ISBN 0-315-59172-2

Multilevel Interface to Database Management System: MIDBMS

Li Zhang

A Thesis

in

The Department

of

Computer Science

Presented in Partial Fulfilment of the Requirements
for the Degree of Master of Computer Science at
Concordia University
Montreal, Quebec, Canada

April 1990

ABSTRACT

MULTILEVEL INTERFACE TO DATABASE MANAGEMENT SYSTEM: MIDBMS

Li Zhang

This thesis presents a model of a multilevel interface to a database management system. In this interface, database users are able to pose their database queries with different languages, such as a natural language or formal query languages, based on their understanding of the database query system.

Under this model, a portable prototype system named Multilevel Interface to Data Base Management System (MIDBMS) is designed and implemented for a heterogeneous distributed database management system (HDDBMS). The MIDBMS includes a natural language interpreter, which interprets English queries and produces SQL queries. The MIDBMS also includes a formal query language translator, which then translates SQL queries into Global Query and Mapping Language (GQML) queries used in the HDDBMS.

The issues considered in building such a portable multilevel interface to a database management system, in interpreting natural language, as well as in representing general knowledge are discussed in this thesis.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere thanks to my thesis supervisor Dr. B. C. Desai for providing guidance, advice, and financial support throughout this research. As well, I would like to thank him for his invaluable suggestions in the preparation of this thesis.

I would also like to thank my colleagues and friends who have unselfishly offered their comments on the thesis itself, and I would especially like to thank Russel Krause who joined me in numerous discussions on the issues of natural language understanding and prototype system design, proof-read the whole thesis, and who offered many helpful suggestions and corrections.

Finally, I would like to thank my parents for their patient encouragement, my wife, Dong, for her great understanding, my daughter, Tian, who I have not even seen since she was born. You cannot know how much you have meant to me. Without your love and moral support, I would never have finished this work.

Contents

List of Figures

Chapter 1

INTRODUCTION

1.1 CHARACTERIZATION OF MIDBMS

A natural language interface (NLI) to a database query system provides its users with the capability of obtaining information stored in a database by querying the database system with a natural language, such as English or French. This ability allows users to construct a query with the language which they are mostly familiar with and frees them from knowing how an artificial query language is used and how information is stored and processed in a database system. Many such NLI systems have been developed in recent years toward this direction. These include The PRE System[19], Team[22], Datalog[23], KID[26], and System-X[33]. Most of the work in these systems has concentrated on the transportability, extensibility, and the knowledge representation of natural language.

In these systems, natural language is the only language that the NLI system intends to provide to the users to communicate with the database systems. However, natural language is, at times, syntactically and/or semanticly ambiguous[28], and these ambiguities cannot always be resolved completely by an natural language interface[17]. Thus, natural language cannot always express database queries as precisely as desired. In these circumstances, the users may wish to turn to a formal query language. Thus, it is desirable to build a database query environment in which multiple database query interfaces can be established. At each such interface, one type of query language would be supported. Thus, the database users could choose one of the query languages which is the most suitable for their requirements and their knowledge about the database system. Based on these concerns, a proposal of "Multilevel Interface to Database Management Systems" (MIDBMS) is offered in this thesis.

Basically, the multilevel interface system consists of three hierarchical interfaces. They are:

1. Natural language interface;
2. General formal query language interface;
3. Target formal query language interface.

These interfaces are at different levels in terms of the database knowledge which users need to have and also in terms of the power of the expression which each query language has.

The natural language interface uses a natural language. This natural language could be English, French, or any other natural languages. The database users can pose their database queries with one of these natural languages. These users need not know anything specific about the database query system except some conceptual organizations about the objects they want to query. This interface is very helpful for those naive users who do not have much knowledge about database system or those who just want to make some casual queries.

At the general query language interface, a general formal language for database query is offered. This formal query language should be independent of any specific database management system so that even though the underlying database management system is replaced, this interface is still able to be functional. It also should be general enough so that most of the database query operations can be included. In addition, it should be widely accepted as a relatively standard database query language so that the database users may have a better chance to get familiar with it. Users working at this interface need to have some knowledge about the database query systems. Thus, they can pose some more precise queries which cannot be made in the natural language interface.

At the target formal query language interface, the formal query language working on the underlying database management system is open to database users. These users have rich knowledge about this target query language and the underlying database management system. By making use of this interface, they can make some special database operations which are related to the database management system and are not offered in the general query language.

As a consequence, in such an environment, the general formal query language can be considered as a bridge linking a natural language and a target formal query language. This can benefit those users who have different levels of knowledge about the database query system. Based on their knowledge about the database query system, users can choose a working interface in this environment to pose their queries.

Along with the richness of their knowledge about the database query system, they can replace their working interface with a more complicated one so that the database queries can be given more precisely and effectively.

Simultaneously, such an environment also improves the transportability of the interface system, since the general formal database query language is independent of any specific database management system. If the underlying database system is changed, the only thing we need to do is to rewrite a query language translator between the general formal query language and the target formal query language. The natural language interface and general query language interface are left untouched.

The main contribution of this thesis in the area of natural language interface to database query system are the following:

1. Offering a model of Multilevel Interface to Database Management System in which database users are able to pose their queries with several different query languages based on their understanding to database query system.
2. Providing a semantic compatible criterion for testing the semantic compatibility between two noun phrases for handling the conjunction problem.
3. Enriching the normal E-R Model used in database management system by adding the semantic content of the database domain knowledge.
4. Establishing a query meaning representation, which is more formal query language oriented, for filling the syntactic gap between a general semantic representation of a natural language sentence and a query specification of a formal query language.

5. Combining the top-down and bottom-up parsing mechanism, which is used for syntactic analysis, and Case Grammar, which is used for semantic analysis, for building a natural language interface which interprets English queries and produces SQL query specifications.

These contributions are described in the remaining chapters of this thesis.

1.2 MIDBMS SYSTEM GOAL AND OUTLINE

Under the proposal given in previous section, a model and an overall design of the MIDBMS system has been developed. For testing the design, a prototype system has been implemented. In the prototype system, a subset of English language is selected as a query language.

At the general formal query language interface, Structured Query Language (SQL) [15][16] is selected as a query language. The reason for choosing SQL as the general formal query language is that SQL is becoming the "de facto" standard ıelational query language. It has clear, meaningful, and structured syntactic format and the powerful semantic ability for database manipulation. It can be independent of any specific database management system.

At the target formal query language interface, Global Query and Mapping Language (GQML) working on the Heterogeneous Distributed Database Management System (HDDBMS) [18][39] is selected. This is because the GQML provides a higher level language interface over several different types of database products each of

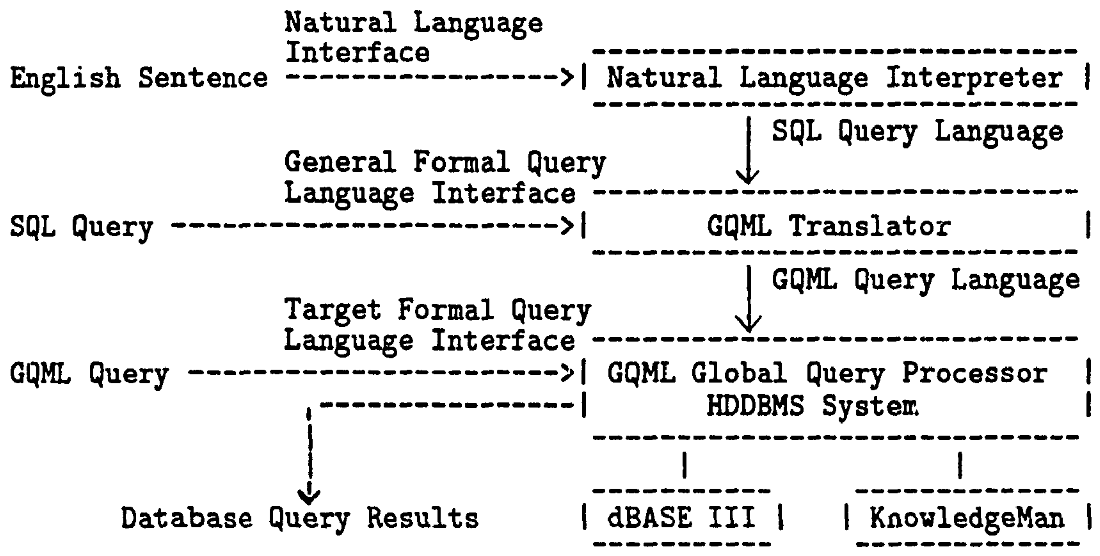
Natural Language Interface
English Sentence
Natural Language Interpreter
SQL Query Language
General Formal Query Language Interface
SQL Query
GQML Translator
GQML Query Language
Target Formal Query Language Interface
GQML Query
GQML Global Query Processor
HDDBMS System
Database Query Results
dBASE III
KnowledgeMan

Figure 1.1: Architecture of MIDBMS System

which has their own query language. This offers a better test bed for the multilevel interface system. In this case, one more local query language interface can be open to the database users. At this interface, users can use different formal query language to give queries to the corresponding local database system. Figure 1.1 describes the architecture of the MIDBMS system.

In the architecture described in Figure 1.1, the Natural Language Interpreter accepts English sentences, performs the syntactic and semantic analyses, and interprets the result of the syntactic and semantic analyses into the general formal query language - SQL. At this processing stage, most of the work concentrates on the natural language understanding. It includes lexicon organization, lexical analysis, syntactic analysis, semantic analysis, and query interpretation. The general issues encourtered in this processing stage are discussed in Chapter 2.

For performing the syntactic and semantic analyses, the syntactic and semantic knowledge about the natural language is required. For producing the formal query language, the database domain knowledge is required. This involves the choice of an appropriate knowledge representation scheme to represent these different types of knowledge. This issue is discussed in Chapter 3. At this interface, the ability of the database update could be provided, but the current MIDBMS system just deals with database queries.

As a whole, at this processing stage, the main concern is the portability of the natural language interface for the three types of database transfer[17]:

- change of DBMS;
- change of application domain;
- conceptual reorganization of the database.

Such a goal can be achieved by applying the principle of modularity to the system design. It includes the modularity of the system functions, such as splitting the syntactic analysis and the semantic analysis into two separate parts, as well as the modularity of the knowledge representation, such as splitting the syntactic and semantic knowledge into a database domain independent part and a database domain dependent part. For performing the principle of modularity, some techniques are investigated and examined in the remaining chapters of this thesis.

At the general formal query language interface, the GQML translator accepts SQL and translates it into GQML. Since GQML is working on a heterogeneous distributed database management system, it has some special operations which are related to the mapping among the heterogeneous database schemas. These special operations are not provided in SQL. However, since both languages are working on the same relational data model, the query specifications of these two query languages have similarity in terms of the syntax and the semantics. Thus, it is possible to translate SQL query specification into GQML query specification. However, the translation from SQL to GQML is not reversible due to the schema mapping operations which are provided in GQML but not in SQL. Therefore, it is reasonable to put GQML into the target formal query language interface. In this sense, if database users need to perform some schema mapping rather than query operations among the heterogeneous database systems, they have to use GQML language at the target formal query language interface. Except these special operations, database query specifications can be given at each level of the interfaces. The issues encountered in this processing stage, such as the different operations between the query specifications from SQL and GQML, are discussed in Chapter 2.

At the target formal query language interface, the global query processor of GQML accepts the GQML query specification and cooperates with the other part of HDDBMS system to handle the global query processing. This includes:

- interpreting the global query into a set of local queries which are sent to the corresponding local DBMS;
- when the local database results are ready, collecting the database results from each of the local database systems which are involved in the global query pro-

cessing.

- combining all the collected database results and sending them back to the users.

As a result, it seems to the database users that they are facing a single conventional database management system rather than a set of different products of the DBMS systems.

For establishing such a facility, a mapping mechanism among the heterogeneous database schemas is required. This mapping can be performed at the different levels so that the different conflicts among the different database schemas can be resolved. Through this mapping mechanism, an integrated view of multiple existing database management systems can be presented to the database users. Therefore, these users do not need to know any details about the heterogeneous and the distributed features of the database system; these features having been absorbed by the mapping mechanism. Figure 1.2 shows a possible logical levels of the database schema mapping architecture.

The levels of the schema architecture presented in Figure 1.2 are explained below:

1. Local host schema: each site participating in the system has a local host schema that describes the local database using the data model of the local DBMS.

2. Translated local host schema: this is a subset of the local host schema after translation into a global data model schema.

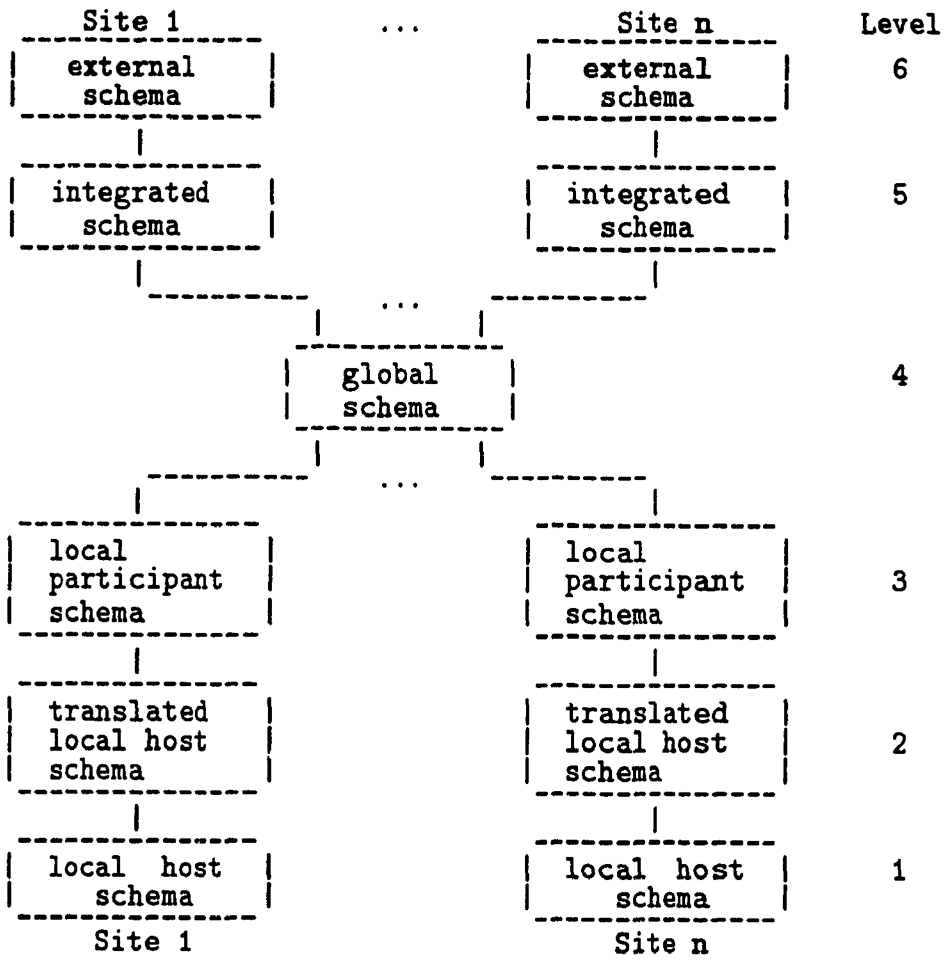

Figure 1.2: Reference HDDBMS Schema Architecture (from [39])

3. Local participant schema: this is defined by any number of mapping operations on the translated local host schema. This mapping may be performed in order to solve a subset of database conflicts.

4. Global schema: this is a collection of all local participant schemas from each site and represents the global database and the maximum amount of DDBMS data available to any one site.

5. Integrated schema: each site has an integrated schema which is defined by any number of mapping operations on the subset of the global schema accessible by that site. These mapping operations resolve any remaining database conflicts and specify integrated objects, properties, and relationships.

6. External schema: when it is necessary for the global database at a given site to be expressed in an external data model which differs from the global data model, an external schema may be derived from the integrated schema at that site.

Under this architecture of the schema, the problems which HDDBMS has to deal with are the followings[4]:

- resolution of naming conflicts, which typically involves either semantically equivalent data items named differently in different databases, or semantically different data items which have the same name in different databases;

- resolution of data representation conflicts for the same data item in different databases;

- resolution of data scaling conflicts, when the same data item is stored in different databases using different units measure;
- resolution of data structure conflicts caused by using various data models by different DBMSs;
- resolution of data inconsistencies for the same data item residing in several databases.

Except the schema mapping mechanism of the HDDBMS system, the abilities of constructing local query commands, distributing the local queries to the corresponding local database system, and collecting the local database results are also required. So far, a brief introduction to HDDBMS is given and this is just trying to give some background knowledge for building the MIDBMS system. For more details, see [3], [4], [18], and [39].

In the remaining chapters of the thesis, discussion concentrates on the general issues of transportability, extensibility, and the knowledge representation. Chapter 2 discusses general issues in building a multilevel interface to database management system. Chapter 3 discusses knowledge representation of the natural language understanding systems. Chapter 4 discusses the issues of the MIDBMS system design and implementation. Chapter 5 concludes this thesis and summarizes future work.

Chapter 2

GENERAL ISSUES IN BUILDING A MULTILEVEL INTERFACE TO DATABASE MANAGEMENT SYSTEM

2.1 SYSTEM COMPONENTS OF A NATURAL LANGUAGE INTERFACE

A natural language interface to database query system is actually a language translator which can convey the meaning of the user's query to database query system. However, since natural language and database query language are totally different from each other in terms of their syntactic and semantic definitions, it is necessary for a natural language interface to be able to understand the meaning contained in a natural language sentence it receives. Based on this understanding, a natural language interface should convert this natural language sentence into a target language sentence which the database query system can understand.

Generally, in the context of the computational linguistics, natural language is described in terms of its syntax, semantics, and pragmatics. These elements can be characterized roughly as follows[25]:

- Syntax is concerned solely with the relations between linguistic expressions;
- Semantics is concerned with the relations between expressions and the objects

to which they refer;

- Pragmatics is concerned with the relations among the expressions, the objects to which they refer, and the users or contexts of use of the expressions.

This suggests that natural language understanding system needs the following processing phases:

1. Lexical analysis phase: Lexical analyst accepts a natural language sentence, searches a dictionary or lexicon to find out a lexical definition for each word appearing in the sentence, and produces a lexical representation which is merely a sequence of syntactic markers, such as noun, verb, etc.

2. Syntactic analysis phase: Syntactic analyst accepts the output of the lexical analysis, applies grammar rules to the lexical representation to check if the sentence is correct grammatically. If the sentence is grammatically correct, it must produce a syntactic representation which is a tree structure containing lexical markers, such as noun, verb, etc., phrase markers, such as NounPhrase, VerbPhrase, etc., and the grammatical markers, such as subject, object, etc. at the different levels. In this processing phase, syntactic and semantic knowledge is required.

3. Semantic analysis phase: It is possible that a syntactically correct sentence could make no sense because of the internal semantic disagreement. In such a case, semantic analysis is required. In this processing phase, the output of the syntactic analysis is accepted and the semantic restriction rules are applied to the syntactic representation. If the sentence is semantically

correct, a meaningful semantic representation is produced. This semantic representation represents the deep structure of a sentence. For supporting this semantic analysis, semantic knowledge is required.

4. Query interpretation phase: After the above syntactic and semantic analyses, a general semantic representation can be obtained. This is what the interface system has understood. However, this meaningful representation is generally linguistic oriented. There still exists a big gap between the general semantic representation and the formal query language. Therefore, for producing a database query, another semantic representation which is more database query language oriented is desirable. This can be done by introducing a query interpreter. The query interpreter has to access the database domain specific knowledge and try to map the general linguistic representation to a specific database domain. If this mapping is successful, a query meaning representation can be produced.

5. Formal query language generation phase: In this processing phase, formal query language sentences can be produced by the query language generator based on the query meaning representation. Sentence generation is essentially the opposite of the sentence parsing. However, since the syntax of the database query language is fairly simple compared with the syntax of natural language as well as the semantics of the query language is well defined, the query language generation is relatively easier than natural language understanding. In this phase, the knowledge about the query language grammar is required.

```
 Natural Language Sentences
             |
             v
 ---------------------
| Lexical Analysis    |----------------------------
 ---------------------                             |
             | Lexical Representation              |
             v                                     |
 ---------------------                   ---------------------
| Syntactic Analysis  |-----------------| Syntactic Knowledge |
 ---------------------               |   ---------------------
             | Syntactic Representation  ------------
             v                                     |
 -----------------------                  --------------------
|  Semantic Analysis    |----------------| Semantic Knowledge |
 -----------------------                  --------------------
             | Semantic Representation
             v
 ---------------------            ---------------------------
| Query Interpretation |----------| Domain Specific Knowledge |
 ---------------------            ---------------------------
             | Query Meaning Representation
             v
 ----------------------------------
| Formal Query Language Generation |
 ----------------------------------
             |
             v
  Formal Query Language
```

Figure 2.1: General Design Architecture of a Natural Language Interface to Database Query System

In the above processing phases, different knowledge is required. It is desirable to represent these knowledge declaratively so that it is easier for a NLI system to extend its coverage of the syntactic phenomena and improve its transportability.

A general design architecture for a natural language interface is given in Figure 2.1. This is similarly defined in [17]. Each of the components of this architecture is discussed in the following sections.

2.2 LEXICAL ANALYSIS

The main task of a lexical analyzer is to provide an associativity between each word in an input natural language sentence and its syntactic and semantic definitions. In a natural language understanding system, the syntactic and semantic definitions are expressed with the syntactic markers, such as noun, verb, adverb, etc., and semantic markers, such as physical_object, abstract_object, etc. These markers are the most basic analytical elements manipulated by the syntactic and semantic analyzers. For being retrieved and used conveniently by the system analyzers, these markers are stored in a dictionary or lexicon which is the major knowledge source of the natural language understanding system.

Once a sentence is received, the lexical analyzer has to search the lexicon and try to find a corresponding lexical entry for each word in the sentence so that the associativity between the words and their syntactic and semantic markers can be established. The following example shows that each word in the sentence "List the names of the students" is associated with its corresponding syntactic marker.

`Word:`	`List`	`the`	`names`	`of`	`the`	`students`
`Syntactic Marker:`	`Vt`	`Det`	`N`	`Prep`	`Det`	`N`

These meaningful symbols construct a sequence of the syntactic markers which presents the lexical construction of a sentence. In this thesis, this sequence of the syntactic markers of a sentence is named as a lexical interpretation.

In English, since a word could play several syntactic roles and/or several semantic roles, syntactic and semantic ambiguities arise. For determining a unique meaning of a English sentence, we have to choose one of the syntactic roles and one of the semantic roles for each word in the sentence based on the context of a discourse. Thus, English is said to be context sensitive. The basic reason for this sensitivity of English is that most of the English words are syntactically and/or semantically overloaded. This is true of other natural language as well. Taking the sentence "List the names of the students" as an example, we can find that this sentence is syntactically ambiguous because the words "list" and "names" are syntactically overloaded. The word "list" has syntactic roles of noun and verb and the word "names" has the syntactic roles of noun and verb as well. As for semantic ambiguity, the word "charge" can give us an example. If we say "He charged the battery," then the meaning of the "charge" is to add an electrical charge to a battery. However, if we say "He charged me," then the meaning of the "charge" is to ask as a price. In these two sentences, the semantic meaning of the word "charge" is different based on the context even though it plays the same syntactic role of verb. Actually, the word "charge" is also syntactically ambiguous since it could be a noun.

