A Data Management Strategy for Transportable Natural Language Interfaces

By

JULIA ANN JOHNSON

B.Sc., The University of Alberta, 1974
M.Sc., The University of British Columbia, 1980

A THESIS SUBMITTED IN PARTIAL FULFILLMENT OF
THE REQUIREMENTS FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

in

THE FACULTY OF GRADUATE STUDIES
(Department of Computer Science)

We accept this thesis as conforming
to the required standard.

THE UNIVERSITY OF BRITISH COLUMBIA

June 1989

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

Department of Computer Science

The University of British Columbia
Vancouver, Canada

Date October 13, 1989

Abstract. This thesis focuses on the problem of designing a highly portable domain independent natural language interface for standard relational database systems. It is argued that a careful strategy for providing the natural language interface (NLI) with morphological, syntactic, and semantic knowledge about the subject of discourse and the database is needed to make the NLI portable from one subject area and database to another. There has been a great deal of interest recently in utilizing the database system to provide that knowledge. Previous approaches attempted to solve this challenging problem by capturing knowledge from the relational database (RDB) schema, but were unsatisfactory for the following reasons: 1.) RDB schemas contain referential ambiguities which seriously limit their usefulness as a knowledge representation strategy for NL understanding. 2.) Knowledge captured from the RDB schema is sensitive to arbitrary decisions made by the designer of the schema. In our work we provide a new solution by applying a conceptual model for database schema design to the design of a portable natural language interface. It has been our observation that the process used for adapting the natural language interface to a new subject area and database overlaps considerably with the process of designing the database schema. Based on this important observation, we design an enhanced natural language interface with the following significant features: complete independence of the linguistic component from the database component, economies in attaching the natural language and DB components, and sharing of knowledge about the relationships in the subject of discourse for database schema design and NL understanding.

Contents

List of Tables

List of Figures

Acknowledgements.

Thanks to the members of my thesis committee, Richard Rosenberg, Paul Gilmore, Bob Goldstein, Carson Woo, and David Poole for their significant contributions to the content and organization of my thesis.

Special thanks to my research supervisor, Richard Rosenberg, with whom progress really started and without whose guidance I would never have been able to complete this work.

Thanks to my mother Emily, my sister Genevieve, and my best girl friends Agnes, Terry, and Amanda for their constant faith in me.

Thanks to my friends at UBC Andrew Csinger, Nou Dadoun, Teresa Przytycka, Heidi Dangelmaier, John Demco, Herbert Kreyszig, Rick Morrison, Marc Majka, Ron Rensink, George Tsiknis, Esfandiar Bandari, Barry Brachman, and Brent Boerlage who have made my work fun and contributed to my thesis either directly or indirectly.

Chapter 1

Introduction

Permitting the user of a database (DB) to ask questions in his or her own natural language (e.g., English) is a highly desirable goal. Systems that provide natural language access to databases are usually called *natural language interfaces* (NLI). An NLI accepts a user's questions expressed in natural language, understands the questions, and responds to them by extracting answers from the DB and presenting them to the user.

The *domain*[1] of an NLI is the subject about which users of the interface may ask questions. For commercial viability an NLI must be useable on more than one domain and on more than one database. Natural language systems of the 1970s (the LUNAR system, Woods [98], domain: lunar rocks, task: analysis of rock characteristics; the PLANES system, Waltz [90], domain: military airplane repairs, task: airplane performance recording) were

[1]We are using the term *domain* as it is used in the area of natural language systems. The term *domain* is also used in the area of relational DB systems to mean "a set of values". The reader is referred to Section 2.2 for a full definition of the term *domain* and for clarification of its use in the areas of relational DB systems and NL systems.

database and domain specific. By the late 1970s Hendrix et al. built a system (LADDER [45]) that could be customized to a new domain and database by the system designer who would be required to have considerable knowledge about the system and about linguistics. The prohibitive cost of the initial customization and further work if the user requirements change prevented natural language interfaces from becoming widespread. In recent years research has focused on providing an interface that can be customized by a typical database administrator (DBA) who does not have knowledge in linguistics but is an expert on the database. Examples include the PRE system [23] transported from one office domain to another, the Linguistic String Project [64] transported from a medical to a Navy domain, and the Datalog system [39] transported from an office to a housing information domain.

A large amount of domain and database specific information is required for a computer to understand natural language requests for information from a database. The system usually operates in two different modes: the *question/answering mode* in which a user asks questions in natural language and the NLI provides answers, and the *knowledge acquisition mode* in which the DBA provides information to the NLI about the domain and database. Figures 1.1 and 1.2 illustrate, respectively, the question/answering mode and the knowledge acquisition mode for a typical natural language interface. The lines preceded by the symbol '$\gg$' are input by the DBA or the user. The others are output by the NLI. For ease of illustration we have assumed in Figure 1.2 that there is at most one type of relationship between any two types of entities.

The *transportability* of a natural language interface is the ease with which it can be adapted to a new domain or database. Henceforth, the term *transportable* will be abbre-

≫ Who are the professors in the computer science department?
P. Jones
L.A. Lawrence
J. Smith
≫ Does Professor Jones teach a math course?
No
≫ Print the courses taken by Bill Black.

Number	**Name**
CPSC100	Programming and Problem Solving
Math115	Calculus II
Chem201	Chemistry of the Non-transition Elements

Figure 1.1: Question/Answering in a Typical NLI

For which database is knowledge to be acquired?
≫ university
What are the entities of interest in the university database?
Separate names of different entities by blanks.
≫ students departments professors courses
What English words and phrases are used to refer to "students"?
Enclose phrases in quotations.
≫ pupil
What entities are related to "students"?
≫ courses professors departments
What English words and phrases are used to refer to the relationship between "students" and "courses"
≫ take "are enrolled in" study
What English words and phrases are used to refer to the relationship between "students" and "professors"
≫ "are supervised by" "are advised by"
⋮
What adjectives are used to describe "students"?
≫ graduate good "first class" "computer science"

Figure 1.2: Knowledge Acquisition in a Typical NLI

viated to *portable*. There have been several ways in which researchers have improved the portability of an NLI:

1. The language that the system can understand is restricted [23]. The description of a restricted language (using a traditional grammar, for example) is expected to be simpler than for non-restricted languages, and therefore, the job of providing the grammar is simpler. Item (2) below describes a competing approach.

2. A grammar is used that generates a large class of natural language requests [64]. If a grammar generates a small class of requests, then it will be unlikely to apply to more than one domain. Therefore, a different grammar will be needed for each new domain.

3. The NLI is designed in a modular fashion [39]. For example, the classical approach of separating syntax from semantics results in a syntactic component that is largely domain independent.

4. A *customization program* is provided that interacts with the DBA in a user friendly way prompting him or her for information about the domain and database [82, 3, 19].

Portability is achieved in Harris' INTELLECT system [41] (commercially available NLI for retrieval of office information) by exploiting knowledge that is already available in the database. Harris proposed the idea of obtaining membership information from the DB (information about which database values are located in which columns). This information is needed by the NLI for understanding natural language requests and is typically provided

by the DBA during customization. Portability is improved because less information must be provided by the DBA.

Although Harris' results are worthwhile, further work is needed on exploiting the capabilities associated with the DB to obtain portable natural language interfaces. In this thesis we expand Harris' results by applying the SET conceptual model, introduced in [32] for database schema design,[2] to the design of a natural language interface. Portability is enhanced because much of the same information can be used for both purposes.

A picture of the overall design of our proposed system is given in Figure 1.3. The box labeled DMS denotes software and a schema for the metadatabase[3] (a metaschema) that implement a data management strategy to obtain transportable NL interfaces. The DMS includes tables for storing SET schemas, and operations for adding, deleting and changing SET schemas. The arrows in the figure denote data flow. The natural language interface (NLI) transforms the NL input to a query formulated in terms of the SET schema. The DMS transforms that query to a relational database (RDB) query. The RDB query is executed against the RDB and the result (a set of zero or more tuples) is returned to the DMS. In this thesis, we focus on the process of formulating a database query that answers the natural language input request. However, for completeness, a brief description of the process of formulating a natural language response follows: The DMS reformulates the tuples returned from the RDB query to conform to the view of the domain expressed by the SET schema.

[2]A DB *schema* describes a particular organization of the data in the database. A function required to get the DB operational is to construct or *design* the DB schema.

[3]The *metadatabase* of a DB system is part of the DB that records knowledge about the DB itself. In particular, it contains the DB schema.

In the case of an empty result (zero tuples), the DMS searches the metadatabase passing to the NLI information needed for formulating an informative natural language response. An informative response to the request 'Which students passed CPSC101', in the event that the course is not offered, would be 'CPSC101 is not offered this term. The next offering is Fall 1990'. In contrast, an uninformative answer would be 'none'.

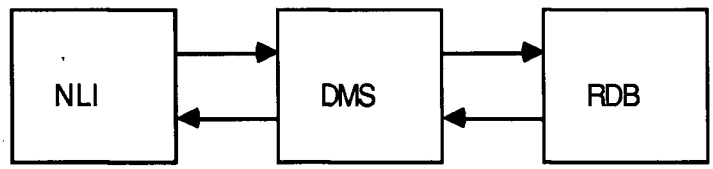


Figure 1.3: A Data Management Strategy for Portable NLI

Our work differs substantially from previous work [40, 51, 93, 55] in which knowledge for NL processing is captured directly from the RDB schema. SET schemas are a refined version of Entity-Relationship diagrams [12] which have become part of common practices for database schema design. Methods exist for translating a SET schema to a relational schema with additional integrity constraints as suggested by the following diagram [33]:

$$| \textit{SET schema} | \longrightarrow | \textit{Relational schema} + \textit{integrity constraints} |$$

Existing relational database systems do not support the additional integrity constraints generated by even a moderately complex SET schema. One kind of constraint, *referential integrity constraints*, are important for formulating a precise statement of the meaning of an NL database request and, whereas SET schemas capture this kind of constraint, RDB schemas do not. Therefore, the SET schema is a more useful source of knowledge for NL

processing than the RDB schema. Furthermore, as will be shown in Section 4.5.1, the information is invariant to the arbitrariness of the design of the SET schema, whereas the information extracted from RDB schemas is sensitive to arbitrariness of the design of the RDB schema.

Thesis Overview

Chapter 2 introduces important terminology, and presents background concepts that are important for understanding the need for a DMS. Specific linguistic problems are introduced that are shown (later in the thesis) to benefit from knowledge that is applied for designing the relational database (RDB) schema.

Chapter 3 introduces a taxonomy of ways in which an NLI can be portable with respect to the domain and database. The taxonomy is used as a framework within which previous approaches to obtaining portable NLI are examined. In addition, Harris' ideas for using the DB system to provide knowledge for natural language understanding are reviewed.

Chapter 4 describes knowledge available from the SET schema that is useful for natural language understanding. It shows how the approaches which capture knowledge from the RDB schema are inferior to the new one of capturing knowledge from the source used to design the RDB schema. This is the main thesis being defended in my dissertation. The chapter also presents examples from three different domains (a university domain, a hospital domain, and a library circulation domain) which illustrate how the knowledge can be used to solve the linguistic problems introduced in Chapter 2.

Chapters 5 and 6 discuss some of the issues associated with the implementation of

our design strategy. They include an overall design for a transportable natural language interface, specification of the interfaces between the NLI and the DMS, heuristics for solving the linguistic problems previously introduced, and expansion of the metadatabase of a standard RDB system to permit it to describe the knowledge needed for natural language processing.

Chapter 7 presents conclusions and directions for further research.

Appendix 1 provides examples of the representation of the meaning of NL database requests using the language DEFINE (introduced by Gilmore in [31]). This language has been selected in the design of a portable NLI because it permits an intermediate representation of the NL request that is independent of the DB structure, a feature that is widely understood to be important for portability.

Appendix 2 illustrates a procedure for translating an NL request to a representation of the meaning of the request expressed using the language DEFINE. The formalism for the translation is Woods' Procedural Semantics [95], and the contribution here has been to adapt the formalism to the new target language DEFINE.

Appendix 3 describes a natural language interface (ALPS [93]) that has been designed to be portable, and that we take as a starting point from which additional portability is provided by exploiting database facilities.

Chapter 2

Background and Thesis Overview

This thesis focuses on identifying knowledge associated with the DB that would be useful for automatically understanding natural language and on designing a portable natural language interface that takes advantage of that knowledge. This chapter provides background for understanding our design strategy and it points out the main results of the thesis. In Sections 2.1-2.3, terminology and background concepts are introduced which leads into a discussion in Sections 2.4-2.6 of the significant contributions of the research.

2.1 Important Linguistic Problems

We are concerned with facilitating communication with a database because the common database query languages such as SQL are inadequate. They have an awkward syntax that makes it difficult for a naive user to communicate his or her request, and they are artificial languages requiring an initial start up time for learning them. Both of these problems would

be alleviated if we could communicate with a database using our own natural language.[1]

In this section five specific linguistic problems that must be handled by any reasonably robust NLI are presented. They are modifier attachment, semantic ambiguities,word sense disambiguation, ellipsis, and conjunction scoping. In Chapter 4 we concentrate on how knowledge associated with the DB can be used to solve three of the given problems, and in Chapter 7 we consider how our methods might be extended to handle the remaining two.

Modifier Attachment (MA)

There are a variety of situations in natural language where one sentence constituent modifies another. Consider the noun phrase "the book on the table with a red cover". There are two possible parses for the phrase as illustrated in Figure 2.1. In the figure nonterminal symbols NP, DET,PP, N, and PREP stand for noun phrase, determiner, prepositional phrase, noun, and preposition, respectively. In P1 the phrases "on the table" and "with a red cover" *both* modify the head noun "book". In P2 the phrase "with a red cover" modifies the noun "table", and the phrase "on the table" modifies the head noun "book".

The sentence constituent being modified is called the *referent*, and the one doing the modification the *modifier*.

The MA problem is to select the most appropriate attachment for the modifiers.

[1] An alternate view [23] that will be discussed in detail in Section 3.2 holds that artificial languages are easier to learn than natural languages, and further that the precision of artificial languages leads to the formulation of correct queries for the problem at hand. The thesis advanced in Chapter 4 holds that the SET schema is a superior source of knowledge over the relational schema for processing natural language requests, and this thesis applies equally well to both natural and artificial languages.

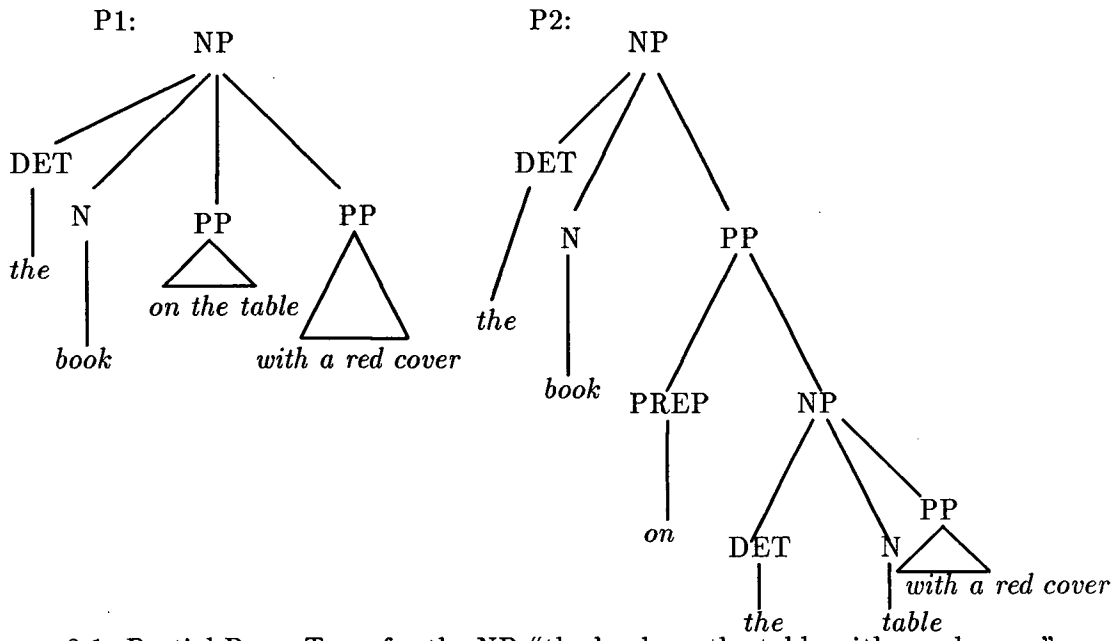


Figure 2.1: Partial Parse Trees for the NP "the book on the table with a red cover"

Semantic Ambiguities (SA)

The term *internal representation* is defined in Section 2.3 of this chapter, but for now we will say that it is a formal representation of the meaning of a sentence. Semantic ambiguities arise when a single syntactic structure (parse tree) maps into more than one internal representation. For example, in a university domain a request such as "Dr. Lee's students" could have any of the following meanings:

1. Students in the same department as Dr. Lee

2. Students enrolled in courses taught by Dr. Lee

3. Students supervised by Dr. Lee

The SA problem is to select the most appropriate meaning for a natural language request.

