in an inverted list linking them to their objects. Each attribute in this list may also have properties which are connected to it by IS-A and HAS-A arcs. All links between objects and their attributes are symmetrical.

Objects and attributes must be defined in terms of the WKB. For instance:

```
teacher is-a human
         has-a office
```

What is an office? It is more than just a location, since it holds furniture, probably has a telephone, is a meeting place, etc. It must be linked to the WKB so that it is explained in terms which both the user and the database understand. Our chosen solution here is to define it as an abstract object in the WKB:

```
office is-a abstractobj
         has-a location
```

At this stage, a more detailed description of office is unnecessary.

Objects are explicitly linked to other objects by the roles they play in relationships. This role actually has nothing to do with the enterprise schema, but is important for the natural language understanding process. Students and courses are related because students register in a course. This can be described as

stu-course (student, course)

What roles do these objects play? This depends on the semantics of the relationship. In the case of ambiguous semantics, all interpretations will be specified. Each relationship has compulsory and optional participants; the use of the compulsory ones aiding in the resolution of interrogative pronouns.

The definition of relationships is symmetrical: The role an object plays in various relationships is defined, and the relationships are defined in terms of their participants. There may be more than one way of perceiving a relationship, and sometimes this may not map 1:1 with the relationships of the domain. In such a case, it is necessary to define a special relationship which is a subset of a domain relationship, and then map it to that relationship. The mapping occurs in the DB mapping module which occurs after the NLI.

A real implementation would need to access the database to build value sets, and for this domain, the value sets are obviously large: the set of valid student numbers, and the set of person names. It is not appropriate to store all of these values in the KB, but a range could be specified for the student numbers. The problem of handling people's names could be solved by storing a set of valid person-names in the WKB or DKB. It should always be possible for the user to add new names interactively. Although the number of people may be large, the variety of names is not as large, and so we could adopt a convention of storing frequently occurring values in the KB. For instance, the entries "smith" and "roy" need only occur once, but could account for a large number of attribute values in the database. As the size of the database grows, the size of the value sets for some objects or attributes will grow less fast than the number of occurrences, as shown in Figure 5.12.

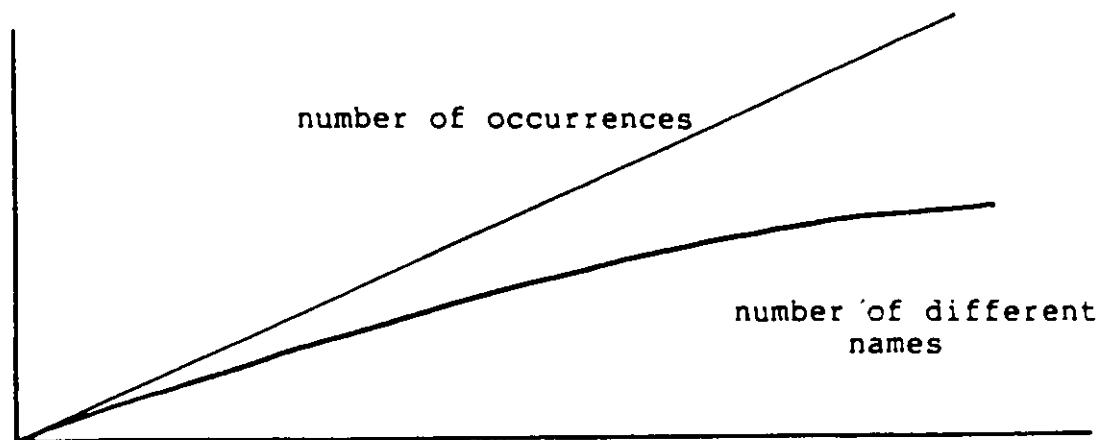


Figure 5.12: Relationship between number of occurrences of entities, and the number of different names.

Some value sets are specific to a domain, and should not be added to the WKB. In the University domain, these would include the range of student numbers, course codes, office numbers, etc. The chosen approach is to maintain a list of special terms which map directly from these values to attributes in the DKB. The WKB does not try to interpret the semantics attached to these values. So, for instance, in the airline domain, if a user asks about a B747, the NLI will recognize it as a value of an attribute called "model-no". Likewise, in the University domain, the value CSC113 will be mapped directly to "coursecode".

The lexicon must be expanded to accept the words which describe the new domain. It already has a store of closed-class words such as determiners, prepositions, conjunctions, etc., as well as a number of content-empty verbs (do, does, has, have, is, are). There are few nouns, so this is the first part of the lexicon to be expanded. We need to know all of the nouns which will be used to refer to the objects in the domain, and so we choose one noun which links directly with a WKB concept, and then define synonyms for it. Some synonyms may refer to several links, although in a narrow domain, this should not be a problem (Blanning, 1984). If an ambiguous term is found, either the WKB will resolve it by intersecting its knowledge about the other



items referred to in the sentence, or interaction must occur with the user.

The WKB concepts are linked to the DKB via a DKB-map, and the mapping may be 1:1 or m:n. The map is not crucial to the success or failure of the understanding process, but it certainly improves the efficiency by narrowing the search area. In those instances where the map cannot be used because no WKB concepts were referred to, the value sets are scanned for hints about the DKB concepts, using either the value or form of the tokens. If this doesn't work, then an exhaustive search of the DKB objects takes place, looking for semantic closeness between the semantic focus of the user's expression and the domain concepts.

The relationships in the domain are referenced by case frames, and the lexicon is augmented by the addition of verbs and the case frames to which they refer. Synonyms will refer to a case frame, and this case frame will either map directly to a case frame in the DKB, or one of the DKB relationships will be a subset of a WKB case frame.

It is also convenient to introduce content-empty verbs into the lexicon which make the use of natural language more convenient. Some of these verbs could have multiple case frames, but rather than define all possibilities (and risk

errors), the understanding of the relationship is left to the DKB, which looks at the attributes to discover an implied relationship. Such ambiguous verbs include "take", "available", "go", "offer", etc.

#### 5.16 Implementation and Testing

In the early stages of the development of DNL, a variety of question forms were tested to determine how to handle different grammatical and semantic structures. Three basic question types are summarized below, and examples are shown in Appendix D.

##### 1) Simple questions about objects and their attributes.

- s23. "Get suppliers in London"
- s24. "Get names of passengers"

In sentence 23, the prepositional phrase is an attribute-value which modifies the preceding noun, while in sentence 24, the noun, "names", is an attribute of the focus of the following prepositional phrase. The fundamental rule of attachment, as described earlier, is sufficient to handle both types of structures.

##### 2. Explicit relationships between objects.

- s25. "which suppliers sell red parts"
- s26. "which passengers are booked on flight 57 on Tuesday"
- s27. "who teaches csc111"

The verbs "sell", "are booked" and "teaches" trigger case frames with appropriate roles to handle the objects mentioned in the questions.

### 3. Implied relationships between objects.

- s28. "get suppliers of red parts"
- s29. "get me a flight to london on Tuesday"
- s30. "which students are in csc113"

The user has not used a verb to specify a relationship between the objects in the question, so it is up to the system to determine what the relationship is. In the second question, the user is probably not even aware that an implied relationship is involved (Flights have destinations, but Departures have departure dates. A Departure is an instance of a Flight.) The WKB will identify the objects and pass them on to the DKB. The DKB will recognize that more than one object has been mentioned, and so search the semantic net for relationships between the objects.

The ability to accept implied relationships is crucial to the success of an intelligent NLI because it truly frees the user from having to know about the structure of stored data. Embellishments to these basic question types as described in Appendix D include the use of pronouns, predicate adjectives, relative clauses, anaphora, conjunctions, disjunctions and so on.

Once the NLI was suitably configured to handle these 3 basic question types, an experiment was tried with a group of 12 "naive" users. These people were undergraduate business students in an introductory microcomputer course. Although they were by no means database experts, they were somewhat computer literate, and understood the purpose of a NLI to a database. None of them had ever seen the Supplier-Parts-Projects database. Although a "snapshot" of the database was presented to them, none of the volunteers referred to it while making up their questions.

Each participant was given a list of imperative commands for the database, examples of which are shown in Figure 5.13. Each command was in a standard form, and they were asked to create two (2) different ways of issuing each command, either in imperative, interrogative or declarative form.

- 1) Get full details of all projects in London
- 2) Get the projects for which s1 is a supplier
- 3) Get the suppliers who supply a London or Paris project with a red part

Figure 5.13: Sample queries in imperative form on the Suppliers-Parts-Projects database

Some of the forms of question 2 in 5.13 as asked by the subjects included:

"s1 supplies which projects?"  
"list projects s1 supplies"  
"which projects does s1 sell to?"

Each participant wrote the questions on a sheet of paper, and these were tested with the program later. This way, the participants were not frustrated by minor bugs in the program, but they also missed out on important feedback.

Figure 5.14 shows the results of the test on the Suppliers-Parts-Projects database:

Q#	Declar- ative	Imper- ative	Interrog. pronoun	Other interrog	total	# correct	% correct
1	2	9	5	0	16	13	81.25
2	1	4	12	0	17	17	100.00
3	0	4	12	0	16	14	87.50
4	0	6	7	4	17	15	88.24
5	1	4	9	0	14	13	92.80
6	0	5	11	1	17	4	23.53
7	1	4	8	0	13	12	92.31
8	1	6	7	0	14	12	85.71
9	1	4	5	1	11	5	45.45
10	1	4	9	0	14	11	78.57
totals	8	50	85	6	149	116	77.85

Figure 5.14: Results of test queries on the Suppliers-Parts-Projects database

The participants developed patterns which they adhered to rather rigidly, and their phrase structures did not deviate much from the samples they were given. The lack of immediate feedback meant that they could make the same error in several queries without being aware of it.