From the above discussion, we can see that in the lexical analysis phase, the lexical analyzer should expose all the possible syntactic roles and semantic roles which the words in a sentence possess. Based on these possibilities, one of the syntactic and semantic meaning of those overloaded words can be chosen by performing the syntactic and semantic analyses. As a result, a unique meaning of a sentence is determined. For performing the syntactic and semantic analyses systematically, it

	List	the	names	of	the	students
1).	N	Det	N	Prep	Det	N
2).	N	Det	Vt	Prep	Det	N
3).	Vt	Det	N	Prep	Det	N
4).	Vt	Det	Vt	Prep	Det	N

Figure 2.2: Example of Lexical Interpretations

is desirable to produce all the possible lexical interpretations for each sentence in this lexical analysis phase. In this case, each time just one lexical interpretation is submitted to the syntactic analyzer. If this lexical interpretation fails to be parsed, then the next lexical interpretation is submitted until one of the lexical interpretations is parsed or no more lexical interpretation is left. This approach is greatly simplifies the syntactic and semantic analyses. Figure 2.2 gives an example which shows all the lexical interpretations for the sentence "List the names of the students."

Producing all the possible lexical interpretations of a sentence simplifies the syntactic and semantic analyses of a NLI system. However, it presents another problem, that is, some of the interpretations are obviously hopeless but they are still submitted to the syntactic analyzer to be processed. This part of the analytical efforts done by the syntactic analyzer is totally useless. As a result, the efficiency of the syntactic analyzer is decreased. Therefore, choosing the most promising lexical interpretation to be submitted first to the syntactic analyzer is a key problem in the lexical analysis phase in terms of the system efficiency.

A possible solution of this problem is to establish some heuristic functions according to the syntactic knowledge. In this way, a processing order for all the lexical interpretations can be established. The most promising interpretation is given the highest priority. In the example given in Figure 2.2, we may find that the third lexical interpretation is the most promising, because:

1. A determiner or an adjective should be followed by an adjective or a noun;
2. There should be, at least, one verb in a sentence.

These two criteria could be taken as the heuristics used in the lexical analysis phase to produce a processing order of the lexical interpretations so that the efficiency of the syntactic analyzer can be improved.

As described above, the lexicon is the major knowledge source of the syntactic analyzer. A lexicon consists of a set of lexical entries. Each word of the natural language used for posing queries has, at least, one lexical entry. Each entry of the lexicon should provide a syntactic or grammatical portion, which contains the syntactic definition for each word; and the semantic portion, which represents each of the distinct senses that each word has in its occurrences[30]. As a result, each word used in a natural language can be defined uniquely in this lexicon.

In the syntactic portion of the lexicon, the syntactic markers are used to identify the possible syntactic uses of a word. For example, the word "person" is a noun and the word "teach" is a verb. However, some words could have more than one syntactic roles to play as described above. The word "list" can be a noun or a verb. This syntactic feature requires that the lexicon must provide a mechanism in which a word can be defined in multiple lexicon entries based on its multiple syntactic and/or semantic definitions. In addition, as a knowledge source, the lexicon should be represented with an appropriate knowledge representation scheme in a natural language understanding system. These knowledge representation schemes and techniques are discussed in the next chapter.

Since the syntactic portion of the lexicon is defined with the syntactic markers which express the syntactic categories for each word, a systematic organization of the knowledge about syntactic classification for words is required. Such an organization is provided by the linguists with word classes. In the word classes, words are classified into groups in a way that provides a relatively small number of classes, each of which has an internal regularity and a relatively small degree of overlap with other classes[44]. Figure 2.3 presents a possible syntactic structure for noun class.

In the semantic portion of the lexicon definition, semantic markers are used to distinguish the various senses of each syntactic use of a word. Katz and Fodor developed a theory[30] which claims that a finite set of semantic markers can be used to describe items in language and to associate the meanings between different words within a sentence. This theory offers a basic foundation for semantic analysis. However, even though a word gets a semantic marker, it is, at some time, still not enough to resolve the semantic ambiguity. For example, the word "charge" could be a noun or a verb. In its noun classification, the word "charge" has a semantic marker of abstract-object. Under this semantic category, this word still could have the meaning of "expense" or the meaning of "accusation." This problem can be

```
                         - Proper
       ------------|- Pronoun
       |                 - Common
       |
       |                 - Definite
       |------------|                    - Question
Noun   |                 - Indefinite --|- Quantified
Class  |                                 - ...
--------|
       |                 - Singular
       |-   Number --|
       |                 - Plural
       |
       |                 - First
       |-   Person --|- Second
       |                 - Third
       |                                   - Determiner
       |                 - Possessive  --|
       |                 |                 - ...
       -   Case ----|
                         |          - Subjective
                                    - ...    - Objective
                                    - Reflexive
```

Figure 2.3: Syntactic Structure for Noun Class (from [44] p518)

resolved by introducing some semantic distinguishers so that the word sense can be further distinguished. As described in the syntactic structure for the word classes, these semantic markers also can be organized into a semantic class so that a systematic organization of the semantic distinction for word senses can be provided. Figure 2.4 gives an example of a semantic structure with the semantic markers and distinguishers.

The syntactic class and the semantic class discussed above are the knowledge about the lexicon definition. They are essential for the natural language processing. Thus, we have to find appropriate knowledge representation schemes to represent them. This knowledge representation issue is discussed in the next chapter.

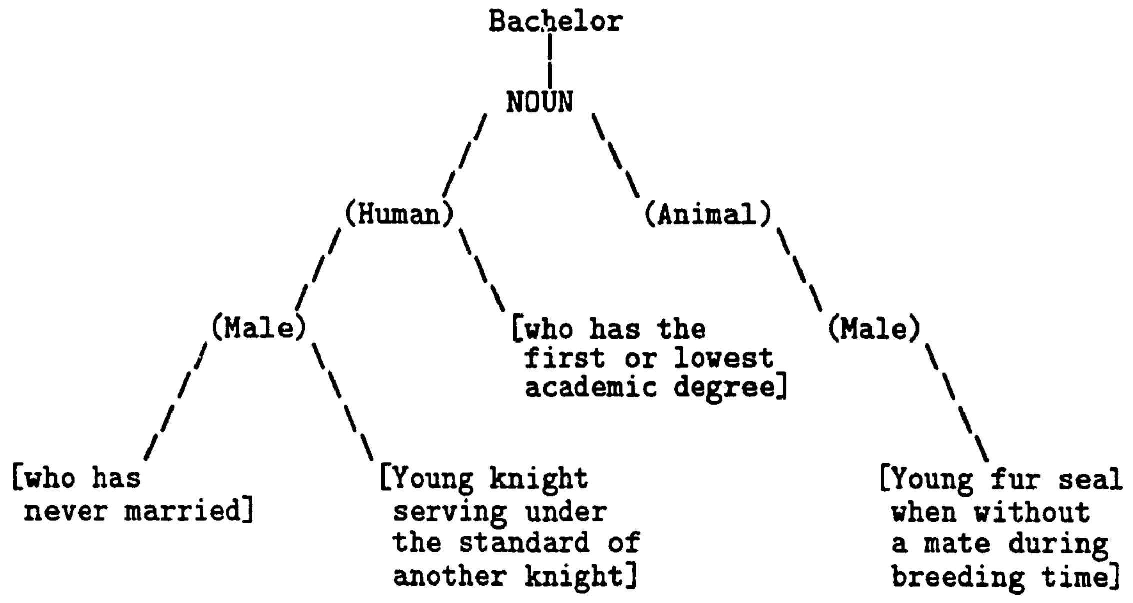
Bachelor
NOUN
(Human)
(Animal)
(Male)
[who has the first or lowest academic degree]
(Male)
[who has never married]
[Young knight serving under the standard of another knight]
[Young fur seal when without a mate during breeding time]

Figure 2.4: Semantic Structure with Semantic Marker and Distinguisher (from [30] p496)

2.3 SYNTACTIC ANALYSIS

Before discussing the syntactic analysis, we need to review some basic concepts of linguistics. Linguistics can be defined most simply as the study of language, in particular, natural language. In general, linguistics is concerned with how languages work and how they are used[25]. Any kind of natural language consists of sentences. One sentence expresses a complete thought and is made up of several phrases. In most cases, each phrase has its unique syntactic and semantic meaning. Each phrase is made up of several words. A word is the smallest linguistic unit which has syntactic and semantic roles and is capable of expressing ideas. Based on the syntactic and semantic analysis, a pair of roles can be decided. Thus, a unique meaning or idea expressed with a word is determined. In this section, syntactic component of linguistics and its analysis are discussed.

The syntactic component of a language has several syntactic roles. These roles are noun, pronoun, verb, adjective, adverb, preposition, conjunction, and interjection.

- A noun is a name for something, which could stand for a person, a place, or a thing. For example, *John, Montreal*, and *ball.*

- Pronoun is used to replace a noun when the noun is already known. For example,

  The boy had a ball, and *he* threw *it.*

  where *he* and *it* are pronouns. Pronoun is changed according to the number and person, and it must agree with the noun which they are replacing. Number has singular and plural. Person has first, second, and third.

- Verb is used to express an action, being, or state of being. For example,

  The boy *threw* the ball. The girl *is* a student.

  Verbs are further classified as transitive, intransitive, and linking. In one sentence, the verb plays a significant role in terms of the sentence meaning. It has following properties: voice, tense, person, and number.

- Adjective is used to modify a noun or a pronoun. For example,

  The boy has a *nice* shirt.

- Adverb is used to modify a verb, an adjective, another adverb, or an entire sentence. For example,

  The boy drove the red car *very fast.*

- Preposition establishes the relationship between a noun or noun groups and some other part of the sentence. For example,

  He will be back *on* Sunday morning.

- Conjunction joins words or phrases together. For example,

  The boy will go home *after* he finishes his work.

- Interjection expresses strong feelings. For example,

  Oh, no. I don't want to go.

Phrases are composed of a string of words. Examples of phrases are noun phrase(NP), verb group(VG), preposition phrase(PP) and so on. For example,

The teacher will give a lecture on Monday morning.

where "The teacher" and "a lecture" are noun phrases, "will give" is a verb group, and "on Monday morning" is a prepositional phrase.

Sentences can be classified by their structures and uses. In terms of the structures, we have simple, compound, and complex sentences. In terms of the uses, we have declarative, interrogative, imperative, and exclamatory sentences. Different classifications of sentences have different word orders. Thus, the meaning of an English sentence is dependent to a large extent on word order[25].

Since one of the main concerns of linguistics is how a language is used, it is necessary to understand how an internal structure of a sentence is constructed. One of the most famous linguists, Noam Chomsky, gives a language structure by developing

```
S   --> NP  VP
NP  --> Art  N
VP  --> Aux  V
Aux --> is
V   --> crying
Art --> the
N   --> boy
```

Figure 2.5: Example of Phrase Structure Grammar

a series of generative grammars which can produce acceptable structural relations within a sentence. Based on this linguistic research, a Phrase Structure Grammar is defined[10][11].

A Phrase Structure(PS) grammar starts with a sentence and defines its parts, then defines each of the sub-parts, continuing the redefinitions until a sentence has been produced. A simple PS grammar is given in Figure 2.5.

Based on this simple grammar, we can produce a sentence "The boy is crying." The corresponding tree structure is given in Figure 2.6.

The PS grammar could serve both in the sentence generation and the sentence analysis. Based on the production rules defined in a PS Grammar, we can perform inference from the left side of the rule to the right side of the rule until the terminal symbols are reached. This approach can be called top-down parsing strategy. If the analysis starts from the bottom level, we can use induction mechanism to abstract syntactic components level by level from the terminal symbols until the top level S is reached. This can be called bottom-up parsing strategy. It is also possible to combine

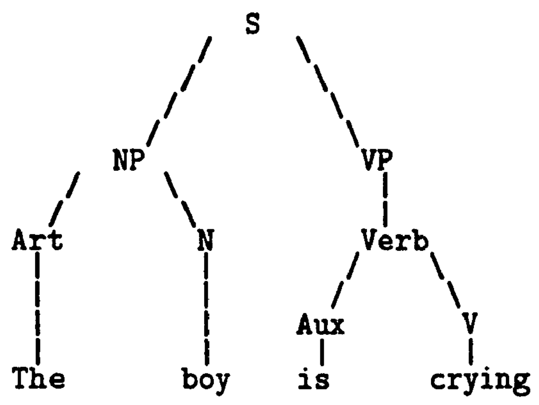

Figure 2.6: The Tree Structure of The Sentence "The boy is crying"

both top-down and bottom-up approaches into one parsing mechanism. Generally, in the context of computational linguistics, syntactic analysis could pursue each of these three different parsing strategies based on context-free grammars[1]. Terry Winograd[44] has given an extensive analysis and discussion of syntactic analysis.

In syntactic analysis, another problem we need to deal with is the conjunction problem, which is one of the most problematic fields in a NLI system. In conjunction processing, one can simply include some rewriting rules for 'NP —> NP Conj NP,' 'S —> S Conj S,' 'PP —> PP Conj PP,' etc. to combine two same type of phrases into one. In some systems, such as LUNAR[46], rather than having separate conjunction rules for each syntactic category in the grammar, there is special process associated with conjunction that produces the same effect. This leads to a short grammar but a more complex parser is required[44].

However, this pure syntactic processing approach does not always work. For example, in sentence "List all the names and the salaries of the employees," we can combine "the names," "and," and "the salaries" into one noun phrase because both the "name" and the "salary" are the attributes of the human object. But in another sentence "List the names of the employees and their salaries," we cannot combine "the employees," "and," and "their salaries" into one noun phrase even though this sentence contains the same syntactic structure of "NP Conj NP" as the previous sentence. If we do combine "the employees" and "the salaries" together, then the meaning of the sentence is becoming "List the names of the employees and the names of the salaries." This certainly does not make sense. The reason is that, in the first sentence, the "name" and the "salary" have the same semantic category which is the attribute of human object. But this condition does not hold in the second sentence where "employees" is a human object while "salaries" is an attribute of the human object. This indicates that the two noun phrases in the first sentence are semanticly compatible but in the second sentence, the two noun phrases are not. Thus, we need to define an operation in the semantic class to test the semantic compatibility between two noun phrases associated with a conjunction. In this thesis, we provide a criterion to test if two noun phrases are semanticly compatible. This criterion consists of two rules:

1. If two noun phrases have the same semantic definition, then these two noun phrases are semanticly compatible;
2. If two noun phrases are the attributes belonging to two different semantic objects and one of the semantic object is the ancestor of the other, then these two noun phrases are semanticly compatible.

With these two rules, we can handle conjunction problem conveniently. For example, since the words "student" and "teacher" have the same semantic category (human object), these two noun phrases are semanticly compatible based on the first rule. If a word is "name" belonging to animate object and the other word is "address" belonging to human object, based on the second rule, these two noun phrases are semanticly compatible as well because the animate object is the ancestor of the human object.

As described above, this syntactic analysis needs to access not only the syntactic knowledge but also the semantic knowledge. This makes the syntactic analysis more sufficient and reliable.

2.4 PARSING STRATEGIES FOR CONTEXT-FREE GRAMMARS

2.4.1 Top-Down Strategy

Top-Down parsing strategy is a goal-directed parsing mechanism. In this parsing process, the rules of a grammar are used in such a way that the right-hand side of the rules are always used to rewrite the symbols on the left-hand side. This expansion process starts from the top level of the grammar, keeps rewriting those constituents as well as producing the parse tree structure simultaneously until the terminal level is reached. The most influential work employing top-down parsing strategy is Augmented Transition Network (ATN)[44][45]. Figure 2.7 illustrates a possible top-down parsing process with the grammar given in Figure 2.5 for the sentence "The boy is

```
S --> NP VP
  --> Art N VP              (rewriting NP)
  --> the N VP              (rewriting Art)
  --> the boy VP            (rewriting N)
  --> the boy Aux V         (rewriting VP)
  --> the boy is V          (rewriting Aux)
  --> the boy is crying     (rewriting V)
```

Figure 2.7: Top-Down Parsing Example

crying."

In practice, the syntactic structure that a NLI system has to analyze is more complicated than the example given in Figure 2.7. If we try to parse the same sentence "The boy is crying" with the grammar given in Figure 2.8, we may find that this parsing process is not that straightforward. In this example, the parser, using rule 1, would find a NP, "The boy," and then using rule 6 for the VP, would find the necessary AUX and VERB but not the following NP. Thus, it backtracks and tries another way to find a symbol S. Rule 2 is tried, and the NP "The boy" is parsed again but the next one is AUX instead of VERB. Thus, it backs up again and tries rule 3. This succeeds after parsing "The boy" as a NP for the third time, "is" as a AUX, and "crying" as a VERB. From this example, we can find that the same part of the parsing process may be repeated many times in searching for a solution. Therefore, with a pure top-down parsing strategy, a lot of backtracking is involved. As a result, many pieces of the parsing results have to be abandoned. This decreases the efficiency of the parser.

```
1)          S  ---->  NP   VP
2)          S  ---->  NP   VERB
3)          S  ---->  NP   AUX   VERB
4)          NP ---->  ART NOUN
5)          NP ---->  ART ADJ   NOUN
6)          VP ---->  AUX VERB NOUN
7)          VP ---->  VERB NP
```

Figure 2.8: Grammar Example (from [1] p67)

2.4.2 Bottom-Up Strategy

The bottom-up parsing strategy is a data-directed combination process. Compared with the top-down strategy, bottom-up strategy starts with the individual words. It looks for rules whose right-hand sides can match the sequence of adjacent words in a sentence. If such a match is found, these words are combined into a constituent as identified by the left-hand side of the rule. Then, the parser tries to combine the constituents with each other and the remaining words in the sentence into a larger constituent. Simultaneously, a parsing tree structure is built up. This process proceeds until all the constituents covering the entire input can be combined into a single structure labeled with the distinguished symbol. The Chart Parser[44][46] is an example of employing bottom-up parsing strategy. Figure 2.9 illustrates a possible bottom-up parsing process with the grammar given in Figure 2.5 for the sentence "The boy is crying."

With the grammar given in Figure 2.10, we are able to parse a more complicated sentence "The large can can hold more water." In this example, we can get NP1 with

```
--> Art boy is crying          (rewriting the)
--> Art N is crying            (rewriting boy)
--> NP is crying               (rewriting Art N)
--> NP Aux crying              (rewriting is)
--> NP Aux V                   (rewriting crying)
--> NP VP                      (rewriting Aux V)
--> S                          (rewriting NP VP)
```

Figure 2.9: Example of Bottom-Up Parsing Process

```
1)                 S  ---->  NP  VP
2)                 NP ---->  ART ADJ NOUN
3)                 NP ---->  ART NOUN
4)                 NP ---->  ADJ NOUN
5)                 VP ---->  AUX VERB NP
6)                 VP ---->  VERB NP
```

Figure 2.10: Grammar Example (from [1] p61)

rule 2 by combining "The large can," NP2 with rule 4 by combining "large can," NP3 with rule 4 by combining "more water," VP1 with rule 5 by combining "can hold" and NP3, VP2 with rule 6 by combining "hold" and NP3, S1 with rule 1 by combining NP1 and VP1, and S2 with rule 1 by combining NP2 and VP1. These phrases are given in Figure 2.11.

From the above example, we find that some of the phrases, such as NP2, VP2, and S2 never lead to a legal sentence. However, this problem cannot be found during the parsing process until the top level of the grammar is reached. As a result, the phrase structures established during the parsing process for the phrases NP2, VP2,

NP1:	the large can
NP2:	large can
NP3:	more water
VP1:	can hold more water
VP2:	hold more water
S1 :	the large can can hold more water
S2 :	large can can hold more water

Figure 2.11: Example of Bottom-UP Parsing Process

and S2 have to be given up later on. Compared with the top-down parsing strategy, the bottom-up strategy can avoid the backtracking problem encountered in the top-down strategy. However, it still produces some undesirable parsing results. This seems to indicate that the mixture of the top-down and bottom-up parsing strategies is quite promising.

2.4.3 Mixture of Bottom-Up and Top-Down Strategy

Based on the discussion and comparison made in the previous sections, we may find that both top-down and bottom-up parsing strategies have their own advantages and disadvantages. In top-down parsing strategy, some pieces of analytical structures built up for keeping the syntactic analytical results have to be given up and subsequently backtracking is required. The reason for the backtracking is the failure of the parser to find an appropriate phrase which can lead to a legal sentence. Such an example is given in Figure 2.8.

In bottom-up parsing strategy, all the possible phrases are found. Based on these phrases, the parser tries to construct larger syntactic constituents until the entire input is combined into a single syntactic structure. However, some of these attempts fail since some of the phrases do not lead to a legal sentence. Such an example is given in Figure 2.11.

Since both parsing strategies discussed above involve some useless parsing efforts, it is desirable to design a parser which uses varying degree of both top-down and bottom-up parsing strategies and gains the advantages of both approaches without the disadvantages. By analyzing both parsing strategies, we find that the problem causing the useless efforts of the parser during its parsing process is in the phrase construction level. If we can use bottom-up parsing strategy to construct phrases and use top-down parsing strategy to parse these phrases based on a grammar, then a lot of useless efforts for parsing a sentence could be avoided. For performing the bottom-up parsing strategy mentioned above, the following three restrictions should apply:

1. The processing order of a sentence is strictly from left to right;
2. The parser should look ahead as many words as needed for finding the largest string match under the grammar rules used in a parsing system;
3. Any phrase cannot overlap the previously recognized phrase.

In a mixed parsing strategy with the left to right processing order, the parser can be divided into two separate parts which are bottom-up parsing processor and top-down parsing processor. In bottom-up parsing stage, all the phrases are found with the restrictions described above. Then, a conventional top-down parsing mechanism

```
Sentence:
              The   large   can   can   hold   more   water

Lexical Interpretations:
1)            det   adj     aux   aux   vt     adj    n
2)            det   adj     n     aux   vt     adj    n
3)            det   adj     aux   n     vt     adj    n
4)            det   adj     n     n     vt     adj    n
5)            det   adj     aux   aux   n      adj    n
6)            det   adj     n     aux   n      adj    n
7)            det   adj     aux   n     n      adj    n
8)            det   adj     n     n     n      adj    n

Legal Phrases from Interpretation 2:
              NP1:  The large can
              VP1:  can hold
              NP2:  more water
```

Figure 2.12: Example of Mixed Parsing Strategy

is employed to parse these phrases. Since most of the unreasonable phrases can be discarded at the phrase construction stage, the useless efforts for parsing a sentence could be reduced to the minimal level.

Example given in Figure 2.12 illustrates how such a mixed parsing strategy is used.