Word Sense Disambiguation (WSD)

Word sense ambiguity arises when a word in a sentence has more than one meaning. Given the sentence "Students run programs", for example, the noun *program* might mean either a computer program or a recreational program, in which case there would also be two senses for the verb *run* : to *execute* a computer program, and to *administer* a recreational program. The WSD problem is to select exactly one meaning for each word in the sentence.

Ellipsis (E)

An ellipsis is an utterance in which one or more of the phrases have been omitted. Consider the following pairs of requests (from [10]):

1. Who are the professors of Computer Science?

   of Mathematics?

2. Who are the students who passed Computer Science?

   The failures?

Ellipses are typically handled by finding a syntactic or semantic parallel between the request in which one or more phrases have been ellided and a previous request in the dialog. A syntactic method for handling ellipses is illustrated by the first pair of requests. The second request of the pair can be interpreted by substituting "of Mathematics" for "of Computer Science" in the parse tree for the first request. Neither a syntactic nor a semantic method has permitted the type of ellipsis illustrated by the second pair of requests to be adequately handled. In the second pair of requests there is neither a syntactic nor a semantic

parallel. The collection of objects retrieved by the second request is the mathematical complement of that retrieved by the first. The E problem has at least three subproblems: First, the request must be recognized as an ellipsis. Second, it must be decided which of the various types of ellipses has occurred. Third, the elliptical fragment must be completed by filling in missing phrases. The resulting sentence can be processed like a normal (non-elliptical) request.

Conjunction Scoping (CS)

The scope of a conjunction is a specification of the arguments of the conjunction. Consider the following example from [56]:

1. Which presidents visited New York and New Jersey after leaving office?

2. Which presidents died in New York and New Jersey after leaving office?

Our knowledge that two different cities may be visited by a president at two different times leads use to admit as a possible meaning of (1) the narrow scope reading

Which presidents visited both New York and New Jersey after leaving office?

Our knowledge that presidents die at most once and that they are not omnipresent leads us to exclude the narrow scope reading of (2) in favor of the wide scope reading

Which presidents died in New York after leaving office, and
which presidents died in New Jersey after leaving office?

In the wide scope reading the word "and" is being used to express logical disjunction. The complexity of the problem of conjunction scoping increases rapidly as the natural language request containing a conjuction becomes more complex. The request

Which departments have employees who drive red and blue cars?

has the following possible readings:

R1: Which departments have employees who drive cars that are both red and blue?

R2: Which departments have employees who drive both a red car and a blue car.

R3: Which departments have both employees who drive a red car and employees who drive a blue car.

The CS problem is to define the scope of conjunctions in the NL request.

2.2 Terminology

The above discussion has described important linguistic problems which demand solutions from the area of database systems. To provide solutions we must first introduce some terminology. Unfortunately, many terms have previously been used in both areas (natural language and database), but with different meanings. For purposes of clarity we will keep terminology in the two areas separate, and where necessary to avoid confusion, introduce our own terms.

2.2.1 Domain

In the area of relational database systems the term *domain* means a set of values associated with a column of a relation. All values in a column are constrained to be members of its domain.

In the area of natural language understanding the term *domain* is taken to mean the subject that forms the content of the dialog between a person who formulates natural language requests and the system that understands and responds to those requests. The domain of a natural language interface comprises a collection of facts from which answers to database requests are drawn.

To avoid confusion we use the term *domain* in the sense that it is used in the area of natural language understanding, and we use the term *value set* to mean *domain* in the relational database sense.

2.2.2 Dictionary

In the area of database the term *data dictionary* refers to a database that records information about the database system. In modern database systems the data dictionary is "integrated" into the database, which is to say that it forms part of the database, and that both the data dictionary and the database are queried through the same interface (the query language provided by the database system). In earlier systems the database and the data dictionary were separate, each requiring its own interface for access.

In the area of natural language understanding the term *dictionary* (sometimes "linguistic dictionary" or "lexicon") is used to mean a repository of syntactic and semantic information

about words that is used for automatic understanding of natural language sentences. An NLI comes with an initial dictionary containing descriptions of words that are common across domains. The initial dictionary is usually extended for use in a particular domain by adding domain specific words and domain specific definitions of words.

To avoid confusion we will use the term *metadatabase* to refer to the data dictionary used by the database system and the term *lexicon* to refer to the lexical dictionary used by the NLI. The metadatabase and the lexicon are different entities that have arisen independently in the two different areas, although as we shall see they could well be integrated into one source of information for the NLI.

2.2.3 Language

The term *language* is potentially confusing because we have a *query language* that is used by the database system to access the database, a *natural language* in which humans communicate with each other, and finally a *sublanguage* (of a natural language) in which the user of an NLI formulates requests of a database. In this thesis the unqualified term *language* will never be used to mean query language of a database system or full natural language used by humans for communicating with each other. The unqualified term *language* will be used only when the term *sublanguage* is intended.

The sublanguage of an NLI is a subset of the sentences of some natural language. The user of an NLI does not know its sublanguage but interacts with the system as if it is operating in full natural language. A formal language is characterized by a grammar, but the processes involved in natural language understanding (reference resolution, word sense

disambiguation, handling ellipsis, to name a few) and in forming appropriate responses (correcting false user presuppositions, making scalar implicatures) are hard to characterize using a grammar. Every NLI has an implicit sublanguage even though we are not able to explicitly generate its sentences.

Knowledge about the structure of a sentence is represented in a *parse tree* (also called a *phrase marker* because it serves to mark the phrases of the sentence). A *sentence constituent* is any subtree of the sentence's parse tree. A parse tree for the sentence "Jack slept on the table" taken from [11] is given in Figure 2.2. The sentence constituents are "the table", "on the table", "slept on the table" and the complete sentence "Jack slept on the table". In addition, each of the words "Jack", "slept", "on", "the", and "table" are sentence constituents. A sentence constituent that appears in the parse tree as a terminal node is called a *primitive constituent*. Primitive constituents are not necessarily single English words. For example, the adjective "Computer Science" is a multiple word primitive constituent in the university domain.

The term *deep structure* comes from Chomsky's transformational grammar formalism [13] for describing the structure of English language. A transformational grammar includes a *base component* that generates certain basis sentences, and a transformational component that generates sentences that are derived from the basic ones. Deep structure phrase markers are those generated by the base component [14].

A *natural language database request* has one of the following forms:

1. a yes/no question (e.g., Does Jones take CPSC 101?)

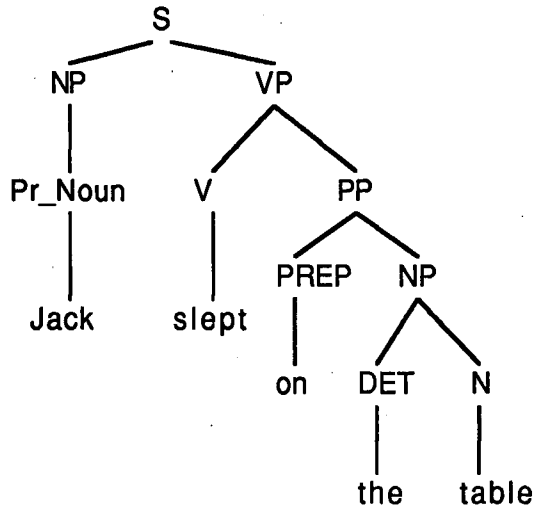


Figure 2.2: Parse Tree for the Sentence "Jack slept on the table."

2. a print question (e.g., Print (Show me) (Who are) the professors in the computer science department.)

3. a question that requests the application of a function (e.g., count, average, sum) to the results of a database query (e.g., How many (Count the) professors in the computer science department.)

An NL DB request specifies three things: the type of the objects to be retrieved, conditions that must be satisfied by the objects, and an operation (e.g., print, test, count) that tells how the objects should be processed upon their retrieval. For each of forms (1), (2), and (3) the NL DB request could be a fragment which omits one or more of the object, operation, or condition (e.g., professors in the computer science department).

An *internal representation* (IR) for a natural language request is a statement of the conditions under which the NL request is satisfied [1]. A possible IR for the request "Print

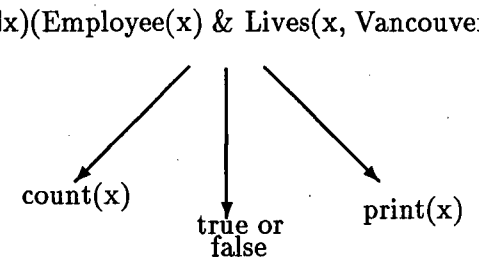


Figure 2.3: A Possible IR for the Request "Print employees who live in Vancouver"

employees who live in Vancouver" is illustrated in Figure 2.3. The IR does not specify whether the result should be counted, evaluated as true or false, or printed.

2.2.4 Knowledge Representation

A knowledge representation strategy (KRS) is a way of representing knowledge about the real world. A *model*, for our purposes, is a KRS in which knowledge is represented as a collection of primitive objects and complex objects constructed from the primitive ones. For example, in the relational model virtual relations are constructed from more primitive base relations, and base relations from more primitive value sets.

Initially, let us assume that we are able to freeze the world in time. A *schema* is a particular organization of raw data into a meaningful description of some part of the real world. A relational schema, for example, comprises a particular collection of value set names V, relation names R, and a function $F : R \rightarrow V^{*}$[2] which associates with each relation name a tuple of value set names. A schema is based on a model which is to say that the KRS of

[2] V^{*} is the set of all tuples of any length over V. If $\mid V \mid$ denotes the cardinality of V and V^{n} is the set of all n-tuples over V, then $V^{*} = \bigcup_{n=1}^{\mid V \mid} V^{n}$. For example, if $V = \{1, 2, 3\}$ then

$V^{*} = \{1, 2, 3, (1, 2), (1, 3), (2, 3), (2, 1), (3, 1), (3, 2), (1, 2, 3), (1, 3, 2), (2, 1, 3), (2, 3, 1), (3, 2, 1), (3, 1, 2)\}$.

the model has been used for constructing the schema. The association between a schema and the raw data provides an *interpretation*. For example, in a relational schema each name in V is implicitly associated with a set of data values in the raw data, and each name in R with a subset of the Cartesian product (in the order given by F) of the value sets of the relation named. For example, if $F('Q') = (V_1, \ldots, V_n)$ then an interpretation associates with the relation whose name is $'Q'$ some subset of $V_1 \times \ldots \times V_n$. Figure 2.4 depicts the relationship between the concepts of data, schema, and interpretation. An arrow from A to B labeled by the name of a process illustrates that the input to the process is A and the output is B.

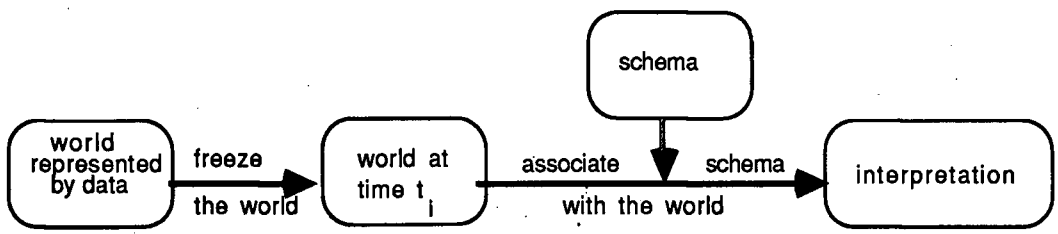


Figure 2.4: The Association between a Schema and Raw Data

A *metaschema* is a description of the characteristics of schemas. In current database systems the metaschema M is itself a schema, and the model upon which M is based is the same as that upon which the schemas described by M are based. If the metaschema is itself a schema, then the data interpreted by the metaschema are called *metadata*. The repository of all metadata is called the *metadatabase*.

Human beings also associate schemas with raw data in the world to permit them to

understand the world. A schema (whether in the head of a human or represented using a computer) together with its raw data is called *knowledge*. Similarly, a metaschema together with its metadata is called *metaknowledge*. The terms knowledge and *information* will be used interchangeably.

Semantic knowledge is knowledge that is useful for automatically interpreting and responding appropriately to NL requests. It need not come from the database. For example, a method for helping to resolve modifier attachment involves examining semantic categories associated with the modifier and each possible referent. A modifier may be attached to a given referent if their semantic categories overlap. Semantic categories provide semantic knowledge which in this example comes from the knowledge acquisition phase.

To assist the user in understanding how to use the KRS of a model a variety of concepts may have to be introduced on top of the model. A *schema design methodology* for a given model is a collection of techniques for constructing schemas based on that model. A schema design methodology may also be concerned with constructing schemas that will guarantee efficient use of storage space when the database is operational.

2.2.5 Portability

Domain portability of an NLI is the ease with which it can be adapted to a new domain. *Database portability* of a NLI is the ease with which it can be adapted to a new database.

2.3 Background

Now that important terminology from the areas of natural language understanding and database systems has been introduced, we are in a position to discuss a number of background topics: In Section 2.3.1 a general paradigm for the modular structuring of an NLI is presented. Later, in Chapter 5, we extend this architecture by providing a more detailed specification of the database components and their attachment to the database. Previous methods which capture knowledge from the structure of the database (as opposed to the approach introduced here of capturing it from the source used to design the structures) are reviewed in Section 2.3.2. In Section 2.3.3 three different strategies for organizing data are presented which are used in Section 2 and in Chapter 5 as a basis for the logical structuring of knowledge associated with the DB that is useful for natural language understanding. Finally, Section 2.3.4 describes an architecture for a standard relational database system that has been proposed by the American National Standards Committee (ANSI [86, 9]), and that we take as the database system upon which the NLI is built.

2.3.1 A Typical Architecture for a Natural Language Interface

A typical architecture for a natural language interface (one that is adopted in this thesis) consists of four main components as illustrated by Figure 2.5 from [54]:

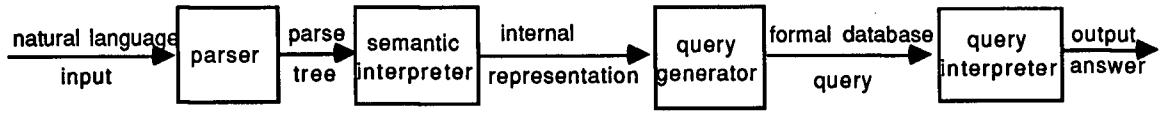


Figure 2.5: A Typical Natural Language Interface

The separation between the syntactic and semantic components is motivated by the view that there is a level of structure in natural language that is independent of semantics (Chomsky 1965 - Aspects of the Theory of Syntax [14]). Semantically similar sentences may vary significantly in their word order. Syntactic analysis removes some of the variation between sentences that is not due to semantic differences. Although this model has been superseded by more recent models [71, 76], it still provides a useful characterization of the natural language phenomena that are of interest in this thesis. An alternate approach (the LADDER system [45]) is the use of a semantic grammar which results in the integration of the syntactic and semantic components. A semantic grammar is based on categories that include semantic information about the domain as opposed to the syntactic categories (noun phrase, verb phrase) of a traditional grammar.

A description of the NLI of Figure 2.5 follows: The parser produces a parse tree for the NL input. The semantic interpreter transforms the parse into an internal representation (IR).

The designers of the TEAM system [36] argue that the language for internal representation should be independent of any particular organization of the database. A database query language represents the meaning of a natural language request in a way that directly reflects the structures of the database schema. Two different databases expressing the same information but organized according to two different schemas may use two different database queries for the same natural language request. For example consider a database of employee information. One database schema might distinguish between male and female employees by using two relations, one to describe female employees and the other to describe male

employees. A different database schema may represent all employees in one relation and use a column that contains 'M' or 'F' to distinguish between tuples of male and female employees. A database query such as one to retrieve all male employees will look quite different for the two different databases.

The query generator transforms the IR to a database query expressed in a standard query language. This query is called the formal database query. The actual database query (not shown in the figure) is a query expressed using the query language of the available database system that has the same meaning as the formal database query. The query interpreter transforms the formal database query to the actual database query, uses the database system to execute the actual database query, and returns the results to the natural language interface.

2.3.2 Semantic Relatedness Measures in Relational DB Schemas

The approach of capturing knowledge for natural language understanding from the DB schema is reviewed here as background for Chapter 4 where we introduce the new approach of capturing knowledge from the source used to design the schema. The latter approach is superior because the knowledge is explicitly available and more accurate than when it is captured indirectly from the structures of the DB.

Semantic relatedness between constituents is the closeness in meaning between the constituents, and it is dependent on the domain of discourse. For example, the nouns *dog* and *bone* may be more semantically related than the nouns *dog* and *computer* in a domain that is concerned with the care and feeding of dogs. If, however, the domain is concerned

with communication patterns, then the nouns *dog* and *computer* may be more semantically related because humans can communicate with dogs and computers, but not bones.