The types of errors made which prevented QNL from processing the query included syntax violations (use of contractions and punctuation was not allowed), and the use of quantifiers. There were many instances of ill-formed questions, including verb-subject mis-match and number agreements, but QNL handled many of these. Two questions resulted in very low success rates because the samples were ill-formed, and many of the students simply copied the phrases from the samples.

As was mentioned in the introduction and cited in Tufig and Cristea (Tufig and Cristea, 1985) and Thompson, (Thompson, 1980), an effective way of training a set of users on the use of a natural language interface is to present them with a set of sample questions. After a very short period of study, users will naturally fall into certain patterns of usage. This phenomenon was observed during this experiment.

For instance, for the sample question

"Who supplies a London or Paris project with a red part?",

7 out of the 10 people who tried this query used the exact expression "London or Paris project with a red part", even though they were instructed to come up with different ways of asking the question. This tendency was especially noticeable as the questions became more complex:

For example,

"Get the projects not supplied with any red part by any london supplier"

saw the partial sentence "not supplied with any red part by any london supplier" repeated 4 out of 11 times, "which projects are not supplied", 5 times, and the phrase "london suppliers" 11 times.

To test the transportability of the interface, QNL has been tested with two other domains: a university database with information about professors, students and courses, and an airline database, with information about flights, departures, employees, passengers, and aircraft. The same types of questions were asked of each domain as were asked of the Suppliers database. This was done to verify that the language understanding capabilities were transportable. If different types of questions had been asked in any one of the domains, then we could not be certain of complete portability. In other words, if a question type works in only one domain, it could be that the peculiarities of that domain allow an interpretation that is accidentally correct. This is possible because QNL will always try to build some interpretation of a question unless certain pre-determined rules are violated.

Moving to the university database required approximately four hours of programmer time, including building the domain semantic net, and augmenting the lexicon and world knowledge base with more world knowledge. Sample queries are shown in Appendix D. The semantics of both the university domain and the suppliers domain are relatively simple, with a small number of entities joined by fewer relationships. The semantics of the airline domain were another matter.

In the airline domain, any single query could require many relationships to be inferred, and the procedures of the domain-specific semantic network were not sufficiently robust to handle multiple inferences. However, when the DKB was reconfigured to reduce the number of relationships (but still retain the semantics of the domain), the performance of the NLI improved considerably. Thus, the problem obviously lies with the DKB semantic net procedures.

#### 5.17 Summary

The process of introducing a new domain involves characterizing the objects and relationships in terms of a semantic network called the DKB, which has a few links to another semantic network called the WKB. Value sets are described so as to map directly to either WKB concepts like "name", or to DKB attributes. The lexicon must be augmented



to include all of the terms and synonyms used to refer to the domain. These terms are either content-empty or are described in the semantic net, and linked to domain concepts. The domain concepts map m:n to objects in the domain.

The final stage is to test the NLI with users of the domain. In the case of the University database, the testing was quite successful with the major exception of the handling of conjunctions and disjunctions. It is apparent the combined syntax-semantic rules which worked in the SPJ domain are not transportable.

A third domain was investigated as part of this thesis, one which is much more semantically rich than either the SPJ or University applications: that of an airline domain. The procedures to answer simple questions about objects and attributes, objects expressed only by attributes or attribute-values, and explicit relationships worked as well as in the SPJ domain. Also, negations, pronoun references and anaphora were correctly processed. The problems emerged with implied relationships and complex relationships, and this failure is directly attributable to the DKB procedures which manipulate the semantic net. In order to reduce the possibility of unrestrained inferencing in the semantic network which could lead to severe inefficiencies,

manipulations of the net are guided by "a priori" knowledge. The procedures look for specific links, and only follow these links a pre-determined distance. Therefore, they often never find a correct inference, but return with whatever information was at their furthest node. These procedures will have to be untethered to respond better to their environment.

Unrestrained inferencing may not be a problem with respect to database access because there are few object types which are of interest in any domain, although there are many occurrences of each type. The current implementation of the semantic net was further constrained by the memory limitations and CPU speed of the machine, and so compromises were made at the expense of robustness. A future implementation must allow data-driven inferencing in a semantic net of arbitrary size.

## Chapter 6

### Concluding Remarks

#### 6.1 Conclusion

This thesis has studied the issues in the design of a transportable natural language database interface. After completing a survey of recent transportable NLI, a modular design was decided upon, and a prototype question-answering system was implemented on an IBM PC/XT using TLC Lisp.

This is a question-answering system in that users form questions or give commands in natural language that will result in the retrieval of information from a database. Commands such as "sort" and "count" are supported, as well as Yes-No questions, Wh questions, and explicit requests for the retrieval of information. It is not possible at this time to use this system to update a database.

The prototype, "Queries in Natural Language" (QNL) divides the process of understanding a natural language question or command into three distinct stages: syntax analysis, domain-independent semantic analysis and domain-specific semantic analysis. Each of the steps uses particular types

of knowledge, divided into declarative and procedural modules.

In the Syntax analysis stage, a deterministic, one-path parser, modelled after the ideas of Marcus (Marcus, 1980) was implemented. A one-path parser was chosen to constrain the overhead caused by local ambiguities, both syntactic and semantic, and therefore improve overall efficiency by reducing the amount of work to be done in the semantic analysis stage. The parser recognizes tokens in the input stream and groups them into phrase structures. The types of tokens include content-empty words, nouns and adjectives which refer to generic concepts, conjunctions, disjunctions, pronouns, and verbs.

The domain-independent semantic analyzer, or World Knowledge Base (WKB), consists of a semantic network, and a set of procedures which operate on the network. The semantic net formalism was chosen to provide an inheritance hierarchy, and to be used as a tool for inference making. All objects in the world model are specified as instances of certain primitives, like physical object, human, and so on, with attributes particular to the object.

Semantic nets have been extensively used in natural language systems, and have been used in conjunction with Case Grammars. They were initially designed as inferencing tools, providing an inheritance hierarchy, and simple means of recognizing objects by their attributes, testing for set membership, determining similarities between objects, and so on. Semantic nets have been used successfully in several recent NLI, and so for these reasons, this formalism was deemed to be an appropriate knowledge representation scheme for QNL. The knowledge in the net is declarative, and it was an objective of this thesis to separate declarative knowledge from procedural knowledge as a means of attaining portability. The procedures which manipulate the semantic net are independent of the entities and relationships of the network.

The domain-independent semantic analyzer attaches modifying phrases according to a "first-fit" strategy encompassed in a fundamental rule: Attach a modifying phrase to the most recently referred-to object to which it makes sense to attach the modifier. The WKB is also responsible for resolving ellipsis, anaphora, and pronouns other than interrogative pronouns. A set of rules to handle conjunction and disjunction was implemented, but it later became obvious that no small set of general rules could handle this issue.

The WKB passes a tree-structured meaning representation to the domain-specific knowledge base (DKB). This is another semantic net which verifies the existence, within the domain, of the entities, attributes, and relationships as determined by the WKB. Interogative pronouns are resolved here, as are other ambiguities which could only be resolved using domain-specific knowledge; such as implied relationships. The output of the DKB is an expression which could be transformed into a relational calculus or relational algebra expression, or even directly into the data manipulation language of a database management system.

Transportability is achieved by changing the declarative knowledge in the DKB; the procedures which manipulate the domain semantic net are left untouched. Currently, the WKB is still small since it is only at the prototype stage, so it is necessary to also augment the declarative knowledge within it, but the effort required to do so will reduce as the WKB grows.

The system was tested by moving the NLI to two other domains other than the original one used in the design process: a university database (Appendix B) and an airline database (Appendix C).

For the university database, a fairly simple, but structurally (and semantically) dissimilar domain, QNL worked successfully within a few hours of initiating the transportation process. For the airline domain, which was much more complex, although the simple questions like asking about objects and their attributes worked consistently, more complex queries, for instance involving implied relationships among several entities, failed. These failures are directly attributable to weaknesses in the procedures which manipulate the semantic networks. Currently, these procedures are limited as to the number of links which may be followed in a search for information.

## 6.2 Future Work

Future work with respect to this research should focus on three (3) areas. First, the NLI should be applied to real database management systems. That is, while analysis shows the output of the DKB to be correct and useful, we still need to study the process of transforming these expressions into those which can be manipulated by a DBMS. The procedures to do this will require knowledge of both the files and records of the database, and the entities and relationships of the DKB. It will basically be a mapping between these two types of knowledge, but may also include query optimization.

The procedures which manipulate the semantic nets must be improved to allow probing through more links than is currently permitted. These procedures were purposely constrained for efficiency considerations, but practice has shown first that it is seldom necessary to search through large areas of the networks, and second, that when significant inferencing is required, artificial constraints are unacceptable.

More work must be done on the grammatical and semantic analysis to improve the handling of conjunctions, disjunctions and pronouns. Work must also be done on complex phrases involving comparatives like "between" and "greater than". As was explained in Chapter 1, the application of syntactic and semantic analysis should cooperate in the understanding process. There are times when syntax does best, and times when semantic analysis is more useful. The development of any language understanding process requires formal decisions about how and when to apply these different types of knowledge.

Natural language communication with computers will make it easier for people to use computers and access complex information, but the process of understanding precisely what a user wants is difficult to achieve. As many researchers have already pointed out, tailoring a NLI for specific



applications is an effective way of allowing easy access to a computer, but it is not necessarily true understanding of language. This latter task is much more complex, and requires a concentration of efforts by linguists, computer scientists, psychologists and others. This thesis has been a small step on a long, long road.

## References

Addis, T.R., Designing Knowledge-Based Systems, Prentice Hall, Englewood Cliffs, New Jersey, 1985.