In this example, all the possible lexical interpretations of the sentence "The large can can hold more water" are listed and the grammar given in Figure 2.10 is used for parsing this sentence. Before starting the parse of this sentence, the heuristics given in section 2.2 can be applied for removing some hopeless lexical interpretations. As a result, the lexical interpretations 1, 3, 5, 6, 7, and 8 are removed. Just the interpretations 2 and 4 are left to be parsed. In the bottom-up parsing stage described above, the second lexical interpretations can be parsed and all the reasonable phrases are given in Figure 2.12. However, the forth lexical interpretation cannot be parsed. In this interpretation, the words "The large can" can be combined into a NP but the next word "can" as a noun, cannot be combined into any phrase. Therefore, this lexical interpretation is discarded at the phrase processing stage. In top-down parsing stage, the phrases obtained from the second lexical interpretation can be parsed.

From this example, we can see that the useless efforts for parsing a sentence are reduced to the minimal level. Thus, the mixed parsing strategy is very helpful for improving the efficiency of the parsing mechanism. Top-Down CFG Parser with a Chart[1] offers another possible solution for combining top-down and bottom-up parsing strategies to parse sentences. For more details, please refer the original paper[44].

2.5 SEMANTIC ANALYSIS

Since English is ambiguous not only syntactically but also semanticly, there could still exist some semantic ambiguities after the syntactic analysis. For example, if we say "A cat teaches the course comp212," we know that this sentence does not make sense based on our general world knowledge. Because it is impossible for a cat to teach a course. However, this sentence is syntactically correct. By analyzing this sentence, it can be found that the action expressed with the verb "teach" needs a human actor as its subject instead of a animal expressed with the word "cat." This indicates that the action and the corresponding actor appearing in this sentence do not semanticly agree with each other[1]. Such a problem of semantic disagreement has to be resolved in the semantic analysis.

In the semantic analysis, we need to find out the verb of a sentence first. Based on the information about the verb, the semantic agreement checking can be performed. In this analysis, the semantic roles of the words in a sentence are used. These semantic roles are defined with the semantic markers and these semantic markers are stored in the lexicon as described in section 2.2. They can be manipulated and accessed by the semantic analyzer conveniently. During the semantic analysis, in addition to the knowledge about the semantic classification for each word, the knowledge about the verb is also required. This knowledge includes the specification of semantic roles required by each verb. For expressing such a knowledge and performing semantic analysis, it is desirable to establish a more semantic oriented framework. In this framework, the knowledge about the verb can be represented and a deep meaning representation of a sentence can be established. This is also helpful to solve the problem that one meaning could corresponds to several different syntactic structures. In practice, for expressing the same meaning, one could use several different sentences. For example, one could say "Which courses are taken by Backer?" or "Which courses does Backer take?" to express the same question. After syntactic analyses to these two sentences, two different syntactic structures are established. This example reveals the fact that the meaning could be expressed with several different syntactic structures. Rephrasing, several different syntactic structures could represent the same deep meaning. Therefore, if we can establish a deep structure to represent the deep meaning that the system has understood, it will simplify the semantic analysis and ease the query interpretation.

Many researchers have attempted different approaches to capture the deep meaning of a sentence and representing it with a deep structure. The most influential work among these research is Case Grammar developed by Fillmore[6][20]. Case Grammar describes a natural language from a more semantically oriented perspective than the transformational grammars of Chomsky[12] and establishes a semantic representation for expressing the deep meaning of a sentence.

Case Grammar is defined with a set of rewrite rules which are used to convert the surface structure of a natural language sentence into its corresponding deep structure. In this definition, the deep structure of a sentence consists of a modality M and a proposition P. The modality M contains some information about verbs. They are tense, aspect, negation, and mood. Proposition P consists of a verb and several cases associated with the verb. From this definition, we can see that verb is the major source of Case Grammar. This makes Case Grammar more verb-oriented. The rewrite rules and the corresponding tree structure of Case Grammar are given in Figure 2.13.

A list of cases which are used in Fillmore's original Case Grammar are given as follows.

```
Rewrite Rules:

S  --> M + P
M  --> tense, aspect, negation, and mood
P  --> V + C1 + C2 + ... + Cn
Ci --> K + NP
K  --> proposition

Tree Structure:

                  S
                /   \
               /     \
              /       \
             /         \
            /           \
           M             P
           |             |
           |     ----------------------
           |     |     |    |          |
       Negation  Verb  C1   C2   ...   Cn
       Tense
       Aspect
          .
          .
          .
```

Figure 2.13: The Rewrite Rules and the Tree Structure of Case Grammar

Agent:	the instigator of the action, an animate being;
Instrumental:	the thing used to perform the action, an inanimate object;
Locative:	the location of the action and other cases;
Dative:	the recipient of the action;
Neutral:	the thing being acted upon.

Depending on different situations, more cases could be assigned for obtaining richer semantic roles.

In Case Grammar, a case frame specification must be set up for each verb to indicate which case is required, optional, or not allowed. For example, the verb "give" could have the following case frame specification.

Verb:	give
Agent:	optional
Instrumental:	optional
Locative:	not allowed
Dative:	optional
Neutral:	required

```
Sentence: Which courses are taken by Backer?
                     S
                   /   \
                  /     \
                 /       \
                /         \
               M           P
               |           |
               |     ----------------------------
               |     |             |              |
Interrogative  Verb          Agent          Neutral
Tense: present take          Backer       which course
Voice: passive
```

Figure 2.14: Example of the Case Frame Representation

Figure 2.14 and Figure 2.15 describe two deep structures represented with case frames for the sentences "Which courses are taken by Backer?" and "Which courses does Backer take?", respectively.

Comparing these two examples, we can find that in these two deep meaning structures, the proposition parts are the same. The only difference occurs in the modalities which contain the information about the verb "take." One is active voice and the other is passive voice.

Based on such a framework, semantic analysis can be carried out. The main task of this analysis is to map the surface structure of a sentence onto its deep structure. This is done by filling up the cases established in the deep meaning representation according to a set of restriction rules which will be discussed in section 4.4. Once the semantic analysis is completed, the deep meaning structure produced in this stage is taken as the output of the semantic analyzer and submitted to the query interpreter.

```
Sentence: which courses does Backer take?
                      S
                    /   \
                   /     \
                  /       \
                 /         \
                /           \
               M             P
               |             |
               |       ----------------------------------
               |       |            |                  |
   Interrogative    Verb         Agent             Neutral
   Tense: present   take         Backer          which course
   Voice: active
```

Figure 2.15: Example of the Case Frame Representation

2.6 AMBIGUITIES

Two common types of ambiguities likely to be encountered in natural language database queries are syntactic and semantic ambiguities [29] [37] [44]. Syntactic ambiguity can have a more detailed classification as follows: lexical ambiguity, structural ambiguity, and referential ambiguity. Several examples are given below to illustrate these different ambiguities.

```
Sentence 1:

    List   the   names   of   the   students.
```

In sentence 1, we could have four different lexical interpretations because of the lexical ambiguity from words "list" and "names."

```
Lexical Interpretations:
```

1)	N	Det	N	Prep	Det	N
2)	N	Det	V	Prep	Det	N
3)	V	Det	N	Prep	Det	N
4)	V	Det	V	Prep	Det	N

Sentence 2:

`Who is talking to the student in the car?`

In sentence 2, we could have two different syntactic interpretation because of the structural ambiguity caused by the preposition phrase.

1. The person expressed with "who" is talking within the car.
2. Only the student is in the car.

Sentence 3:

`Does the teacher give the boy the book which he borrowed from the library?`

In sentence 3, we also could have two interpretations because of the referential ambiguity caused by the pronoun.

1. The teacher borrowed the book from the library.

2. The boy borrowed the book from the library.

Sentence 4:

Who has gone to the bank?

In sentence 4, because of the semantic ambiguity, we could have two interpretations.

1. The bank is a financial institute.
2. The bank is a river bank.

In the above sample sentences, the lexical ambiguity can be resolved by listing all the possible lexical interpretations and using the grammar rules to check which interpretation is syntactically correct as discussed in section 2.3. For structural and referential ambiguities, very little syntactic and semantic information can be used. Therefore, the resolution of the structural and the referential ambiguity is to reference the world knowledge related to the application domain, or consult users in order to get more instructions. In sentence 2, if we define that "teacher possesses a car" as a world knowledge about the verb "possess" in the domain specific knowledge base, then the ambiguity in Sentence 2 can be solved because the agent of the verb "possess" is "teacher" not "student." Another approach for dealing with the structural and referential ambiguities is to adopt the principle of Right Association[1]. For example, in the second sentence listed above, the second interpretation is picked up because the "student" is the right-most referent for the preposition phrase. While in the third sentence, because the "boy" is the right-most referent for the pronoun "he," the second interpretation is considered as the result of the syntactic analysis. In this way, a unique parsing tree structure can be obtained. In addition to the principle of Right Association mentioned above, there are two more principles we can follow during the disambiguity processing. They are Minimal Attachment, and Lexical Preferences[1]. For more details, please refer to the original paper.

In semantic ambiguity, it could be partly resolved by combining the knowledge stored in the case frame for verbs with the selectional restrictions on what types of objects can fill those cases. For example, the word "bank" has at least two meanings as a noun. One is the bank of a river. The another is the financial institution. Therefore, they have two different semantic markers. For the bank of a river, a "physical-object" semantic marker can be assigned and for the financial institution, an "organization" semantic marker can be assigned. Thus, if we are talking about a bank transferring money from Montreal to Ottawa, the semantic marker "organization" is chosen because the verb "transfer" needs the "human" or "organization" semantic object as its agent. If a verb has more than one semantic meaning, then the resolution of the semantic ambiguity has to be dependent on the domain specific knowledge. Since in a specific application, most of the verbs have been already restricted to a small domain. In this domain, each verb has its unique meaning. Thus, once the meaning of a verb is fixed, the corresponding semantic roles around the verb can be determined accordingly.

For resolving the ambiguities listed above, different knowledge is required. From the viewpoint of system portability, it is desirable to divide these knowledge into two separate parts. One part is domain independent and the other part is domain dependent. As a result of this separation, the syntactic and semantic analyzers which access the general syntactic knowledge, such as syntactic class, and general semantic knowledge, such as semantic class, are able to be independent of the application domain. Therefore, the portability of the system can be enhanced. The techniques for representing different knowledge is discussed in the next chapter.

2.7 QUERY INTERPRETATION AND FORMAL QUERY LANGUAGE GENERATION

After intensive syntactic and semantic analyses, a general semantic representation can be obtained and therefore, the formal query language generation can start. For obtaining modularity, it is desirable to divide the query generation process into two separate parts. One part is the query interpretation and the another part is the formal query language construction. As a result of this separation, the query interpretation is able to be independent of the syntactic format of any specific query language. This requires a query meaning representation scheme which is query language oriented so that this representation can be easily used by the formal query language constructor. This query meaning representation should be as general as possible and contain all the information needed in the formal query language construction. Compared to the network and hierarchy data models used in database system, the relational data model does not need any navigation information in its query specification.

This makes the relational query language quite simple and straightforward. Based on the structure of relational query specification, the general information required in a relational database query can be classified into three sets: the attribute-set which contains all the attributes requested by the users, the relation-set which contains all the relations referenced in the query, and the qualification-set which contains all the selection constraints posed by the users. Due to the generality and the simplicity of these sets representation, it is reasonable to take these three sets as the query meaning representation.

The main task of a query interpreter is to map the general meaning representation obtained from the semantic analysis onto the query meaning representation. For example, suppose we have a query:

```
list the names of the students who are taking comp212.
```

After the syntactic and semantic analyses, the corresponding case frame based general semantic representation might be:

`main sentence:`

agent	action	neutral	modifier	dative	instrument	locative
nil	list	the names	the student	nil	nil	nil

`relative clause:`

agent	action	neutral	modifier	dative	instrument	locative
who	take	comp212	nil	nil	nil	nil

According to the above general meaning representation, we may find that the attribute-set is contained in the neutral case of the main sentence, and the qualification-set is contained in the cases of the relative clause or a locative case. As for the relation-set, it could be obtained by checking database schema after the attribute-set is established. At this point, several general questions arise:

1. Because the attribute name used in a natural language is not necessarily the same as the one used in a database schema, we need to define a correspondence mechanism;
2. An attribute name used in a natural language could refer to several attributes used in a database schema, such as name, which could refer to the teacher's name and the student's name. Thus, this type of ambiguity has to be considered;
3. In the relative clause, we also need to resolve the referential problem and unknown word.

The first and the second questions, generally, can be resolved by introducing a domain mapping table which establishes an one to one correspondence between the words used in a natural language and the ones used in a database schema. The third question can be resolved by using world knowledge about verbs as discussed in section 2.6. Furthermore, the join relation and the corresponding join qualifications have to be fixed in this stage. Since the solution for these problems is closely related to the different system design, a more detailed discussion is given in section 4.5.

The task of constructing specific formal query language from the query meaning representation described above is trivial. It simply involves putting the attribute-set, the relation-set, and the qualification-set into the format of a specific query language.

2.8 FORMAL QUERY LANGUAGE TRANSLATION

In multilevel interface system, the natural language interface produces a general formal query language. Since this general formal query language is independent of any specific database management system, a formal query language translator is required to translate the general formal query language into target formal query language. Under the support of this translator, the multilevel interface system can interface different database management system. Consequently, the protibility of the multilevel interface system is enhanced.

In the formal query language translation phase, the main work is to translate the query specification of one type of formal query language, say source language, into the query specification of another type of query language, say target language. In most cases, these two query specifications would not be the same. In the most situations likely some operations existing in a target query specification do not exist in a source query specification. Basically, there are no general criteria to handle this

```
lnj
    EMPLOYEE,
    lnj  WORKSIN, DEPTMNT;
  where  TOTSALARY > 30000  and
         DEPTNM = ''ARCHITECTURE''
  attrs  EMPID, FIRSTNAME, LASTNAME, TOTSALARY;
```

Figure 2.16: Example of GQML Query Language

kind of problem. It has to be treated case by case. However, if these two query specifications are working on the same database model, then there would be some similarities between them in terms of the syntax and semantics.

For illustrating the problem discussed above, we can take the problem of translating from SQL to GQML query specifications. In this example, both the source and the target language are formal and relational. However, GQML has a special natural join operation which SQL does not have. The basic format of natural join operation used in GQML is given below.

```
lnj    <relation1>, <relation2>

where  <predicate>

attrs  <attribute list>;
```

In the natural join operation, the thing which users need to do is to give two relations which need to be joined without join qualifications. Figure 2.16 gives an example of GQML query specification.

In this query specification, WORKSIN is a join relation. It associates the

```
SELECT    EMPID, FIRSTNAME, LASTNAME, TOTSALARY
FROM      EMPLOYEE, WORKSIN, DEPTMNT
WHERE     TOTSALARY > ''30000''  AND
          DEPTNM = ''ARCHITECTURE''  AND
          EMPLOYEE.EMPID = WORKIN.EMPID  AND
          DEPTMNT.DEPTID = WORKIN.DEPTID
```

Figure 2.17: Example of SQL Query Language

DEPTMNT relation with the EMPLOYEE relation and no join qualification is required in the WHERE clause. However, in SQL, the situation is different. Figure 2.17 gives an example of the same query.

In SQL query specification, the join qualifications have to be given explicitly but no special join operator is required. In SQL, the join operation is performed automatically by the SQL query processor. By comparing these two examples, we can find that even though SQL has no specific natural join operator, it contains enough information to construct corresponding GQML query specification. In the above example, we need to determine the join relation which links two separate relations. This can be done by analyzing the qualifications contained in the WHERE clause. Then, the join relation and the remaining relations can be used to construct natural join clause of GQML query specification. Since the similarity of these two query specification, the attribute-set of SQL is left untouched. This set can be taken as the attribute-set of GQML directly. The qualification-set of GQML can be constructed from remaining part of SQL qualification-set after removing those join qualifications.

In the formal query language translation phase, we need to deal with the different operations existing within two different formal query languages. Since both source and target query languages are formal, have quite simple syntax, well defined semantics as well as the syntactic and semantic similarity, the effort required in building a formal query language translator is relatively less than the one required in building a formal query language generator. In this way, even though the database management system is changed, the only thing we need to do is to rewrite a formal query language translator. As a result, the portability of a NLI system is enhanced.

From the discussion about each processing phase required in a natural language understanding system, we can find that at each processing phase, different knowledge is required. They are syntactic knowledge used in the syntactic analysis, semantic knowledge used in both syntactic and semantic analysis, and domain specific knowledge used in the query interpretation. For the knowledge being accessed conveniently by a NLI system, it needs to be organized in a knowledge base and represented with appropriate knowledge representation schemes. This issues are discussed in the next chapter.

Chapter 3

KNOWLEDGE REPRESENTATION

3.1 GENERAL ISSUES IN KNOWLEDGE REPRESENTATION

A natural language interface system, as discussed in the previous chapters, needs to reference several different kinds of information in its syntactic and semantic analyses. This information includes syntactic, semantic, and domain specific knowledge. How to organize and represent this knowledge is a central problem that arises in a natural language processing system. In the most general sense, a knowledge representation of a natural language is any framework in which information about the language and its words can be stored and retrieved[1]. This framework could be built with procedures or declarative statements.

If this framework is procedural, the knowledge is encoded into programs. Whenever the knowledge is required, a specific program is invoked. In practice, procedural knowledge can be used very effectively in well-defined domains but it often lacks generality and is difficult to maintain and upgrade. Furthermore, in procedural systems, a prescribed order of accessing different pieces of knowledge has to be given. If a piece of knowledge needs to be changed, then the corresponding program which contains that knowledge must also be rewritten.

If the knowledge is declarative, then the knowledge component of a NLI system is independent of the programs which access it and could also be independent of the other knowledge used in the NLI system. Compared to the procedural approach, declarative knowledge can be more easily defined and used more conveniently. Information can be removed, added, or modified without affecting the other knowledge components or the program mechanism. System functions may use some piece of knowledge to determine what piece of information is best utilized to solve the problem at hand. In this case, no prescribed processing order needs be defined.

Comparing the two approaches discussed above, the first one seems more flexible and perhaps more efficient while the second one is more clear, rigorously defined, and easily modified. The second approach seems more advantageous to the natural language understanding system because this one can make the NLI system more portable. However, it is difficult for the declarative approach to represent some exceptional cases. This leads to a third knowledge representation scheme which is the combination of the first two approaches. In this third approach, main pieces of knowledge are represented declaratively and the exceptional cases are handled by procedures.

In this chapter, several kinds of knowledge required for a NLI system and its representation schemes are discussed. We do not intend to give a survey of the major research projects in knowledge representation, nor to describe the work of individual theories in detail. Instead, a brief review and analysis of some main work done by researchers in each stage of the natural language understanding process is attempted.

Since some existing knowledge representation schemes are inadequate or unsuitable for a NLI system, several ideas and approaches for enhancing these existing knowledge representation schemes are provided as well.

3.2 SYNTACTIC KNOWLEDGE AND ITS REPRESENTATION

3.2.1 Lexicon

The first task given to the syntactic analyzer is to provide an associativity between each word in an input sentence and its syntactic and semantic definitions. Thus, we need to construct a framework in which those definitions about each word used in a natural language can be stored and retrieved. Such a framework can be called a dictionary or lexicon. It is the major knowledge resource used by a natural language understanding system.

Generally, a lexicon consists of a set of lexical entries. Each word used in a natural language for posing query has, at least, one entry. Each entry of the lexicon contains a syntactic portion and a semantic portion as described in section 2.2. For building up such a lexicon, an appropriate knowledge representation scheme has to be selected. There are several lexical features that can help us to judge which approach is the most appropriate.

1. User Friendliness: The lexicon is a knowledge exchange interface between a computerized natural language understanding system and humans. At this interface, humans provide the knowledge about a natural language to

```
{
   word entry:          student;
   syntactic portion:   class:common_noun,  number:singular,
                        person:third,  case:subjective;
   semantic portion:    semantics:human
}
```

Figure 3.1: Example of Declarative Lexicon Entry

the system. Thus, this interface should be user-oriented to give humans a natural and convenient way to express their knowledge.

2. Lexicon Extensibility: Because of the possibility of changes in application domains and/or the conceptual reorganization of the database[17], the lexicon needs to be extensible.

3. Domain Independence: Since the application domain may change, the domain independence is desirable.

Given the above criteria for a lexicon, a declarative approach to lexical knowledge representation seems more suitable than the others. Based on the syntactic structure and the semantic structure described in section 2.2, a possible declarative lexicon entry for the word "student" is given in Figure 3.1.

The declarative notation given in Figure 3.1 satisfies the above three characteristic requirements. In terms of both syntax and semantics, it is very clear and convenient for the users to define and modify their lexicon.

From the viewpoint of the system space efficiency, however, it is difficult for a NLI system to accept this word-string notation as an internal representation of the lexical knowledge. This is because there is a high demand on memory space. It would be desirable to keep this form as the external representation used by the users to define their language knowledge, because of its convenience and clarity. A possible solution for this problem is to introduce an internal notation used by the computer system that differs from the external one. This notation should be more efficient than the external one in terms of the space. A bit-string representation seems to be a good candidate which can highly compress the information contained in the word-string into several bytes. To enable this, an interpreter has to be established to perform the transformation between these two notations.

From the lexicon entry example of Figure 3.1, we see that each word may have several attributes in its syntactic portion and several in its semantic portion. In addition, each attribute has several possible values. For example, the word "student" has the attributes "class," "number," "person," "case," and "semantics." For the attribute "number," it has the attribute value "first," "second," or "third." To map a word-string onto a bit-string, we have to cut the bit-string into several small sections, each of which contains several bits. Each section is assigned to an attribute used in the word-string notation and a bit-combination in each section represents a corresponding attribute value. This results in different notations for the knowledge representation. In this case, the system users can make use of the word-string notation to define their lexicon. The system can make use of the bit-string notation to improve the efficiency of the system memory space. Thus, these two notations benefit the users and the system functions, respectively.

Under the above knowledge representation scheme, a remaining issue is the systematic mapping between a word-string and a bit-string. If we recall the syntactic and the semantic classes described in section 2.2, we find that the two classes given in Figure 2.3 and Figure 2.4 show us how the syntactic and semantic definitions are organized. In fact, these two classes provide the guidelines we need during the transformation between the two notations. Because these guidelines are the knowledge about how to define the lexicon knowledge, they can be called metaknowledge. It is also desirable for this metaknowledge to be defined declaratively for the same reasons as the knowledge itself. In the declarative definition of the metaknowledge, the tree structure described in Figure 2.3 and Figure 2.4 can be retained. In such a declarative tree structure each node, representing an attribute of the syntactic or the semantic class, has a value which corresponds to a bit-string notation. The system performs its mapping based on the word-string notation used in the lexicon definition. The interpreter searches the syntactic or the semantic tree structure to find an attribute match. Once a match is found, the corresponding bit-string notation is obtained. As a result, the word-string can be compressed into several bytes. This approach is extremely effective for information compression.

After a word-string notation is transformed into a bit-string notation, another problem arises: it is difficult for the system functions to recognize the bit-string notation directly. In this case, during the syntactic or semantic analyses, the system functions need to know the exact meaning of all the possible bit-combinations when they determine the syntactic and semantic classifications for recognizing a word. If a word-string notation could be used directly by the system functions, the system programming and the system maintenance would be eased. Thus, it is desirable for the system functions to use the same notation of the knowledge representation as the one used by human users. This requires the interpreter to reverse the internal notation back into the external notation. In this way, the internal notation is transparent to both the users and the system functions. The price of obtaining the advantages of the space efficiency and the interface friendliness is the processing time for the compression and the expansion.