In natural language interfaces to database systems the denotational part of the meaning of a constituent (as opposed to the procedural part) is typically [36, 53, 21] an object in the database schema. The object may be simple or complex (defined in terms of other objects). Semantic relatedness between constituents is typically estimated by measuring the distance through the database schema between the objects denoted by the constituents. Distance between two relations in a relational schema is measured by counting the number of links required to join the relations together.

In a relational database there may be more than one way of joining two relations together. For example, relations P and Q may be joinable through relations X and Y and also through relations R_1, R_2, and R_3. The sequence of relations from P to Q is called a join path. In mapping a natural language question to a corresponding relational database query a decision must be made on which join path to follow when there is more than one possibility. Each join path determines a possible internal representation for the request. The problem of resolving ambiguity is to determine for each interpretation the likelihood that it is the one intended by the user. The likelihood of an interpretation is measured relative to the other interpretations, rather than in absolute terms.

The solution based on link counts is to select the join path that requires the fewest links. [3] For example, to join relations P and Q together through relations R_1, R_2, and R_3

[3] A link corresponds to a join condition in the relational query that produces the join. A join condition is a statement $R.A = S.B$ in the query that links relations R and S together by specifying a join of R on

requires four links. In general, the join of relations R_1 and R_n, $n \geq 2$, through relations $R_2, \ldots, R_{n-1}$ requires $n-1$ links.

The solution based on link counts is based on two assumptions: first, that the constituents in a sentence tend to refer to semantically related objects and, second, that semantically related objects appear close to each other in the database schema where distance between objects is measured by the number of links between them. In Chapter 4 of this thesis, the notion of a link between relations is expressed in terms of more fundamental concepts which permits a better understanding of what is being measured by semantic relatedness measures based on links.

To illustrate the heuristic based on links consider the example (adapted from [93]) which has been previously introduced in Section 2.1: The phrase

Dr. Lee's Students.

can have at least three different meanings in a university domain. To illustrate the heuristic, we will focus on two of them:

1. Students supervised by Dr. Lee

2. Students taught by Dr. Lee

A partial schema for the university domain is given in Figure 2.6: The *Student* relation records for each student the student identifier and the student name. The *Professor* relation records for each professor the professor identifier and the professor name. The

column A with S on column B.

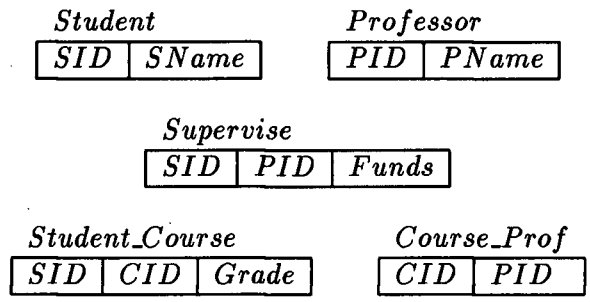


Figure 2.6: Relational Database Schema for the University Domain

Supervisor relation records for each graduate student, the professor that supervises the student's research, and the type of funding the student receives. *Student_Course* records which courses are taken by which students, and the grades obtained by students in courses. *Course_Prof* records which courses are taught by which professors.

Interpretation 1 will require the *Student* and *Professor* relations to be joined through the *Supervisor* relation. Interpretation 2 will require the *Student* and *Professor* relations to be joined through the *Student_Course* and *Course_Prof* relations. For the given domain the heuristic based on link counts rates (2) more favorable than (1) as the interpretation intended by the user.

A measure of semantic relatedness based on the domain (which is described by the database schema) is an important part of a natural language interface to a database system, but other measures are needed as well. In particular, a measure of semantic relatedness based on the context in which the natural language dialog occurs is needed. The subject of context is addressed in Chapter 4 where we present a measure of semantic relatedness based on the SET schema. For more information, the reader is also referred to [20] in which the context of a conversation is modeled by a sequence of queries corresponding to the sequence

of previous requests in the dialog, and to work in artificial intelligence [37, 38] in which, for task-oriented domains, context is modeled by the structure of the task and its subtasks.

2.3.3 Data Models

Four different data models are of concern in the thesis. They are the SET Conceptual model [31], a model that has been specified as part of the thesis and that we refer to as the NF^2 object model, the NF^2 data model [75, 49, 50], and the relational data model [15]. The reader is assumed to be familiar with the relational model. The other three are described here as background for understanding our data management strategy.

The SET Conceptual Model

The SET Conceptual model [31, 32] is intended to provide a single model for what is usually a two step process in the design of a database schema:

1. Enterprise modeling in which knowledge is captured and recorded about the information and processes important to the enterprise, as well as the information exchanges between different functioning subunits of the enterprise.

2. Database schema design for whatever database management system is to be used to record the information obtained in the first step.

For simplicity we will abbreviate the name SET Conceptual model to SET model. It is illustrated in [31], that the SET model can be used to automatically design relational database schemas from the information in (1). The broader purpose of the model is emphasized in that report and especially in the paper [32].

Fundamental Concepts

The fundamental notions in the SET model are those of set and ordered pair. The concept of set is used in other models such as the Entity-Relationship (ER) model [12], however, in the ER model the notion of set is an intuitive one. In the SET model the notions of set and ordered pair are based on the provably consistent set theories of [30] and, hence, a sound foundation is provided upon which the richer models of data needed for NL applications can be built.

The *intension* of a set is a property that determines membership in the set. The *extension* of a set is the membership of the set (the collection of objects that satisfy the intension of the set). The notion of membership is a relation between objects and set extensions. When we say that object x is a member of set A, we mean that x is a member of the extension of A.

The extension of a set changes with time. Given that, the notions of subset and Cartesian product need to be clarified. We understand these notions as they apply to set extensions. If set $S1$ is a subset of set $S2$ then at every time instance t the extension of $S1$ at time t is a subset of the extension of $S2$ at time t. Similarly, at every time instance t the extension of the Cartesian product $(S1 \times S2)$ of sets $S1$ and $S2$ is the Cartesian product of the extensions of $S1$ at time t and $S2$ at time t.

Binary Associations

A *binary association* is a subset of the Cartesian product $(L \times R)$ of two sets L and R. L is called the left parent and R the right parent. The extension of an association with left

parent L and right parent R is a mapping from the extension of L to the extension of R.

In the following discussion the names A and B refer to set extensions. A mapping in addition to being one-to-one, one-to-many, or many-to-many may be either *in* or *on* the source extension and either *into* or *onto* the target extension. In a mapping from A to B the source extension is A, and the target extension is B. The mapping is *on* A if every member of A occurs in the mapping, and it is *in* A otherwise. The mapping is *onto* B if every member of B occurs in the mapping, and it is *into* B otherwise. Each of the 16 possible combinations will be referred to as a mapping type.

Min/max values provide a notation for specifying the mapping type. Associated with each of the source and target extensions of a mapping is a pair of values (p, q) where p has the value either 0 or 1, and q the value either 1 or n. p is called the min value and q the max value.[4] A min value of 0 specifies an *in* or *into* mapping, and a min value of 1 an *on* or *onto* mapping. A max value of 1 specifies a *one* mapping on the source or target extension, and a max value of n a *many* mapping. Table 2.1 shows some of the common mapping types and their specification using min/max values: The source extension is assumed to be

[4]The *standard* values for min are 0 and 1 and for max are 1 and n. More than the standard values may be obtained during database conceptual modeling to serve as design parameters for database implementations. For example, Kumar and Stonebraker [58] use statistics on the sizes of relations to choose the best plan for processing an incoming database query. As a second example, Smith and Genesereth use information about set sizes for estimating the cost of solving a conjunctive problem. The authors state that " a conjunctive problem is a set of propositions which share variables and must be solved simultaneously". A gain in efficiency can be realized if the set of conjuncts is ordered for the problem solving system. Assuming the availability of knowledge about sizes of sets, methods for ordering conjuncts are developed and the overall efficiency of including conjunct ordering in a problem solver is considered.

((1,1),(0,n)) a function on A into B,
((1,1),(0,1)) a one-to-one mapping on A into B,
((1,n),(0,n)) a relation on A into B,
((0,n),(0,n)) a relation in A into B,
((0,1),(1,n)) a relation in A onto B.

Table 2.1: Common Mapping Types Specified Using Min/Max Values

A and the target extension B. The min/max values associated with the source extension of the mapping are given first, and those for the target extension second.

The min/max values of an association apply to its extension at every instance in time. That is, at every instance in time the extension of the association is a mapping of the type specified by the association's min/max values.

m-ary Associations

An m-ary association S is a subset of the Cartesian product $(S_1 \times \ldots \times S_m)$ of not necessarily distinct sets $S_1, \ldots, S_m$. Each of the sets $S_1, \ldots, S_m$ is called a parent set of S and each may itself be an association. For example, the *Manager* association is a binary association which is a subset of the Cartesian product $(Employee \times Employee)$ where *Employee* is a set of employees. The *Manager* association has one distinct set *Employee* which plays a role as two different parent sets. Since the parent roles cannot be distinguished by the name *Employee*, they are referred to as the left and right parent sets.

A tuple $(s_1, \ldots, s_m) \in S$ is called an association entity. An entity $e \in S_i$ *participates* in association entity $(a_1, \ldots, a_m) \in S$ as the i^{th} component if and only if $e = a_i$. If both a and b are members of *Employee*, and a manages b, then the pair (a, b) will be an association entity in the *Manager* association. a participates in (a, b) as the left component, and b

participates as the right component.

For each parent set S_i the *min/max value* of S on S_i is a pair of values (p,q) where p is the minimum and q is the maximum number of association entities in S in which any given entity in S_i participates. An m-ary association has m min/max values. Usually only the values 0 and 1 are used for the min value and 1 and n for the max value. A max value of n means 'one or more'. A min/max value of $(0,n)$ for the *Manager* association on the left parent set states that there may be employees who are not managers, and that a manager manages one or more employees. The min/max values have been widely studied with respect to their role in database schema design. They receive the name least and maximum participation in [57], lower and upper degree in [32], minimum and maximum cardinality in [84], and min/max value in [79].

The SET Schema

Every set must be explicitly declared which involves giving it a name and supplying other information such as its *intension* and *min/max values*. *Primitive sets* are those whose members are not drawn from some previously declared set. Every non-primitive declared set is a subset of the Cartesian product of one or more previously declared sets. The SET schema comprises all the declared sets of an enterprise.

For each primitive set there exists a set of computer representable surrogates and a one-to-one correspondence between the surrogate set and the primitive set. Members of the surrogate set rather than the entity set appear in the database. For sets of computer representable entities the set itself can serve as the surrogate set. Such sets are called *value*

sets, and those that need a surrogate set different from themselves for representating their members are called *non-value sets.*

The intension of a set may be expressed either in a natural language intended to be read by humans or in a formal language that can be interpreted by a machine. Sets with the former type of intension are called *base sets* and with the latter type are called *defined sets*. Value sets such as those usually provided by a programming language (integer, real, character) are assumed to be pre-declared and each is considered to be a defined set. The language DEFINE is introduced in [31] as part of the SET model for expressing the intensions of non-primitive defined sets. The key point to recognize about base sets is that they cannot be defined in terms of other declared sets. The parent sets of a base set may be defined sets, but the selection of its members from the Cartesian product of its parents sets requires human intervention. All non-value primitive sets are base sets.

The language DEFINE is reviewed in Appendix A and used there to represent different types of noun modifiers. The method assumed for translating the parse tree for a request to its internal representation (IR) is Wood's procedural semantics [95], but adapted to use a new target language (the language DEFINE rather than the functional calculus used by Woods).

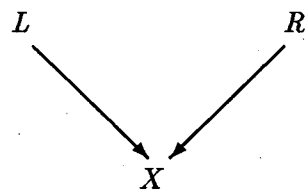
The advantage of the language DEFINE over the functional calculus is that it permits new sets to be defined in terms of previously declared ones. The advantage of defined sets for portable natural language interfaces is a greater simplicity in the semantic rules used for building the internal representations of requests, and hence, an enhancement in portability. The semantic rules are simplified because the name of a defined set and not its

full definition, appears in every rule which uses the set. If the full definition must appear, then the rules are longer and they contain redundant segments.

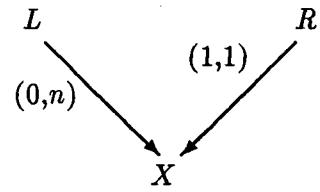
An example of the application of Wood's framework to a particular domain is given in Appendix B, and examples of IRs expressed in the language DEFINE appear throughout Chapters 4 and 5.

The Domain Graph

A collection of sets and associations can be represented as a directed graph in the following way: The nodes of the graph are labeled with the names of sets and associations. An association X with left parent L and right parent R is denoted by a pair of directed edges as illustrated by the following figure:



For the examples given in the thesis the left parent always appears on the page to the left of the right parent. Min/max values for the association are given as labels on the edges. Min/max values associated with the left parent label the left edge and those associated with the right parent, the right edge. For example, the following min/max values for association X state that the extension of X is a function from the extension of L to the extension of R and that it is an *on* and *into* mapping.



A domain graph [31] is "a graphic representation for the inclusion relationships that exist between sets and their subsets, and between subsets of cartesian products of sets and the sets forming the product."[33] The origins of the domain graph are to be found in the entity relationship diagrams of the entity relationship approach to database conceptual modeling[12].

The domain graph (DG) is a directed-acyclic graph. Association S with parent sets $S_1, \ldots, S_m$ is denoted by $m+1$ vertices labeled $S, S_1, \ldots, S_m$ and m directed edges $(S_1, S), \ldots, (S_m, S)$. Direction on an edge indicates the parentage of sets. The edge (S_i, S) directed from S_i to S indicates that S_i is a parent set of S. A directed edge (S_i, S) of the DG is labeled with the min/max value of S on S_i.[5] A DG for the University domain appears at the end of this chapter as Figure 2.15.

[5]A notation similar to the domain graph appears in [16] where it is called the association graph and in [88] where it is called the database graph. The association graph of [16] is represented as a first normal form binary relation AG. A pair (e, a) belongs to AG if e names a primitive set and a names an association for which e is a parent set. The association graph is a restriction of the domain graph because parent sets of associations cannot themselves be associations. No such restrictions are imposed in the database graph of [88]. Although the database graph is an undirected graph, different types of vertices are distinguished (entity set, association) which permits the edges to be treated as if they are directed. In contrast with the domain graph, the association graph has no edge labels, and the edges of the database graph are labeled with max values but not min values.

DEPARTMENTS					
DNO	MGR	PROJECTS			
		PNO	PNAME	MGR	CITY
314	516	17	CGA	582	VAN
		23	HP	621	LA
		29	AB	582	LA
218	713	18	NBX	582	WA
		37	NFL	621	LA

Figure 2.7: An NF^2 Data Relation

Assumptions

It is assumed that min/max values are available for labeling the edges of the DG. A second assumption is that the min/max values assigned to the edges of the DG are consistent; that is, that extensions exist for the sets named in the DG for which the constraints expressed by the min/max values are satisfied.

The NF^2 Data Model

The Non-First Normal Form (NF^2) data model [75, 49, 50] is obtained by extending the relational model by permitting elements of relations to themselves be relations. In the example of Figure 2.7 the DEPARTMENTS relation has columns DNO, MGR, and PROJECTS. The PROJECTS column in its role as a relation has columns PNO, PNAME, MGR, and CITY. An NF^2 relation is a hierarchy of relations. The relation at the top of the hierarchy is called the root relation and the others are called subordinate relations. A relation with no column whose elements may also be relations is called a leaf relation.

Column names are unique within the immediate subordinates of any relation. *Full path names* for columns are obtained by naming all the relations in the hierarchical path starting

at the root and moving to subordinate relations. The name $R[A]$ references column A of relation R. If A is itself a relation then the notation can be used again to reference a column (say B) of A. $R[A][B]$ parses as $(R[A])[B]$.

The first normal form requirement of the relational model imposes on relations the condition that column values are indivisible. An NF^2 relation is not in first normal form because column values may have a substructure. Such relations are said to be unnormalized.

The NF^2 Object Model

The NF^2 object model is an expansion of the NF^2 data model to include the concept of an entity. In the NF^2 data model the user must define his or her own keys. In the NF^2 object model, user defined keys are permitted but, in addition, the system implements a key for each non-leaf relation by associating with each a set of surrogates used to denote entities.

A relation is viewed as describing one type of entity. In the NF^2 data model, if the name $R[A]$ references a column which is itself a relation, then an inclusion constraint involving $R[A]$ (e.g. $R[A] \subseteq \ldots$) is not meaningful. In the NF^2 object model, inclusion constraints of this form are meaningful because the name $R[A]$ refers not to the column A of R but to the set of entities describe by that column.