Aho, A.V., and Jeffery D. Ullman, The Theory of Parsing, Translation and Compiling, Prentice Hall, Englewood Cliffs, New Jersey, 1972.

Barr, Avron and Feigenbaum, Edward A., The Handbook of Artificial Intelligence, William Kaufmann, Inc., Los Altos, California, Vol. I, 1981.

Berwick, Robert C., The Acquisition of Syntactic Knowledge, The MIT Press, Cambridge, Mass., 1985.

Bibel, W. and Petkoff, B., Artificial Intelligence: Methodology, Systems, Applications, Proceedings of the International Conference on Artificial Intelligence: Methodology, Systems, Applications (AIMSA '84), Varna, Bulgaria, (North Holland), September, 1984.

Bic, Lubomir and Gilbert, Jonathan P., "Learning From AI: New Trends in Database Technology", IEEE Computer, March, 1986, pp. 44-54.

Bobrow, D.G., "Natural Language Input for a Computer Problem-Solving System", in Minsky, Marvin (ed.) Semantic Information Processing, MIT Press, Cambridge, Mass., 1968.

Boguraev, B.K., and Sparck-Jones, Karen, "A Natural Language Analyser for Database Access", Information Technology: Research and Development, 1982, pp. 23-29.

-----, "How to Drive a Database Front End Using General Semantic Information," Proceedings of the 1983 Conference on Applied Natural Language Processing, 1983, pp. 81-88.

Bruce, Bertram, "Case Systems for Natural Language", Artificial Intelligence, Vol. 6, 1975, pp. 327-360.

Carbonnell, James G., Mark W. Boggs and Michael L. Mauldin, "XCALIBUR Project Report 1: First Steps Towards an Integrated Natural Language Interface", Department of Computer Science, Carnegie-Mellon University, 1983.

Castro, Luis, Jay Hanson and Tom Rettig, Advanced Programmer's Guide Featuring dBase III and dBase II, Ashton Tate, Culver City, California, 1985.

Charniak, Eugene, "A Common Representation for Problem Solving and Language Comprehension Information", Artificial Intelligence, Vol. 16, 1981, pp. 225-255.

-----, and Drew McDermott, Introduction to Artificial Intelligence, Addison-Wesley Publishing Company, Reading, Mass., 1985.

Chen, Peter Pin-Shan, "The Entity-Relationship Model - Toward a Unified View of Data", ACM Transactions on Database Systems, Vol. 1, No. 1, March, 1976, pp. 9-36.

-----, "The Entity-Relationship Model - A Basis for the Enterprise View of Data", Proceedings of the AFIPS Conference, National Computer Conference, NCC-77, 1977, pp. 77-84.

Chomsky, Noam, Syntactic Structures, Mouton, The Hague, Netherlands, 1957.

-----, Aspects of the Theory of Syntax, MIT Press, Cambridge, Mass., 1965.

Craig, J.A., Berenner, S.C., Carney, H.C., and Longyear, C.R., "DEACON: Direct English Access and Control", Proceedings of the Fall Joint Conference, Montvale, New Jersey, AFIPS Press, 1986.

Damereau, Fred J., "Problems and Some Solutions in Customization of Natural Language Front Ends", ACM Transactions on Office Information Systems, Vol. 3, No. 2, April, 1985 pp. 165-184.

Cullingford, R.R., and Mallory Selfridge, "Real-World Natural Language Interfaces to Expert Systems", Proceedings, Trends and Applications, 1983: Automating Intelligent Behaviour Applications and Frontiers, IEEE, Gaithersburg, Maryland, May 25-26, 1983. pp. 89-93.

Date, C.J., An Introduction to Database Systems, 3rd Edition, Addison-Wesley Publishing Company, Reading, Mass., 1982, pg. 114.

Earley, J., "An Efficient Context-Free Parsing Algorithm", Communications of the ACM, Vol. 13, 1970, pp. 94-102.

Epstein, Samuel S., "Transportable Natural Language Processing Through Simplicity - The PRE System", Transactions of Office Information Systems, Vol. 3 No. 2 April, 1985, pp. 107-120.

Fillmore, Charles, "A Case for Case", in Bach, Emmon and R.T Harms (ed.) Universals in Linguistic Theory, Holt, Rinehart and Winston, CBS College Publishing, 1968.

Findlar, Nicholas V., ed. Associative Networks: Representation and Use of Knowledge by Computers, Academic Press, New York, 1979.

Friedman, Joyce, Brett, Thomas H., Doran, Robert W., Pollack, Barry W., and Theodore S. Martner, A Computer Model of Transformational Grammar, American Elsevier, New York, 1971.

Frost, R. A., "Using Semantic Concepts to Characterise Various Knowledge Representation Formalisms: A Method of Facilitating the Interface of Knowledge Base System Components", The Computer Journal, Vol. 28, No. 2, 1985, pp. 112-116.

Ginsparg, Jerrold M., "Natural Language Processing in an Automatic Programming Domain", (Phd. Thesis) Stanford University, Department of Computer Science, 1978.

Ginsparg, Jerrold M., "A Robust Portable Natural Language Database Interface", Proceedings of the Conference on Applied Natural Language Processing, Association for Computational Linguistics, Feb. 1983, pp. 25-30.

Griffith, Robert L., "Three Principles of Semantic Nets", ACM Transactions on Database Systems, Vol. 7, No. 3, 1982, pp. 417-442.

Grishman, Ralph, "Natural Language Processing", Journal of the American Society for Information Science, Vol. 35 No. 5, 1984, pp. 291-296.

-----, Hirschman, L., Friedman, C., "Isolating Domain Dependencies in Natural Language Interfaces", Proceedings of the Conference on Applied Natural Language Processing, Association for Computational Linguistics, February, 1983, pp 46-53.

Grosz, Barbara J., "Transportable Natural Language Interfaces: Problems and Techniques", Proceedings of the Conference, 20th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, June, 1982, pp. 46-50.

-----, "TEAM: A Transportable Natural Language Interface System", Proceedings of the Conference on Applied Natural Language Processing, Association for Computational Linguistics, February, 1983, pp.39-45.

-----, Appelt, Douglas E., Martin, Paul A., and Pereira, Fernando C.N., "TEAM: An Experiment in the Design of Transportable Natural Language Interfaces", Artificial Intelligence, Vol. 32, 1987, pp 173-243.

Guida, Giovanni and Tasso, Carlo, "IR-NLI: An Expert Natural Language Interface to Online Databases", Proceedings of the Conference on Applied Natural Language Processing, Association for Computational Linguistics, February, 1983, pp.31-38

Hafner and Godden, "Portability of Syntax and Semantics in Datalog", ACM Transactions on Office Information Systems, Vol. 3 # 2, April, 1985, pp. 141-164.

Hammer, Michael and McLeod, Dennis, "Database Description with SDM: A Semantic Database Model", ACM Transactions on Database Systems, Vol. 6, No. 3, 1981, pp 351-381.

Harris, L.R., "The ROBOT System: Natural Language Processing Applied to Database Query", in ACM 78, "Proceedings of the 1978 Annual Conference (Washington, D.C., Dec. 1978), ACM, New York, pp. 165-172.

-----, "Experience with ROBOT in 12 Commercial Natural Language Data Base Query Applications", International Joint Conference on Artificial Intelligence, 1979, pp. 365-368.

Harris, Mary Dee, Introduction to Natural Language Processing, Reston Publishing Company, Inc., Reston, Virginia, 1985.

Hendrix, Gary G. and Lewis, William H., "Transportable Natural-Language Interfaces to Databases", Proceedings of the Conference, 19th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, July, 1981, pp. 159-164.

Hendrix, Gary G., Sacerdoti, Earl D., Sagalowicz, Daniel, and Slocum, Jonathan, "Developing a Natural Language Interface to Complex Data", ACM Transactions on Database Systems, Vol. 3, No. 2, June, 1978, pp. 105-147.

Hopcroft, J.J., and Jeffery D. Ullman, Formal Languages and their Relation to Automata, Addison-Wesley, Reading, Mass., 1969.



Ishikawa, H., Izumida, Y., Yoshino, T., Hoshiai, T., Makinouchi, A., "A Knowledge-Based Approach to Design a Portable Natural Language Interface to Database Systems", Proceedings of the 1986 International Conference on Data Engineering, IEEE Computer Society, 1986, pp. 134-143.

Jarke, Matthias, Turner, John A., Stohr, Edward A., Vassiliou, Yannis, White, Norman, and Ken Michielsen, "A Field Evaluation of Natural Language for Data Retrieval", IEEE Transactions on Software Engineering, Vol. Se-11, No. 1, January, 1985, pp.97-113.

Kaplan, S. Jerrold, "Designing a Portable Natural Language Database Query System", ACM Transactions on Database Systems, Vol. 9, No. 1, March, 1984, pp. 1-19.

Katz, J.J., and J. A. Fodor, "The Structure of a Semantic Theory", Language, Vol. 39, Number 2, (Part 1), 1964, pp. 170-210.

Kay, M., "The Mind System", in Rustin, R., (ed), Natural Language Processing, New York, Algorithmics Press, 1973, pp. 155-188.

King, Jonathan J., "Modelling Concepts for Reasoning About Access to Knowledge", Proceedings of the Workshop on Data Abstraction and Databases, ACM, 1980, pp. 138-140.

Konolige, K., "A Framework for a Portable Natural Language Interface to Large Databases", SRI International (Technical Note 197), Menlo Park, California, 1979.

Marcus, Mitchel P., A Theory of Syntactic Recognition for Natural Language, The MIT Press, Cambridge, Mass., 1980.

Martin, James, Fourth Generation Languages: Volume I: Principles, Prentice Hall, Englewood Cliffs, New Jersey, 1985.