3.2.2 Syntactic Class

In the syntactic analysis, the knowledge about the syntactic categories for each word is required. This knowledge is expressed with syntactic markers, such as noun, verb, adverb, etc. and contained in the lexicon. As described in Figure 2.3, the syntactic markers are organized in a syntactic class which presents a tree structure. Such a syntactic class needs to be represented with an appropriate knowledge representation scheme so that the knowledge about the syntactic categories can be used by a NLI system conveniently.

Based on the discussion in the previous section, we can see that the only operation we need on the syntactic class is to search the syntactic class tree structure. Thus, the syntactic class can be defined as a common tree structure declaratively. Figure 3.2 illustrates how a noun syntactic class is organized. The completed syntactic class definition used in the MIDBMS system is given in Appendix A.

In addition, the modality information about verbs is also required in a NLI

```
                                   ---- Common_Noun
                -------- Genus ----|---- Proper_Noun
               |                   ---- Pronoun
               |
               |                   ---- Class_Noun
               |-------- Category--|---- Collective_Noun
               |                   |---- Material_Noun
               |                   ---- Abstract_Noun
Noun_Class ----|
               |                   ---- Nominative_Case
               |-------- Case -----|---- Objective_Case
               |                   |---- Possessive_Case
               |                   ---- Reflexive_Case
               |
               |                   ---- Masculine
               |-------- Gender ---|---- Feminine
               |                   ---- Neuter
               |
               |                   ---- Plural
               |-------- Number ---|---- Singular
               |                   ---- Uncountable
               |
               |                   ---- First
                -------- Person ---|---- Second
                                   ---- Third
```

Figure 3.2: Example of Syntactic Noun Class

```
                                ---- Past
             ------- Tense ----|---- Present
            |                   ---- Future
            |
            |                   ---- Simple
            |------- Aspect ---|---- Perfect
            |                  |---- Continuing
            |                   ---- Perfect_Continuing
            |
            |                   ---- Active
Modality ---|------- Voice ----|
            |                   ---- Passive
            |
            |                   ---- Positive
            |------- Negation -|
            |                   ---- Negative
            |
            |                   ---- First
            |------- Person ---|---- Second
            |                   ---- Third
            |
            |                   ---- Singular
             ------- Number ---|
                                ---- Plural
```

Figure 3.3: Structure of Modality

system. This information is collected during the syntactic analysis and used during the semantic analysis. Figure 3.3 gives the structure of the modality under a tree structure which can also be defined declaratively.

3.2.3 Context-Free Grammar

In the syntactic processing stage, grammar is another form of syntactic knowledge required by the syntactic analyzer. A grammar gives the basic syntactic structure of natural language sentences acceptable to the syntactic analyzer. Grammars consisting of rules of the form "<symbol> —> <symbol>1 ... <symbol>n," for n>=1, are called context-free grammars[1]. The context-free grammars are a very important class of grammars for two reasons. The formalism is powerful enough to be able to describe most of the structures in natural languages, and yet it is restrictive enough so that efficient parsers can be built to analyze sentences[1]. A widely accepted notation for representing a grammar is a transition network[44][45]. A transition network is based on the application of the mathematical notions of graph theory and the finite state machine to the study of grammars. Generally, there are several types of transition networks. The simplest one is the finite state transition network.

A finite state transition network is a directed-graph consisting of a set of states connected by arcs. Each arc represents a transition between two states. The states are expressed by circles and the arcs by lines. Each arc is ended with an arrowhead which indicates the direction of the transformation between two states. A transition network can be considered as a pattern for recognizing or generating sequences of words or phrases if sub-graphs are used. In both ge. erating and recognizing, the process should follow the form of the net in a step by step manner. Each transition along an arc corresponds to a single word or phrase in a sequence. The process starts from an initial state and ends in a terminal state. A simple grammar and its corresponding finite state transition network is given in Figure 3.4. In the transition network given in Figure 3.4, the initial state is S0 and the terminal state is Sn. It can parse sentences, such as "The boy is eating an apple."

Based on this technique, more complicated transition networks may be developed. Examples of this are the recursive transition network and the augmented tran-

```
Phrase Structure Grammar:
            S   ---> NP  VP  NP
            NP  ---> [Art]  N
            VP  ---> Aux  V
            Art --> the | an
            N ----> boy | apple
            Aux --> is
            V ----> eating
```

Finite State Transition Network:

$$\begin{tikzcd}
(S0) \arrow[r, "Art"] \arrow[rr, bend right, "N"'] & (S1) \arrow[r, "N"] & (S2) \arrow[r, "Aux"] & (S3) \arrow[r, "V"] & (S4) \arrow[r, "Art"] \arrow[rr, bend right, "N"'] & (S5) \arrow[r, "N"] & (Sn)
\end{tikzcd}$$

Figure 3.4: Example of Simple Phrase Structure Grammar

sition network[44]. According to overall system design, different types of transition networks can be selected as the grammar knowledge representation scheme and the parsing mechanism. It is also desirable to define the transition network declaratively to enable the grammar modification. In the example given in Figure 3.4, the finite state transition network can only prove if a sentence is syntactically valid. It cannot produce any analytical syntactic structure. The augmented transition network can solve this problem by augmenting a set of registers which reserve the analytical syntactic structures for a sentence. However, if a combination parsing strategy of bottom-up and top-down is used in the syntactic analysis, then the analytical syntactic structures of phrases can be produced in the bottom-up stage. As well, the finite state transition network can be used to parse these phrases in the top-down stage. In this case, if the grammatical markers, such as subject, object, etc., are needed, then the finite state transition network described in Figure 3.4 has to be augmented. For the reason discussed above, we augment the finite state transition network by introducing an action component into the transition network. In such a augmented finite state transition network, each transition state is associated with an action. During the grammar checking stage, the transition network is traversed. In this traversal, when a new state is reached, a corresponding action is carried out so that a grammatical marker can be produced. For example, in the demonstrative augmented finite state transition network given in Figure 3.5, if the control goes to the state1 from the state0, then the grammatical marker "subject" can be assigned to the current noun phrase. If the control goes to the state2 from the state1, then the grammatical marker "predicate_verb" can be assigned to the current verb phrase. And if the control goes to the state3 from the state2, then the "object" can be assigned to the last noun phrase. This simplifies the semantic analysis without need of much effort of syntactic analysis. The corresponding declarative definition of the augmented finite state transition network is also given in Figure 3.5. In the example given in Figure 3.5, the first column is the start state of a transition; the second column is the condition of the transition; the third column is the end state of a transition; and the forth column is the action component associated with each transition state. The S0 is the start state of the grammar and the Sn is the terminal state of the grammar. Based on this notation, a grammar checker which makes use of the grammar knowledge can check the sentence with the string matching mechanism. This provides a possibility of building a table driven grammar checker. The declarative grammar definition used in the MIDBMS is given in Appendix B.

```
Augmented Finite State Transition Network:

                     the boy      is eating      an apple
 phrase marker:         NP            VP            NP
                (s0)--------->(s1)----------->(s2)--------->(Sn)
                                |               |             |
                                |               |             |
 grammatical marker:       (subject)  (predicate_verb)  (object)
```

Declarative Definition:

Start_state:	Condition:	End_state:	Action:
S0,	NP,	S1,	Subject;
S1,	VP,	S2,	Predicate_Verb;
S2,	NP,	Sn,	Object.

Figure 3.5: An Augmented Finite State Transition Network with Grammatical Markers

3.3 SEMANTIC KNOWLEDGE AND ITS REPRESENTATION

3.3.1 Case Grammar

Case Grammar, discussed in section 2.5, is defined with a set of rewrite rules which are used to convert the surface structure of a natural language sentence into its deep structure. In this definition, the deep structure of a sentence consists of a modality M and a preposition P. The modality M contains some information about verbs. They are tense, aspect, negation, and mood. Preposition P consists of a verb and several cases associated with the verb. From this definition, we can see that verb is the major source of Case Grammar. This makes Case Grammar more verb-oriented. Figure 3.6 gives an example of the Case Grammar graph representation for the sentence "The boy is eating an apple."

```
            ---------(SENT)----------
            |                       |
            |                       |
          MODL                ---(PROP)------
 -----------------            |             |
 Tense:  Present              |             |
 Aspect:Simple             VERB: eat      (CARG1)----------
 Form:   Progressive      ------------      |                |
 Mood:   Declarative      Agent:   Req      |                |
                          Inst.:   Non    CaseType: Agent   (CRG2)
                          Dat. :   Non    ---------------    |
                          Neutral:Req      NounPhrase:       |
                          Loc. :   Opt     Det:   the     CaseType: Neutral
                                           Noun:  boy     -----------------
                                                             NounPhrase:
                                                             Det:  an
                                                             Noun: apple
```

Figure 3.6: Example of Case Grammar Graph Representation

From this example, we find that each case corresponds to a phrase structure. These syntactic phrase structure can be produced by a syntactic analyzer. However, the relationships between the cases and the phrases need to be established by applying a set of rewrite rules as described in section 2.5. These rules convert the syntactic roles in the surface structure of a sentence into the corresponding semantic roles represented in its deep structure. The rules used in the MIDBMS are represented with a procedure and are given in the next chapter. In addition, the knowledge about each verb needs to be represented as well. This knowledge includes which cases are required, optional, or not allowed. As described in section 2.5, it can be represented with a case frame specification. Figure 3.7 gives a possible example of the case frame specification for verb "teach."

Based on such a representation structure, semantic analysis can be carried out. Once the cases are filled with the corresponding semantic roles according to a case frame specification for verb ''teach'':

agent	action	instrumental	dative	neutral	locative
optional	teach	not-required	optional	required	optional

Figure 3.7: Example of Case Frame Specification for Verb "teach"

specific sentence, a deep meaning representation of a sentence is established. This deep meaning representation is the output of the semantic analysis.

3.3.2 Semantic Class

For performing the syntactic and semantic analyses, the semantic knowledge about word senses is required. This knowledge can be expressed with semantic markers, such as physical_object, abstract_object, etc. These semantic markers are used to define the words used in a natural language as a semantic portion of the lexicon definition. Thus, the knowledge about word senses can be used by a NLI system conveniently. As described in Figure 2.4, these semantic markers are organized in a semantic class which presents a tree structure. With the semantic class knowledge, a natural language understanding system is able to resolve the ambiguity of word senses and the conjunction problem as discussed in section 2.6.

Based on the above discussion, we need to find an appropriate knowledge representation scheme to represent the semantic class. For this purpose, a semantic network is introduced in this section. In most of the research of NLI system, the semantic network is employeed to express the general semantic knowledge[1][13][25].

```
                          ALL
                       /       \
                     /           \
                   /               \
                 /                   \
          PHYSOBJ                     ABSTRACT
          /       \                  /         \
        /           \              /             \
  NON_ANIMATE        ANIMATE    TIME            LOCATION
   /        \        /     \
 /            \    /         \
NON_LIVING  VEGETABLE  DOG      PERSON
```

Figure 3.8: A Semantic Network Describing Subtype Relationship (from [1] p208)

This knowledge includes the information about word senses and the relationships between these senses. In this thesis, a semantic network is employed to represent the semantic class. The semantic compatibility operation based on the semantic compatibility criteria discussed in section 2.3 is defined on the semantic network.

Generally, a semantic network is a graph structure with labeled links between labeled nodes. The word sense is represented by node and the relationship between the nodes is represented by links. In semantic network, two types of hierarchy can be represented. One is the IS-A hierarchy which defines the subtype relationship and the other is the HAS-A hierarchy which defines the subpart relationship[13]. Under such a hierarchy, all the subpart attributes which a supertype object has can be inherited by its subtype objects. In this knowledge representation scheme of the semantic network, the semantic class can be represented naturally. Figure 3.8 describes a possible structure of the subtype relationships contained in the semantic class.

Figure 3.9 describes a possible structure of the semantic network describing the subpart relationship for ANIMATE object. In this semantic network, each node

```
           |     HAS-A
        ANIMATE ---------(NAME, GENDER, ...)
       /        \
     /           \ IS-A
   /              \      HAS-A
...              PERSON ---------(ID, ADDRESS, ...)
                  /    \
                /   ...  \
             ...          ...
```

Figure 3.9: A Semantic Network Describing Subpart Relationship

represents a semantic object. Each semantic object could have several attributes. All the attributes of a semantic object can be inherited by its subpart semantic objects.

The semantic network can be defined declaratively. The corresponding declarative definition of the semantic class used in the MIDBMS system is given in Appendix C.

3.4 DOMAIN DEPENDENT KNOWLEDGE AND ITS REPRESENTATION

The knowledge representations we discussed so far are, to some degree, domain independent. After semantic analysis, a general semantic representation can be obtained. However, this general semantic representation has to be further interpreted to produce a query meaning representation as discussed in Chapter 2. The purpose of establishing query meaning representation is to ease the formal query language generation. This interpretation processing is closely related to the database domain. Thus, domain dependent knowledge is required. By accessing the domain specific knowledge, the interpreter is able to fill up the gap between the general semantic representation and the formal query language.

From the viewpoint of a database system, considerable efforts have been made in capturing the semantic content of the database system with conceptual modeling tools. The most influential work are Entity Relationship model[8][9] and the Semantic Data Model[24]. From the viewpoint of AI knowledge representation, there also exist several formalisms, such as Logic[2], Procedural Representation[43], Frames[34], and Semantic Network[40]. Comparing these different knowledge representation schemes, it seems to us that E-R model is the most natural and understandable scheme for representing database schema information. It is data independent, and represents the view of the application environment. It also can be mapped to three database models: the hierarchical, the network, and the relational model. The drawback of the E-R model is its lack of semantic content. In the E-R model, the relationships between attribute and relation as well as the ones between the relation and the relation are defined explicitly. However, this definition just shows whether the relationship exist or not. It fails to show what kind of relationship it is. In other words, the semantic portion of the E-R model is missing. Based on this observation, we assign an explicit semantic meaning to each relationship for enhancing the semantic content of the E-R model. Figure 3.10 gives an example of extended E-R model. In this example, each relation is assigned to a semantic object. The join relation "stud-cou" is assigned to the action "take." The join relation "teac-cou" is assigned to the action "teach." The semantic objects "STUDENT," "COURSE," and "TEACHER" are connected with these two actions. Thus, this extended E-R model can be viewed as three levels,

```
Semantics
Level:
       STUDENT -----> TAKE -----> COURSE <---- TEACH <----- TEACHER
          |                          |                          |
          |                          |                          |
Relation  |                          |                          |
Level:    |                          |                          |
     ---------   ----------   --------   ----------   ---------
    | student |--| stud-cou |--| course |--| teac-cou |--| teacher |
     ---------   ----------   --------   ----------   ---------
          |           |           |           |           |
          |           |           |           |           |
Attribute |                                               |
Level:    |                                               |
        (      .........      attributes       ........    )
```

Figure 3.10: Extended E-R Model

Relation_Name:	Attribute_Name:	Feature:
"STUDENT",	"STUDENTID",	"key",
"STUDENT",	"STUDENTNM",	"default",
"COURSE",	"COURSEID",	"key",
"COURSE",	"COURSEID",	"default",
"COURSE",	"COURSENM",	"",
"STUDENT_COURSE",	"STUDENTID",	"key",
"STUDENT_COURSE",	"COURSEID",	"key".

Figure 3.11: Example of Database Schema

which are attribute-level, relation-level, and semantics-level.

The attribute-level and the relation-level can be represented with a table which is more database schema oriented. Figure 3.11 shows a possible database schema representation.

For the semantics-level, case frame is an ideal representation scheme. For the purpose of portability, it is reasonable to separate the knowledge about verbs into

```
Case Frame Specification for Verb ''teach'':
    agent     action    instrumental    dative    neutral   locative
    ------------------------------------------------------------------
Domain Independent Portion:
   optional   teach     not-required   optional  required  optional

Domain Dependent Portion:
   teacher    teach         null          null     course  university
```

Figure 3.12: Example of Separated Case Frame

two parts. One part is relatively domain independent as described in the previous section and the other part is domain dependent. Figure 3.12 shows an example of such a case frame based knowledge representation for verb "teach."

By making use of the domain dependent part of the case frame, some degree of inference ability can be obtained. For example, let us consider the following sentence:

```
Who is teaching comp212?
```

In this sentence, the action is "teach." Based on the domain dependent portion of the case frame, we can find that the agent of the verb "teach" is "teacher" and the meaning of the sentence is "which teacher is teaching course comp212?" Therefore, the agent case can be filled up with "teacher" and the neutral case can be filled up with "course comp212." Similar techniques are introduced and employed in [1][29].

Since one noun phrase used in a natural language query could imply several attributes used in a database schema, an ambiguity from the mapping between the

Noun:	Entity:	Attribute:
name	teacher	TEACHERNM
name	student	STUDENTNM
age	student	STUDENTAGE
student	student	STUDENTNM
		STUDENTAGE

Figure 3.13: Example of Word Mapping Table

general semantic representation and the query meaning representation arises. Basically, this ambiguity can be resolved by providing a mapping table. This mapping establishes all the possible references between the noun phrases used in natural language queries and the names of the attributes and the relations used in a database schema. A possible representation for such a mapping table is given in Figure 3.13. Two declarative domain knowledge bases used in the MIDBMS are given in Appendix D and Appendix E.

3.5 FINAL MEANING REPRESENTATION

Some of the research about natural language interface to database query system choose logic representation as the final meaning representation, such as relational calculus, logical form, etc. In these systems, the query language generator needs to be built up on the basis of the logic representation. From the viewpoint of the system construction, this logical form is not convenient for a formal query language generator to interface the semantic analyzer of a NLI system. From the viewpoint of the system use, this logical form cannot be used directly by the users to express their query requirements. Therefore, if we can find a final meaning representation which is able to replace the logical form representation and also can be used by the users directly, this would benefit both the NLI system and the users. SQL is a relatively standard query language. It has very clear syntactic structure and very powerful semantic expressing ability. Thus, it is reasonable to choose SQL as the final meaning representation of a natural language interface to database query system. As a result, such an interface system is able to provide users with one more window in which they can express their query. Because both of SQL and any other relational query language are formal languages and they have clear syntactic and semantic definitions, the task of the translation between two formal languages is fairly trivial. Based on this observation, SQL is selected as the final meaning representation of the MIDBMS system.

In this chapter, several different kinds of knowledge required in the each stage of the natural language understanding process are discussed. The corresponding knowledge representation schemes are examined. Since the finite state transition network is inadequate for the syntactic analysis in a NLI system, we provide a augmented finite state transition network for syntactic analysis. This augmented finite state transition network, based on the syntactic analysis, produces the corresponding syntactic structure with phrase markers and grammatical markers for a sentence. Such a syntactic structure benefits the semantic analysis. In addition, since the E-R model used to express database domain structure in a database system lacks semantic content we enhance the E-R model. This enhancement provides the database domain semantics. Such an enhanced E-R model is used to perform inferences during converting a general semantic representation into a query meaning representation. Both of these modified approaches are adopted in the MIDBMS prototype system design and implementation.

Based on the discussion about knowledge representation schemes used in a NLI system, an overall design of MIDBMS is given in Chapter 4. In this system design, the bottom-up and top-down parsing mechanism is adopted for syntactic analysis. The Case Grammar is adopted for semantic analysis. Both mechanisms of the syntactic and semantic analyses are effectively intergrated into one NLI system. As the result of the syntactic and semantic analyses, a general meaning representation of a database query is produced. For interpreting the general meaning representation and producing a query oriented meaning representation, a query interpreter is constructed. Finally, a formal query language translator is established for producing query specifications. For making the system transportable and maintainable, the principle of modularity is followed throughout the system design. The details design of the system is given in next chapter.

Chapter 4

MIDBMS SYSTEM DESIGN AND IMPLEMENTATION

4.1 OVERALL SYSTEM DESIGN

A prototype system called MIDBMS is designed and implemented for testing the proposal made in this thesis. The MIDBMS system consists of eight procedural modules, which are the System Initiator, the Sentence Reader, the Lexical Analyzer, the Syntactic Analyzer, the Semantic Analyzer, the Query Interpreter, the SQL Generator, and the GQML Translator. Figure 4.1 shows the architecture of the MIDBMS system. The function of each of these modules with more detailed design are given in the following sections. Section 4.2 discusses the lexicon definition; section 4.3 discusses the grammar definition; section 4.4 discusses the syntactic analysis; section 4.5 discusses semantic analysis; section 4.6 discusses the query interpretation; section 4.7 discusses the SQL generation; and section 4.7 discusses the GQML translation.

1. System Initiator initiates the system, reads in the lexicon definition and compresses it into bit-string representation.

2. Sentence Reader reads in a sentence from either a system terminal or a specified file.

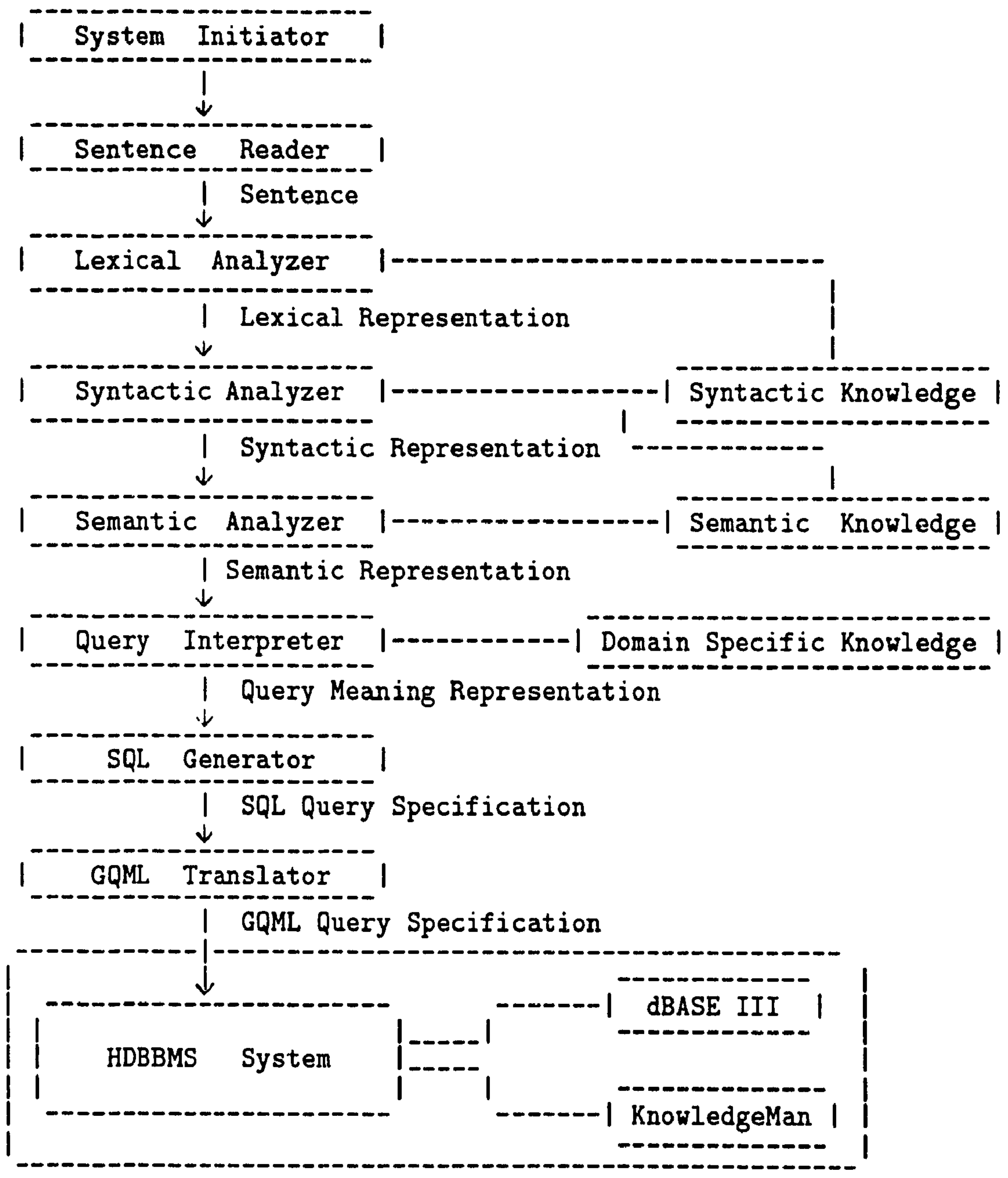

Figure 4.1: Architecture of the MIDBMS

3. Lexical Analyzer accepts a sentence, finds all the lexical entries for each word in the sentence, and constructs all its possible lexical interpretations.