Figure 2.8 illustrates the NF^2 object relation that corresponds with the NF^2 data relation of Figure 2.7. The columns *Dept* and *Proj* contain surrogates which denote department and project entities, respectively. The entity surrogates are generated and maintained by the system. They are not part of the users view of the database, which is to say that the columns

DEPARTMENTS							
Dept	DNO	MGR	PROJECTS				
			Proj	PNO	PNAME	MGR	CITY
*d*1	314	516	*p*1	17	CGA	582	VAN
			*p*2	23	HP	621	LA
			*p*3	29	AB	582	LA
*d*2	218	713	*p*4	18	NBX	582	WA
			*p*5	37	NFL	621	LA

Figure 2.8: An NF^2 Object Relation

Dept and *Proj* are actually hidden from the user. The user references column *Dept* by the name DEPARTMENTS, and column *Proj* by the name DEPARTMENTS[PROJECTS]. Suppose that the schema includes a name COMPANIES[PROJECTS]. The following statement is an example of an *inclusion constraint*, and this particular one states that (in the database associated with the schema) every entity described by the DEPARTMENTS[PROJECTS] column is also described by the COMPANIES[PROJECTS] column:

$$\text{DEPARTMENTS[PROJECTS]} \subseteq \text{COMPANIES[PROJECTS]}$$

The inclusion constraint is meaningful only if the columns referred to on either side of the symbol $\subseteq$ describe the same types of entities (e.g., projects). The NF^2 object model is described in greater detail in Chapter 6.

2.3.4 An Architecture for a Current Database System

We take as a current architecture for database systems the one described in [63]. The motivation for choosing this one is that it is a standard architecture, and we wish to assume only standard database capabilities to support the natural language interface. The architec-

ture is based on one that has been accepted by ANSI/SPARC and that is being considered by ISO as a reference model for database systems. ANSI is the American National Standards Committee on Computers and Information Processing, and ISO is the International Standards Organization. The relevant reports are [86] and [9].

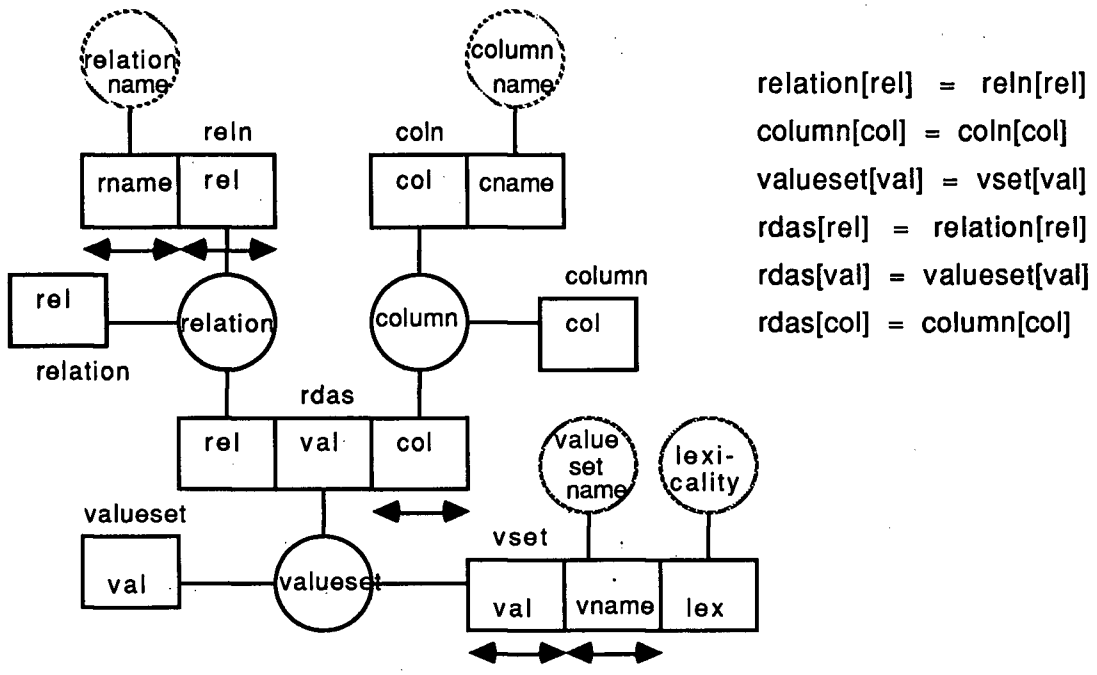


Figure 2.9: The Core Metaschema

The Core Metaschema

The core metaschema is a schema for a relational database that records information about relational database schemas. Figure 2.9 adapted from [63] illustrates the core metaschema for a self-describing database system. The inclusion constraints have been added for clarity. Broken circles denote primitive value sets and full circles primitive non-value sets. A box

denotes a column, and a contiguous string of boxes denotes a relation. The value set of a column is indicated by connecting the circle that denotes the value set with the box that denotes the column.

Double headed arrows point out columns the values of which uniquely identify tuples in the relation. For example, each value of the column *col* of relation *rdas* occurs in at most one tuple of the relation. Such a column is called a key. The notation can also be used to point out compound keys. In this case the double headed arrow will span multiple columns of the relation.

The unary relations *relation*, *valueset*, and *column* represent relations, value sets, and columns in existence. The columns *reln*[*rname*], *coln*[*cname*], and *vset*[*vname*] associate relation names, column names, and value set names with existing relations, columns, and value sets. The relation *rdas* gives the value set and relation associated with each column. Every column in existence is associated in *rdas* with a value set and a relation. Every value set in existence and every relation in existence is associated in *rdas* with at least one column.

Operations in the Metaschema

Insert and delete operations are defined in [63] on each of the relations *relation*, *column*, *reln*, and *rdas*. The operations are specified in a Prolog like language and they enforce the inclusion and key constraints that are stated in Figure 2.9. An operation comprises a collection of assertions called update dependencies. The language for writing update dependencies is briefly described here, and the insert operation on *relation* is provided for

illustration.

Primitive operations and predicates are those provided by the language. The primitive predicate $var(x)$ is *true* if variable x is uninstantiated and *false* otherwise. Primitive predicate $nonvar(x)$ is the negation of $var(x)$. The primitive operations are *assert*, *retract*, *read*, *write*, *new*, and *break*. $assert(r(t))$ adds tuple t to relation r, and $retract(r(t))$ deletes t from r. $read(x)$ reads a value from the screen and binds it to x, and $write(x)$ writes the value bound to x to the screen. $new(r(x))$ with r a unary relation defined on non-value set D, binds a new entity surrogate of D to x. All of the primitive operations evaluate to *true*.

Each update dependency has the form $< op > \rightarrow < cond >, < op_1 >, \ldots, < op_n >$ where $< op >$ is the operation being defined, $< cond >$ is a condition which may be either one of the primitive predicates or a predicate of the form $< relation\_name >$ ($< variable/constant\_list >$) where *variable/constant_list* is a list of variables or constants (e.g., $rdas(x, y, c)$), and each $< op_i >$ is either an operation, itself defined by a collection of update dependencies, or one of the primitive operations. Each $< op_i >$ is referred to as an implied operation.

$< op >$ has the form $< op\_name >$ ($< relation\_name >$ ($< variable\_list >$)). For the existing operations (the ones that have been defined in [63] as part of the core metaschema) $< op\_name >$ is one of *insert*, *delete*, or *modify*, and $< relation\_name >$ is the name of a core metaschema relation. All variables in $< variable\_list >$ are assumed to be universally quantified. All variables in the variable lists of the $< op_i >$ which do not appear in $< variable\_list >$ of $< op >$ are assumed to be existentially quantified. All variables range over primitive sets.

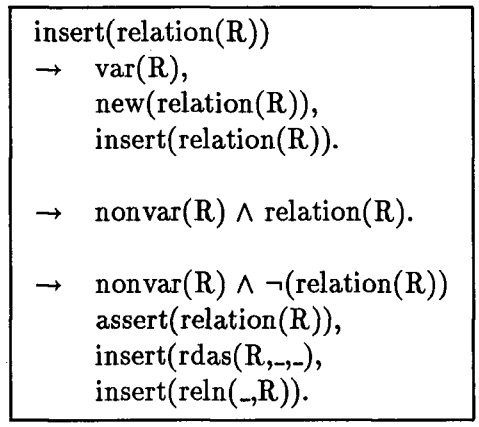


Figure 2.10: Insertion into *relation*

Figure 2.10 illustrates the insert operation on the relation *relation* of the core metaschema. The first update dependency is read as follows: If the variable R is uninstantiated, then a new relation surrogate is created and inserted into relation *relation*. The second is read as follows: If R is instantiated and R is already in relation *relation* then do nothing. The operation succeeds with the DB unchanged. The third update dependency is read as follows: If R is instantiated and R is not in *relation*, then insert it into *relation*. The insertion triggers insertions into relations *rdas* (because every relation has at least one column) and *reln* (because every relation has a name). Insertion of the tuple (R,_,_) into relation *rdas*, for example, will result in a prompt to the user to provide values for the missing entries denoted by '_'.

This concludes our coverage of background concepts needed to understand the thesis. In the remainder of this chapter the significant contributions of the research are outlined.

2.4 Knowledge Sharing for DB Schema Design and NL Understanding

Different demands are imposed on a knowledge representation strategy by the NLI and the DB system. The NLI needs a language for representing the meaning of NL requests that is independent of the structures of the DB. The DB system, on the other hand, is concerned with structures for representing data efficiently. To solve this dilemma, it is of great importance to choose the appropriate knowledge representation strategy.

Noticing that Gilmore's SET model [31] provides a suitable knowledge representation strategy both for NL processing and for constructing the DB schema, it was natural to choose this model as the basis for our design strategy. Figure 2.11 gives an overall design of our proposed system. Here the arrows denote information flow. It is particularly noteworthy that the SET schema not only provides knowledge for constructing the relational DB schema (previously researched by Gilmore [31] and Storey and Goldstein [80]) but more significantly provides knowledge for the purpose of adapting the natural language interface (NLI) to a new domain.

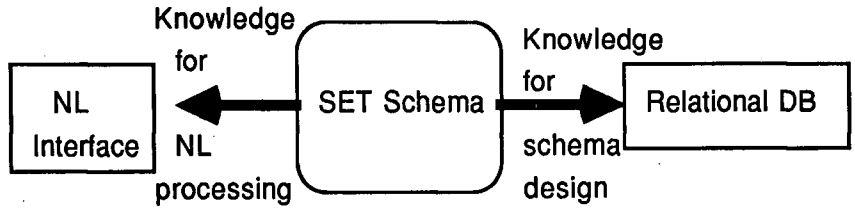


Figure 2.11: The SET Schema - a Bridge between the NLI and the RDB

Portability is the ease with which an NLI can be adapted to a new domain and database, and the enhancement of portability is a result of our design strategy. Capturing knowledge from the SET schema reduces the amount of work required for adapting the NLI to a new domain because the same work has previously been carried out for the purpose of designing the DB schema.

2.4.1 A Semantic Relatedness Measure in SET Schemas

A measure of semantic relatedness in relational schemas and its application for resolving semantic ambiguity have been illustrated in Section 2.3.2. In this approach, the relational schema is represented as a graph G whose nodes denote relations of the schema. An arc $(R1, R2)$ is an arc of G if the relations denoted by $R1$ and $R2$ are allowed to be joined. The edges of G are undirected and each has a weight of 1. A join path for set of vertices V is a subtree of G that contains the vertices in V and each of whose leaf nodes is in V. If a natural language database request has more than one possible join path, the minimum weight one is taken to be the best one for the request.

A semantic relatedness measure (SRM) in Entity-Relationship (ER) [12] schemas has been proposed by Wald and Sorenson [89, 88, 87] to solve the query inference problem.[6] The ER-schema is represented as a graph the nodes of which denote entity sets, relationship sets, and attributes as defined in [12]. The weight of an edge is dependent on the direction

[6] A goal in designing a query language (QL) is that the QL should free the user from concerns about the structure of the database (eg., which relations are used to store the data, which types of data are stored in each of them). A QL query may map to more than one DB query. The query inference problem is to choose the best DB query from among the possible ones for a given QL query.

in which it is traversed (e.g. If it is traversed from the entity set to the relationship set it may have a different weight than if it tranversed from the relationship set to the entity set.) Certain QL queries called *tree queries* can be represented as subtrees of the ER-graph. The weight of a subtree is measured with respect to a target graph TG which is the set of vertices referenced by the query. The weight relative to a vertex $v \in TG$ is the sum of the weights of the edges as they are traversed starting at v. The weight of a subtree is the minimum of the relative weights over all $v \in TG$. When a given QL query must be represented by more than one subtree, the minimum weight subtree determines the best database query for the QL query.

In this thesis a measure of semantic relatedness in SET schemas is proposed as a basis for resolving other types of ambiguities in natural language requests, specifically, word sense ambiguities, semantic ambiguities, and post noun modifier attachment. The overall process is illustrated in Figure 2.12 where the arrows indicate data flow. The possible interpretations

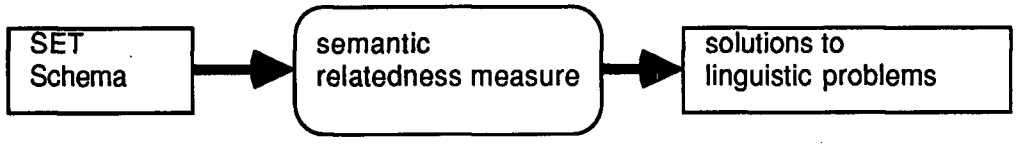


Figure 2.12: Capturing Knowledge from the SET Schema to Solve Linguistic Problems

for a request are ordered by the semantic relatedness measure from most likely to least likely, with some interpretations ruled out completely.

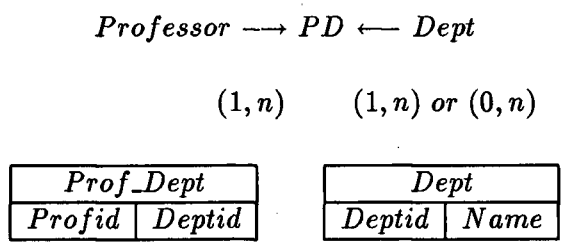
Wald and Sorenson's measure uses knowledge expressed by the max value, but not the min. Our SRM in SET schemas is similar to Wald and Sorenson's, but more general because

it uses knowledge expressed by both the max and the min.

An SRM in SET schemas (and ER-schemas) is better than one in relational schemas for the following reasons:

1. The arbitrariness of the relational schema design affects the results of the measure. A common way for two relational schemas to differ is that one relation is used in one schema to express some information while two relations are used in the other schema to express the same information. The join path consisting of the one relation and the join path consisting of the two relations will have different weights. In contrast, our SRM in SET schemas is very insensitive to the arbitrariness of the design of the SET schema. In Section 4.5.1 we investigate the sensitivity of our heuristic to variations in the schema.

2. Distance in relational schemas is based on the structure of the database. Although knowledge expressed by min/max values is used to make decisions about DB structure, we cannot get back from the structure, all of the knowledge that was used for designing it. To illustrate, consider the following domain graph and the relational schema designed from it which describes relations *Prof_Dept* and *Dept*. Some of the information in the domain graph is lost in the process of designing the database schema. Whether the min value of PD on *Dept* is 1 or 0, the given relational schema is a correct design. It is not possible to determine from the relational schema whether the min value of PD on *Dept* is 1 or 0. The information that every department has at least one professor is available in the domain graph but is lost in the process of designing the relational schema.



2.5 Using the Database Extension to Provide Knowledge

A central problem in translating a natural language request into a relational database query is that of associating primitive constituents in the input request with names of columns in the formal query. Consider the natural language request "List all the red cogs". The information is not available in the request that *red* is a color and *cog* is the name of a part.

The problem is typically solved by storing the information in the lexicon. Sample entries in the lexicon of an NLI to a relational database system follow:

(cog (coln (pname of part)))

(red (coln (color of part)))

The entry for *red* states that *red* is a member of the color column of the part relation, and similarly for the entry for *cog*. Using the database to provide column information [41] is useful for improving portability. A method of capturing this knowledge from a current database system is described in Chapter 5.

2.6 A Portable Natural Language Interface

Current database systems are designed to be extendable to support a variety of applications each of which may use a different knowledge representation strategy. In this thesis, a

new architecture for natural language interfaces is proposed that takes advantage of the extensibility features of a standard RDB system to permit the NLI to be easily ported between domains and databases. Figure 2.13 illustrates an architecture for an enhanced NLI, denoted NLI$^+$. Previous portable natural language interfaces can be logically separated into a linguistic core [72] and a database component. The DB component is involved with *fitting* the NLI to the available database system. An NLI$^+$ is an NLI enhanced with a component that is involved with *fitting* the database system to the NLI. We will refer to this new component as the *data management strategy* (DMS), and its purpose is to enhance portability.[7]

A relational meta-DB describes relational schemas, but we wish to capture knowledge for natural language understanding from SET schemas. To fulfill our goal, we had to make the following additions to a standard RDB system. We added new relations to the metadatabase, added corresponding operations for updating those relations, and extended operations used for updating the existing metadatabase relations. These additions are part of the DMS which provides the NLI with a richer source of knowledge than would be available from the RDB alone, while making use of knowledge that is already available as a result of the DB schema design process.