-----, and Joe Leben, Fourth Generation Languages: Volume II: Representative 4GLs, Prentice Hall, Englewood Cliffs, New Jersey, 1986.

Miller, Harry, "Let HAL Do It", PC World, May, 1987, pp. 206-213.

Minsky, Marvin, "A Framework for Representing Knowledge", in P. Winston (ed.) The Psychology of Computer Vision, McGraw Hill, New York, 1975, pp. 211-277.

Mylopoulos, John, Bernstein, Philip A., Wong, Harry K. T., "A Language Facility for Designing Database-Intensive Applications", ACM Transaction on Database Systems, Vol. 5, No. 2, 1980, pp. 185-207.

Nenova, I., "On an Implementation of the ATNL-Language", in W. Bibel and B. Petkoff (eds.) Artificial Intelligence: Methodology, Systems, Applications, North Holland, Amsterdam, 1984, pp. 185-190.

Newell, Allen, "The Knowledge Level", Artificial Intelligence, Vol 18, pp. 87-127, 1982.

Oppacher, F., "A Programming Approach for Constructing Natural Language Processors, (M. Comp. Sci. Thesis), Concordia University, Montreal, 1981

Orman, Levant, "Nested Set Languages for Functional Databases", Information Systems, Vol 9, No. 3/4, 1984, pp 241-249.

-----, "Design Criteria for Functional Databases", Information Systems, Vol. 10, No. 2, 1985, pp. 201-217.

Pavlov, Radoslav, Angelova, Galia and Paskaleva, Elena, "On Experimental Linguistic Processors for Man-Computer Dialogue in Bulgarian", in W.Bibel and B.Petkof (eds.), Artificial Intelligence: Methodology, Systems, Applications,; North Holland, Amsterdam, 1985, pp. 169-176.

Quillian, M.R., Semantic Memory, in Minsky, Marvin, Semantic Information Processing, MIT Press, Cambridge, Mass., 1975.

Rich, Elaine, Artificial Intelligence, McGraw-Hill, Book Company, New York, 1983.

Ritchie, Graeme and Henry Thompson, "Natural Language Processing", in Tim O'Shea and Marc Eisenstadt, Artificial Intelligence: Tools, Techniques and Applications, Harper and Row, New York, 1984, pp. 358-388.

Sager, Naomi, Natural Language Information Processing: A Computer Grammar of English and its Applications, Addison Wesley Publishing Company, Reading, Mass., 1981.

Scha, Remko J. H., "English Words and Data Bases: How to Bridge the Gap", Proceedings of the Conference, 20th Annual Meeting of the Association for Computational Linguistics, Association for Computational Linguistics, June, 1982, pp. 57-59.

Schank, Roger C., "Conceptual Dependency: A Theory of Natural Language", in Roger Schank and Kenneth Colby (eds.) Computer Models of Thought and Language, W.H. Freeman and Co., San Francisco, 1973.

-----, and Abelson, Robert P., Scripts, Plans, Goals and Understanding: An Inquiry into Human Knowledge Structures, Lawrence Erlbaum Associates, Publishers, Hillsdale, New Jersey, 1977.

-----, and Colby, Kenneth Mark, Computer Models of Thought and Language, W.H. Freeman and Company, San Francisco, 1973.

Shipman, David W., "The Functional Data Model and the Data Language DAPLEX", ACM Transactions on Database Systems, Vol. 6, No. 1, 1981, pp.140-173.

Siklossy, Laurent and Herbert A. Simon, "Some Semantic Methods for Language Processing", in Herbert A. Simon and Laurent Siklossy, (eds.) Representation and Meaning: Experiments with Information Processing Systems, Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1972.

Silva, Georgette, and Montgomery, Christine, "Knowledge Representation for Automated Understanding of Natural Language Discourse", Computers and the Humanities, Vol. 11, 1978, pp. 223-234.

Simmons, Robert. F., "Answering English Questions by Computer: A Survey", Communications of the ACM, 1965, Volume 8, pp. 53-70.

-----, "Semantic Networks: Their Computation and Use for Understanding English Sentences", in Roger Schank and Keith Colby (eds.) Computer Models of Thought and Language, W.H. Freeman and Co., San Francisco, 1973, pp. 63-113.

Sowa, John F., "A Conceptual Schema for Knowledge-Based Systems", Proceedings of the Workshop on Data Abstraction and Databases, ACM, 1980, pp. 193-195.

Stockwell, R.P., Schacter, P.P., and B.H. Partee, The Syntactic Structures of English, Rinehart and Winston, New York, 1973.

Tanimoto, Stephen L., The Elements of Artificial Intelligence: An Introduction Using Lisp, Computer Science Press, Rockville, Maryland, 1987.

Templeton, Marjorie, and Burger, John, "Problems in Natural Language Interface to DBMS with Examples from EUFID", Proceedings of the Conference on Applied Natural Language Processing, Association for Computational Linguistics, February, 1983, pp 3-16.

Tennant, Harry, Natural Language Processing: An Introduction to an Engineering Technology, Petrocelli Books, Inc, Princeton, 1981.

Thompson, Barbara H., "Linguistic Analysis of Natural Language Communication with Computers", in Proceedings of the 8th International Conference of Computational Linguistics, 1980

-----, and Thompson, Frederick B., "Introducing ASK, a Simple Knowledgeable System", Proceedings of the Conference on Applied Natural Language Processing, Association for Computational Linguistics, February, 1983, pp.17-24.

-----, "ASK is Transportable in Half a Dozen Ways", ACM Transactions on Office Information Systems, Vol. 3, No. 2, April, 1985, pp 185-203.

Thompson, Henry, and Graeme Ritchie, "Implementing Natural Language Parsers", in Tim O'Shea and Marc Eisenstadt, Artificial Intelligence: Tools, Techniques and Applications, Harper and Row, New York, 1984, pp. 245-301.

Thompson, Henry S., "Chart Parsing and Rule Schemata in GPSG", Proceedings of the 19th Annual Meeting of the Association for Computational Linguistics, Alexandria, VA., Association for Computational Linguistics, 1981.

Tomita, Masaru, "An Efficient All-Paths Parsing Algorithm for Natural Languages", Technical Report, Department of Computer Science, Carnegie-Mellon University, 1984.

Tufis, Dan and Cristea, Dan, "IURES: A Human Engineering Approach to Natural Language Question-Answering Systems", in W.Bibel and B.Petkof (eds.), Artificial Intelligence: Methodology, Systems, Applications,; North Holland, Amsterdam, 1985, pp. 177-184.

Ullman, Jeffery D., Principles of Database Systems, 2nd edition, Computer Science Press, Rockville, Maryland, 1982, pp. 16-19.

Waltz, David. L., "Natural Language Access to a Large Database", Computers and People, April, 1976, pp. 19-26.

-----, "An English Language Question Answering System for a Large Relational Database", Communications of the ACM, July, 1978, Vol. 21, pp. 526-539.

----- "The State of the Art in Natural Language Understanding", in Lehnert, Wendy G., and Ringle, Martin H., Strategies for Natural Language Processing, Lawrence Erlbaum Associates, Hillsdale, New Jersey, 1982.

Weizenbaum, Joseph, "ELIZA: A Computer Program for the Study of Natural Language Communication Between Man and Machine", Communications of the ACM, 1965, Vol 9, pp. 36-45.

Wilks, Yorick, "An Intelligent Analyzer and Understander of English", Communications of the ACM, Vol. 18, No. 5, May, 1975, pp.264-274.

Winograd, T., Understanding Natural Language, Academic Press, New York, 1972.

Woods, W.A., "Transition Network Grammars for Natural Language Analysis", Communications of the ACM, 1970, Vol. 13, pp. 591-606.



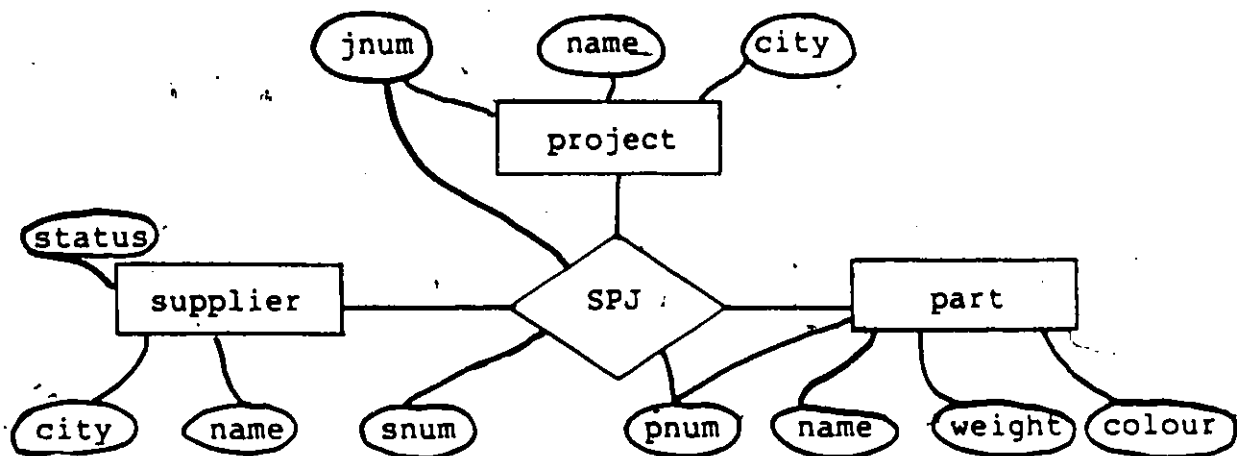
-----, "Context Sensitive Parsing", Communications of the ACM, 1970, Vol. 13, pp. 437-445.