4. Syntactic Analyzer accepts a lexical interpretation, performs the syntactic analysis, and produces a syntactic tree structure if the analysis succeeds. If the sentence fails to be parsed, then the syntactic analyzer asks for next lexical interpretation to process until a suitable lexical interpretation is successfully parsed or no more lexical interpretation can be obtained. If a sentence is parsed, then a corresponding parse tree is produced. This parse tree is the syntactic representation of the sentence. This analyzer accesses the syntactic class knowledge and semantic class knowledge. The parsing strategy is the mixture of Bottom-Up and Top-Down. In the Bottom-Up parsing stage, all the phrases are grouped up. In the Top-Down stage, the parser checks if these phrases are in good order in terms of grammar.

5. Semantic Analyzer accepts the syntactic representation, performs semantic analysis, and produces a semantic representation if the semantic analysis succeeds. If the semantic analysis fails, then the semantic analyzer asks for next syntactic representation to process until a suitable syntactic representation is processed successfully or no more syntactic representation can be obtained. This analyzer accesses the case frame based knowledge about verbs and some procedural semantic rules. In semantic analysis, Case Grammar[6][20] is used to represent the deep structure of a sentence. This deep meaning representation is more natural language oriented.

6. Query interpreter accepts the semantic representation, performs the query interpretation, and produces a query meaning representation if the interpretation succeeds. If the semantic representation fails to be interpreted correctly, the interpreter asks for next semantic representation to process until a suitable semantic representation is interpreted correctly or no more semantic representation can be obtained. The interpreter accesses the domain dependent knowledge and some procedural rules. The query meaning representation is defined with Attribute-Set, Relation-Set, and Qualification-Set. This representation is more query language oriented.
7. SQL Generator accepts the query meaning representation and constructs the SQL query specification.
8. GQML Translator accepts the SQL query specification and translates it into the GQML query specification. The GQML query specification is submitted to the HDDBMS system for obtaining the final database results.

4.2 LEXICON DEFINITION

The lexicon is the most important resource in the natural language understanding system. It must contain all the vocabulary which the system needs and all its necessary syntactic and semantic information for each lexical item to be processed. In the MIDBMS system design, each lexical item is defined with its syntactic portion and semantic portion as described in section 3.2. The syntactic portion contains the syntactic categories of a word, such as noun, verb etc., and the corresponding features, such as person, number, gender etc. The semantic portion contains the semantic category of a word, such as animate, abstract-object, etc. Example 1 shows the lexicon entry for the word "students." In this example, the "class," "genus," "gender," "number," "person," "root," and "semantics" are key words defined in the system syntactic class. The "noun," "common_noun," "neuter," "plural," and "third" are the syntactic categories and features defined in the system syntactic class. The "animate" is the semantic category defined in the system semantic class. The system syntactic class definition and the semantic class definition are given in Appendix A and C.

```
Example 1:

      students (class:noun  genus:common_noun  gender:neuter
                number:plural  person:third  root:student
                semantics:human)
```

Some words have several different syntactic or semantic categories. This can be expressed by giving several lexicon entries as demonstrated in Example 2. In this example, the word "lists" can be a noun or a verb.

```
Example 2:

      lists (class:noun  genus:common_noun  gender:neuter
             number:plural  person:third  root:list
semantics:abs_obj;
class:verb  genus:vt  form:present  person:third
number:singular  root:list)
```

Further, some words have the same syntactic category but have several different syntactic features. For example, the word "study" is a verb. It can be a transitive verb or an intransitive verb. In this case, the lexical entry can be defined with logical AND (&) operator. Example 3 illustrates such a definition.

Example 3:

```
study (class:noun  genus:common_noun  gender:neuter
       number:singular  person:third  root:study
       semantics:abs_obj;
       class:verb  genus:vt&vi  form:infinitive
       person:first&second  number:singular&plural
       root:study;
       class:verb  genus:vt&vi  form:infinitive
       person:third  number:plural  root:study)
```

The biggest advantage of this style of lexicon definition is to provide users a very clear and intuitive interface to define their own lexicon. However, in terms of system space efficiency, this lexicon definition is really expensive. The solution of this problem is to compress the word-string definition into a bit-string definition.

The compression approach discussed in section 3.2 is implemented in MIDBMS. This compression process is done by a system function called the Lexicon Builder during the system initiation stage. For offering the system functions a convenient handle to access the syntactic and semantic information, the restoration from the bit-string to a word-string is desirable. In MIDBMS, whenever a system function wants to check if a word has an expected syntactic or semantic category, it just needs to call a predicate by giving a word-string notation of the expected syntactic definition and the bit-string notation of a specified word. The predicate checks if these two representations have the same syntactic or semantic definition. If they have the same syntactic or semantic definition, then a true value is returned. Otherwise, a false value is returned. Let us suppose the system wants to know if a word contains the following syntactic and semantic categories and syntactic features: "common noun," "singular," and "animate." In this case, a system function called the syntax_checker is invoked and the corresponding parameters are given as follows.

```
syntax_checker("class:noun genus:common number:singular
                  semantics:human",
                  syntactic_bit_string, semantic_bit_string)
```

where the syntactic_bit_string and the semantic_bit_string come from the lexicon entıy of the specified word.

Generally, the mechanism of the restoration provides the system functions with a convenient handle to access the syntactic and the semantic knowledge. It also makes the system more maintainable. The price of obtaining these benefits is to spend more time on the mapping operation.

Besides, the feature of the multiple lexical definition makes the lexicon more domain independent. Since even though the domain is changed, the only thing users need to do is to add some new definitions into the lexicon instead of changing it. When the lexicon contains enough vocabulary, then it can reach a relatively stable state.

For supporting semantic analysis, case frame specification for each verb is required. In MIDBMS, Fillmore's original cases are used and the domain independent portion of the case frame specification is put into the lexicon because of its feature of the domain independency as discussed in section 3.4. A domain independent portion of the case frame specification for verb "study" is listed below.

```
study (agent:req instrumental:non locative:opt
       dative:non neutral:req)
```

This definition is quite abstract. It contains the most basic requirements for the semantic roles related to a verb. In most of the different database domains, the case specifications for each verb are almost the same. This is so since the meaning of most of the verbs does not change along with the change of the application domains. Therefore, this case frame specification can be viewed to some degree as a domain independent knowledge.

4.3 GRAMMAR DEFINITION

Imperative sentences and interrogative sentences are acceptable in MIDBMS. The corresponding Phrase Structure Grammar used in this system is defined in Figure 4.2.

In the syntactic analysis, the bottom-up strategy is applied first to build simple phrase structures. This greatly reduces the burden of the top-down analysis and also simplifies the construction of the transition network employed for representing the grammar as described in Chapter 3. In this system, a finite state transition network is employed and augmented with action components as described in section 3.2.3. In this way, phrase structures produced in the bottom-up processing phase can be parsed and a grammatical marker can be attached to each of these phrases. This means that once a certain state of the transition network is reached, the defined action is executed and a corresponding grammatical marker can be produced for a phrase. The corresponding transition network definition is given in Appendix B. This definition is declarative but at the action part, the transition network invokes some procedures to perform the defined actions. This is an application of the combination of the different knowledge representation schemes as described in section 3.1.

```
S------------------>  Imperative | Interrogative
Imperative --------->  VP + NP + Rest
Imperative --------->  VP + P_Pronoun_Obj + NP + Rest
Imperative --------->  VP + P_Pronoun_Obj + Rel_Clause
Interrogative ------>  Rel_Clause1 | Rel_Clause2
Rest --------------->  (Conj + NP)* + Modifier + Rest
Rest --------------->  Nil
Modifier ----------->  PrepP | Rel_Clause
PrepP -------------->  Prep + NP + Rest
Rel_Clause --------->  Rel_Clause1 | Rel_Clause2 |
                       Rel_Clause3 | Rel_Clause4
Rel_Clause1 -------->  Q_R_Pronoun + VP + NP + Rest
Rel_Clause2 -------->  Q_R_Pronoun + NP + VP + Rest
Rel_Clause3 -------->  Q_R_Pronoun + P_Pronoun_Obj + VP + Rest
Rel_Clause4 -------->  Q_R_Pronoun + NP + Be + (Neg) +
                       Rel_Operation + NP
Rel_Operation ------>  Rel_Operator + (Or + Rel_Operator)
Rel_Operator ------->  more than | less than | equal to
Q_R_Pronoun -------->  who | whom | whose | which
NP ----------------->  (det) + (adj)* + (noun)* + noun
NP ----------------->  (P_Pronoun_Poss) + (noun)* + noun
NP ----------------->  P_Pronoun_Sub
VP ----------------->  (Aux) + (Neg) + V
VP ----------------->  Be + (Neg)
V ------------------>  Vt | Vi | Be
```

Figure 4.2: Phrase Structure Grammar Definition

4.4 SYNTACTIC ANALYSIS

4.4.1 Lexical Analysis

The task of the lexical analyzer is to accept a sentence from the Sentence Reader, find the lexicon entries for all the words in the sentence, and construct a set of lexical interpretations. In addition, some pre-processing is carried out in this stage to simplify further syntactic analysis. Some problems which are encountered in this processing stage are discussed below.

Once the syntactic analyzer receives a sentence, it needs to search the lexicon and find all the possible lexicon entries for each word in the sentence. In database queries, attribute values can appear in a query sentence. For example, "List the names of the students taking comp212." Here, "comp212" is an attribute value for course code in a student relation. If we set up an entry for such an unknown word in a lexicon, we would be forced to put all the attribute values into the lexicon. In an extreme case, the database would be duplicated[1]. Therefore, in MIDBMS, we assume that the database attribute values are treated as unknown words. These unknown words are considered nouns. If an unknown word is encountered, a special marker is assigned to the unknown word and it is left to be processed in the syntactic analysis phase and again in the query interpretation phase.

In natural language, one word can have several syntactic or semantic definitions. As a result, one sentence could have several lexical interpretations. An example of multiple lexical interpretation is given in Figure 2.2. For simplifying the syntactic analysis, all possible lexical interpretations are produced at this stage. Each interpretation is then submitted to syntactic analyzer. These interpretations can later be rejected at the phrase analysis stage, at the grammatical analysis stage, or at the semantic analysis stage. If a lexical interpretation fails to be parsed, the next interpretation is submitted to the syntactic analyzer until a suitable interpretation is parsed or there are no more lexical interpretations.

It is possible that some of the lexical interpretations are more favorable than others. If those favorable interpretations can be submitted to the syntactic analyzer first, much useless work can be avoided and the system efficiency can be improved. For this reason, a heuristic function is used to give less hopeful lexical interpretations a lower priority. The heuristic is based on the syntactic knowledge listed below.

1. A determiner or an adjective should be followed by an adjective or a noun;
2. There should be, at least, one verb in a sentence.

More heuristics could be applied here so that the system performance could be improved.

To simplify syntactic analysis, it is desirable to make some syntactic transformations in the given sentences. For example, in natural language, people often prefer saying "I can't do this" to saying "I can not do this." We have two methods to deal with this syntactic phenomenon. One is to change the word "can't" into two words "can not." Another one is to define the special word "can't" in the lexicon. The second method enlarges the size of the lexicon as well as complicates the syntactic analysis. Therefore, the first method is adopted in the system design. Because the transformation is purely string substitution and does not require any specific knowledge, this work can be easily done before the words in a sentence are localized in the lexicon.

Another problem related to the syntactic transformation is that people often use abbreviation to express relative clauses. For example, in the sentence "List the names of the students taking comp212," the present participle phrase "taking comp212" can be expressed as the relative clause "Who are taking comp212." For past participle phrase, the same strategy can be used. For example, in the sentence "List all the courses taken by Richard," the past participle phrase "taken by Richard" also can be transformed into a relative clause "which are taken by Richard." After this transformation is performed, the syntactic analyzer needs only to process the relative clauses. As a result, the syntactic phenomena, which the syntactic analyzer has to deal with, are reduced.

4.4.2 Bottom_Up Phrase Construction

Based on the lexical interpretation, a bottom-up analysis is carried out to produce simple phrase structures, such as noun phrases, verb phrases, etc. This analysis is procedural. One type of phrase corresponds to one procedure. During the analysis, all these procedures look ahead as many words in the sentence as needed and try to get a match based on the phrase structure defined in the grammar. Once a phrase is recognized, a phrase marker is assigned. Simultaneously, the head and the tail of the phrase are recorded in the corresponding phrase structure. If all the words in the sentence can be grouped up into several phrases properly, then the bottom-up analysis is completed. As a result, a set of phrase structures are established. This set of phrase structures are then passed to the grammatical analysis stage. If the words in the sentence cannot be grouped properly into phrases, then the current lexical interpretation fails in bottom-up parsing and the next lexical interpretation is attempted.

To simplify the grammatical analysis, some conjunctions are processed after the phrase analysis is finished. For example, in the sentence "List the names and the salaries of the employees," the noun phrase "the names" and the noun phrase "the salaries" can be combined to produce a compound noun phrase. This is because "the name" and "the salary" have the same syntactic category, are associated with the conjunction, and are semantically compatible. However, in another sentence such as "List the names of the employees and their departments," we can not combine the noun phrases "the employees" and the noun phrase "their departments." This is because that the word "employee" and the word "department" are not semantically compatible. Therefore, the combination of two noun phrases associated with a conjunction has to consult not only the syntactic information but also the semantic information as discussed in section 3.3.2. A predicate called semantics_checker based on the criteria defined in section 3.3.2 is provided to test if two noun phrase are semanticly compatible.

After the bottom-up syntactic analysis, we can get a sequence of phrase markers

```
1)        VP ------------------------ V  tell

2)        P_Pro_Obj --------------- Pro  me

                               -- Det  the
                   -- NP -------|
                   |           ---- N  names
                   |
3)        NP -----|-------------- Conj  and
                   |
                   |           -- Det  the
                   -- NP -------|
                               ---- N  salaries

                   --------------- Prep  of
4)        PrepP --|
                   |           -- Det  the
                   -- NP -------|
                               ---- N  employees

                   --------------- Prep  in
5)        PrepP --|
                   |        -- unknown  computer
                   -- NP -- |
                            -- unknown  science
```

Figure 4.3: Example of Phrase Structures

which describes the phrase structures of a sentence. Figure 4.3 gives an example of all the phrase structures for the sentence "Tell me the names and the salaries of the employees in computer science."

In this bottom-up processing stage, when a verb phrase is constructed, its corresponding modality is built up as well. For example, based on the definition of the modality given in section 3.2.2, the verb phrase "has been teaching" has the "Third Person," "Singular Number," "Present Tense," and "Perfect_Continuing Aspect." The information about verbs is used in the semantic analysis.

4.4.3 Top_Down Grammar Parsing

In top-down parsing processing stage, a set of phrase structures produced in the phrase analysis stage are processed. First, the phrase markers contained in the phrase structures are scanned to check if the sequence of the phrase markers is in good order according to the syntactic grammar defined in the system. This is done by traversing the transition network as discussed in section 3.2.3. If the sentence is syntactically valid, then the grammatical markers are assigned to the corresponding phrases and the syntactic tree structure is produced. The Right Association principle[1] is adopted for resolving the structural and referential ambiguities as discussed in section 2.6. As a result, a deterministic parse tree is produced.

The case-frame semantic analysis described in [25] processes phrase structures, such as noun phrases, verb phrases, etc., rather than individual words. During this analysis, the semantic analyzer needs to find out the subject, object, etc. contained in a sentence by applying a set of syntactic transformation rules. Then, a set of semantic transformation rules are used so that the syntactic roles in the sentence can be converted into the corresponding semantic roles. This approach seems to complicate the semantic analysis. If the syntactic roles such as subject, object, etc. can be produced in the syntactic analysis stage, then the semantic analysis can be simplified without much syntactic effort. Thus, it is required that the transition network used for parsing the sentences be augmented with an action component. In each action component, a grammatical marker is defined. During the traverse of the transition network, when such a grammatical marker is encountered, a corresponding procedure

```
                 The teacher      teaches        course.
Phrase Marker:       NP              VP              NP
               (s0)--------->(s1)----------->(s2)--------->(S3)
                               |               |              |
                               |               |              |
Grammatical Marker:       (subject)  (predicate_verb)    (object)
```

Figure 4.4: Example of Augmented Finite State Transition Network with Grammatical Marker

is invoked to establish this marker in the parse tree structure. This technique is discussed in section 3.2 and employed in MIDBMS to meet the requirement proposed above. In this way, all the syntactic roles in the sentence are produced in the top-down parsing stage and contained in the parse tree structure. Figure 4.4 gives a demonstrative example to show how such an augmented transition network works.

In the semantic analysis, the semantic analyzer needs only to search the parse tree to find out the syntactic roles in the sentence. Then, it can perform the semantic transformation rules to produce the corresponding semantic roles. Appendix B gives the augmented finite state transition network definition used in the MIDBMS system.

Combining the bottom-up processing and top-down processing, Figure 4.5 shows a comprehensive example of syntactic representation produced in the syntactic analysis.

```
           --- Predicate_Verb ----------- VP -------- V    list
          |
          |                                   ---- Det  the
          |--- Direct_Object ------------ NP ---|
          |                                   ---- N    names
          |
          |                        -------------- Prep  of
S ----|--- Attribute --- PrepP ---|
          |                        |          ---- Det  the
          |                         -- NP ---|
          |                                   ---- N    students
          |
          |--- Subject ----------------- Q_R_Pro --- Pro  who
          |
          |                                   ---- Aux  are
          |--- Predicate_Verb ----------- VP ---|
          |                                   ---- V    taking
          |
           --- Direct_Obj --------------- NP ----- unknown comp212
```

Figure 4.5: Comprehensive Example of Syntactic Representation

4.5 SEMANTIC ANALYSIS

In semantic analysis, Case Grammar is employed to resolve the semantic ambiguity and produce a general semantic representation. The cases used in the MIDBMS system are Fillmore's original cases mentioned in section 3.3. These cases are "agent," "instrumental," "neutral," "dative," and "locative." For easing the query interpretation, one more case "modifier" is added. This modifier case contains a preposition phrase headed with the preposition "of." In the semantic analysis phase, the modality containing verb knowledge is required. In addition, a set of semantic transformation rules are applied to the syntactic representation so that the corresponding cases can be filled with the semantic roles. The main rules used in this implementation are listed below.

1. If the voice of the verb in a sentence is active and the subject of the sentence is animate, then the subject is converted into agent case and the object is converted into neutral case if the object exists.

2. If the voice of the verb in a sentence is active and the subject of the sentence is not animate, then the subject is converted into instrument case.

3. If the voice of the verb in a sentence is passive, then the subject of the sentence is converted into neutral case. In this circumstance, if there exists an adverbial, which is a preposition phrase modifying verbs, following a verb phrase and headed with a preposition BY, and the noun phrase contained in the adverbial is animate or unknown words, then the noun phrase is converted into agent case. If the adverbial is headed with a preposition WITH or BY and the noun phrase contained in this adverbial is not animate, then the noun phrase is converted into instrument case.

4. If the voice of the verb in a sentence is active and there exists a indirect object, then this indirect object is converted into dative case.

5. All the adverbials, except those cases occurred in rule 3, are converted into locative cases.

6. All the attribute, which is a preposition phrase modifying nouns, following a noun phrase and headed with a preposition OF are converted into modifier case.

These rules establish a mapping between the syntactic representation and the semantic representation by referencing the syntactic knowledge stored in the modality and the semantic knowledge stored in the semantic network. For example, the sentence "Which courses are taken by Richard?" has the following cases according to the rule 3.

```
Agent:   "Richard"
Action:  "take"
Neutral: "which courses"
```

After all the grammatical components find the corresponding cases, we still need to check if these cases filled up by the semantic analyzer satisfy the requirements defined in the case frame specification for each verb as described in section 2.5. If the requirements are satisfied, then the semantic analysis is completed and this case frame based semantic representation is submitted to the query interpreter. If the semantic analysis cannot be passed, this lexical interpretation is rejected and some error messages are given to help users to find out where the problem is. Then, the next syntactic representation is processed. Figure 4.6 shows a semantic representation of the sentence given in Figure 4.5. In this representation, all the syntactic analytical information is retained.

4.6 QUERY INTERPRETATION

The query interpretation is to transform the general semantic representation which is more linguistic oriented into a query meaning representation which is more query specification oriented. In this processing phase, a semantic representation of a sentence is accepted as input form the syntactic analyzer and transformed into the

```
        --- Action ----------------- VP -------- V     list
       |
       |                                  ---- Det   the
       |--- Neutral ----------------- NP ---|
       |                                  ---- N     names
       |
       |                           -------------- Prep  of
S ----|--- Modifier  --- PrepP ---|
       |                          |       ---- Det   the
       |                           -- NP ---|
       |                                  ---- N     students
       |
       |--- Agent ------------------- Q_R_Pro --- Pro   who
       |
       |                                  ---- Aux   are
       |--- Action ----------------- VP ---|
       |                                  ---- V     taking
       |
        --- Neutral ----------------- NP ----- unknown  comp212
```

Figure 4.6: Example of Semantic Representation

query meaning representation, which is an attribute set, a relation set, and a qualification set as described in section 3.5. This query meaning representation is produced by referencing the domain specific knowledge. By establishing a query meaning representation as discussed in section 2.7, the gap between the general meaning representation and the formal query language can be filled. Simultaneously, the portability of the NLI system is enhanced. The query interpretation procedure implemented in this MIDBMS system consists of four stages which are discussed below.