Chapter 5 presents an overall design for an NLI$^+$, outlines the interfaces between the NLI and the DMS, and introduces heuristics for using the knowledge in SET schemas to

[7]The essential difference between fitting an NLI to the DB system and fitting a DB system to the NLI is that capabilities of the DB system are used to fit the DB system to the NLI, whereas in the other case programming language and other more general capabilities are used. The two are complementary rather than competing approaches for interfacing a natural language understander with a database.

resolve ambiguities in the meaning of NL requests. Chapter 6 describes the DMS in more detail. A preview of the DMS is given in Subsection 2.6.1 that follows:

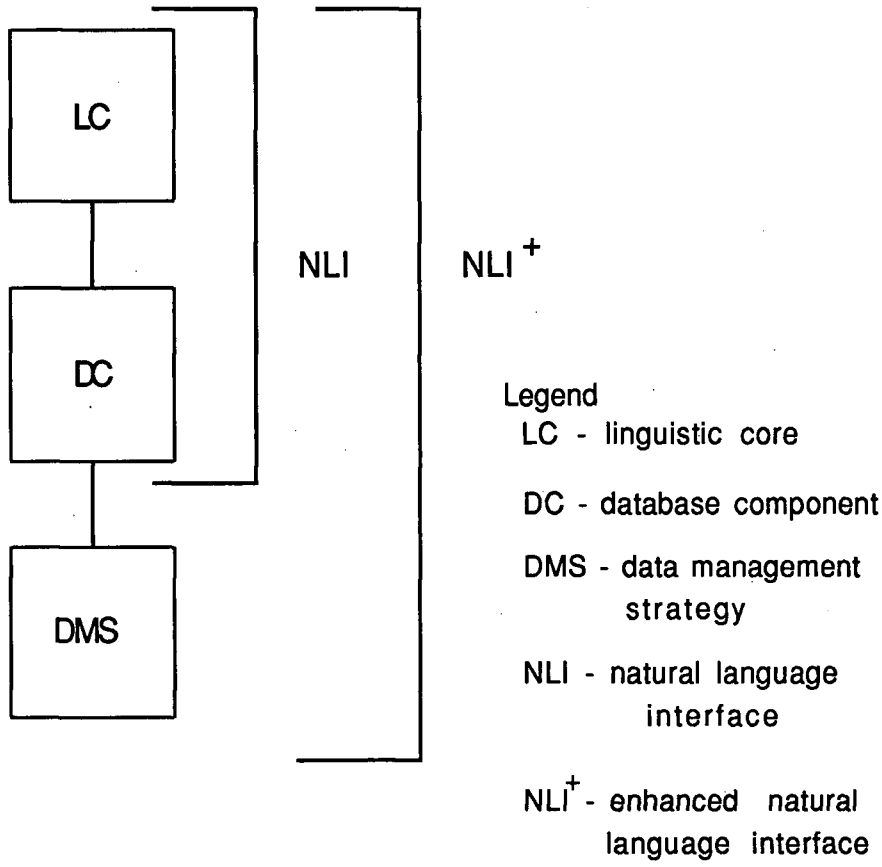


Figure 2.13: An Enhanced Natural Language Interface

2.6.1 The Data Management Strategy

Figure 2.14 illustrates a layering of schemas each one based on a different data model which we use as a basis for structuring the DMS. The problem of mapping between a SET schema and a relational schema structures naturally in this fashion. The arrows indicate the association of a schema with raw data which yields an *interpretation* as defined in Section

2.2. However, the raw data may itself be a schema. The process of describing the different schemas moves in a direction opposite to that indicated by the arrows, because knowledge needed for the design of the NF^2 object, NF^2 data, and relational schemas is provided by the SET schema.

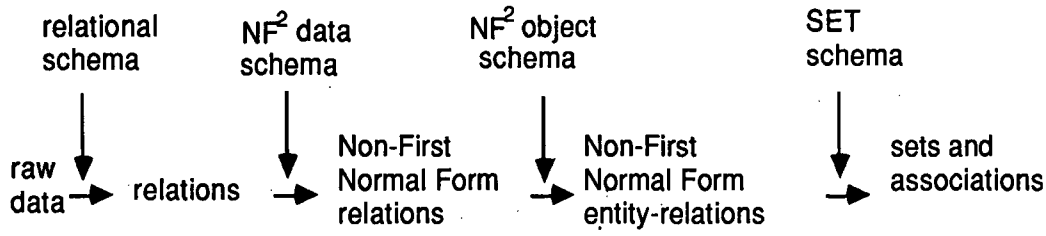


Figure 2.14: A Layering of Schemas to Support SET Schemas

An Example from the University Domain

This subsection gives an example of the relationship between the different types of schemas illustrated in Figure 2.14. A SET schema for the university domain is illustrated using a graphical notation in Figure 2.15.

The entities of interest in the university domain are students, courses, professors, and departments. When reading the SET schema, we can think of the sets *Student*, *Course*, *Prof*, and *Dept* as containing these entities, but when the SET schema interprets a collection of data it is necessary for each entity to be denoted by a data object. The associations of interest are as follows:

SC associates with a student the courses that he or she is taking

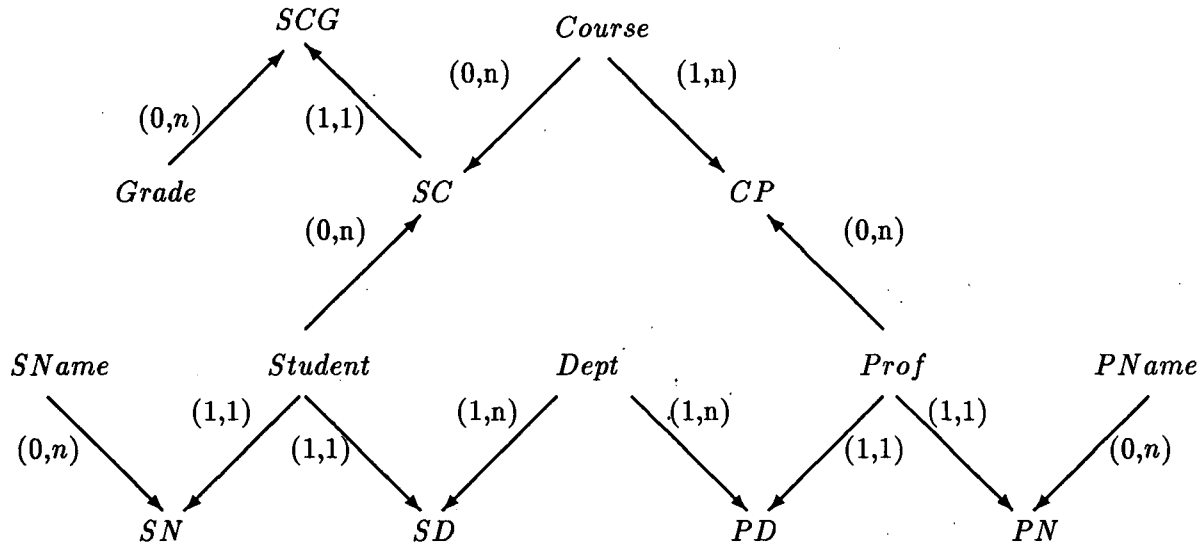


Figure 2.15: Partial SET Schema for the University Domain

CP associates with a course the professor who teaches the course

SD associates with a student the department in which he or she is registered

PD associates with a professor the department in which he or she works

The sets *Student*, *Dept*, *Prof*, and *Course* are sets of entities that exist in the world, and the sets *SName*, *PName*, and *Grade* are value sets. An *attribute* is a binary association with one parent an entity set and the other a value set. The attributes in the university domain (illustrated in Figure 2.15) are as follows:

SCG associates with a student s and course c the grade obtained by s in c

SN associates with a student his or her name

PN associates with a professor his or her name

The min/max value of *SC* on *Course* is $(0, n)$ which states that there are at least zero and at most any number of students registered in a course. This is to say that there is no

constraint expressed by the min/max value of SC on $Course$. The min/max value of SD on $Dept$ is $(1, n)$ which states that there is at least one student registered in every department.

The remainder of this example illustrates the NF^2 object, NF^2 data, and relational schemas for part of the SET schema for the university domain.

The many-to-many association between $Student$ and $Prof$ through $Course$ is represented by three NF^2 object relations, one for $Student$, one for $Prof$, and one for $Course$, and a pair of inclusion constraints to link the courses taken by students with the courses taught by professors. The names of columns serve no purpose in specifying the *types* of objects that may appear in columns. The term *type* will be used to refer to the primitive sets in the domain which, for the university domain, include the value sets $SName$, $Grade$, and $Pname$ and the non-value sets $Student$, $Course$, $Prof$, and $Dept$.

An *on* or *onto* association is represented by a pair of inclusion constraints. In the NF^2 object schema the following inclusion constraint states that every course is taught by at least one professor:

$$\text{COURSES} \subseteq \text{PROFESSOR[COURSES]}$$

In transforming the object schema (Figure 2.16) to a corresponding NF^2 data schema (Figure 2.17), columns must be introduced to explicitly represent entities. In the example, (Figure 2.17) the columns that have been introduced for this purpose are STUDENTS[SID], STUDENTS[COURSES][CID], PROFESSORS[PID], PROFESSORS[COURSES][CID], and COURSES[CID].

The mapping from an NF^2 data schema to a corresponding relational schema involves

STUDENTS		
SNAME	COURSES	
	CNAME	GRADE

PROFESSORS		
PNAME	COURSES	
	CNAME	TIME

COURSES
CNAME

$$\begin{aligned} \text{STUDENTS[COURSES]} &\subseteq \text{COURSES} \\ \text{PROFESSORS[COURSES]} &\subseteq \text{COURSES} \\ \text{COURSES} &\subseteq \text{PROFESSORS[COURSES]} \end{aligned}$$

Figure 2.16: Partial NF^2 Object Schema for the University Domain

STUDENTS			
SID	SNAME	COURSES	
		CID	GRADE

PROFESSORS			
PID	PNAME	COURSES	
		CID	TIME

COURSES	
CID	CNAME

$$\begin{aligned} \text{STUDENTS[COURSES][CID]} &\subseteq \text{COURSES[CID]} \\ \text{PROFESSORS[COURSES][CID]} &\subseteq \text{COURSES[CID]} \\ \text{COURSES[CID]} &\subseteq \text{PROFESSORS[COURSES][CID]} \end{aligned}$$

Figure 2.17: Partial NF^2 Data Schema for the University Domain

STUDENTS	
SID	SNAME

PROFESSORS	
PID	PNAME

COURSES	
CID	CNAME

STUD_COUR		
SID	CID	GRADE

PROF_COUR		
PID	CID	TIME

$$
\begin{aligned}
\text{STUD\_COUR[SID]} &\subseteq \text{STUDENTS[SID]} \\
\text{STUD\_COUR[CID]} &\subseteq \text{COURSES[CID]} \\
\text{PROF\_COUR[PID]} &\subseteq \text{PROFESSORS[PID]} \\
\text{PROF\_COUR[CID]} &\subseteq \text{COURSES[CID]} \\
\text{COURSES[CID]} &\subseteq \text{PROF\_COUR[CID]}
\end{aligned}
$$

Figure 2.18: Partial Relational Schema for the University Domain

for each root NF^2 data relation. The relational schema corresponding to the previous NF^2 object schema is illustrated in Figure 2.18

The NF^2 relational, NF^2 data, NF^2 object and SET schemas are redundant descriptions of the same knowledge. The additional storage requirement is not a significant disadvantage. However, it would be a significant disadvantage if the database administrator (DBA) had to design each schema and record it in the metadatabase. Fortunately, we can make use of the results of a large body of research aimed at automating this process. In particular, work has been done on generating a relational schema that expresses the same knowledge as a given SET schema [31, 80] and, further, on generating an NF^2 data schema that expresses the same knowledge as a given relational schema [73].

The advantage of our approach lies in the conceptual clarity that it introduces to the database component of the natural language interface. It is easier to translate the internal representation of a natural language request (expressed in the context of a SET schema) to

an NF^2 object query and the NF^2 object query to a relational query than it is to go directly from the internal representation to a relational query. Furthermore, knowledge about how the objects in one schema correspond to those of another is useful for automatically generating the formal DB query.

2.7 Summary

This chapter has introduced terminology and background concepts, and summarized the research contributions of the thesis. As part of the background concepts, five linguistic problems and four data models have been described.

The five linguistic problems serve as a focus for designing an NLI that is portable from one domain to another. Knowledge is captured from the SET schema to solve the given problems. Domain portability is enhanced because the SET schema is already available as a result of designing the relational database schema.

The four data models serve as a basis for designing an NLI that is portable from one database to another. Knowledge about the mappings between schemas based on the different data models is useful for translating the internal representation of a request to a relational database query. Knowledge about the mappings is generated as a result of designing the relational database schema. If that knowledge can be automatically provided to the NLI, then the ease with which it can be adapted to a new database will be enhanced.

Chapter 3

Portable Natural Language Interfaces

This chapter provides a survey of approaches to achieving portable NL interfaces. In Section 3.1 a classification of the ways in which an NL interface can be portable between domains and databases is given. In Sections 3.2-3.5 previous approaches to achieving the various types of portabilities identified in Section 3.1 are examined. Section 3.6 includes an examination of some ideas proposed by Harris and implemented in the ROBOT system (now called INTELLECT) for using the database to achieve portability. The chapter concludes with a description of the TEAM system (Transportable English database Access Medium). TEAM is a very ambitious project that began in 1980. A detailed description of the system has recently appeared in the literature[36].

3.1 Types of Portabilities

3.1.1 Domain Portability

> Eskelin said "99 per cent of the calls **PTL** received in response to Bakker's resignation were favorable."
> *The Vancouver Sun, March 21, 1987*

Knowledge about the domain is probably necessary for a human to understand ambiguous sentences such as the one above. Similarly, a natural language interface must be provided with knowledge about the domain before it is capable of understanding natural language database requests. The process of providing that knowledge is said to be the process of adapting the NLI to a new domain. The term *domain portability* has been defined in Section 2.2. Within domain portability we distinguish between syntactic and semantic portability as in [39].

Syntactic portability of an NL interface is the ease with which the syntactic component (parser) can be adapted to a new domain. A high degree of syntactic portability can be achieved by using a traditional grammar (rather than a semantic grammar) because it is more domain independent. The major disadvantage of a semantic grammar is that it tends to be domain specific. In order to adapt the NL interface to a new domain the grammar must be rewritten.

Semantic portability of an NL interface is the ease with which the semantic component (semantic interpreter) can be adapted to a new domain. A high degree of semantic portability can be achieved by isolating domain specific semantic information within a separate

module of the system.

3.1.2 Database Portability

The term *database portability* has been defined in Section 2.2. Within database portability we distinguish several types of portabilities.

Data Model Portability

A data model is a set of rules for structuring data together with a set of operations that are permitted on those structures. For example, the structures of the relational model are relations and the operations typically permitted are join, project, and select. By data model portability of an NL interface I mean the ease with which the interface can be adapted to a new data model.

The usual solution to achieving data model portability is illustrated by the architecture of Figure 2.5. (However, this approach does not solve the problem completely.) An internal representation is constructed from each natural language input expression. The internal representation is data model independent in the sense that it can be mapped to the constructs of various data models. The query generator and query interpreter require modification to adapt the system to a new data model but the syntactic and semantic components remain unchanged.

Schema Portability

Two different databases may use different schemas even if they describe the same domain and are based on the same data model. By schema portability of an NL interface I mean the ease with which the interface can be adapted to a new database schema.

The usual approach to improving schema portability is to use a language for internal representation that is independent of any particular database schema. In this way the syntactic and semantic components of the interface do not require modification to adapt it to a new database schema. This approach can be improved as in the TEAM system [36]. Here the need for modification of the query generator is avoided as well. A distinction is made between the *conceptual schema* in which the internal representation (IR) is expressed and the database schema. A definition of each object in the conceptual schema in terms of database relations is supplied by the DBA as part of the customization process. The system uses the definitions to convert the IR of a request to a database query. In the previous approach, knowledge about how to translate the IR for a particular conceptual schema to a database query is built into the query generator. This approach used in the TEAM system can still be further improved.

Database System Portability

By database system portability of an NL interface I mean the ease with which the interface can be plugged in to the database system. Database system portability is concerned with issues such as the following:

1. Translating the formal database query produced by the NL interface to a query expressed in the query language of the available database system.

2. Translating between the data structures by which metadata in the available database system are organized and the structures that are expected by the NL interface.

3. Adapting the NL interface to the physical files of a new domain.

3.2 Portability through Simplicity

One way to achieve portability of an NL system is to restrict the language that the system can understand. The term *habitable* was introduced in [92]. A language is *habitable* if

- Users are able without conscious effort to construct statements in the language.

- Users are able without conscious effort to avoid constructing statements that are not in the language.

The PRE (Purposefully Restricted English) system described in [23] is intended to provide a habitable subset of English for database access with linguistic coverage broad enough to satisfy a large proportion of user requests. Every PRE expression is of the following form: "any number of attributes in a projection at its beginning, any number of links in its middle, and any number of conjoined selection conditions at its end" [23].