-----, "Transition Network Grammars", in Rustin, R. (ed), Natural Language Processing, Algorithmics Press, New York, 1973, pp. 111-154.

## Appendix A: The Suppliers-Parts-Projects Database

This appendix demonstrates the first of three (3) knowledge bases used in the development and testing of QNL. For each knowledge base, an Entity-Relationship diagram is given, followed by the Lisp description of the entities, attributes and relationships of the domain. The Lisp expressions define a semantic network which is parts of the Domain Knowledge Base (DKB).

A noticeable difference between this and other Knowledge Bases like SDM, KID, etc., is that we are not defining the objects and relationships in terms of strings and integers, but rather in terms of semantic markers, and semantic properties, and how the objects relate to each other. Thus, if the low-level definitions in the database change, this KB is not affected.



E-R Diagram for the Suppliers-Parts-Projects Database

We describe the entities of the domain:

```
(setq sup-objects
  ((dpart (physobj) ; An object called part
    ((has-a pnum) ; has attributes
      (has-a name)
      (has-a colour)
      (has-a weight))
    ((obj dsupply) ; plays roles in relationships
      (obj duse)
      (agent weigh)
      (obj weigh)))

  (dproject (abstractobj) ; an abstract object : project
    ((has-a jnum) ; has attributes
      (has-a name)
      (has-a city))
    ((rcpt dsupply) ; relationships
      (agent duse)))

  (dsupplier (animate) ; supplier is animate
    ((has-a snum) ; has attributes
      (has-a name)
      (has-a status)
      (has-a city))
    ((agent dsupply))) ; one relationship
))
```

Next, the relationships of the domain.

```
(setq sup-relns
  '((dsupply (dsupplier agent) ; Each entity and its
            (dpart object) ; role is specified.
            (dproject rcpt)
            (dqty attribte))
    (weigh (dpart agent)
           (dpart object))
    (duse (dproject agent)
          (dpart object)
          (dqty attribte))))
```

Next, we need a way of identifying an object from its attributes. Each attribute name may indicate more than one entity in the domain.

```
(setq sup-attributes '(
  (status-value (has-a number)
                (attrib-of dsupplier))
  (pnum (attrib-of dpart))
  (name (attrib-of dpart)
        (attrib-of dproject)
        (attrib-of dsupplier))
  (colour (has-a colour)
          (attrib-of dpart))
  (weight (has-a number)
          (attrib-of dpart))
  (snum (attrib-of dsupplier))
  (location (has-a location)
            (attrib-of dsupplier)
            (attrib-of dproject))
  (jnum (attrib-of dproject))
))
```

The list of special terms may include specific values or ranges of values: in this case supplier-numbers, part numbers and project numbers.

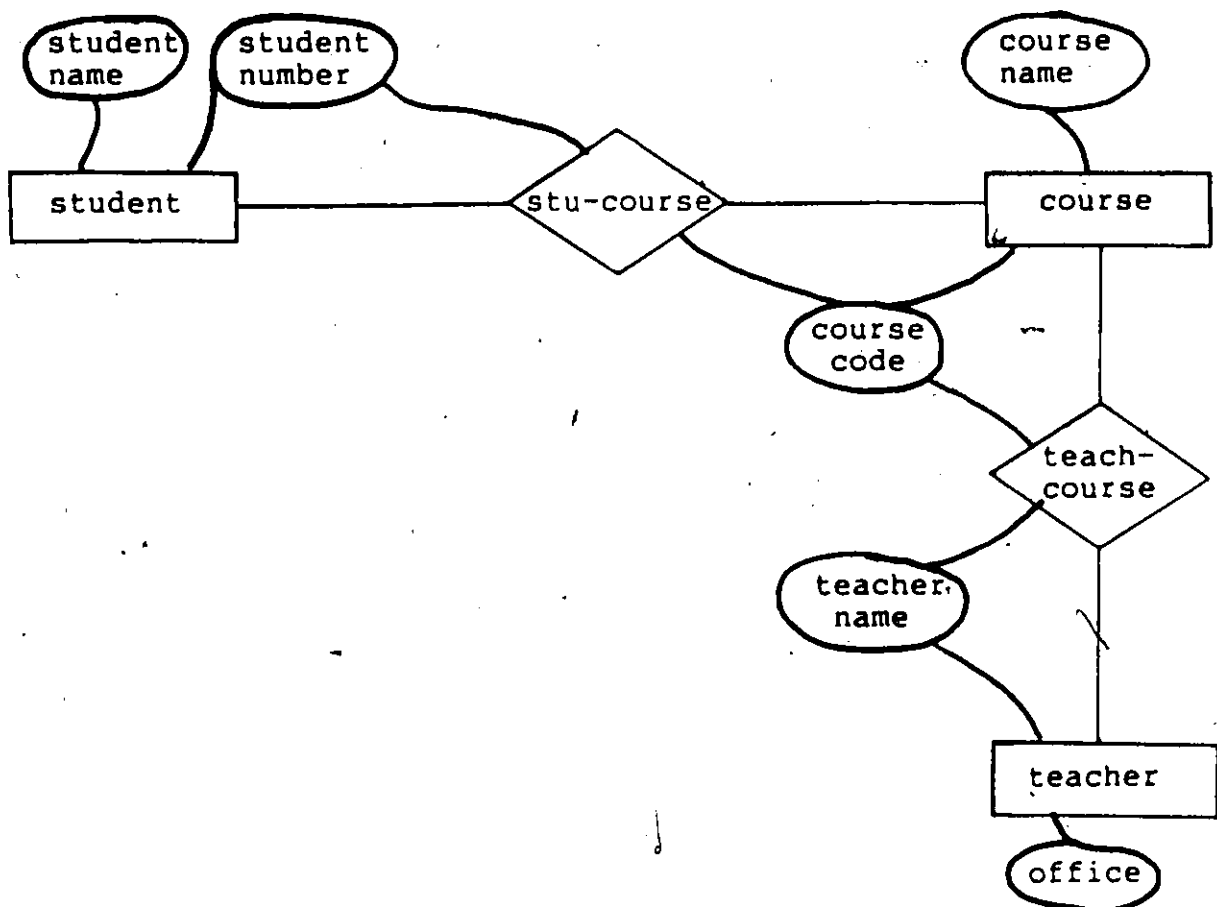
```
(setq sup-special-terms '((sl snum )
                           (pl pnum )
                           (jl jnum ))))
```

Finally, we need a direct link between the DKB and the WKB. This is accomplished by the following map. This connects the lexical descriptions of words to concepts in the DKB. It is possible for a description in the WKB to map to several concepts in the DKB. It is also possible for several definitions from the WKB to describe the same concept in the DKB, not because they are identical, but rather because they are semantically close. The determination of this mapping should be automated at some time in the future.

```
(setq sup-map '((part dpart)
                 (supplier dsupplier)
                 (project dproject)
                 (use duse)
                 (weigh weigh)
                 (buy dsupply)
                 (sell dsupply)
                 (supply dsupply)
                 ))
```

## Appendix B: The University domain

The University Database is structurally different from the Suppliers database in that there are three entities joined by two relationships, rather than one. There is plenty of possibility of ambiguity since all three entities have a name, and in fact, since teachers and students are both human, there is significant semantic similarity between these entities.



E-R Diagram of the University Domain

First, the objects of the domain:

```
(setq univ-objects '(
  (student (human)
    ((has-a snumber)      ; attributes
     (has-a name)
     (has-a major))
    ((agent stu-course))) ; relationship

  (course (abstractobj)
    ((has-a coursecode)   ; attributes
     (has-a coursename)
     (has-a dept))
    ((object stu-course)  ; two relationships
     (object teach-course)))

  (teacher (human)
    ((has-a name)          ; attributes
     (has-a office))
    ((agent teach-course)) ; one relationship
```

The relationships:

```
(setq univ-relns '(
  (teach-course (teacher agent) ; Each entity has its
    (course object)) ; role specified.
  (stu-course (course agent)
    (student object))))
```

Next, we need a way of identifying an object from its attributes. Each attribute name may indicate more than one entity in the domain.

```
(setq univ-attributes '(
  (name (attrib-of student)
        (attrib-of teacher))
  (office (attrib-of teacher))
  (coursecode (attrib-of course))
  (coursename (attrib-of course))
  (dept (attrib-of course))
  (snumber (attrib-of student))
  (major (attrib-of course)))
```

These special terms will include the range of valid student numbers, valid office numbers, valid course codes, and valid course names.

```
(setq univ-special-terms '(
  (csc113 coursecode)
  (csc114 coursecode)
  (8700 snumber)
  (fortran coursename)
  (pascal coursename)
  (code coursecode)
  (codes coursecode)
  (compsci dept)
  (bus dept)
  (business dept major)
  (csc major)
  (eng major)
  (major major)
  (j117 office)
  (h906 office)))
```