1. Constructing attribute set:

   In this stage, all the attributes which users are asking for must be determined. In the semantic representation produced in the semantic analysis phase, the noun phrases representing the attributes are stored in the neutral case of the main sentence as discussed in section 2.7. In a main sentence, there could exist several neutral cases which are associated by conjunction.

For example, in the imperative sentence "List all the names of the employees and their departments," the neutral cases are filled up with "the names" and "their departments." Thus, the interpreter has to search all the neutral cases belonging to the main sentence and pick up the nouns, such as "names" and "departments," contained in the noun phrases. Since these noun words used in natural language query are not necessarily the same with the ones used in the database schema as the attribute names, a mapping between the words used in the natural language sentences and the ones used in the database schema is required as described in section 2.7. For example, in the sentence "List the names of the students," the noun "names" stands for an attribute of the student relation. However, in the database schema, this is expressed as STUDENTNM. A mapping table is established for this mapping processing as discussed in section 3.4. Figure 4.7 gives an example to illustrate this mapping.

To disambiguate the noun phrases used in a natural language query and find the exact database attribute name which a noun word represents, user has to indicate which entity each noun belongs to. This can be expressed with the noun-noun modifier, the noun with possessive case, or the preposition phrase headed with preposition OF. For example, in the noun phrases "student names," "student's name," and the "the names of the students," the word "student," "student's," and the preposition phrase "of the student" indicate that the word "name" belongs to the student entity. Once these entities are determined, we can find the corresponding database at-

```
Noun:             Entity:            Attribute:

id ---------> student ----------> STUDENTID
         |--> teacher ----------> TEACHERID
         |--> course  ----------> COURSEID

name ------> student ----------> STUDENTNM
         |--> teacher ----------> TEACHERNM
         |--> course  ----------> COURSENM

student -----> student -------------> STUDENTID
                               |---> STUDENTNM

teacher -----> teacher -------------> TEACHERID
                               |---> TEACHERNM
                               |---> OFFICE

course ------> course -------------> COURSEID
                               |---> COURSENM

office ------> office -----------> OFFICE
```

Figure 4.7: Example of Word Mapping Table

tribute name for a noun phrase stored in a neutral case by making use of the mapping table mentioned in Figure 4.7.

As discussed in Chapter 1, natural language is less precise than a formal query language in some aspects[44]. One particular aspect of this lack of precision is the fact that in natural language a user may explicitly or implicitly ask for entire relations rather than for individual attributes of these relations. When interpreting natural language queries as query meaning representations, the interpreter has to decide which attributes of the relations being demanded are to be presented. The simplest strategy is to present all of the relation's attributes. Another strategy is to specify a default set of the attributes for each relation. In this MIDBMS system, the entire attribute set of each relation is specified to be represented in

Attribute:	Relation:	Feature:
"STUDENT",	"STUDENTID",	"key",
"STUDENT",	"STUDENTNM",	"default",
"COURSE",	"COURSEID",	"key",
"COURSE",	"COURSEID",	"default",
"COURSE",	"COURSENM",	"",
"TEACHER",	"TEACHERID",	"key",
"TEACHER",	"TEACHERNM",	"default",
"TEACHER",	"OFFICE",	"",
"STUDENT_COURSE",	"STUDENTID",	"key",
"STUDENT_COURSE",	"COURSEID",	"key",
"TEACHER_COURSE",	"TEACHERID",	"key",
"TEACHER_COURSE",	"COURSEID",	"key",

Figure 4.8: Example of Database Schema

the query specification if the name of the relation appears in the natural language query sentence. For example, in the sentence "List all the students," the attribute set of the student relation, which are STUDENTID and STUDENTNM in the example given in Figure 4.7, are picked up and placed into the attribute set.

2. Constructing relation set:

   Once the attribute set is constructed, the corresponding relations of the attributes can be determined by searching the database schema. A possible database schema is given in Figure 4.8.

3. Constructing qualification set:

   Qualification analysis is concentrated on the locative cases and the relative clauses. In this stage, the most difficult problem is to resolve the pronoun references. This problem can be resolved in two ways as discussed in section 2.6. One is to use the Right Association principle[1] to find out the previous

Agent	Action	Instrument	Dative	Neutral	Locative
student	take	nil	nil	course	nil
teacher	teach	nil	nil	course	nil

Figure 4.9: Example of World Knowledge about Verbs

noun phrase located in front the pronoun. For example, in sentence "List the names of the students who are taking comp212," the pronoun "who" refers to the noun "students." Another approach is to make use of the world knowledge about the verb. For example, if we are talking about the university and suppose we have the database schema given in Figure 4.8, then we could have the following world knowledge about the verbs "teach" and "take" shown in Figure 4.9. This kind of knowledge is represented in the form of case frames as described in section 3.4.

If the sentence is "List the all the students who are taking comp212," then we can say that the relative pronoun "who" represents students because the agent of the verb "take" is "student." The unknown word "comp212" is related to the course because the neutral case of the verb "take" is "course." In database schema, a default attribute is defined in each relation. This default attribute is used when user specifies an attribute value in the query without giving the corresponding attribute name. In the above example, by referencing the database schema, we can find the default attribute of the course relation is COURSECODE. Therefore, the qualification of this relative clause is COURSECODE = "comp212."

Making use of this domain specific world knowledge about the verbs is very helpful in resolving the referential problem. This also makes the knowledge representation of the whole system more consistent in terms of the case framework representation. In MIDBMS, both approaches are employed. The Right-Association approach is used when the verb is "to be." And the verb knowledge is used when the verb expresses a behavior.

4. Join relation:

   In a query, one or more join relations could be involved even though those relations are not mentioned explicitly in the sentence. For example, in the sentence "List the names of the students taking comp212," two relations are mentioned explicitly. One is STUDENT relation and the another is the COURSE relation. These two relations are associated with the join relation STUDENT_COURSE according to the database schema defined in Figure 4.8. For exposing all the information contained in the natural query, it is necessary to make the implied join relations explicit. Thus, more information can be provided to the formal query language generator. The implied join relations can be found with following steps.

   (a) Check the relation set established in constructing relation set stage. If there exists one relation which contains all the key attributes of the remaining relations, then the join relation has been included in the relation set already. Then, do step 3.

   (b) If there does not exist such a join relation, then pick up all the key attributes of each relation in the relation set. Search database schema and find the join relation which contains all the key attributes of all the relations in the relation set. Put this join relation into relation set. Then, do step 3.

(c) Construct join qualifications with an equality sign and the key attributes. Then, put the join qualifications into the qualification set.

For the sentence mentioned above, the relation set contains two relations which are STUDENT and COURSE. Because each of these two relations does not contain the key attribute of the another relation, the database schema is searched. Since the relation STUDENT_COURSE contains both key attributes of the STUDENT and the COURSE relations, it is determined as a join relation and is placed into the relation set. The corresponding join qualifications are listed below and they are placed into qualification set.

```
STUDENT.STUDENTID = STUDENT_COURSE.STUDENTID
COURSE.COURSEID = STUDENT_COURSE.COURSEID
```

4.7 SQL GENERATION

In the MIDBMS system, the query specification of SQL is implemented. The corresponding syntax of the query specification used in this implementation is given in Figure 4.10.

From this syntax, we can find the correspondence between the query meaning

```
<query_specification> ::=
SELECT <attr_reference_lst>
<table_expression>

 <attr_reference_lst> ::=
<attr_reference> [{,<attr_reference>}...]

<table_expression> ::=
<from_clause>
[<where_clause>]

<from_clause> ::=
FROM <rel_reference_lst>

<rel_reference_lst> ::=
<rel_reference> [{,<rel_reference>}...]

<where_clause> ::=
WHERE <predicate>

 <predicate> ::=
<expression> <comparator> <expression> |
                <predicate> <bin_log_op> <predicate>

 <attr_reference> ::=
<character string, first character must be a letter>

 <rel_reference> ::=
<character string, first character must be a letter>

 <expression> ::=
<arith_expr> | <string_expr>

 <arith_expr> ::=
<num_lit> | <attr_reference>

 <string_expr> ::=
<string_lit> | <attr_reference>

 <num_lit> ::=
<digit_string>[.<digit_string>]

 <string_lit> ::=
"<char_string>" | ""

 <comparator> ::=
< | > | >= | =< | <>

 <bin_log_op> ::=
AND
```

Figure 4.10: Syntax of SQL Specification Subset

representation and the query specification. The attribute set of query representation corresponds to the "attr_reference_lst" of the query specification; the relation set corresponds to the "rel_reference_lst," and the qualification set corresponds to the "predicate." Thus, in this SQL generation stage, most of the work involved is to fill the attr_reference_lst with the attribute set, the rel_reference_lst with the relation set, and the predicate with the qualification set. Then, according to the syntax of SQL query specification, the SQL generator produces the SELECT clause, FROM clause, and WHERE clause. Based on the database schema given in Figure 4.8, Figure 4.11 gives an example of query specification of SQL for the sentence "List all the teachers who are teaching comp212 and working in AD301." In this example, the attribute-set includes attributes of TEACHERID and TEACHERNM; the relation-set includes relations of TEACHER, TEACHER_COURSE, and COURSE; and the qualification-set includes the following qualifications:

```
TEACHER.TEACHERID = TEACHER _COURSE.COURSEID

COURSE.COURSEID = TEACHER _COURSE.COURSEID

COURSEID = ''comp212''

OFFICE = ''AD301 ''
```

The corresponding query qualification is given in Figure 4.11.

```
SELECT   TEACHERID,TEACHERNM
FROM     TEACHER,TEACHER_COURSE,COURSE
WHERE    TEACHER.TEACHERID = TEACHER_COURSE.TEACHERID  AND
         COURSE.COURSEID = TEACHER_COURSE.COURSEID  AND
         COURSEID = "comp212"  AND
         OFFICE = "AD301";
```

Figure 4.11: Example of Query Specification of SQL

1).	u:	union
2).	int:	intersection
3).	dif:	difference
4).	div:	division
5).	lim:	limit
6).	lnj:	limited natural join
7).	ren:	rename attributes
8).	onj:	outer natural join
9).	alt:	alteration
10).	grp:	grouping
11).	trc:	transpose row to column
12).	tcr:	transpose column to row

Figure 4.12: GQML Operations

4.8 GQML TRANSLATION

GQML is a formal relational database query language. Since it works on a heterogeneous distributed database management system, it has not only database query operations but also some special operations about database schema mapping. GQML has twelve operators which are listed in Figure 4.12.

For establishing the target formal query language interface, GQML translator is implemented. This translator translates SQL query specification defined in the previous section into the corresponding GQML query specification. The corresponding query operations are "lim" clause, "lnj" clause, "where" clause, and "attrs" clause. Figure 4.13 gives their syntax.

Under this syntax specification, two query examples are given below.

```
1) English: List the employees whose salaries are more than 30000.


   SQL:    SELECT EMPID, FIRSTNAME, LASTNAME, TOTSALARY
           FROM   EMPLOYEE
           WHERE  TOTSALARY > 30000;


   GQML:   lim EMPLOYEE
             where TOTSALARY > 30000
             attrs EMPID, FIRSTNAME, LASTNAME, TOTSALARY;


2) English: List the employees whose salaries are 30000 and
            whose department is architecture.


   SQL:       SELECT EMPID, FIRSTNAME, LASTNAME, TOTSALARY
```

```
<rel_op> ::=
       lim <rel_opnd> [where <predicate>]
                      [attrs <attr_nm_lst>]; |
       lnj <rel_opnd>,<rel_opnd> [where <predicate>]
                      [attrs<attr_nm_lst>];

<rel_opnd> ::=
       <rel_op> | <rel_nm>

<attr_nm_lst> ::=
       <attr_nm> {, <attr_nm>}

<attr_nm> ::=
       <character string, first character must be a letter>

<rel_nm> ::=
       <character string, first character must be a letter>

<predicate> ::=
       <expression> <comparator> <expression> |
       <predicate> <bin_log_op> <predicate>

<expression> ::=
       <arith_expr> | <string_expr>

<arith_expr> ::=
       <num_lit> | <attr_nm>

<str..ng_expr> ::=
       <string_lit> | <attr_nm>

<num_lit> ::=
       <digit_string>[.<digit_string>]

<string_lit> ::=
       "<char_string>" | ""

<comparator> ::=
       < | > | >= | =< | <>

<bin_log_op> ::= and
```

Figure 4.13: Syntax of Subset GQML (from [39])

```
FROM   EMPLOYEE, WORKSIN, DEPTMNT
WHERE  EMPLOYEE.EMPID = WORKIN.EMPID   AND
       DEPTMNT.DEPTID = WORKIN.DEPTID  AND
       TOTSALARY > 30000  AND
       DEPTNM = "ARCHITECTURE";
```

```
GQML:    lnj
           EMPLOYEE,
           lnj WORKSIN, DEPTMNT;
         where TOTSALARY > 30000 and
               DEPTNM = "ARCHITECTURE"
         attrs EMPID, FIRSTNAME, LASTNAME, TOTSALARY;
```

By analyzing the above two sets of examples, we may find that the main difference existing between these two types of query specifications is the join operation. In SQL, users have to specify not only the join relations in the FROM clause but also the join qualifications in the WHERE clause explicitly. However, in GQML, users do not need to give join qualifications explicitly in the WHERE clause but they have to use the "lnj" operator to specify the two join relations. This difference existing these two query specifications can be resolved. This is because both query specifications are defined based on the relational data model. Thus, they have quite strong similarity in terms of syntax and the semantics as discussed in section 2.8. The basic algorithm for resolving the difference of join operations is given below.

1. Check the WHERE clause of the SQL to see if there exist join relations. This needs to check out the join qualifications first. The characteristics of the join qualification are that the both sides of the equality sign contain the different relations but same attributes. After checking out the join qualifications, try to find out which relation appeared more than one time in the join qualifications. The one appeared more than one time is the join relation.

2. If there exists a join relation, then remove the corresponding join qualifications from the WHERE clause. Construct "limited natural join" clause by joining those relations appearing in the join qualifications with the "lnj" operator.

The remaining part of these two query specifications is almost the same except the different key words are used. Thus, this part of the translation is fairly trivial.

4.9 SUMMARY OF SYSTEM DESIGN

In this chapter, an overall prototype system design is provided. In this design, we integrate the top-down and bottom-up parsing mechanism and Case Grammar into one system organically. The top-down and bottom-up parsing mechanism is used for the syntactic analysis. Case Grammar is used for the semantic analysis. Since most knowledge used in the prototype system is declarative and the system is divided into modules according to their functions, the prototype system is highly transportable. Based on this design, the prototype system called MIDBMS is implemented.

Chapter 5

CONCLUSION AND FUTURE WORK

5.1 CONCLUSION

This thesis has investigated the issues in the design of a transportable, multilevel interface to database management system. The main concerns are the feasibility and the portability of such a system. Based on the investigation provided in this thesis, a prototype system is designed. For testing this system design, the Multilevel Interface to Data Base Management System (MIDBMS) is implemented on UNIX system with ANSI C programming language. For integrating the pre-existing system HDDBMS[39] into the MIDBMS to form the target formal query language interface, the MIDBMS system is transferred to an IBM PC/XT where the HDDBMS system resides. Since the programming language used in the implementation is ANSI C, no extra programming effort is needed except for recompiling the whole system code on an IBM PC/XT machine during the system transformation.

Most existing natural language interface systems, such as [18], [22], [26], [29], and [33], choose one formal query language as their final output. In such cases, if the formal query language is changed, the whole formal query language generator has to be rewritten. The amount of the rewriting work really depends on how the query language generator interfaces with the rest of the NLI system and how the intermediate processing results are represented. Because of the variety of the formal query languages, it is desirable to choose a relatively standard formal query language as the final output of the NLI system. Based on this standard formal query language, any other specific target formal query language translator can be generated, and thus only the translator associated with the target query language has to be rewritten. Moreover, since the translator maps from a formal language to a formal language, little effort is required in its rewriting. As a consequence, users receive an extra interface in which they can express what they want from a database system more precisely.

In the MIDBMS system, SQL is chosen as the final output of the NLI system. GQML working on HDDBMS is chosen as a target query language. Both the NLI system and the GQML translator are designed and implemented in the MIDBMS system. Building the GQML translator involved three days' programming work. The final results have shown that such a system architecture that establishes multiple level interfaces on a database management system benefits not only the NLI system building itself but also the users who can submit their queries in the different interfaces with the different query languages.

Generally, the transportability of the NLI system depends on the modularity of the system functions and the knowledge representation schemes. This principle is pursued throughout the MIDBMS system design. Consequently, the MIDBMS system is highly modular in its system functions and its knowledge :epresentation. This can be observed through the system architecture given in Figure 4.1.

As a test bed of the transportability of the MIDBMS system, two different database domains are prepared. In the transit between these two application domains, only several hours of work were required to define a new domain specific knowledge base, which consists of a database schema, a word mapping, and a case frame based world knowledge about verbs. The lexicon is left untouched. The domain specific knowledge bases used in this thesis and a set of query examples are given in Appendix D and E. This set of comprehensive results is very encouraging and shows that the system design goal has been achieved.

As the part of the thesis, two domain specific knowledge bases are provided in Appendix D and Appendix E. Based on these two different database domains, two sets of natural language query sentences and the corresponding SQL and GQML query specifications are given. This set of comprehensive results is very encouraging and shows that the system design goal has been achieved.

5.2 FUTURE WORK

A framework of the multilevel interface system has been built up in the MIDBMS system. However, more work needs to be done in the following areas: 1. Ellipse processing:

This capability is missing from the current MIDBMS implementation. With this ability, users are able to use pronouns to reference the objects mentioned in the previous sentences. For example, users may give the sentence "List the names of the employees in computer science" in the first query and give an abbreviation sentence "their salaries" in the second query. For processing ellipses, an ellipse processor must remember the processing history it has performed. This can be done by keeping the general meaning representation described in section 2.5 for a previous processed sentence. Based on this general meaning representation, the ellipse processor has to find out the corresponding referent that the pronoun represents. The solution used in resolving the referential and structural ambiguities as described in section 2.6 can be applied here for determining a correct referent. Once the corresponding referent is found out, the old object contained in the corresponding referent case is replaced with the new object contained in the abbreviation sentence. Then, this modified general meaning representation is submitted to the query interpreter to be processed further. Generally, if a NLI system just keeps the history of one previous sentence, the ellipse processing is relatively easy. However, if a NLI system tries to keep the history of several previous sentences, then the situation becomes complex. Since in such a processing history kept by a NLI system, there could exist several possible referents contained in several processed sentences. Thus, the ellipse processor has to decide which referent is the right one which the user references. This issue needs more investigation.

2. Enlargement of syntactic and semantic coverage:

Although the current MIDBMS system is capable of understanding a relatively diverse set of syntactic and semantic phenomena, some even more complex phenomena need to be examined. For example, since imperative sentences and special interrogative sentences headed with interrogative pronouns are acceptable in the current MIDBMS, the sentence "Could you tell me who are taking course comp212?" would not be recognized. For accepting this type of common interrogative sentences with relative clauses, the system needs to enlarge the grammar. Correspondingly, the system abilities of syntactic and semantic analyses need to be enhance as well.

3. Enhancement of the response ability of the system:

In the syntactic analysis of the current MIDBMS implementation, the response to the ill-formed sentences is not detailed enough. This should be enhanced so that better guidance can be provided to the users to help them construct acceptable sentences.

4. Augmentation and optimization of the SQL generator:

The current SQL generator handles only the query specification of SQL. This query specification is also incomplete in terms of the SQL standard[7]. Group_By clause and Having clause are missing from the current query specification. In addition, in the SQL standard, the relation reference is defined as following:

```
<relation reference> ::= <relation name>[<correlation name>]
```

We may find that the optional part, that is, [¡correlation name¿] is missing from the current implementation. Under this circumstance, when the natural language query sentence "Who are teaching comp212 and comp231?" is posed to the system, the corresponding SQL query specification produced by the SQL generator is given as follows.

```
SELECT   TEACHERID,TEACHERNM
FROM     TEACHER,TEACHER_COURSE,COURSE
WHERE    TEACHER.TEACHERID = TEACHER_COURSE.TEACHERID  AND
         COURSE.COURSEID = TEACHER_COURSE.COURSEID  AND
         COURSEID = "comp212"  AND
         COURSEID = "comp231";
```

However, the correct query specification should be the following one:

```
SELECT   A.TEACHERID, A.TEACHERNM
FROM     TEACHER A, TEACHER_COURSE B, TEACHER_COURSE C,
         COURSE D, COURSE E
WHERE    A.TEACHERID = B.TEACHERID  AND
         A.TEACHERID = C.TEACHERID  AND
         B.COURSEID = D.COURSEID  AND
         C.COURSEID = E.COURSEID  AND
         D.COURSEID = "comp212"  AND
         E.COURSEID = "comp231";
```

Without specifying the correlation names, the SQL cannot be generated correctly in the case described above. This requires to modify the SQL generator. In addition, the example given above also can be optimized. The COURSE relation is redundant. The optimized query specification is listed below.

```
SELECT   A.TEACHERID, A.TEACHERNM
```

```
FROM     TEACHER A, TEACHER_COURSE B, TEACHER_COURSE C,
WHERE    A.TEACHERID = B.TEACHERID  AND
         A.TEACHERID = C.TEACHERID  AND
         B.COURSEID = "comp212"  AND
         C.COURSEID = "comp231";
```

Except those future projects listed above, another extension which can be done in the MIDBMS is a natural language generation sub-system [31] [32]. In this subsystem, the input is SQL query specification and the output is a corresponding natural language sentence. Under the support of this sub-system, when users give a query sentence from the natural language interface, the MIDBMS is able to return a SQL query specification or a natural language sentence. This natural language sentence expresses what the system has understood. Thus, a loop, which is from natural language to SQL and then from SQL to natural language, can be constructed. Figure 5.1 shows a possible enhanced MIDBMS architecture in which a natural language generation sub-system is integrated.