An example of a PRE expression follows:

> What are the names, ids, and categories of the employees who are assigned to schedules whose items include appointments that are executions of orders

whose addresses contain "maple", whose dates are later than 12/15/83, and whose statuses are other than "comp".[23]

PRE expressions are processed from end to beginning. Thus, every PRE expression corresponds to a database navigation which begins by selecting a collection of records from a file, follows links from those records through arbitrarily many files, and projects the records in the final file on a subset of its fields.

Ambiguity is avoided in PRE (rather than resolved) by restricting the possible interpretations of expressions. Every PRE expression has exactly one interpretation, even when its English equivalent has multiple readings.

Epstein hypothesises that the PRE language is habitable. Porting PRE to a new domain requires the filling in of tables that capture domain information. No new grammar is required because PRE does not use a grammar. A high degree of syntactic portability is trivially provided. Epstein argues that the simplicity that results from restricting the language permits the system to be portable, and that if the restricted language is habitable it will be adequate for database access.

3.3 Portability through Generality

Marsh and Friedman [64], on the other hand, argue that a broad coverage grammar is crucial for portable NL systems. They report on the Linguistic String Project (LSP) system which processes Navy equipment failure messages. An example of such a message follows [64]:

Request NAVSTA Guantanamo Bay Cuba coord SRD assist for repair of KW-7.

Door adjustments made without success. Unable to make contact at center of door. Door possibly warped. Ships force technicians unable to determine exact cause of failure or make repairs. Failure of one of four KW-7's has minimal impact on training.

The objective of the LSP system is to represent the information content of the messages in a form that can be automatically processed. The LSP system was originally used in a medical domain for representing the information in patient records and journal articles. Both types of messages are characterized by their terseness because in both domains the messages are written under space and time constraints.

The system includes a broad coverage grammar. The collection of messages in each of the medical and Navy domains is referred to as a sublanguage. The LSP system permits productions of the broad coverage grammar to be overwritten with sublanguage specific productions. In this way the broad coverage grammar can be tailored to different sublanguages.

In both domains it was necessary to extend the grammar to handle sentence fragments. Figure 3.1 from [64] illustrates the mechanism. The figure provides sample productions from the full and sublanguage grammars of the medical and Navy domains. In each domain a sublanguage specific production for FRAGMENT and an updated production for CENTER (center string of sentence) are provided.

The LSP system executes the grammar in the following way: New productions occurring in the sublanguage grammar (eg. FRAGMENT) are added to the full grammar. For rules that occur in different forms in the full grammar and the sublanguage grammar (eg.

General English Productions

```
<CENTER>     ::= <ASSERTION>/<QUESTION/
                 <PROSENT>/<PERMUTATION>/
                 -<ASSERTION>/<IMPERATIVE>.
```

Medical Productions

```
<CENTER>     ::= <ASSERTION>/<FRAGMENT>/
                 <IMPERATIVE>
<FRAGMENT>   ::= <SA>/<VINGSTG>/<TOVO>/
                 <TVO>/<SOBJBESHOW>/
                 (<NSTG>/<ASTG>/<PN>)/<SA>/
                 <VENPASS>).
```

Navy Productions

```
<CENTER>     ::= <ASSERTION>/<FRAGMENT>/
                 <QUESTION>
<FRAGMENT>   ::= <SA>/<TVO>/<SOBJBESHOW>/
                 <VINGO>/<VENPASS>)/(<NSTG>/
                 <ASTG>/<PN>)<SA>).
```

Figure 3.1: Pruning Down a Broad Coverage Grammar

CENTER), the sublanguage rule takes precedence.

The authors maintain that a broad coverage grammar is required for syntactically portable NL systems. Their argument is that it is easier to prune down a large grammar for a particular sublanguage than to build up a grammar that has been developed for a very restricted domain.

3.4 Portability through Modularity

A recursive transition network is a formalism for specifying language structure that is equivalent in generative power to a context free grammar. An augmented transition network (ATN) [99, 96] is a recursive transition network augmented with a potentially infinite number of registers which renders it equivalent in power to a Turing machine. The augmentation permits the ATN to handle transformational rules which cannot be expressed using a context free grammar.

In the classical architecture for ATN-based NL interfaces (illustrated in Figure 2.5), the syntactic and semantic components are implemented as separate modules that run in series. The syntactic component(parser) produces a collection of parse trees from NL inputs. The semantic interpreter reads the parse trees rejecting meaningless ones and producing interpretations for the ones that are meaningful.

The classical approach leads to inefficiencies because the parser spends time generating parse trees which are later rejected.[1] However, the classical approach provides a high degree

[1]There is a trade-off, however. As pointed out by Harris [41] and Woods [95], doing semantic processing during the parse also leads to inefficiencies because the semantic processor spends time generating meanings

of syntactic portability.

The use of a semantic grammar provides an alternate architecture for NL interfaces (PLANES [90], LADDER [45]). The grammar includes both syntactic and semantic information which results in the integration of the semantic and syntactic components into one module. Such an architecture for an NL interface provides efficient processing but at a cost in portability. Porting the interface to a new domain requires that the grammar be rewritten.

An intermediate approach between the syntactic grammar and the semantic grammar is the cascaded ATN grammar formulated by Woods in 1980 [97]. In the cascaded ATN approach the syntactic and semantic components run in parallel communicating information back and forth during the parse.

Both efficiency and portability are provided within the cascaded ATN architecture. Since the syntactic and semantic components can communicate, meaningless parse trees can be rejected early in the process. The syntactic and semantic components can be implemented as separate modules providing the same portability as the classical approach.

The Datalog system [39] is based on a cascaded ATN architecture. The parser invokes the semantic interpreter to add a constituent of the current clause or phrase to its interpretation. The semantic interpreter passes back a T or NIL, T if the constituent makes sense and is consistent with previous semantic assignments, and NIL otherwise. If NIL is returned the parser must back up and try another path.

---

for parses that will eventually fail syntactically.

3.4.1 Semantic Portability

Datalog's semantic component is separated into two parts: general semantics and domain semantics. General semantics comprises a collection of semantic procedures which provide interpretations for phrases.[2] Domain semantics comprises a base lexicon, an application lexicon, and a semantic network of domain specific information. The base lexicon contains syntactic and semantic definitions for general vocabulary. The application lexicon contains syntactic and semantic definitions for domain specific vocabulary.

To adapt the system to a new domain, the semantic network must be constructed, and the application lexicon must be updated to include domain specific vocabulary. In addition, the base lexicon must be updated to reflect nonstandard interpretations for general words. This is done by setting a pointer in the lexicon entry for the word to point to a semantic procedure other than the standard one required for its interpretation.

The separation between general semantics and domain semantics provides the basis for Datalog's semantic portability. The semantic procedures manipulate concepts such as entity, class, subclass, and a variety of link types in the semantic network. Given that these concepts are general, the general semantics component of Datalog will not have to be changed very often in adapting the system to a new domain.

[2] An interpretation of a clause or phrase is expressed in terms of entities, attributes, restrictions on attributes, and actions such as DISPLAY and TEST. An interpretation in Datalog serves the function of the standard query representation of [93] and the internal representation of [10]. It is a data model independent statement of the meaning of the natural language input expression.

3.4.2 Syntactic Portability

The separation between syntax and semantics provides the basis for Datalog's syntactic portability. Syntactic portability is usually tested by applying the system to a new domain and checking whether the syntactic component runs without requiring modification. An alternate test is used in [39].

The authors report on an extension to the system to permit it to interpret a new type of noun phrase. The new type of noun phrase is characterized by a head noun which represents an informational object such as *average* or *data*. An example of such a noun phrase is "the average salary of mathematicians". The system was able to answer a question such as "List the average of mathematicians' salaries", but not "List the average salary of mathematicians". In the first case the head noun of the noun phrase is the informational object *average*. In the second case the head noun is *salary*.

Extension of the system to handle the new type of noun phrase constitutes a test of syntactic portability because the new type of noun phrase fails semantically not syntactically. The new category of head noun was represented in the general semantics component of the system and semantic procedures were written to interpret noun phrases of the new type. The authors were gratified and surprised to find that no changes to the syntactic component were required.

3.5 Tools for Customization

NL systems typically require a large amount of information to be supplied to adapt them to a new domain. A customization program assists the DBA by prompting for the required information, by making sure that the information provided is consistent, and by deciding, based on its knowledge of the domain so far, what further information should be provided. The customization program provides automation of some of the DBA functions thus reducing the size of the task of porting the system to a new domain or database. In this section the customization programs of three different NL systems are described.

3.5.1 ASK - A Simple Knowledgeable System

In the ASK system [82] (domain: shipping and cargo information) natural language is used for retrieving as well as adding to the knowledge base. The addition of new vocabulary, attributes, and classes of objects is illustrated by the sample session of Figure 3.2 taken from [82]. The lines preceded by the symbol '>' are input by the DBA or the user.[3] The others are output by the NLI. In the first four commands the DBA creates a class named *person*, creates a particular person named *Capt. Ahab*, creates an attribute *captain*, and attributes to *Capt. Ahab* the property of being captain of the Karlgren. The first four commands result in the NLI acquiring knowledge about the domain, and hence the system is operating in knowledge acquisition mode. The final command is a question from the user about what

[3]In the ASK system the user interface for question answering and for knowledge acquisition are the same, but this is not generally the case. See Section 3.5.2 of this chapter for a description of the LDC-1 system [3] which provides separate interfaces for knowledge acquisition and for question/answering.

> Create the class named person
The class named person has been added.
> Create a person named Capt. Ahab
Capt. Ahab has been added as a member of the class person.
> attribute: captain
The attribute captain has been added.
> The captain of the Karlgren is Capt. Ahab.
Capt. Ahab has been added as the captain of the Karlgren.
> What person is the captain of each ship?
SHIP CAPTAIN
Karlgren Capt. Ahab

Figure 3.2: Knowledge Acquisition in ASK

information is in the database, and hence the system is operating in question/answering mode.

In adapting the system to a new domain, it is assumed that data from the new domain exist in formated text files. A program for bulk data input is provided which engages in a dialogue with the user to obtain specifications of the format and content of the input files. The content specifications provide the relationship between conceptual objects of the domain and fields of the physical files. For example, a statement such as "$< 1,2 >$ is a ship" where ship has been previously defined as a class and $< 1,2 >$ identifies a particular field in the file(the second field of record one) associates the field <1,2> with the conceptual object *ship*.

Once the format and content specifications have been obtained, the bulk data program adds new vocabulary and data from the input files to the knowledge base. The motivation for providing such an interface is to permit the NL system to be adapted to physical files with a variety of formats.

3.5.2 LDC-1(Layered Domain Class)

LDC-1 [3]is a natural language interface to domains in which the primary structuring relation is that of decomposition. Such domains are referred to by the authors as layered domains, and hence the name for the system.

The authors provide a useful classification of the types of knowledge required to adapt an NL system to a new domain. They distinguish the following:

- domain structures and vocabulary
- semantics for verbs, adjectives, and other modifiers
- information about the physical data files

Domain Structures and Vocabulary

ENTITY NAME? section
SYNONYMS: class
TYPE (PERSON,NUMBER,LIST,PATTERN,NONE)? pattern
GIVE TWO OR THREE EXAMPLE NAMES: cps51.12, cps212.2, rel34.1
NOUN SUBTYPES: none
ADJECTIVES: large, small
NOUN MODIFIERS: none
COMPOSES INTO: course
DECOMPOSES INTO:
MULTIPLE ENTITY? yes
ORDERED ENTITY? no

Figure 3.3: Domain Structures and Vocabulary Acquisitions in LDC-1

Figure 3.3 (adapted from [3]) illustrates a terminal session in which the system acquires knowledge about a particular entity. Synonyms for the entity name are requested. The system then asks the user for the entity type. If the type PATTERN is given, the system

asks for some examples and infers a pattern based on the examples. For the examples given in the figure the inferred pattern will be three letters followed by some digits followed by a dot followed by some digits. When a string belonging to the inferred pattern is encountered in an input query the system will deduce that it is a section entity without referring to the database.

The DECOMPOSES INTO and COMPOSES INTO fields provide the decomposition relation for layered domains. The section entity is part of (COMPOSES INTO) the higher level entity *course*, and does not decompose into lower level entities. The MULTIPLE ENTITY field tells whether the entity is a single object or a class of objects. For a class the system asks whether the objects are considered as ordered within the class.

Semantics Acquisition

A language is provided for describing verb and adjective meanings. For example, a desirable instructor might be defined as an instructor who gives a grade of B or above to more than half his students. The language permits this statement to be expressed and associated with the term *desirable instructor.* In a natural language request for desirable instructors, only those instructors who satisfy the meaning of desirable will be retrieved.

Information about the Physical Files

LDC-1 was designed for small databases (on the order of hundreds of records). Efficient retrieval and data integrity were not of concern to the designers. The primary focus was on providing simple file formats so that the database could be maintained by office workers

without help from programmers or database administrators.

The customization program permits the relationships between fields of the physical files and conceptual objects of the domain to be specified. Both ASK and LDC-1 are addressing the data independence problem which is one of the fundamental problems solved by a database system. They are also providing the standard solution to this problem as proposed by the ANSI/X3/SPARC Study Group [83]. Data model portability was not a design goal of either ASK or LDC-1. Both systems interface to a file system rather than a database system.

3.5.3 TQA - Transformational Question Answering

TQA [19] is an NL interface to IBM SQL-based program products. A design goal was to reduce the linguistic information which must be supplied so that database administrators could customize the system to their own databases without help from linguistic experts.

A customization program is supplied which engages in a dialogue with the DBA to obtain domain information. The TQA customization program differs from those of ASK and LDC-1 in that it analyses the database to determine what information should be requested from the DBA.

Automatic Association of Lexicon Information with Primitive Constituents in the Input Request

Primitive constituents appear as leaf nodes in the parse tree, and are recognized by the NL interface using rules that are built into it (e.g. rules of morphology) and information in the

Primitive Constituent	*Descriptor*
88-H04	D02-L01D02
81-T14	D02-L01D02
29-211	D02-D03
27-125	D02-D03
89-R07	D02-L01D02
BERNEN	L06
18-138	D02-D03
36-068	D02-D03
07-059	D02-D03
71-D19	D02-L01D02
SJ-028	D02-D03

Table 3.1: Primitive Constituents and Their Shape Descriptors

lexicon. Some primitive constituents appear as values in the database.

The *shape* of a primitive constituent is the format of the primitive constituent. TQA uses a limited language for describing formats. Presumeably the term "shape" rather than "type" as used in programming languages has been used by the author to reflect this limitation.

A variety of primitive constituents and their associated shape descriptors are illustrated in Table 3.1 taken from [19]. The shape descriptor specifies the number of consecutive digits, the number of consecutive letters, and the relative ordering of these substrings in the entry. For example, the shape descriptor D02-L01D02 for entry 88-H04 says that it contains two digits (D02) followed by a dash (-) followed by one letter (L01) followed by two digits (D02).

When the NL system is in knowledge aquisition mode, the NL interface automatically analyses non-numeric database columns to obtain shape descriptors for the column entries.

For each different shape descriptor the names of columns that contain entries of the given shape are recorded in a master table of shapes.

When the system is in question/answering mode the shapes of primitive constituents in the input request are computed using the same algorithm for computing shapes that was used for creating the master table of shapes. The master table of shapes is interrogated for the resulting descriptors and their associated column information.

Automatic Entry of Information in the Lexicon

The TQA grammar includes semantic categories such as human, organization, and place. A table of redundancy rules associates with each semantic category, the names of columns whose entries belong to that category. For example, the entry

(HU (=COLN (EMP# PHONE)))

in the table states that the entries of the EMP# column in the PHONE relation denote humans. Common words such as 'who', 'where', and 'person' which do not occur as column entries in the database are associated with column information by means of redundancy rules. For example, the common word lexicon may contain an entry (=ADDCOLS (HU)) associated with the word 'person' which states that objects denoted by the word 'person' are human. The customization program will add to the lexicon entry for 'person' all of the column names associated by the redundancy rules with the category human.

Customization of Output

Values in the database are commonly coded to save space. For example, an employee database might use codes for employee departments (eg. 021 for the Finance department). The customization program guesses the columns that contain coded values by comparing the number of distinct entries in the column with the number of tuples in the relation. If the number is small the DBA is asked to supply the name of a table that decodes the values. For example, the department code table may contain department codes together with department names.

The customization program does a word frequency analysis on columns whose entries comprise multiple words. For words that occur frequently (inc., co., corp.) the DBA is asked to provide synonyms (incorporated, company). The DBA is also asked whether these words serve to mark words preceding or following them in the input request as coming from the same column. For example, if all of the entries in the COMPANY column are names of companies (ACME BRO Inc., B.C.L.M. Enterprises Inc.) the word 'incorporated' marks the preceding words (ACME BRO, B.C.L.M. Enterprises) as coming from the COMPANY column.