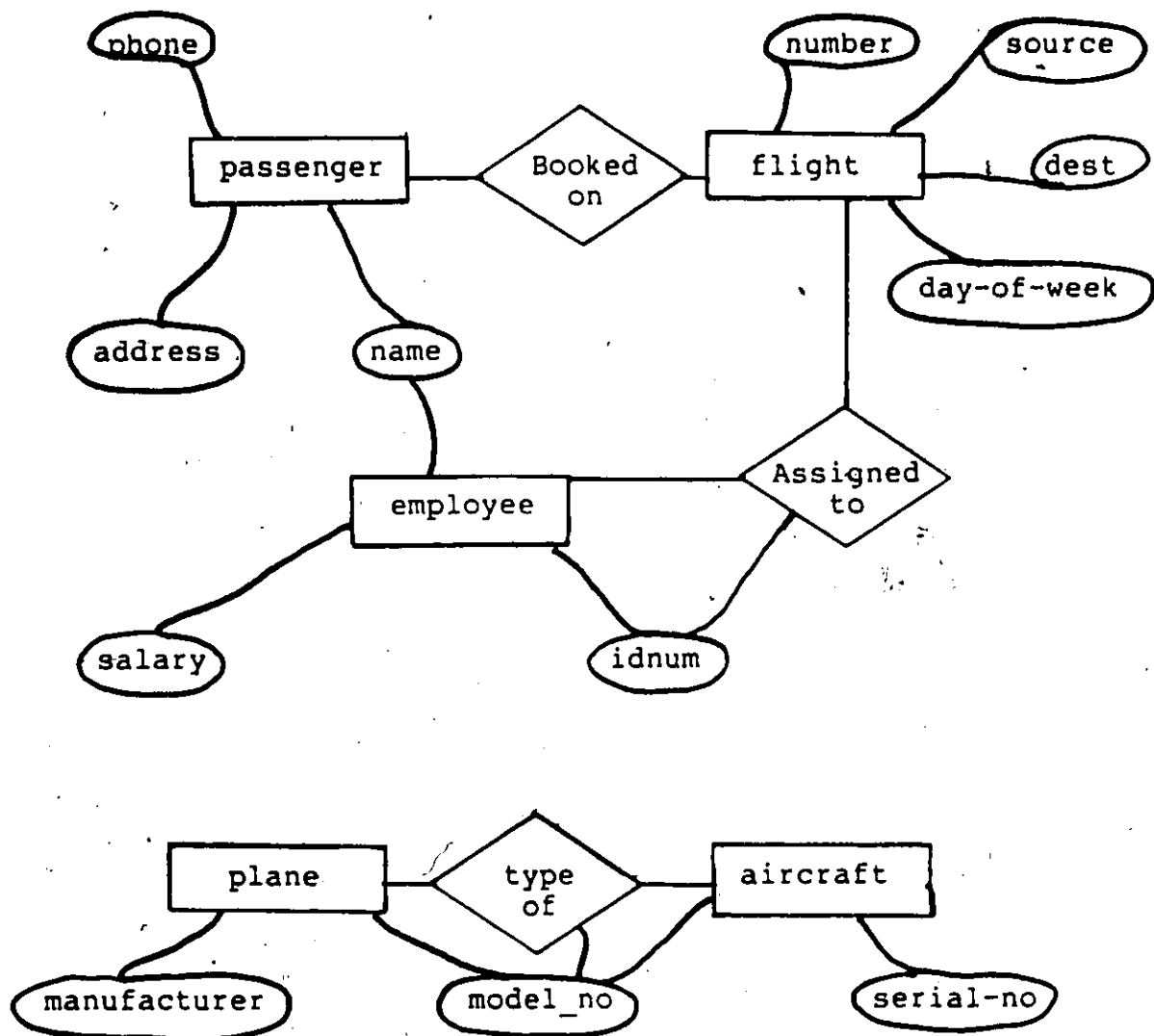


Finally, we must link the DKB to the WKB.

```
(setq univ-map '(
  (student student)
  (course course)
  (teacher teacher)
  (teach teach-course)
  (major student)
  (department course)
  (register stu-course)
  (office teacher)
  (classlist stu-course)
  (take stu-course))))))
```

## Appendix C. The Airline domain

This domain is more complex than either of the previous ones, in that there are more relationships, and many more ways of referring to these relationships.



E-R Diagram of the Airline Domain

First, the entities of the airline domain:

```
(setq air-objects '(
  (passenger (human)
    ((has-a name)           ; attributes
     (has-a address)
     (has-a phone-no))
    ((humanagent booked-on))) ; relationship

  (aircraft (physobj)
    ((has-a serial-no)      ; attributes
     (has-a model-no))
    ((object type-of)))     ; relationship

  (flight (ptrans)
    ((has-a number)         ; attributes
     (has-a source)
     (has-a dest)
     (has-a day-of-week))
    ((agent departure)      ; relationships
     (obj booked-on)
     (obj assigned-to)))

  (employee (human)
    ((has-a name)           ; attributes
     (has-a address)
     (has-a salary)
     (has-a idnum))
    ((humanagent assigned-to))) ; relationship

  (plane (physobj)
    ((has-a manufacturer)   ; attributes
     (has-a model-no))
    ((agent make)           ; relationships
     (agent type-of))))))
```

The relationships of the Airline domain:

```
(setq air-relns '(
  (booked-on (passenger humanagent) ; Each entity has
              (flight obj))         ; its role specified
  (departure (flight agent))
  (type-of (plane agent)
            (aircraft obj))
  (make (plane agent)
         (plane obj))
  (assigned-to (employee agent)
               (flight obj))
  (plane (plane agent)))
```

The attributes of the entities. Some attributes may link with more than one entity.

```
(setq air-attributes
  ((idnum (has-a number)
           (attrib-of employee))
   (address (attrib-of employee)
             (attrib-of passenger))
   (phone-no (attrib-of passenger))
   (name (attrib-of passenger)
          (attrib-of employee))
   (source (has-a location)
            (attrib-of flight))
   (salary (has-a number)
            (attrib-of employee))
   (dest (has-a location)
          (attrib-of flight))
   (day-of-week (attrib-of flight))
   (number (attrib-of flight)
            (is-a phone-no)
            (is-a idnum))
   (manufacturer (attrib-of plane))
   (model-no (has-a number)
              (attrib-of aircraft)
              (attrib-of plane))
   (flight-no (has-a number)
               (attrib-of flight)))
```

The special terms, which could include the range of valid flight numbers, employee numbers aircraft serial numbers, and will include specific values like airplane model numbers.

```
(setq air-special-terms '((B727 model-no )
                           (727 model-no )
                           (B747 model-no )
                           (747 model-no )
                           (e56 idnum)
                           (Boeing manufacturer )
                           (a100 serial-no ))))
```

Finally, the map which links the DKB with the WKB. Notice that some concepts from the WKB map to more than one concept in the DKB. Also notice that some verbs map to entities rather than relationships.

```
(setq air-map '(
  (booked booked-on)
  (booked assigned-to)
  (depart flight)
  (leave flight)
  (make make)
  (phone passenger)
  (fly booked-on)
  (passenger passenger )
  (employee employee)
  (manufacturer plane)
  (plane plane aircraft)
  (type type-of)
  (work assigned-to)
  (aircraft aircraft plane)
  (flight flight)
  (arrive flight)))
```

#### Appendix D: Sample Queries Handled by QNL

The first part of this appendix demonstrates queries applied to the Suppliers-Parts-Projects database of Appendix A, followed by questions asked of the University database of Appendix B, and of the airline database of appendix C. Each English question is followed by the expression as understood by QNL. This expression is passed to the query generator to be transformed into a relational calculus or relational algebra expression, and then into the DML of the particular database. All expressions are in fully parenthesized form.

First, simple queries about objects and their attributes:

```
(get full details of all suppliers)
```

```
(for all)  
((var-166 dsupplier (animate)))
```

```
(get details of suppliers in london)
```

```
(for all)  
((var-168 dsupplier (animate)) ((location (london))))
```

These next simple queries demonstrate that "noise words" can often be ignored. Noise words include personal pronouns, some adverbs, and so on. Also, some of the different acceptable grammatical forms are demonstrated.

```
(i need to know about the projects in london)

(for all)
((var-169 dproject (abstractobj)) ((location (london))))
```

In this next question, "who" is treated as a relative pronoun. It is later discarded along with the leading noise words, and the relative clause is treated as the complete sentence. However, this ends with a verb without an object. QNL tries to generate a meaning with whatever it is given.

```
(i need to know who the london suppliers are)

(for all)
((var-170 dsupplier (animate)) ((location (london))))
```

```
(what do you know about the suppliers in london)

(for all)
((var-172 dsupplier (animate)) ((location (london))))
```

```
('full details of all projects are needed)

(for all)
((var-174 dproject (abstractobj)))
```

(can you tell me about the paris jobs)

```
(for all)
((var-171 dproject (abstractobj)) ((location (paris)))))
```

(is there a project in toronto)

```
(exists)
((var-175 dproject (abstractobj)) ((location (toronto)))))
```

(can you tell me the weight of each red part)

```
(for all)
((var-178 dpart (physobj)) ((colour (red) (weight (var-179)))))
```

One of the problems we can expect to encounter is that some attributes may be perceived as relationships, and also, as attributes. The DKB will specifically recognize it both ways, and the database mapping will translate the relationship or attribute appropriately. By this means, if the actual files of the database change, only the mapping needs to be altered, not the entire DKB.

(how much does p1 weigh)

```
(for all)
((var-188 dpart (physobj)) ((pnum (p1))))
((weigh) ((var-188 dpart (physobj)) ((pnum (p1)))))
```

.. (print the names of the london suppliers)

```
(for all)
((var-181 dsupplier (animate))
  ((location '(london)) (name (var-182)))))
```



(tell me about j1)

(for all)  
((var-186 dproject (abstractobj)) ((jnum (j1))))

(what colour is p1)

(for all)  
((var-190 dpart (physobj)) ((pnum (p1) (colour (var-191))))))

This next question ends with a preposition.

(what colour does p1 come in)

(for all)  
((var-184 dpart (physobj)) ((pnum (p1) (colour (var-185))))))

In this next question, the adjective, "red", follows the noun. This is unusual because normally the noun ends a noun phrase. In this case, "red" is treated as an incomplete noun phrase, and is attached to "p1" in the general semantic analyzer.

(is p1 red)

(exists)  
((var-187 dpart (physobj)) ((colour (red) (pnum (p1))))))

These next questions ask about relationships between objects.