In this loop, users can repeatedly modify their natural language sentences until they believe that the system has understood their demands. Then, this confirmed SQL query specification can be submitted to the GQML translator. W. S. Luk and Steve Kloster in their paper called ELFS: English Language From SQL[31] described a generation system from SQL to natural language. The original applications of the ELFS system are the SQL tutorial and the SQL programming aid. If this system is integrated into the multiple interfaces system, that will be greatly improve the

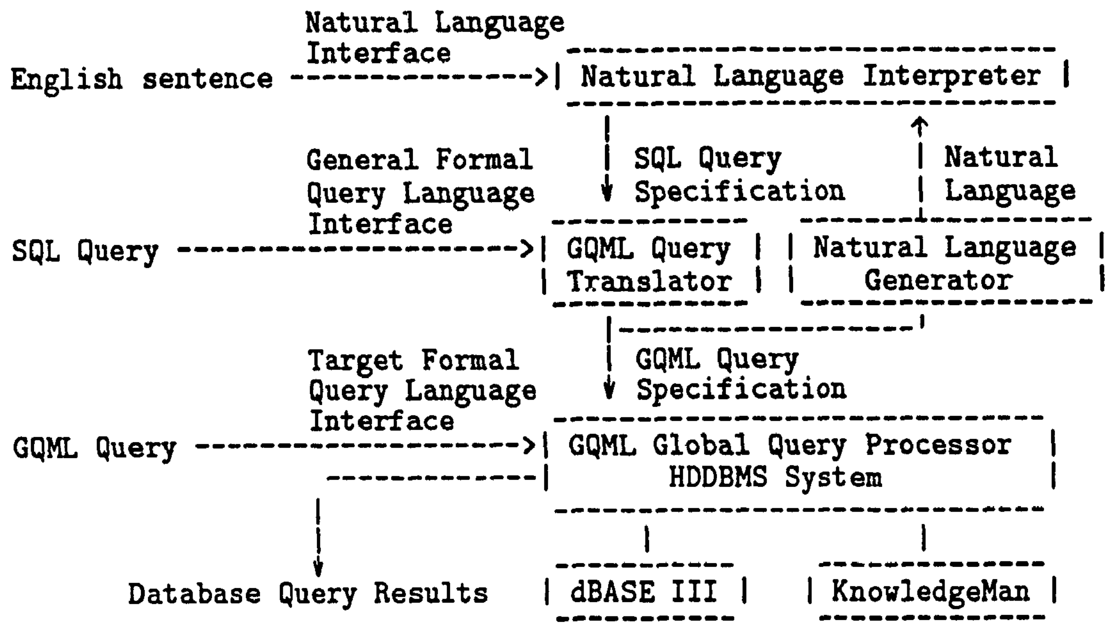
Natural Language Interface
English sentence
Natural Language Interpreter
SQL Query Specification
Natural Language
General Formal Query Language Interface
SQL Query
GQML Query Translator
Natural Language Generator
Target Formal Query Language Interface
GQML Query Specification
GQML Query
GQML Global Query Processor HDDBMS System
Database Query Results
dBASE III
KnowledgeMan

Figure 5.1: Enhanced Architecture of MIDBMS System

friendliness of the interface system and make the system, as a whole, more practical.

This thesis is a step toward establishing a database query environment. In this environment, different kinds of tools are provided with which database users can access database information conveniently. For achieving such a goal, we have offered a system model and implemented a prototype system. The result is encouraging. However, more investigation and research are required. This needs the joint efforts from the researchers working in the fields of database management system and artificial intelligence.

Bibliography

[1] Allen, James. Natural Language Understanding, University of Rochester, The Benjamin/Cummings Publishing Company, Inc., 1987

[2] Barr, Avron, and Feigenbaum, Edward A. The Handbook of Artificial Intelligence, William Kaufman Inc., California, Vol.1, 1981

[3] Belford, G. G., Liu, J. W. S., and etc. Report Generation Facility: A High-Level Interface for Coherent Access to Heterogeneous Database Systems, International Conference on Data Engineering, Los Angeles, CA, USA, 5-7 Feb. 1986

[4] Breitbart, Yuri, Olson, Peter L., and Thompson, Glenn R. Database Integration in a Distributed Heterogeneous Database System, International Conference on Data Engineering, Los Angeles, CA, USA, 5-7 Feb. 1986

[5] Brodie, Michael L., and Mylopoulos, John. On Knowledge Base Management Systems: Integrating Artificial Intelligence and Database Technologies, Springer-Verlag, New York Inc., 1986

[6] Bruce, Bertram. Case System for Natural Language, Artificial Intelligence, No. 6, 1975

[7] CAN/CSA-Z243.47-88 (ISO 9075-1987). Information Processing System - Database Language SQL, Canadian Standards Association, 1988

[8] Chen, Peter Pin-Shan. The Entity-Relationship Model Toward a Unified View of Data, ACM Transaction on Database Systems, Vol.1, No.1, March, 1976

[9] Chen, Peter Pin-Shan. English Sentence Structure and Entity-Relationship Diagrams, Information Science, Vol. 29, 1983

[10] Chomsky, Noam. Syntactic Structures, The Hague: Mouton & Co., 1957

[11] Chomsky, Noam. Aspects of the Theory of Syntax, The M.I.T. Press, 1965

[12] Chomsky, Noam. Studies on Semantics in Generative Grammar, The Hague: Mouton & Co., 1972

[13] Cullingford, Richard E. Natural Language Processing: A Knowledge-Engineering Approach, Rowman & Littlefield, 1986

[14] Damerau, Fred J. Problems and Some Solutions in Customization of Natural Language Database Front Ends, , ACM Transactions on Office Information System, April 1985, Vol.3, No.2

[15] Date, C. J. An Introduction to Database System, Addison-Wesley, Reading, Mass., 1981

[16] Date, C. J. A Critique of the SQL Database Language, ACM SIGMOD Rec. 14,3, 1984

[17] Desai, C. Bipin, McManus, J., and Vincent, P.J. A Portable Natural Language Interface, AFIPS Conference Proceedings, Vol.56, 1987

[18] Desai, C. Binpin, Pollock, Richard J., and Vincent, Philip J. A Natural Language Interface to Multiple Database Office Information System, SIGOIS Bulletin, Vol.9-4, December 1988

[19] Epstein, Samuel S. Transportable Natural Language Processing through Simplicity-The PRE System, ACM Transactions on Office Information Systems, Vol.3, No.2, April 1985

[20] Fillmore, Charles. A Case for Case, Universals in Linguistic Theory, 1968 by Holt, Rinehart and Winston, CBS College Publishing

[21] Griffith, Robert L. Three Principles of Representation for Semantic Networks, ACM Transactions on Database Systems, Vol.7, No.3, September 1982

[22] Grosz, Barbara J., Appelt, Douglas E., and etc. TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces, Artificial Intelligence 32, 1987

[23] Hafner, Carole D., and Kurt Godden. Portability of Syntax and Semantics in Datalog, ACM Transactions on Office Information System, April 1985, Vol.3, No.2

[24] Hammer, Michael., and etc. Database Description with SDM: A Semantic Database Model, ACM Transaction on Database System, Vol.6, No.3, 1981

[25] Harris, Mary Dee. Introduction to Natural Language Processing, Prentice-Hall Company, 1985

[26] Ishikawa, H., and etc. A Knowledge-based Approach to Design A Portable Natural Language Interface to Database System, Software Lab., Fujitsu Laboratories Ltd. Japan, Data Engineering Proceedings, 1986

[27] Jacobs, Paul S. Acknowledge Framework for Natural Language Analysis, IJCAI, 1987

[28] Janas, Jurgen M. The Semantics-Based Natural Language Interface to Relational Databases, Cooperative Interfaces to Information Systems (Chapter 6), L. Bok and M. Jarke, Springer-Verlag, Berlin, 1986

[29] Kaplan, S. Jerrold. Designing A Portable Natural Language Database Query System, Stanford University, ACM Transactions on Database System, Vol.9, No.1, March 1984, Page 1-19

[30] Katz, J.J., and Fodor, J.A. The Structure of a Semantic Theory, The Structure of Language, Ed. J. Fodor and J. Katz, Englewood Cliffs, New Jersey: Prentice-Hall, 1964

[31] Luk, W. S., and Kloster, Steve. ELFS: English Language From SQL, ACM Transactions on Database Systems, Vol.11, No.4, December 1986

[32] Marakakis, Emmanouil J. Entity Relationship Approach to the Generation of Sentences for Database Queries, Master Thesis of Computer Science of Concordia University, 1984

[33] McFetridge, P., Hall, G., and etc. Knowledge Acquisition in System X: Natural Language Interface to Relational Databases, Proceedings of the Seventh Biennial Conference of the Canadian Society for Computational Studies of Intelligence, Edmonton, Alta., Canada, June 1988

[34] Minsky, Marvin. A Framework for Representing knowledge, in The Psychology of Computer Vision, McGraw Hill, New York, 1975

[35] Motro, Amihai. Constructing Queries from Tokens, ACM SIGMOD, Vol. 15, No.2, June 1986

[36] Myers, Brad A. Tools for Creating User Interfaces: An Introduction and Survey, Computer Science Department, Carnegie Mellon University, January 1988

[37] Oppacher, F. A Programming Approach for Constructing Natural Language Processors, Master Thesis of Computer Science of Concordia University, 1981

[38] Pitrat, J. Using Declarative Knowledge for Understanding Natural Language, Natural Language Parsing Systems, Springer-Verlag, 1987

[39] Pollock, Richard J. The Design and Implementation of a Heterogeneous Distributed Database Management System Prototype, Master Thesis of Computer Science of Concordia University, 1988

[40] Simmons, Robert. Answering English Questions by Computer: A Survey, Communications of the ACM, Vol.8, 1965

[41] Tstrahan, M. M., and Chamberlin, D. D. Implementation of a Structured English Query Language, Comm. ACM Vol.18, No.10, October 1975

[42] Vincent, Philip J. The Design and Implementation of a Potable Natural Language Interface System, Master Thesis of Computer Science of Concordia University, 1988

[43] Winograd, Terry. Understanding Natural Language, Academic Press, New York, 1972

[44] Winograd, Terry. Language as A Cognitive Process, Volume 1: Syntax, Addison-Wesley Publishing Company, 1983

[45] Woods, William A. Transition Network Grammars for Natural Language Analysis, CACM 13-10, 1970

[46] Woods, William A. Progress in Natural Language Understanding: An Application to Lunar Geology, AFIPS Conference Proceeding, 1973

APPENDIX A

STRUCTURE OF SYNTAX

In MIDBMS, the syntactic class is defined with C language structure definition and given below. This declarative definition reflects the tree structure of the syntactic class.

```
element category[]={
"4",NULL,
"class",NULL,
"collective",NULL,
"material",NULL,
"abstract",NULL,
};


element cas[]={
"5",NULL,
"subjective",NULL,
"objective",NULL,
"possessive",NULL,
"adjective",NULL,
"reflexive",NULL,
};


element gender[]={
"3",NULL,
"masculine",NULL,
"feminine",NULL,
"neuter",NULL,
};


element number[]={
"3",NULL,
"singular",NULL,
"plural",NULL,
"uncountable",NULL,
};


element person[]={
"3",NULL,
"first",NULL,
"second",NULL,
"third",NULL,
};


element form[]={
"5",NULL,
"infinitive",NULL,
"present",NULL,
"past",NULL,
"present_participle",NULL,
"past_participle",NULL,
};


element grade[]={
"3",NULL,
"positive",NULL,
"comparative",NULL,
"superlative",NULL,
};


element position[]={
"2",NULL,
"front",NULL,
"behind",NULL,
};


element adj_property[]={
"10",NULL,
"common",NULL,
"equal",NULL,
"more",NULL,
"less",NULL,
"long",NULL,
"short",NULL,
"big",NULL,
"small",NULL,
"high",NULL.
"low",NULL,
};


element adv_property[]={
"9",NULL,
"common",NULL,
"negation",NULL,
"time",NULL,
"location",NULL,
"direction",NULL,
"position",NULL,
"grade",NULL,
"manner",NULL,
"cause",NULL,
};


element prop_property[]={
"10",NULL,
"in",NULL,
"on",NULL,
"by",NULL,
"with",NULL,
"of",NULL,
"for",NULL,
"at",NULL,
"to",NULL,
"over",NULL,
"under",NULL,
};


element distance[]={
"2",NULL,
"near",NULL,
"far",NULL,
};


element ques_p_category[]={
"7",NULL,
"who",NULL,
"whom",NULL,
"whose",NULL,
"which",NULL,
"what",NULL,
"when",NULL,
"where",NULL,
};


element quan_p_category[]={
"10",NULL,
"all",NULL,
"any",NULL,
"anyone",NULL,
"anything",NULL,
"some",NULL,
"someone",NULL,
"somebody",NULL,
"something",NULL,
"everyone",NULL,
"everything",NULL,
};


element common_noun[]={
"5",NULL,
"category",(element *)category,
"case",(element *)cas,
"gender",(element *)gender,
"number",(element *)number,
"person",(element *)person,
};


element proper_noun[]={
"5",NULL,
"category",(element *)category,
"case",(element *)cas,
"gender",(element *)gender,
"number",(element *)number,
"person",(element *)person,
};


element personal_pronoun[]={
"5",NULL,
"category",(element *)category,
"case",(element *)cas,
"gender",(element *)gender,
"number",(element *)number,
"person",(element *)person,
};


element demons_pronoun[]={
"4",NULL,
"distance",(element *)distance,
"gender",(element *)gender,
"number",(element *)number,
"person",(element *)person,
};


element question_pronoun[]={
"2",NULL,
"category",(element *)ques_p_category,
"case",(element *)cas,
};


element quantified_pronoun[]={
"2",NULL,
"category",(element *)quan_p_category,
"number",(element *)number,
};


element noun_genus[]={
"6",NULL,
"common_noun",(element *)common_noun,
"proper_noun",(element *)proper_noun,
"personal_pronoun",(element *)personal_pronoun,
"demons_pronoun",(element *)demons_pronoun,
"question_pronoun",(element *)question_pronoun,
"quantified_pronoun",(element *)quantified_pronoun,
};


element vt[]={
"3",NULL,
"form",(element *)form,
"person",(element *)person,
"number",(element *)number,
};


element vi[]={
"3",NULL,
"form",(element *)form,
"person",(element *)person,
"number",(element *)number,
};


element vl[]={
```

```
"3",NULL,
"form",(element *)form,
"person",(element *)person,
"number",(element *)number,
};


element be[]={
"3",NULL,
"form",(element *)form,
"person",(element *)person,
"number",(element *)number,
};


element aux[]={
"3",NULL,
"form",(element *)form,
"person",(element *)person,
"number",(element *)number,
};


element verb_genus[]={
"5",NULL,
"vt",(element *)vt,
"vi",(element *)vi,
"vl",(element *)vl,
"be",(element *)be,
"aux",(element *)aux,
};


element definite[]={
"1",NULL,
"number",(element *)number,
};


element indefinite[]={
"1",NULL,
"number",(element *)number,
};


element art_genus[]={
"2",NULL,
"definite",(element *)definite,
"indefinite",(element *)indefinite,
};


element adj[]={
"3",NULL,
"position",(element *)position,
"grade",(element *)grade,
"property",(element *)adj_property,
};


element adj_genus[]={
"1",NULL,
"adj",(element *)adj,
};


element ordinary[]={
"2",NULL,
"grade",(element *)grade,
"property",(element *)adv_property,
};


element interro[]={
"2",NULL,
"grade",(element *)grade,
"property",(element *)adv_property,
};


element relative[]={
"2",NULL,
"grade",(element *)grade,
"property",(element *)adv_property,
};


element conjunctive[]={
"2",NULL,
"grade",(element *)grade,
"property",(element *)adv_property,
};
```

```
element negative[]={
“2”,NULL,
“grade”,(element *)grade,
“property”,(element *)adv_property,
};


element adv_genus[]={
“5”,NULL,
“ordinary”,(element *)ordinary,
“interrogative”,(element *)interro,
“relative”,(element *)relative,
“conjunctive”,(element *)conjunctive,
“negative”,(element *)negative,
};


element prop[]={
“1”,NULL,
“property”,(element *)prop_property,
};


element prop_genus[]={
“1”,NULL,
“preposition”,(element *)prop,
};


element num_genus[]={
“6”,NULL,
“cardinal”,NULL,
“ordinal”,NULL,
“fraction”,NULL,
“decimal”,NULL,
“date”,NULL,
“time”,NULL,
};


element conjun_category[]={
“3”,NULL,
“and”,NULL,
“or”,NULL,
“than”,NULL,
};
```

```
element mark_category[]={
"3",NULL,
",",NULL,
".",NULL,
"?",NULL,
};


element conjunction[]={
"1",NULL,
"category",(element *)conjun_category,
};


element marks[]={
"1",NULL,
"category",(element *)mark_category,
};


element others_genus[]={
"2",NULL,
"conjunction",(element *)conjunction,
"marks",(element *)marks,
};


element syntax[]={
"8",NULL,
"noun",(element *)noun_genus,
"verb",(element *)verb_genus,
"article",(element *)art_genus,
"adjective",(element *)adj_genus,
"adverb",(element *)adv_genus,
"preposition",(element *)prop_genus,
"numeral",(element *)num_genus,
"others",(element *)others_genus,
};
```

APPENDIX B

DECLARATIVE DIFINITION OF GRAMMAR

In MIDBMS, grammar is represented with the finite state transition network. This network is defined declaratively with C language structure definition. In this definition, the first column is the start state of a transition. The second column is the condition of a transition. The third column is the end state of a transition. The fourth column is a grammatical marker which is assigned to a corresponding phrase. The declarative grammar definition used in MIDBMS is given below.

```
machine machine_tab[]={
    "s0",     VP,        "s1",    Predicate_Verb,
    "s0",     RelPro,    "s8",    Subject_Object,
    "s1",     NP,        "s2",    Object,
    "s1",     ProObj,    "s7",    Indirect_Object,
    "s2",     Dot,       "sn",    0,
    "s2",     Conj,      "s3",    Conjunction,
    "s2",     PrepP,     "s4",    Attribute_Adverbial,
    "s2",     RelPro,    "s8",    Subject_Object,
    "s3",     NP,        "s2",    Keep,
    "s4",     NP,        "s5",    Keep,
    "s5",     Dot,       "sn",    0,
    "s5",     NP,        "s5",    Keep,
    "s5",     PrepP,     "s4",    Attribute_Adverbial,
    "s5",     Conj,      "s6",    Conjunction,
    "s5",     RelPro,    "s8",    Subject_Object,
    "s6",     NP,        "s2",    Object,
    "s7",     NP,        "s2",    Object,
    "s7",     RelPro,    "s8",    Subject_Object,
    "s8",     NP,        "s9",    Subject,
    "s8",     ProSub,    "s9",    Subject,
    "s8",     VP,        "s10",   Predicate_Verb,
    "s9",     VP,        "s10",   Predicate_Verb,
    "s9",     PrepP,     "s16",   Attribute_Adverbial,
    "s10",    Dot,       "sn",    0,
    "s10",    NP,        "s11",   Object,
    "s10",    PrepP,     "s13",   Attribute_Adverbial,
    "s11",    Dot,       "sn",    0,
    "s11",    Question,  "sn",    0,
    "s11",    RelPro,    "s8",    Subject_Object,
    "s11",    Conj,      "s12",   Conjunction,
    "s11",    PrepP,     "s13",   Attribute_Adverbial,
    "s11",    NP,        "s11",   Keep,
    "s11",    VP,        "s11",   Predicate_Verb,
    "s12",     NP,         "s11",    Object,
    "s12",     RelPro,     "s8",     Subject_Object,
    "s13",     NP,         "s14",    Keep,
    "s14",     Dot,        "sn",     0,
    "s14",     Question,   "sn",     0,
    "s14",     NP,         "s14",    Keep,
    "s14",     PrepP,      "s13",    Attribute_Adverbial,
    "s14",     Conj,       "s15",    Conjunction,
    "s15",     NP,         "s11",    Object,
    "s15",     RelPro,     "s8",     Subject_Object,
    "s16",     NP,         "s9",     Keep,
    "end",     0,          "",       0,
};
```

APPENDIX C

STRUCTURE OF SEMANTICS

In MIDBMS, the semantic class is defined with C language structure definition and given below. This declarative definition reflects the tree structure of the semantic class.

```
element object_attr[]={
"2",NULL,
"name",NULL,
"identifier",NULL,
};


element phy_obj_attr[]={
"4",NULL,
"size",NULL,
"shape",NULL,
"weight",NULL,
"color",NULL,
};


element animate_attr[]={
"2",NULL,
"age",NULL,
"gender",NULL,
};


element human_attr[]={
"1",NULL,
"address",NULL,
};


element animal_attr[]={
"0",NULL,
};


element inanimate_attr[]={
"0",NULL,
};
```

```
element vegetable_attr[]={
"0",NULL,
};


element non_living_attr[]={
"0",NULL,
};


element abs_obj_attr[]={
"0",NULL,
};


element time_attr[]={
"0",NULL,
};


element location_attr[]={
"0",NULL,
};


element organization_attr[]={
"0",NULL,
};


element concept_attr[]={
"0",NULL,
};


element info_container_attr[]={
"0",NULL,
};


element human[]={
"1",NULL,
"attr",(element *)human_attr, };


element animal[]={
"1",NULL,
"attr",NULL,
};


element vegetable[]={
"1",NULL,
"attr",NULL,
};


element non_living[]={
"1",NULL,
"attr",NULL,
};


element time[]={
"1",NULL,
"attr",NULL,
};


element location[]={
"1",NULL,
"attr",NULL,
};


element organization[]={
"1",NULL,
"attr",NULL,
};


element concept[]={
"1",NULL,
"attr",NULL,
};


element info_container[]={
"1",NULL,
"attr",NULL,
};


element animate[]={
```

```
"3",NULL,
"attr",(element *)animate_attr,
"human",(element *)human,
"animal",(element *)animaL,
};


element inanimate[]={
"3",NULL,
"attr",NULL,
"vegetable",(element *)vegetable,
"non_living",(element *)non_living,
};


element phy_obj[]={
"3",NULL,
"attr",(element *)phy_obj_attr,
"animate",(element *)animate,
"inanimate",(element *)inanimate,
};


element abs_obj[]={
"6",NULL,
"attr",NULL,
"time",(element *)time,
"location",(element *)location,
"organization",(element *)organization,
"concept",(element *)concept,
"info_container",(element *)info_container,
};


element object[] ={
"3",NULL,
"attr",(element *)object_attr,
"phy_obj",(element *)phy_obj,
"abs_obj",(element *)abs_obj,
};


element semantics[]={
"1",NULL,
"object",(element *)object,
};
```

APPENDIX D

DOMAIN KNOWLEDGE BASE I

In MIDBMS, domain specific knowledge base is defined with three separate components which are database schema, word mapping table, and world knowledge about verbs. These knowledge is defined declaratively with C language structure and given below.

```
*******************************************************
DATA BASE SCHEMA ABOUT EMPLOYEES:


EMPLOYEE (KEY:EMPID,FIRSTNAME,LASTNAME,
              TOTSALARY, BIRTHDATE )
DEPTMNT  ( KEY:DEPTID, DEPTNM, STREET, CITY )
WORKSIN  ( KEY:EMPID, KEY:DEPTID, START )
_______________________________________________________
```

DECLARATIVE DEFINITION OF DATABASE SCHEMA:

```
reference schema_tab[]={
    "12",          "",             "",
    "EMPLOYEE",    "EMPID",        "key",
    "EMPLOYEE",    "FIRSTNAME",    "default",
    "EMPLOYEE",    "LASTNAME",     "",
    "EMPLOYEE",    "TOTSALARY",    "",
    "EMPLOYEE",    "BIRTHDATE",    "",
    "DEPTMNT",     "DEPTID",       "key",
    "DEPTMNT",     "DEPTNM",       "default",
    "DEPTMNT",     "STREET",       "",
    "DEPTMNT",     "CITY",         "",
    "WORKSIN",     "EMPID",        "key",
    "WORKSIN",     "DEPTID",       "key",
    "WORKSIN",     "START",        "",
};
```

DECLARATIVE DEFINITION OF WORD MAPPING TABLE:

```
element attr1[]={
"1",NULL,
"DEPTID",NULL,
};

element attr2[]={
"1",NULL,
"EMPID",NULL,
};

element attr3[]={
"1",NULL,
"DEPTNM",NULL,
};

element attr4[]={
"2",NULL,
"FIRSTNAME",NULL,
"LASTNAME",NULL,
};

element attr5[]={
"1",NULL,
"FIRSTNAME",NULL,
};

element attr6[]={
"1",NULL,
"LASTNAME",NULL,
};

element attr7[]={
"1",NULL,
"CITY",NULL,
};

element attr8[]={
"1",NULL,
"STREET",NULL,
};
```

```
element attr9[]={
"1",NULL,
"BIRTHDATE",NULL,
};


element attr10[]={
"1",NULL,
"START",NULL,
};


element attr11[]={
"1",NULL,
"TOTSALARY",NULL,
};


element attr12[]={
"2",NULL,
"STREET",NULL,
"CITY",NULL,
};


element attr13[]={
"5",NULL,
"EMPID",NULL,
"FIRSTNAME",NULL,
"LASTNAME",NULL,
"TOTSALARY",NULL,
"BIRTHDATE",NULL,
};


element attr14[]={
"4",NULL,
"DEPTID",NULL,
"DEPTNM",NULL,
"STREET",NULL,
"CITY",NULL,
};


element id[]={
"2",NULL,
"department",(element *)attr1,
"employee",(element *)attr2,
};


element name[]={
"6",NULL,
"department",(element *)attr3,
"employee",(element *)attr4,
"first",(element *)attr5,
"last",(element *)attr6,
"city",(element *)attr7,
"street",(element *)attr8,
};


element date[]={
"2",NULL,
"birth",(element *)attr9,
"start",(element *)attr10,
};


element address[]={
"1",NULL,
"department",(element *)attr12,
};


element street[]={
"1",NULL,
"department",(element *)attr8,
};


element city[]={
"1",NULL,
"department",(element *)attr7,
};


element salary[]={
"1",NULL,
"employee",(element *)attr11,
};


element employee[]={
```

```
"1",NULL,
"employee",(element *)attr13,
};


element department[]={
"1",NULL,
"department",(element *)attr14,
};


element obj_ref_tab[]={
"9",NULL,
"id",(element *)id,
"name",(element *)name,
"date",(element *)date,
"address",(element *)address,
"street",(element *)street,
"city",(element *)city,
"salary",(element *)salary,
"employee",(element *)employee,
"department",(element *)department,
};
```

DECLARATIVE DEFINITION OF WORLD KNOWLEDGE ABOUT VERBS:

```
action action_tab[]={
  "1",          "",      "",   "",   "",
  "employee",   "work",  "",   "",   "department",
};
```

A SET OF SAMPLE SENTENCES AND CORRESPONDING
QUERY SPECIFICATIONS:

<1>
English:

List the employees in computer science.