The features of TQA for customization of the output are particularily interesting because the system uses knowledge of its own knowledge. It finds out what information it needs by analysing the database and it prompts the DBA for the required information.

The customization programs of three natural language systems have been described in this section. All three assist the DBA by prompting for the information required to adapt the

NL system to a new domain. The customization program contributes to semantic portability because it eases the task of providing knowledge required by the semantic processor such as domain structure information and semantics for adjectives and verbs.

3.6 Using the Database to Achieve Portability

Two ideas for achieving portable NL systems originate in the ROBOT system developed by Harris in 1978 and now commercially available from Artificial Intelligence Corporation as the INTELLECT system [41, 43]. First, Harris recommends using the database as a definition of world knowledge. Second, he recommends using the database as if it were part of the lexicon. These two uses of the database for achieving portable NL interfaces are examined in this section.

3.6.1 Using the Database as a Definition of World Knowledge

World knowledge is "... any piece of information available to the system, distinct from the sentence itself, that aids in the understanding of the sentence" [41]. To help illustrate the usefulness of world knowledge for answering NL requests, we will use a database which comprises two relations that record information about cars. The first one has columns CAR, COLOR, and MANUFACTURER that together specify the color and manufacturer for each car. The second one has columns CAR and OWNER that together specify the owner of each car.

Consider the request "Tell me about green ford cars". If 'green' appears in both the COLOR and OWNER columns, and 'ford' in both the OWNER and MANUFACTURER

columns, then the sentence will have four interpretations.

1. color = green and manufacturer = ford

2. color = green and owner = ford

3. owner = green and manufacturer = ford

4. owner = green and owner = ford

Harris uses the following heuristic for selecting the interpretation intended by the user: A search is performed for each of the possible interpretations. If all of the searches fail then the answer to the question is *none*.[4] If only one search succeeds then the interpretation associated with that search is selected as the one intended by the user. If more than one search succeeds, then the heuristic is of no help. In this case the user is asked to select an interpretation from among the possible ones.

The heuristic is based on the premise that users tend to ask questions about entities that are described in the current extension of the database. A disadvantage of using the given heuristic is that, since the extension of the database changes with time, the interpretation selected today for an input request may differ from that selected tomorrow for the same request. For example, if no green cars are described in the database today, then the interpretation in which the primitive constituent 'green' is understood as a color will not be considered, even though tomorrow it will be considered, if in the intervening period the database is updated with the description of a car that is green in color.

[4]The NL system should report on the presuppositions of the NL inputs [66]. Here the presupposition is that there are colors, owners, and manufacturers in the database.

A better approach would be to use constraints that hold in the domain (eg., A car cannot have two owners.) as a source of world knowledge. Harris apparently did not have available a database system which was capable of recording such constraints. The advantage of using the constraints rather than the database extension to constrain the possible interpretations of an NL request is that a set of interpretations is obtained that is independent of time.

3.6.2 Using the Database as Part of the Lexicon

Any values that occur in the database are expected to occur in natural language requests. Harris recommends using the database to provide the associations between column names and primitive constituents in the input request that appear as entries in the given columns. Harris reports on the general concepts but not the mechanism as this information is proprietary to the Artificial Intelligence Corporation.

Consider the EMPLOYEE relation of Figure 3.4 and the request "List the employees who live in Vancouver and work in the finance department?" An internal representation for the request is produced that has *holes* in it corresponding to information that is to be obtained from the database:

(relation (EMPLOYEE))

(print (name EMP#))

(search (*unknown_column* = 'finance')

and (*unknown_column* = 'Vancouver'))

A complete representation can be generated when the facts that 'finance' appears in the DEPT column and 'Vancouver' appears in the ADDRESS column are obtained from the

EMP#	NAME	DEPT	SALARY	ADDRESS
E72	Smith	personnel	10000	Vancouver
E89	White	purchasing	10000	Vancouver
E799	Black	finance	20000	Vancouver
E900	Jones	personnel	30000	Ottawa
E1001	Lakes	finance	50000	Ottawa

Figure 3.4: Employee Relation

DEPT	PTR1	PTR2	PTR3	PTR4
personnel	0	480	/	/
purchasing	160	/	/	/
finance	320	640	/	/

Figure 3.5: Index on DEPT

ADDRESS	PTR1	PTR2	PTR3	PTR4
Vancouver	0	160	320	/
Ottawa	480	640	/	/

Figure 3.6: Index on ADDRESS

database. This approach contributes to domain portability because it relieves the DBA of the task of providing column information for primitive constituents that appear as values in the database. Since the database may contain thousands of primitive constituents, a substantial saving in the time required to adapt the NL interface to a new domain is gained.

Column information must be obtained from the lexicon for primitive constituents in the input request which do not appear as values in the database. For the given request if the primitive constituent 'Vancouver' does not appear in any database column, then an internal representation for the request cannot be constructed by relying on column information from the database.

In the INTELLECT system the answer 'none' is given for requests for which column information cannot be obtained from either the lexicon or the database. Such a solution

does not permit the distinction between the answer 'No' and an answer which reports on the system's lack of knowledge. For example, if values in the ADDRESS column of the EMPLOYEE relation are drawn from the domain CITIES of names of cities, and 'Vancouver' is not a member of CITIES, then a more appropriate answer to the given request (rather than 'none') is "I don't know of any city named Vancouver".

In the CO-OP system [56] the problem of unknown primitive constituents is treated as a special case of word sense disambiguation. A lexicon entry is constructed for the unknown primitive constituent which designates each of the character columns in the database as a column in which the primitive constituent occurs as a value. During word sense disambiguation one of the columns is selected for the unknown primitive constituent using a number of different heuristics. The advantage of this approach over Harris' is that column information is provided for input primitive constituents using the same method whether or not the primitive constituents appear as values in the database.

For efficiency Harris uses indices on relations rather than the relations themselves as a source of column information. Indices are maintained by a database system to provide efficient processing of queries.

Assume that the EMPLOYEE relation is indexed on DEPT and on ADDRESS. Relations for storing the indices are illustrated in Figures 3.5 and 3.6. The columns contain addresses of tuples in the EMPLOYEE relation. Consider the request "List the employees who live in Vancouver and work in the finance department?". The information needed for constructing the internal representation is available from the indices. For example, the primitive constituent 'finance' appears in the DEPT column of the index on DEPT which

indicates that the primitive constituent 'finance' is a member of the DEPT column of the EMPLOYEE relation. The efficiency advantage that is realized in the INTELLECT system is presumably a saving in disk accesses because the indices are more likely to be in main memory than the indexed relations.

It is important to distinguish between the efficiency of obtaining information about membership in columns using the indices, and the efficiency obtained in query processing by using the indices. As an illustration of the latter, consider the steps that would be required for answering the database query corresponding with the given request:

1. From the first index the set of addresses for EMPLOYEE tuples with 'finance' in their DEPT column is determined. For the given database this set is {320, 640}.

2. From the second index the set of addresses of EMPLOYEE tuples with 'Vancouver' in their ADDRESS column is determined. For the given database this set is {0, 160, 320}.

3. The intersection of the two sets of addresses gives the set of addresses of EMPLOYEE tuples for emplyees who live in Vancouver and work in the finance department. For the given database this set is {320} indicating that there is only one such employee.

The EMPLOYEE relation will be accessed to retrieve information on the given employee which if all contained in one page of storage will require only one disk access. If only sequential access of the EMPLOYEE relation is permitted then many more disk accesses can be expected to process the query.

The designers of the LADDER system [45] argue against Harris' idea based on efficiency

concerns. A second concern expressed by the designers of LADDER in disfavor of Harris' idea is that their databases contain mostly coded abbreviations. The coded abbreviations are unsuitable as entries in the lexicon because they are unlikely to occur in natural language requests.

More recent systems (LDC-1 [3] and TQA [19] which have been described in Sections 3.5.2 and 3.5.3, respectively) employ alternate approaches to automatically providing column information. In LDC-1 a pattern for the values in an entity set[5] is determined by the system based on a few examples given by the user. In TQA shape descriptors for column values are automatically computed by analysing the database. The two systems provide heuristics for computing column information which provide an improvement in efficiency over Harris' algorithm.

In spite of arguments in disfavor of Harris' approach and recent improvements in efficiency that have been realized by the use of heuristics, we adopt Harris' approach and provide an expansion of it, motivated by the following observations: 1.) The LADDER system [45] uses a semantic grammar approach within which it would be prohibitively expensive to query the DB during the parsing process. Within an architecture for NL interfaces that separates between syntax and semantics, the efficiency problem is of lesser concern. 2.) If the DB contains mostly coded values that are not referenced in NL requests, then a table for encoding and decoding such as that used in the TQA system [19] can help. The table needs to be set up once. If the DB contains thousands of primitive constituents, which is the usual case, the work required to set up the table would be substantially outweighed by the

[5]LDC-1 does not distinguish between value and non-value sets as do the Entity-relationship based models.

work needed to provide column information for primitive constituents that appear as DB values. 3.) The heuristics of LDC-1 and TQA do not guarantee that all columns of which a given primitive constituent t is a member will be identified by the heuristic, nor that the columns identified do, in fact, contain t as a member. This is to say that the heuristics are not guaranteed to work in every case.

3.7 TEAM (Transportable English Database Access Medium)

The TEAM system is a very ambitious project. It is an experiment in the design of portable natural language interfaces. A major hypothesis underlying the experiment is stated in the designers' words as follows: "A major hypothesis underlying TEAM is that, if an NLI (Natural Language Interface) is constructed in a sufficiently well-principled manner, the information needed to adapt it to a new database and its corresponding domain can be acquired from users who have general expertise about computer systems and the particular database, but who do not possess any special knowledge about natural-language processing or the particular NLI" ([36], page 175).

In natural language it is common to use a property of an object to refer to the object, especially when a single object in the domain has the given property. For example, if there is only one secretary, then the phrase 'the secretary' refers to the person who possesses the property of having the job of secretary.

In TEAM the DBA must identify the columns in the database that represent properties that may be used for referring to objects. The DBA also identifies the type of objects represented by each column and each relation. The properties represented by a column are

assumed to be possessed by the entities represented by the relation of the column. Columns that are of the value set *name* are automatically assumed to represent properties that may be used to refer to objects (i.e., The DBA need not specifically identify columns which represent names).

TEAM distinguishes between a name and the object named. If no such distinction is made, then it is difficult to handle requests in which the primitive constituent 'name' is being used as a verb uniformly with those in which a name is being used to refer to an object. For illustration consider the following two requests:

1. "Which books belong to John?"

2. "Which books belong to the person named John?"

If 'John' denotes a person in request (1) and a name in request (2) then different mechanisms will be required to understand the two requests. In TEAM a process called *coercion* permits the above requests to be handled uniformly. Argument restrictions for the verb 'belong' require that the prepositional phrase following the verb denote a person and not a proper noun. The conflict is resolved by coercing the proper noun 'John' into an entity that denotes the person named 'John'. The coerced value will agree with the argument restrictions on the verb, and an interpretation for the question will be formed.

TEAM uses information about database keys to determine which columns of a relation contain entries that may be coerced into entities of the type represented by the relation. This is a particularly interesting feature of TEAM, for our purposes, because the system uses information about the database (intensional information) for interpreting natural language

requests.

The need for coercion is recognized by conflicting type requirements within a sentence. Conflicting type requirements between sentences are not considered. In fact, in TEAM the interpretation of a natural language request is not influenced in any way by the context in which the natural language dialog occurs. The system will give an inappropriate response to a pair of requests such as the following:

> Show me the students with an A average in computer science.
>
> Which students are computer science majors?

Most people will recognize that the second request refers to students who have been retrieved by the first request. For the second request TEAM will retrieve all computer science majors rather than all computer science majors that have an A average in computer science.

For each relation the DBA must explicitly identify the columns that denote modifiers of objects of the type represented by the relation. In this way some ambiguities that arise when more than one column of a relation is defined on the same type are ruled out. For example, in the *ship* relation if the word 'US' occurs in both the destination and registry columns, but only the registry column has been identified as a modifier, then the potential ambiguity for the phrase 'US ships' is ruled out.

Among all the systems that are examined in this chapter, only the TEAM system provides an internal representation that is independent from concerns about how the result should be presented. In the Datalog system [39] the language for the internal representation includes the operations DISPLAY and TEST. Similarly, the semantic primitives proposed

by Wood's [95] include the commands TEST and LIST. The system ALPS [93] permits the internal representation to specify which columns should be printed and whether duplicates in the answer should be removed.

The TEAM system determines when it has acquired the minimum amount of knowledge needed to answer questions. The system is in control of the acquisition in the sense that it prompts the DBA for information and stops when it has acquired a sufficient amount. The DBA can provide information beyond the minimum by his or her own initiative.

The characteristic of control of the acquisition appears to be unique to TEAM. A number of natural language systems are compared in [36] on characteristics which include control of acquisition. For the systems reviewed there, as well as for all of the ones that have been examined in this chapter (except for TEAM), the system designer or a superuser decides when enough knowledge has been acquired.

Since TEAM is such an ambitious and recent project, and since the major hypothesis underlying TEAM is the same as that which underlies the research reported in this thesis, the subject of similarities and differences between the TEAM approach and our approach will be addressed later in the thesis in Appendix C.

3.8 Summary

In this chapter different types of portabilities have been identified, and a number of portable natural language interfaces have been examined. For each one the examination has focused on a particular way of obtaining portability that is illustrated by the given system. The approaches used in TQA [19], INTELLECT [41], and TEAM [36] are most relevant to the

problem addressed in this thesis - that of using the database system to improve portability of the interface. In the TQA system the database is accessed during the customization process to determine what knowledge should be requested from the DBA. In the INTELLECT system the inverted indices of the database system are accessed to obtain semantic information required for understanding natural language requests. In the TEAM system information about database keys is used for the same purpose.

Chapter 4

A Measure of Semantic Relatedness for Resolving Ambiguities in Natural Language Database Requests

4.1 Introduction

A measure of semantic relatedness based on distance between objects in the relational database schema has previously been used as a basis for solving a variety of natural language understanding problems including word sense disambiguation, resolution of semantic ambiguities, and attachment of post noun modifiers. The use of min/max values which are usually recorded as part of the process of designing the database schema is proposed as a

basis for solving the given problems as they arise in natural language database requests. The min/max values provide a new source of knowledge for resolving ambiguities and a framework for understanding what knowledge has previously been captured by distance measures in relational database schemas.

Figure 4.1 illustrates general classes of ambiguities that arise in natural language requests. The outermost circle denotes all possible internal representations (IRs) for a given request. The white area denotes IRs that express interpretations that are inconsistent with the constraints[1] in the domain. The middle circle denotes IRs that express interpretations that are consistent with the constraints in the domain, but not acceptable by humans as possible interpretations for the request. For example, in the university domain, possible interpretations for the request "Dr. Lee's students" (an obvious favorite) are as follows:

1. Students in the same department as Dr. Lee

2. Students taught by Dr. Lee

3. Students supervised by Dr. Lee

A human understander of natural language who is familiar with the university domain will exclude (1) as a possible interpretation, whereas (2) and (3) would be considered as possibilities. The inner most circle denotes IRs that express interpretations that are considered by humans as possibilities for the request. Interpretations (2) and (3) above fall into this class.

[1]A (static) *constraint* is a condition in the domain that is invariant over time [60].

The line between the inner most circle and the middle one is actually quite blurred. Different humans will admit different interpretations as possible ones. In this chapter a heuristic is presented which orders the interpretations denoted by the two inner circles from most likely to least likely, with some interpretations being ruled out completely.

A great deal of research has been done to automatically generate interpretations for natural language requests that would be considered by humans to be possible in the given domain. Examples include case grammars (Filmore [26]), semantic grammars (Hendrix et al. [45]), and Woods' ATN grammar coupled with a taxonomic lattice [99]. Each uses some source of knowledge about the domain for determining likely interpretations. Here, the SET schema is used as a source of knowledge, the advantage being that the knowledge is already available as a result of designing the DB schema. In the methods cited above, the knowledge must be explicitly provided as part of the customization process.