(list all suppliers who sell red parts)

```
(for all)
((var-193 dpart (physobj)) ((colour (red))))
((var-192 dsupplier (animate)))
((dsupply) ((var-193 dpart (physobj)) ((colour (red))))
            ((var-192 dsupplier (animate))))
```

(which suppliers in london sell red parts to a project in paris)

```
(for all)
((var-196 dproject (abstractobj)) ((location (paris))))
((var-195 dpart (physobj)) ((colour (red))))
((var-194 dsupplier (animate)) ((location (london))))
((dsupply) ((var-196 dproject (abstractobj))
              ((location (paris))))
            ((var-195 dpart (physobj))
              ((colour (red))))
            ((var-194 dsupplier (animate))
              ((location (london))))))
```

The next questions require the domain-specific analyzer to resolve the interrogative pronoun "who". It recognizes the relationship "sell", investigates the appropriate case frame, and generates a variable for the missing agent of the case frame.

(who sells red parts)

```
(for all)
((var-197 dpart (physobj)) ((colour (red))))
((var-198 dsupplier (animate)))
((dsupply) ((var-197 dpart (physobj)) ((colour (red))))
            ((var-198 dsupplier (animate))))
```

(who sells parts to projects)

```
(for all)
((var-213 dproject (abstractobj)))
((var-212 dpart (physobj)))
((var-214 dsupplier (animate)))
((dsupply) ((var-213 dproject (abstractobj)))
            ((var-212 dpart (physobj)))
            ((var-214 dsupplier (animate)))))
```

(who sells red parts to projects in london)

```
(for all)
((var-216 dproject (abstractobj)) ((location (london))))
((var-215 dpart (physobj)) ((colour (red))))
((var-217 dsupplier (animate)))
((dsupply) ((var-216 dproject (abstractobj))
            ((location (london)))
            ((var-215 dpart (physobj))
             ((colour (red))))
            ((var-217 dsupplier (animate)))))
```

This next question contains a relative clause.

(are there any suppliers in toronto who sell red parts  
to a project in paris)

```
(exists)
((var-201 dproject (abstractobj)) ((location (paris))))
((var-200 dpart (physobj)) ((colour (red))))
((var-199 dsupplier (animate)) ((location (toronto))))
((dsupply) ((var-201 dproject (abstractobj))
            ((location (paris)))
            ((var-200 dpart (physobj))
             ((colour (red))))
            ((var-199 dsupplier (animate))
             ((location (toronto)))))
```

The next question involves an "implied relationship". The user has not explicitly stated the relationship between parts and suppliers, so the relationship must be inferred by

the domain-specific analyzer. This is done by identifying the two entities, and then searching the semantic net for a relationship between them.

```
(get suppliers of red parts)

(for all)
  ((var-203 dpart (physobj)) ((colour (red))))
  ((var-202 dsupplier (animate)))
  ((dsupply) ((var-203 dpart (physobj)) ((colour (red))))
              ((var-202 dsupplier (animate))))
```

Next, three entities are involved in an implied relationship.

```
(get names of suppliers of red parts to projects in
paris)

(for all)
  ((var-207 dproject (abstractobj)) ((location (paris))))
  ((var-206 dpart (physobj)) ((colour (red))))
  ((var-205 dsupplier (animate)) ((name (var-208))))
  ((dsupply) ((var-207 dproject (abstractobj))
              ((location (paris))))
              ((var-206 dpart (physobj))
              ((colour (red))))
              ((var-205 dsupplier (animate))
              ((name (var-208))))))
```

(which projects get red parts from a supplier in london)

```
(for all)
  ((var-210 dpart (physobj)) ((colour (red))))
  ((var-211 dsupplier (animate)) ((location (london))))
  ((var-209 dproject (abstractobj))
  ((dsupply) ((var-210 dpart (physobj))
              ((colour (red))))
              ((var-211 dsupplier (animate))
              ((location (london))))
              ((var-209 dproject (abstractobj))
```

This next question asks for the suppliers who sell parts which are both red and blue. It involves a conjunction and an implied relationship. QNL handles this by generating two similar relationships, but note the attributes of dpart.

```
(who sells red and blue parts)

(for all)
((and) ((var-218 dpart (physobj)) ((colour (blue))))
        ((var-218 dpart (physobj)) ((colour (red))))))
((var-219 dsupplier (animate)))
((and)
  ((dsupply) ((var-218 dpart (physobj)) ((colour (red))))
              ((var-219 dsupplier (animate))))
  ((dsupply) ((var-218 dpart (physobj)) ((colour (blue))))
              ((var-219 dsupplier (animate))))))
```

This last question demonstrates a subtle problem of semantic overshoot. The system will search for parts which can be both red and blue, but within this domain, it is not clear whether a part can be multi-coloured. Although the query can be posed, and will result in a null set, there must be a means of alerting the user that the results may not reflect the user's intent. This could be avoided by paraphrasing the query, and repeating it back to the user.

The next question asks for suppliers who supply both red parts and blue parts. This involves an implied relationship, and a conjunction. To solve the conjunction, a variable must be generated to handle the adjective "red", and QNL realizes this is different from the blue part.

(who sells both red and blue parts)

```
(for all)
((and) ((var-220 dpart (physobj)) ((colour (blue))))
        ((var-221 dpart (physobj)) ((colour (red))))))
((var-222 dsupplier (animate)))
((and)
  ((dsupply) ((var-221 dpart (physobj)) ((colour (red))))
              ((var-222 dsupplier (animate))))
  ((dsupply) ((var-220 dpart (physobj)) ((colour (blue))))
              ((var-222 dsupplier (animate))))))
```

Next we have an implied relationship and a conjunction. Each conjunct is an object which is explicitly stated by the user. QNL processes this by generating two distinct queries and conjoining them. Note that two different parts are specified in this query.

(who sells blue parts and red parts)

```
(for all)
((and) ((var-224 dpart (physobj)) ((colour (red))))
        ((var-223 dpart (physobj)) ((colour (blue))))))
((var-225 dsupplier (animate)))
((and)
  ((dsupply) ((var-223 dpart (physobj)) ((colour (blue))))
              ((var-225 dsupplier (animate))))
  ((dsupply) ((var-224 dpart (physobj)) ((colour (red))))
              ((var-225 dsupplier (animate))))))
```

These next sentences are disjunctions. In the second sentence one of the disjuncts is implied by its attribute.

(who sells blue parts or red parts)

```
(for all)
((or) ((var-227 dpart (physobj)) ((colour (red))))
      ((var-226 dpart (physobj)) ((colour (blue)))))
((var-228 dsupplier (animate)))
((or)
  ((dsupply) ((var-226 dpart (physobj)) ((colour (blue)))))
            ((var-228 dsupplier (animate))))
  ((dsupply) ((var-227 dpart (physobj)) ((colour (red)))*)
            ((var-228 dsupplier (animate)))))
```

(who sells blue or red parts)

```
(for all)
((or) ((var-229 dpart (physobj)) ((colour (red))))
      ((var-229 dpart (physobj)) ((colour (blue)))))
((var-230 dsupplier (animate)))
((or)
  ((dsupply) ((var-229 dpart (physobj)) ((colour (blue)))))
            ((var-230 dsupplier (animate))))
  ((dsupply) ((var-229 dpart (physobj)) ((colour (red))))
            ((var-230 dsupplier (animate)))))
```

This next disjunction refers to a single part which is either red or blue. This also involves an interrogative pronoun in place of the agent of the verb, and a relative clause.

(who sells parts which are red or blue)

```
(for all)
((or) ((var-231 dpart (physobj)) ((colour (red))))
      ((var-231 dpart (physobj)) ((colour (blue)))))
((var-232 dsupplier (animate)))
((or)
  ((dsupply) ((var-231 dpart (physobj)) ((colour (blue)))))
            ((var-232 dsupplier (animate))))
  ((dsupply) ((var-231 dpart (physobj)) ((colour (red))))
            ((var-232 dsupplier (animate)))))
```

The next questions demonstrate how negations are handled. In the first question, an object must not have a certain attribute-value.

```
(get suppliers not in london)
(for all)
((var-233 dsupplier (animate)) ((not (location (london)))))
```

In the next question, it is the relationship which is negated.

```
(which projects do not use red parts)
(for all)
((var-238 dpart (physobj)) ((colour (red))))
((var-237 dproject (abstractobj))
((not duse) ((var-238 dpart (physobj)) ((colour (red))))
              ((var-237 dproject (abstractobj)))))
```

This next one includes a relative clause and a negation. Actually, two relationships are involved, one of which is negated.

```
(which projects which use red parts do not use blue parts)
(for all)
((var-235 dpart (physobj)) ((colour (red))))
((var-236 dpart (physobj)) ((colour (blue))))
((var-234 dproject (abstractobj))
((and) ((not duse) ((var-236 dpart (physobj)) ((colour (blue))))
          ((var-234 dproject (abstractobj)))))
        ((duse) ((var-235 dpart (physobj)) ((colour (red))))
              ((var-234 dproject (abstractobj)))))
```



```

(get suppliers not in london who sell red parts)

(for all)
((var-240 dpart (physobj)) ((colour (red))))
((var-239 dsupplier (animate)) ((not (location (london)))))
((dsupply) ((var-240 dpart (physobj))
              ((colour (red))))
              ((var-239 dsupplier (animate))
              ((not (location (london))))))

```

This next section deals with similar types of queries on the University database. In the first question, no variable for offices is generated because we may want more information to be returned than the user has specified, depending on the definition of the user view.

```

(get full details of all courses)

(for all)
((var-54 course (abstractobj)))

```

In this domain, "office" is an attribute of "professor"

```

(which offices have been assigned to which professors)

(for all)
((var-595 teacher (human)))

```

```

(which professors are in which offices)

(for all)
((var-597 teacher (human)))

```

In these next questions, the domain-specific terms, "compsci", "csc", and "eng" are used. In order for it to be recognized, the terms have been added to the DKB where they may be accessed by the procedures which access the lexicon.

```
(list all students who major in eng)
(for all)
((var-599 student (human)) ((major (eng)))))
```

```
(what are the courses offered by compsci)
(for all)
((var-600 course (abstractobj)) ((dept (compsci)))))
```

```
(are there any students majoring in csc)
(exists)
((var-602 student (human)) ((major (csc)))))
```

```
(are there any eng majors)
(exists)
((var-603 student (human)) ((major (eng)))))
```

The number 8700 is recognized as a value in the range of valid student numbers. The noun-noun modification rule concerning numbers states that the number should be attached as a modifier to the immediately preceding noun. eg. Flight 257, Room 61, etc.