SQL:

```
SELECT   EMPID,FIRSTNAME,LASTNAME,TOTSALARY,BIRTHDATE
FROM     EMPLOYEE,WORKSIN,DEPTMNT
WHERE    EMPLOYEE.EMPID = WORKSIN.EMPID   AND
         DEPTMNT.DEPTID = WORKSIN.DEPTID   AND
         DEPTNM = "COMPUTER SCIENCE";
```

GQML:

```
lnj  EMPLOYEE,
     lnj  DEPTMNT, WORKSIN;
  where  DEPTNM = "COMPUTER SCIENCE"
  attrs  EMPID, FIRSTNAME, LASTNAME, TOTSALARY, BIRTHDATE;
```

<2>
English:

List the employees whose salaries are 3000 and whose start date is 11/1/1986.

SQL:

```
SELECT   EMPID,FIRSTNAME,LASTNAME,TOTSALARY,BIRTHDATE
FROM     EMPLOYEE,WORKSIN
WHERE    EMPLOYEE.EMPID = WORKSIN.EMPID   AND
         TOTSALARY = "3000"   AND
         START = "11/1/1986";
```

GQML:

```
lnj  EMPLOYEE, WORKSIN
  where  TOTSALARY = 3000  and
         START = 11/1/1986
  attrs  EMPID, FIRSTNAME, LASTNAME, TOTSALARY, BIRTHDATE;
```

<3>
English:

List the names, salaries and the birth date of the employees.

SQL:

```
SELECT  FIRSTNAME,LASTNAME,TOTSALARY,BIRTHDATE
FROM    EMPLOYEE;
```

GQML:

```
lim  EMPLOYEE
  attrs  FIRSTNAME, LASTNAME, TOTSALARY, BIRTHDATE;
```

<4>
English:

Tell us the names and the salaries of the employees working in the computer science department.

SQL:

```
SELECT  FIRSTNAME,LASTNAME,TOTSALARY
FROM    EMPLOYEE,WORKSIN,DEPTMNT
WHERE   EMPLOYEE.EMPID = WORKSIN.EMPID   AND
        DEPTMNT.DEPTID = WORKSIN.DEPTID   AND
        DEPTNM = "COMPUTER SCIENCE";
```

GQML:

```
lnj  EMPLOYEE,
     lnj  DEPTMNT, WORKSIN;
  where  DEPTNM = "COMPUTER SCIENCE"
  attrs  FIRSTNAME, LASTNAME, TOTSALARY;
```

<5>
English:

List the employees who are working in the department of computer science and their start date.

SQL:

```
SELECT  EMPID,FIRSTNAME,LASTNAME,TOTSALARY,BIRTHDATE,START
FROM    EMPLOYEE,WORKSIN,DEPTMNT
WHERE   EMPLOYEE.EMPID = WORKSIN.EMPID   AND
        WORKSIN.DEPTID = DEPTMNT.DEPTID   AND
        DEPTNM = "COMPUTER SCIENCE";
```

GQML:

```
lnj  EMPLOYEE,
     lnj  DEPTMNT, WORKSIN;
  where  DEPTNM = "COMPUTER SCIENCE"
  attrs  EMPID, FIRSTNAME, LASTNAME, TOTSALARY, BIRTHDATE, START;
```

<6>
English:

Tell me the names of the employees whose last name is Backer and whose department is computer science.

SQL:

```
SELECT  FIRSTNAME,LASTNAME
FROM    EMPLOYEE,WORKSIN,DEPTMNT
WHERE   EMPLOYEE.EMPID = WORKSIN.EMPID   AND
        DEPTMNT.DEPTID = WORKSIN.DEPTID   AND
        LASTNAME = "BACKER"   AND
        DEPTNM = "COMPUTER SCIENCE";
```

GQML:

```
lnj  EMPLOYEE,
     lnj  DEPTMNT, WORKSIN;
  where  LASTNAME = "BACKER"  and
         DEPTNM = "COMPUTER SCIENCE"
  attrs  FIRSTNAME, LASTNAME;
```

<7>
English:

List the employees whose salaries are not more than 20000.

SQL:

```
SELECT  EMPID,FIRSTNAME,LASTNAME,TOTSALARY,BIRTHDATE
FROM    EMPLOYEE
WHERE   TOTSALARY <= "20000";
```

GQML:

```
lim  EMPLOYEE
  where  TOTSALARY <= 20000
  attrs  EMPID, FIRSTNAME, LASTNAME, TOTSALARY, BIRTHDATE;
```

<8>
English:

List the employees whose salaries are less than 4000 and more than 20000.

SQL:

```
SELECT  EMPID,FIRSTNAME,LASTNAME,TOTSALARY,BIRTHDATE
FROM    EMPLOYEE
WHERE   TOTSALARY < "4000"   AND
        TOTSALARY > "20000";
```

GQML:

```
lim  EMPLOYEE
  where  TOTSALARY < 4000  and
         TOTSALARY > 20000
  attrs  EMPID, FIRSTNAME, LASTNAME, TOTSALARY, BIRTHDATE;
```

<9>
English:

List the employees whose salaries are not less than or equal to 30000.

SQL:

```
SELECT  EMPID,FIRSTNAME,LASTNAME,TOTSALARY,BIRTHDATE
FROM    EMPLOYEE
WHERE   TOTSALARY > "30000";
```

GQML:

```
lim  EMPLOYEE
  where  TOTSALARY > 30000
  attrs  EMPID, FIRSTNAME, LASTNAME, TOTSALARY, BIRTHDATE;
```

<10>
English:

List the employees whose salaries are more than or equal to 20000 and less than 50000.

SQL:

```
SELECT  EMPID,FIRSTNAME,LASTNAME,TOTSALARY,BIRTHDATE
FROM    EMPLOYEE
WHERE   TOTSALARY >= "20000"   AND
        TOTSALARY < "50000";
```

GQML:

```
lim  EMPLOYEE
  where  TOTSALARY >= 20000  and
         TOTSALARY < 50000
  attrs  EMPID, FIRSTNAME, LASTNAME, TOTSALARY, BIRTHDATE;
```

APPENDIX E

DOMAIN KNOWLEDGE BASE II

In MIDBMS, domain specific knowledge base is defined with three separate components which are database schema, word mapping table, and world knowledge about verbs. These knowledge is defined declaratively with C language structure and given below.

```
********************************************************
DATE BASE SCHEMA ABOUT UNIVERSITY:


STUDENT ( KEY:STUDENTID, STUDENTNM )
TEACHER ( KEY:TEACHERID, TEACHERNM, OFFICE )
COURSE ( KEY:COURSEID, COURSENM )
STUDENT_COURSE ( KEY:STUDENTID, KEY:COURSEID )
TEACHER_COURSE ( KEY:TEACHERID, KEY:COURSEID )
--------------------------------------------------------
```

DECLARATIVE DEFINITION OF THE DATABASE SCHEMA:

```
reference schema_tab[]={
    "12",                "",              "",
    "STUDENT",           "STUDENTID",     "key",
    "STUDENT",           "STUDENTNM",     "default",
    "COURSE",            "COURSEID",      "key",
    "COURSE",            "COURSEID",      "default",
    "COURSE",            "COURSENM",      "",
    "TEACHER",           "TEACHERID",     "key",
    "TEACHER",           "TEACHERNM",     "default",
    "TEACHER",           "OFFICE",        "",
    "STUDENT_COURSE","STUDENTID",     "key",
    "STUDENT_COURSE","COURSEID",      "key",
    "TEACHER_COURSE","TEACHERID",     "key",
    "TEACHER_COURSE","COURSEID",      "key",
};
```

DECLARATIVE DEFINITION OF WORD MAPPING TABLE:

```
element stuid[]={
"1",NULL,
"STUDENTID",NULL,
};


element teaid[]={
"1",NULL,
"TEACHERID",NULL,
};


element couid[]={
"1",NULL,
"COURSEID",NULL,
};


element stunm[]={
"1",NULL,
"STUDENTNM",NULL,
};


element teanm[]={
"1",NULL,
"TEACHERNM",NULL,
};


element counm[]={
"1",NULL,
"COURSENM",NULL,
};


element stu[]={
"2",NULL,
"STUDENTID",NULL,
"STUDENTNM",NULL,
};


element tea[]={
"2",NULL,
"TEACHERID",NULL,
"TEACHERNM",NULL,
};

element cou[]={
"2",NULL,
"COURSEID",NULL,
"COURSENM",NULL,
};

element off[]={
"1",NULL,
"OFFICE",NULL,
};

element id[]={
"3",NULL,
"student",(element *)stuid,
"teacher",(element *)teaid,
"course",(element *)couid,
};

element name[]={
"3",NULL,
"student",(element *)stunm,
"teacher",(element *)teanm,
"course",(element *)counm,
};

element student[]={
"1",NULL,
"student",(element *)stu,
};

element teacher[]={
"1",NULL,
"teacher",(element *)tea,
};

element course[]={
"1",NULL,
"course",(element *)cou,
};


element office[]={
"1",NULL,
"office",(element *)off,
};


element obj_ref_tab[]={
"6",NULL,
"id",(element *)id,
"name",(element *)name,
"student",(element *)student,
"teacher",(element *)teacher,
"course",(element *)course,
"office",(element *)office,
};
```

DECLARATIVE DEFINITION OF WORLD KNOWLEDGE ABOUT VERBS:

```
action action_tab[]={
  "3",          "",         "",          "",    "",
  "teacher",    "teach",    "course",    "",    "",
  "student",    "take",     "course",    "",    "",
  "teacher",    "work",     "",          "",    "office",
};
```

A SET OF SAMPLE SENTENCES AND CORRESPONDING
QUERY SPECIFICATIONS:

<1>
English:

List the names of the teachers who are teaching course comp212.

SQL:

```
SELECT  TEACHERNM
FROM    TEACHER,TEACHER_COURSE,COURSE
WHERE   TEACHER.TEACHERID = TEACHER_COURSE.TEACHERID   AND
        COURSE.COURSEID = TEACHER_COURSE.COURSEID   AND
        COURSEID = "comp212";
```

GQML:

```
lnj  TEACHER,
     lnj  COURSE, TEACHER_COURSE;
  where  COURSEID = "comp212"
  attrs  TEACHERNM;
```

<2>
English:

List all the teachers who are teaching comp212 and working in AD301.

SQL:

```
SELECT  TEACHERID,TEACHERNM
FROM    TEACHER,TEACHER_COURSE,COURSE
WHERE   TEACHER.TEACHERID = TEACHER_COURSE.TEACHERID   AND
        COURSE.COURSEID = TEACHER_COURSE.COURSEID   AND
        COURSEID = "comp212"   AND
        OFFICE = "AD301";
```

GQML:

```
   lnj  TEACHER,
        lnj  COURSE, TEACHER_COURSE;
     where  COURSEID = "comp212"  and
            OFFICE = "AD301"
     attrs  TEACHERID, TEACHERNM;
```

<3>
English:

List the courses taken by Russel.

SQL:

```
   SELECT  COURSEID,COURSENM
   FROM    COURSE,STUDENT_COURSE,STUDENT
   WHERE   COURSE.COURSEID = STUDENT_COURSE.COURSEID   AND
           STUDENT.STUDENTID = STUDENT_COURSE.STUDENTID   AND
           STUDENTNM = "Russel";
```

GQML:

```
   lnj  COURSE,
        lnj  STUDENT, STUDENT_COURSE;
     where  STUDENTNM = "Russel"
     attrs  COURSEID, COURSENM;
```

<4>
English:

List the courses which Russel is taking.

SQL:

```
SELECT  COURSEID,COURSENM
FROM    COURSE,STUDENT_COURSE,STUDENT
WHERE   COURSE.COURSEID = STUDENT_COURSE.COURSEID   AND
        STUDENT.STUDENTID = STUDENT_COURSE.STUDENTID   AND
        STUDENTNM = "Russel";
```

GQML:

```
lnj  COURSE,
     lnj  STUDENT, STUDENT_COURSE;
  where  STUDENTNM = "Russel"
  attrs  COURSEID, COURSENM;
```

<5>
English:

List all the teachers working in AD301.

SQL:

```
SELECT  TEACHERID,TEACHERNM
FROM    TEACHER
WHERE   OFFICE = "AD301";
```

GQML:

```
lim  TEACHER
  where  OFFICE = "AD301"
  attrs  TEACHERID, TEACHERNM;
```

<6>
English:

Tell me who are working in office AD301.

SQL:

```
SELECT  TEACHERID,TEACHERNM
FROM    TEACHER
WHERE   OFFICE = "AD301";
```

GQML:

```
lim  TEACHER
  where  OFFICE = "AD301"
  attrs  TEACHERID, TEACHERNM;
```

<7>
English:

Who are teaching comp212?

SQL:

```
SELECT  TEACHERID,TEACHERNM
FROM    TEACHER,TEACHER_COURSE,COURSE
WHERE   TEACHER.TEACHERID = TEACHER_COURSE.TEACHERID   AND
        COURSE.COURSEID = TEACHER_COURSE.COURSEID   AND
        COURSEID = "comp212";
```

GQML:

```
lnj  TEACHER,
      lnj  COURSE, TEACHER_COURSE;
  where  COURSEID = "comp212"
  attrs  TEACHERID, TEACHERNM;
```

<8>
English:

Which courses are taken by Russel?

SQL:

```
SELECT  COURSEID,COURSENM
FROM    COURSE,STUDENT_COURSE,STUDENT
WHERE   COURSE.COURSEID = STUDENT_COURSE.COURSEID   AND
        STUDENT.STUDENTID = STUDENT_COURSE.STUDENTID   AND
        STUDENTNM = "Russel";
```

GQML:

```
lnj  COURSE,
     lnj  STUDENT, STUDENT_COURSE;
  where  STUDENTNM = "Russel"
  attrs  COURSEID, COURSENM;
```

<9>
English:

Which courses does Russel take?

SQL:

```
SELECT  COURSEID,COURSENM
FROM    COURSE,STUDENT_COURSE,STUDENT
WHERE   COURSE.COURSEID = STUDENT_COURSE.COURSEID   AND
        STUDENT.STUDENTID = STUDENT_COURSE.STUDENTID   AND
        STUDENTNM = "Russel";
```

GQML:

```
lnj  COURSE,
     lnj  STUDENT, STUDENT_COURSE;
  where  STUDENTNM = "Russel"
  attrs  COURSEID, COURSENM;
```

<10>
English:

List all the student taking comp212.

SQL:

```
SELECT  STUDENTID,STUDENTNM
FROM    STUDENT,STUDENT_COURSE,COURSE
WHERE   STUDENT.STUDENTID = STUDENT_COURSE.STUDENTID   AND
        COURSE.COURSEID = STUDENT_COURSE.COURSEID   AND
        COURSEID = "comp212";
```

GQML:

```
lnj  STUDENT,
     lnj  COURSE, STUDENT_COURSE;
  where  COURSEID = "comp212"
  attrs  STUDENTID, STUDENTNM;
```

APPENDIX F

SYSTEM MODULE DESCRIPTIONS

File: main.c
Description: This file includes the major function of the system which controls the execution of the natural language interface.

File: par1.c
Description: This file includes the following functions:

SENTENCE_READER: reads English sentences from the system terminal or a specified file.

SYNTACTIC_MARKER_PRODUCER: searches the lexicon and finds a lexicon entry for each word in the input sentence.

INTEPRETATION_PRODUCER: produces all the possible lexical interpretations.

INTEPRETATION_FINDER: gets a lexical interpretation and passes it to the syntactic analyzer.

PHRASE_MARKER_PRODUCER: analyzes the lexical interpretation with the bottom-up parsing strategy and produces phrase structures.

MODALITY_PRODUCER: constructs a modality for each verb phrase.

NOUN_PHRASE_COMBINER: deals with the conjunction problem.

File: par2.c
Description: This file includes the following functions:

MODALITY_FINDER: analyzes a verb phrase and produces the information required for constructing the modality.

PARTICIPLE_PHRASE_MODIFIER: modifies the present and past participle phrases and converts them into relative clauses.

File: par3.c
Description: This file includes the following functions:

LEXICON_BUILDER: reads the lexicon definition from the file lexicon.txt and establishes the internal representation of the lexicon.

GRAMMER_CHECKER: analyzes the phrase structures with the top-down parsing strategy and produces the syntactic representation of a sentence.

SYNTAX_CHECKER: performs the verification of the word syntactic definition between its word-string notation and bit-string notation.

SEMANTICS_CHECKER: performs the verification of the word semantic definition between its word-string notation and bit-string notation.

File: par4.c
Description: This file includes the following function:

CASE_FRAME_BUILDER: accepts the syntactic analytical result, performs the semantic analysis, and produces the general meaning representation.

File: par5.c
Description: This file includes the following functions:

QUERY_INTERPRETER: interprets the general meaning representation and produces the query meaning representation.

SQL_GENERATOR: accptes the query meaning representation and constructs SQL query specification.

File: lexicon.txt
Description: This file includes all the lexicon definition used in the NLI system.

File: parser.h
Description: This file includes all the system data structure definition used in MIDBMS.

File: table.h
Description: This file includes system parameters.

File: syntax.h
Description: This file includes all the syntactic class definition used in MIDBMS.

File: semantics.h
Description: This file includes all the semantic class definition used in MIDBMS.

File: machine.h
Description: This file includes the grammar definition which is represented with an augmented finite state transition network.

File: employee.h

Description: This file includes the database domain specific knowledge base which is about employees.

File: university.h
Description: This file includes the database domain specific knowledge base which is about university.

File: mapping.h
Description: This file is used to contain a database domain specific knowledge base which is either the employee.h or university.h in the current system.

File: tran.c
Description: This file includes all the functions used in the GQML translator.

APPENDIX G

MIDBMS USER'S GUIDE

1) Compiling:

MIDBMS can be compiled either with Apollo C on Apollo system or with TCC on IBM-PC personal computer system. The following MAKEFILE will help users to finish the system compiling and linking and finally produce two executable files. On Apollo system, they are PARSER and TRAN. On PC system, they are PARSER.EXE and TRAN.EXE.

```
makefile:

# usage:
#       make emp  - to make parser for interpreting employee queries
#       make univ - to make parser for interpreting university queries
#       make      - normal make
#   or make {module}

# on unix, CC=cc. on pc CC=\lang\tc\tcc
CC=cc
CFLAGS=
TOUCH=touch

all: parser tran

emp: cpemp parser
univ: cpuniv parser
cpemp:
cp employee.h mapping.h
$(TOUCH) mapping.h
cpuniv:
cp universi.h mapping.h
$(TOUCH) mapping.h

tran: tran.c
$(CC) -g -o tran tran.c
par1.o: parser.h syntax.h semantics.h machine.h par1.c
$(CC) -g -c par1.c
par2.o: parser.h syntax.h semantics.h machine.h par2.c
$(CC) -g -c par2.c
par3.o: parser.h syntax.h semantics.h machine.h table.h par3.c
$(CC) -g -c par3.c
par4.o: parser.h syntax.h semantics.h machine.h par4.c
$(CC) -g -c par4.c
```

```
par5.o: parser.h mapping.h par5.c
$(CC) -g -c par5.c
parser: parser.h syntax.h semantics.h machine.h main.c par1.o par2.o
        par3.o par4.o par5.o
$(CC) -g -o parser main.c par1.o par2.o par3.o par4.o par5.o
```

2) Running:

On either Apollo system or PC system, users can give following commands to run the system:

a) parser ⟨return⟩: The parser accepts English sentences and produces SQL query specifications. The input and output are terminal.

b) tran ⟨return⟩: The tran accepts SQL query specifications and produces GQML query specifications. The input and output are terminal.

c) parser input_file output_file ⟨return⟩: The parser reads English sentences from a specified input file and puts SQL query specification into specified output file.

d) tran input_file output_file ⟨return⟩: The tran reads SQL query specification from a specified input file and puts GQML query specification into specified output file.

e) parser | tran ⟨return⟩: The parser accepts English sentences and produces SQL query specifications. This result is passed to the tran through the "pipe". The tran accepts SQL and translates it into GQML. The GQML query specifications are given on terminal.

f) parser -q output_file⟨return⟩: The parser accepts English sentences from the terminal and produces SQL query specifications. This result is written into output_file.sql. Then, parser will call tran which reads SQL from output_file.sql, translates SQL, and writes GQML into output_file.gql. Then, HDDBMS is invoked to get the final database result.

g) parser -vq output_file⟨return⟩: This command has the same effect with the command f) except that the intermediate results of the processing is printed out on terminal.