(get full details of student 8700)

(for all)  
((var-605 student (human)) ((snumber (8700)))))

In this next query, 8700 is recognized as being an attribute-value of a student.

(get name of 8700)

(for all)  
((var-622 student (human)) ((snumber (8700)))))

There is a relationship between students and courses, but the user has not specified it in this next question. QNL recognizes the entities, then identifies the relationship(s) between them.

(get all students in csc113)

(for all)  
((var-607 course (abstractobj)) ((coursecode (csc113))))  
((var-606 student (human)))  
((stu-course) ((var-607 course (abstractobj))  
((coursecode (csc113))))  
((var-606 student (human)))))

This next question involves a pronoun reference to the previous query.

(who is teaching it)

```
(for all)
((var-609 course (abstractobj)) ((coursecode (csc113))))
((var-610 teacher (human)))
((teach-course) ((var-609 course (abstractobj))
                  ((coursecode (csc113))))
                 ((var-610 teacher (human)))))
```

(is there a course in fortran)

```
(exists)
((var-64 course (abstractobj)) ((coursename (fortran)))))
```

This next question is an example of how QNL handles anaphora. Since no verb is used, the verb(s) from the previous question are used. QNL also merges the new entities with the entities of the previous question.

(what about pascal)

```
(for all)
((var-64 course (abstractobj)) ((coursename (pascal)))))
```

This next two questions involve a pronoun reference to the previous question.

{

(who teaches it) ,

```
(for all)
((var-65 course (abstractobj)) ((coursename (pascal)))))
((var-66 teacher (human)))
((teach-course) ((var-65 course (abstractobj))
                  ((coursename (pascal)))))
                 ((var-66 teacher (human)))))
```

(what students are taking it)

```
(for all)
((var-68 course (abstractobj)) ((coursename (pascal))))
((var-67 student (human)))
((stu-course) ((var-68 course (abstractobj))
                ((coursename (pascal))))
               ((var-67 student (human)))))
```

Next, an implied relationship.

(get me the students in csc113)

```
(for all)
((var-612 course (abstractobj)) ((coursecode (csc113))))
((var-611 student (human)))
((stu-course) ((var-612 course (abstractobj))
                ((coursecode (csc113))))
               ((var-611 student (human)))))
```

In the university domain, the verb "taking" will point to a very specific case frame involving humans and courses.

(which students are taking a business course)

```
(for all)
((var-615 course (abstractobj)) ((dept (business))))
((var-614 student (human)))
((stu-course) ((var-615 course (abstractobj))
                ((dept (business))))
               ((var-614 student (human)))))
```

In this next question, a relationship is specified, and only one of the objects, and that object is identified by an attribute-value.

(what does smith teach)

```
(for all)
((var-616 teacher (human)) ((name (smith))))
((teach-course) ((var-616 teacher (human)) ((name (smith)))))
```

(does wilson teach csc113)

```
(exists)
((var-618 course (abstractobj)) ((coursecode (csc113))))
((var-617 teacher (human)) ((name (wilson))))
((teach-course) ((var-618 course (abstractobj))
                  ((coursecode (csc113))))
                  ((var-617 teacher (human))
                  ((name (wilson)))))
```

Wilson is a human name, students and teachers are human, but only students have majors in this domain. So in this next question, wilson is recognized as a student.

(what major is wilson)

```
(for all)
((value student (human)) ((name (wilson))))
```

In this next question, smith is recognized as a person's name, possibly a student or professor. J117 is recognized as an office number. Professors have an office attribute, but students do not.

(is smith in j117)

```
(exists)
((var-619 teacher (human)) ((office (j117))))
```

In the next question, H960 is recognized as an office. The interrogative pronoun "who" expects a human, so a variable is generated to refer to a professor. This requires domain-specific knowledge.

```
(who is in h906)
(for all)
((var-626 teacher (human)) ((office (h906)))))
```

In this next question, name is assumed to refer to the name of a course, since csc113 is recognized as a course identifier.

```
(get name of csc113)
(for all)
((var-625 course (abstractobj)) ((coursecode (csc113)))))
```

In this next query, there is a specified relationship, "taking", but only one of the participants is mentioned, and that one only by its attribute. 8700 is a student number, and the relationship "taking" is recognized as linking students and courses. QNL accesses that relation, but does not specify what attributes are to be returned

```
(what is 8700 taking)
(for all)
((var-627 student (human)) ((snumber (8700)))))
((stu-course) ((var-627 student (human)) ((snumber (8700)))))
```

An object, student, is recognized by the attribute-value 8700, and the relationship "taking" is recognized. From this information, we can generate the other object in the relationship, "course".

(can you please tell me what 8700 is taking)

```
(for all)
((var-632 student (human)) ((snumber (8700))))
((stu-course) ((var-632 student (human)) ((snumber (8700)))))
```

(what courses does compsi offer)

```
(for all)
((var-634 course (abstractobj)) ((dept (compsci))))
```

(what compsci courses does smith not teach)

```
(for all)
((var-636 teacher (human)) ((name (smith))))
((var-635 course (abstractobj)) ((dept (compsci))))
((not teach-course)
  ((var-636 teacher (human)) ((name (smith))))
  ((var-635 course (abstractobj)) ((dept (compsci)))))
```

This next question is an anaphoric reference to the previous one.

(wilson)

```
(for all)
((var-637 teacher (human)) ((name (wilson))))
((var-635 course (abstractobj)) ((dept (compsci))))
((not teach-course)
  ((var-637 teacher (human)) ((name (wilson))))
  ((var-635 course (abstractobj)) ((dept (compsci)))))
```



```

(which students are taking a compsci or business course)

(for all)
((or) ((var-639 course (abstractobj)) ((dept (business))))
      ((var-640 course (abstractobj)) ((dept (compsci)))))
((var-638 student (human)))
((or)
  ((stu-course)
    ((var-640 course (abstractobj)) ((dept (compsci))))
    ((var-638 student (human))))
  ((stu-course)
    ((var-639 course (abstractobj)) ((dept (business))))
    ((var-638 student (human)))))

```

These next questions apply to the airline domain. First, questions about entities and their attributes.

```

(get me details of flight 57 on monday)

(for all)
((var-57 flight (ptrans)) ((day-of-week (monday)
                                         (number (57)))))

```

```

(are there any flights to paris on friday)

(exists)
((var-65 flight (ptrans)) ((day-of-week (friday)
                                         (dest (paris)))))

```

The relationships in this domain are more complex than in the previous domain, and there are many more ways of users perceiving them. The relationship "booked-on" links passengers to flights. Both of these questions include noun-noun modifications. The basic rule is that if two nouns occur together, and the second noun is a number, the number is an attribute of the first noun, else the first noun is an attribute of the second.

```

(list passengers on flight 106 on tuesday)

(for all)
  ((var-59 flight (ptrans)) ((number (106))))
  ((var-60 flight (ptrans)) ((day-of-week (tuesday))))
  ((var-58 passenger (human)))
  ((booked-on) ((var-59 flight (ptrans)) ((number (106))))
                ((var-58 passenger (human))))

```

```

(list passengers on london flights)

(for all)
  ((var-63 flight (ptrans)) ((location (london))))
  ((var-62 passenger (human)))
  ((booked-on) ((var-63 flight (ptrans)) ((location (london))))
                ((var-62 passenger (human))))

```

This next sequence shows simple questions between entities, where each entity may or may not be precisely specified, or the relationship may be implied.

```

(who is working on monday)

(find all)
  ((var-72 flight (ptrans)) ((day-of-week (monday))))
  ((var-73 employee (human)))
  ((assigned-to) ((var-72 flight (ptrans))
                  ((day-of-week (monday))))
                 ((var-73 employee (human))))

```

```

(what kind of plane is a100)

(for all)
  ((var-76 aircraft (physobj)) (serial-no (a100))))
  ((var-75 plane (physobj)))
  ((type-of) ((var-76 aircraft (physobj)) (serial-no (a100))))
              ((var-75 plane (physobj))))

```

(what plane is a100)

```
(for all)
((var-79 aircraft (physobj)) ((serial-no (a100))))
((var-78 plane (physobj)))
((type-of) ((var-79 aircraft (physobj)) ((serial-no (a100))))
            ((var-78 plane (physobj))))
```

(who makes the 747)

```
(for all) +
((var-81 plane (physobj)) ((model-no (747))))
((make) ((var-81 plane (physobj)) ((model-no (747)))))
```

This next sequence demonstrates the interaction between anaphora and pronoun references.

(can i get a flight to toronto on tuesday)

```
(for all)
((var-67 flight (ptrans)) ((day-of-week (tuesday)
                                           (dest (toronto)))))
```

(wednesday)

```
(for all)
((var-67 flight (ptrans)) ((day-of-week (wednesday)
                                           (dest (toronto)))))
```

(to montreal)

```
(for all)
((var-67 flight (ptrans)) ((day-of-week (wednesday)
                                           (dest (montreal)))))
```