The remainder of this chapter is organized as follows: In Section 4.2 we examine the relationship between the min/max values and the meanings of words. In Section 4.3 our heuristic is presented, and in Section 4.4 it is applied to the problems of post noun modifier attachment (MA), word sense disambiguation (WSD), and semantic ambiguities (SA). In Section 4.5 we establish that the heuristic works well for resolving ambiguities: First, we show that the heuristic is unaffected by the arbitariness of the design of the SET schema (Section 4.5.1). Second, we show that the parameters of the heuristic can be varied without affecting the outcome of the heuristic (Section 4.5.2). Finally, we show that previous heuristics that capture knowledge from relational schemas for resolving ambiguities are actually using knowledge expressed by the min/max values (Section 4.5.3). Related work is reviewed

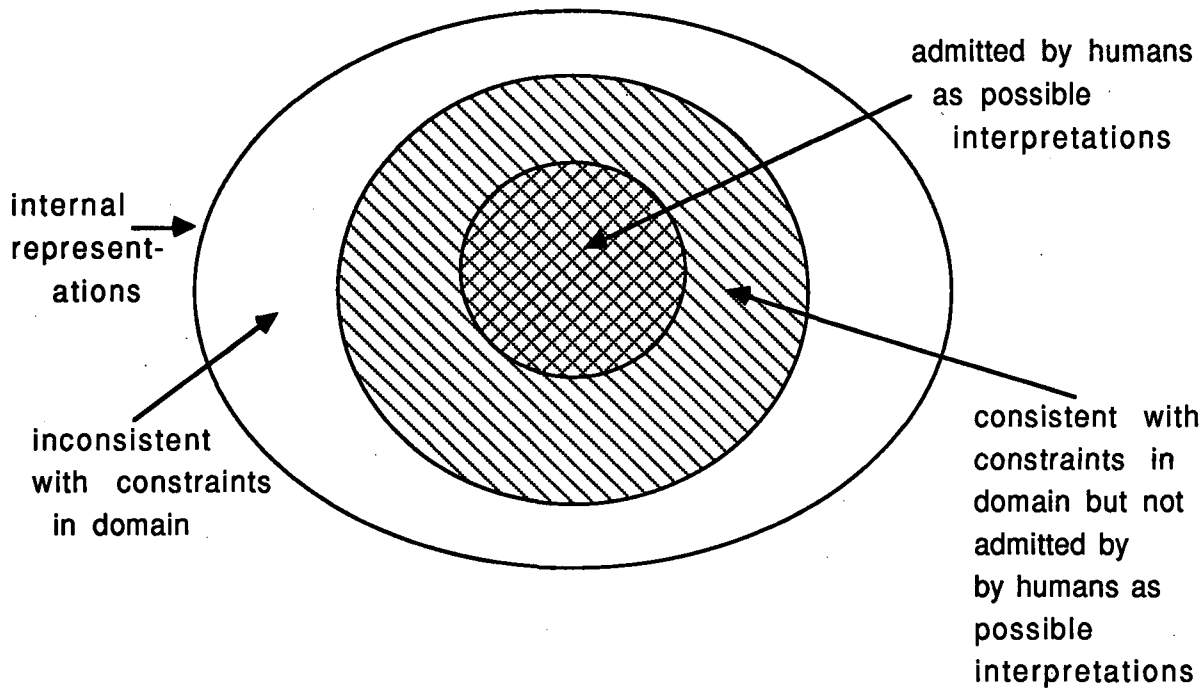


Figure 4.1: Classes of Ambiguities in Natural Language Requests

in Section 4.6, and a summary of the results of the chapter is presented in Section 4.7.

4.2 The Relationship between Min/Max Values and Word Meanings

We will focus on disambiguating prepositional phrase (PP) attachments that use the prepositions "with" and "in" and on choosing the most appropriate meaning for pre-noun modifiers (in particular, possessives).

The preposition "with" has many different meanings. The heuristic developed here deals with only one of them - the *part of* relationship which involves an "inseparable possession", or "possession by nature, not accident" [52]. Examples include "fish with bones", "vase with handles", "man with sinister expression", and "holiday with all expenses paid". Note that fish bones do not exist without the fish and the bones belong to no fish other than the given one, the handles belong to the vase and no other, the sinister expression exists as part of the man and the same expression will not exist as part of any other man, and the paid expenses are not relevant except with respect to the holiday.

The min/max values represent the *part of* relationship by what has become known as an existence dependency association. Given an association X between A and B, the set B is said to be *existence dependent* on A if an entity in B cannot exist independently from an entity in A. (e.g., a volume of a book is existence dependent on the book, a ward of a hospital is existence dependent on the hospital). If X with parent sets A and B is an existence dependent association, then the min/max values of X on the dependent set B are

(1,1). Note that not all associations with min/max values (1,1) are existence dependent associations. Existence dependency between 2 sets is easily generalized to an existence dependency among m sets, $m \geq 2$. In this case, the existence dependent set is dependent on the remaining $m-1$ sets. If R is an m-ary set and S is an existence dependent parent set of R, then it is always possible to define a binary association $T \subseteq R \times S$ such that S is existence dependent on R.

A weaker form of relationship is the *exclusive association*. Given an association X between A and B, the set B is exclusively associated with A if the min/max value of X on B is (0,1).

Pre-noun modifiers that indicate possession (e.g., Dr. Lee's students) are described by an exclusive association between the sets denoted by the noun and the modifier. (The set denoted by the noun is exclusively associated with the set denoted by the modifier.) The most likely meaning for the phrase "Dr. Lee's students" is the one in which a given student belongs to Dr. Lee, and no other. Weaker forms of pre-noun modifiers (e.g., Jones' courses) are possible. In the interpretation "courses taken by Jones", although Jones takes the courses, they may also be taken by others. However, if there are two possible interpretations, and one of them denotes an exclusive association, then that one is the most likely.

The preposition "in" according to the Oxford dictionary [81] means "inclusion or position within limits of space, time, circumstance, etc.". The favored interpretation for the preposition "in" is an existence dependency between the modifier and the referent. The second choice is an exclusive relationship between the two.

4.3 A Heuristic for Measuring Semantic Relatedness

Knowledge expressed by the min/max values is available as a result of the analysis of the domain that is undertaken for the purpose of designing the database schema. A product of that analysis is a description of the domain in terms of sets and associations among sets called the SET schema. A domain graph (DG) is a graphical representation of the SET schema.

To facilitate our presentation of a heuristic for measuring semantic relatedness, an expanded version of the DG that appeared in Chapter 2 is given in Figure 4.2. The sets *CName* and *DName* are sets of course names and department names, respectively. CPSC_Course is the set of courses offered by the computer science department. The new associations are defined as follows:

Sup associates with a student the professor who supervises the student's research

DN associates with a department the name of the department

CN associates with a course the name of the course

4.3.1 What is a Word Meaning?

The primitive constituents of a request map to vertices in the DG. The mapping is specified as part of the process of adapting the natural language interface to a new domain, and it gives the meanings (denotations) of the primitive constituents. Some primitive constituents do not denote vertices in the DG. Examples include noise words ("please" and "quickly", as in "Please print the good students quickly") which can be ignored without changing

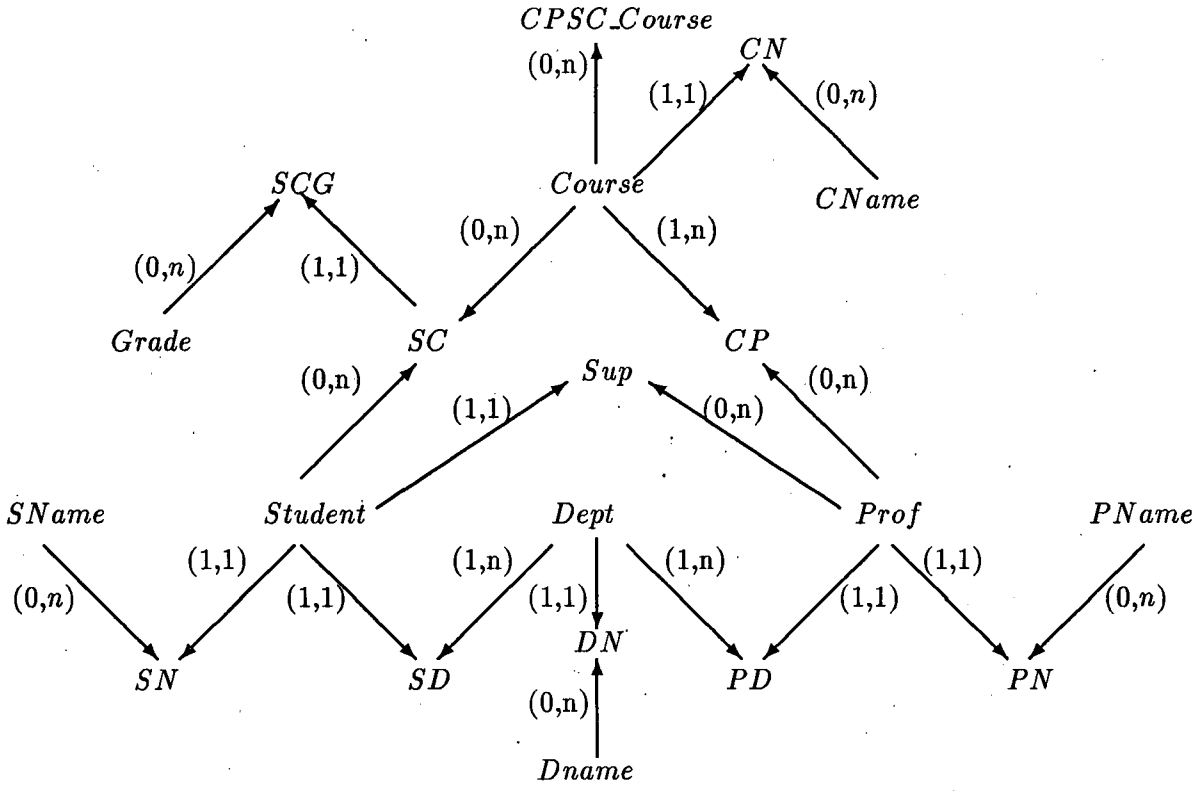


Figure 4.2: A Complete Domain Graph for the University Domain

the meaning of the request and determiners ("a", "the", "some", "all") which map into restrictions on vertices. Some primitive constituents denote more than one vertex, and this is the source of word sense ambiguities in natural language database requests.

In the examples given here, the following rules govern the assignment of meanings to primitive constituents.

1. Database values ("CPSC101", "computer science", "Dr. Lee") denote value sets (*CName*, *DName*, *PName*, respectively)

2. Nouns (student, course, department) denote non-value sets (*Student*, *Course*, *Dept*).

3. Verbs (take, teach, receive) denote associations (*SC*, *CP*, *SCG*).

4.3.2 Query Graphs

To provide a measure of relatedness between primitive constituents of a natural language request we use the notion of a *query graph* which has previously been used as a means of representing database queries [88, 6]. Here the query graph is used as a means of representing natural language requests. Since NL requests are far more complex than DB queries, it is necessary to restrict the complexity of the NL requests under consideration.

A natural language request is *simple* if it requests information about a collection of related entities. Examples of simple requests in the university domain are:

1. a professor in a department with a student who takes a course named CPSC101

2. a student who received a grade of 'B' in CPSC101

In request (1), for example, a relationship exists between professors and departments, departments and students, students and courses, and courses and course names. An example of a request that is not simple is the conjunction of the above two requests:

a professor in a department with a student who takes a course named CPSC101

and a student who received a grade of 'B' in CPSC101

The student referred to in the left piece of the request bears no relationship to the student referred to in the right piece. Simple requests are the building blocks for more complex requests.

The *target graph* for a simple request Q, $TG(Q)$, is the set of vertices denoted by primitive constituents of Q. A *query graph* for Q is any subgraph of the DG that

1. is a tree each leaf node of which is contained in $TG(Q)$

2. contains the vertices in $TG(Q)$.

Not every simple request can be represented by a tree. For example, the following request would be represented by an undirected-cyclic subgraph of the university domain graph (Figure 4.2).

a student in a course taught by a professor

who is the student's research supervisor

Such requests are referred to as *cyclic requests*. A cyclic request is represented as a collection of trees by removing for each cycle one of the edges that creates the cycle. Since there is for each cycle more than one edge whose removal will break the cycle, there will be more than one possible resulting tree. A cyclic request is represented by the collection of all such possible trees (a forest).

Each edge of a query graph is labeled with a min/max value from which a weight for the edge and, therefore, a weight for the query graph can be computed. The weight of an edge labeled with min/max value (p, q) is calculated as follows:

1. If $p = q = 1$, then the weight is 0.

2. If $p = 1$ or $q = 1$, then the weight is 1.

3. If $p = 0$ and $q = 0$, then the weight is some large value such as the number of vertices in the query graph.

This extends the work of Wald and Sorenson in which the weight of a query graph is based solely on the max value.

Given a tree with root v and directed edges, the *forward edges* relative to v are the edges that point away from v.

The weight of a query graph G relative to $v \in TG(Q)$ is the sum of the weights on forward edges relative to v. The absolute weight (or simply the weight) of G is the minimum of the relative weights over all $v \in TG(Q)$.

Example 4.1. Given the University DG, a query graph for $TG(Q) = \{Student, Prof\}$ is $(Student \longrightarrow SC \longleftarrow Course \longrightarrow CP \longleftarrow Prof)$. The forward edges relative to *Student* are $(Student, SC)$ and $(Course, CP)$. The weight relative to *Student* is the sum of the weights on the two forward edges relative to *Student* $(5+1) = 6$. (The number of vertices in the query graph is 5.) The forward edges relative to $Prof$ are $(CP, Prof)$ and $(SC, Course)$. The weight relative to $Prof$ is $(5 + 5) = 10$. The absolute weight is the minimum of the weights relative to *Student* and relative to $Prof$. Therefore, the weight of the query graph is 6.

The weight of a cyclic query graph is the weight of the minimum weight tree among those in the forest trees that represent it.

Let us refer to an edge in the DG labeled with min/max value (p, q) as a (p, q)-edge. Query graphs are compared by comparing their absolute weights. A query graph with many $(0, n)$-edges will have a large weight. If all of the edges are $(0, n)$-edges, then the weight will be $a \times b$ where a is the number of edges and b is the number of vertices. The semantic

relatedness measure (SRM) introduced here requires knowledge to be useful. If all of the query graphs to be compared have many $(0, n)$-edges, then the SRM provides little basis for comparison.

4.3.3 Complementary Heuristics

Hirst [47] points out five information sources or mechanisms that are necessary for resolving word sense ambiguities. They are:

1. a knowledge of context
2. a mechanism to find associations between nearby words
3. a mechanism to handle syntactic disambiguation cues
4. a mechanism to handle selectional restriction negotiations between ambiguous words
5. inference, as a last resort

This thesis investigates the use of min/max values to accomplish items (2), (3), and (4). A knowledge of context figures in a number of the examples to be presented in Section 4.4. In preparation for those examples, context analysis (item (1)) will be discussed in greater detail here. A method similar to one presented in [37, 38] which applies contextual information to the problem of resolving word sense ambiguities is described. Our aim is not to provide a general method for using knowledge about context for resolving ambiguities, but simply to illustrate that context alone is not in practice sufficient.

Context is defined as that part of the database that is in the current focus. The current focus is a subgraph of the domain graph determined by the vertices that have been referenced

in previous requests in the dialog. The focus changes with time. If the focus gets too big, then it isn't very useful for resolving ambiguities. In the case of word sense ambiguities, for example, if the current focus is too big then a given word may refer to more than one vertex in the current focus. Context analysis on the current focus will be needed to resolve that ambiguitiy. On the other hand, the current focus cannot be too small because it will not contain the contextual information needed for resolving ambiguities.

The focus changes during the course of the dialog in the following way: At the start of the dialog the focus is empty. When a request is processed, a new focus is constructed which is the union of the current focus and the minimum weight query graph for the request. If the request contains word sense ambiguities then it will determine more than one target graph and, hence, more than one query graph.

Knowledge about context is used for resolving word sense ambiguities in the following way: Only those vertices in the current focus that are denoted by an ambiguous word are considered as possible meanings for the word.

Once ambiguities are resolved in the request, the *unique* query graph for the request can be added to the current focus. To ensure that the query graph does not get too big, vertices must be dropped from the current focus. For the purpose of the examples, however, it will suffice to ignore this step.

Example 4.2. Given the request "Which students run programs", assume that the noun "program" can mean either a computer program or a recreational program, and that there are two senses for the verb *run*, one for each sense of the noun "program". A possible DG for this situation follows:

$$(1,1) \qquad (0,n) \qquad (0,1) \qquad (0,n)$$

$$(C\_program \longrightarrow Run1 \longleftarrow Student \longrightarrow Run2 \longleftarrow R\_program)$$

The min/max values state that a computer program is run by exactly one student, a student runs any number of computer programs, a student runs at most one recreational program, and a recreational program is run by any number of students.

If the noun "program" denotes the vertices $C\_Program$ and $R\_Program$, the noun "student" denotes $Student$, and the verb "run" denotes vertices $Run1$ and $Run2$, then the possible target graphs and their associated query graphs for the request are:

1. $TG_1 : \{C\_program, Run1, Student\}$

   $QG_1 : C\_program \longrightarrow Run1 \longleftarrow Student$

2. $TG_2 : \{R\_program, Run2, Student\}$

   $QG_2 : Student \longrightarrow Run2 \longleftarrow R\_program$

3. $TG_3 : \{C\_Program, Student, Run2\}$

   $QG_3 : C\_program \longrightarrow Run1 \longleftarrow Student \longrightarrow Run2$

4. $TG_4 : \{R\_Program, Student, Run1\}$

   $QG_4 : Run1 \longleftarrow Student \longrightarrow Run2 \longleftarrow R\_program$

QG_3 has a weight greater than or equal to that of QG_1 because QG_1 is a subgraph of QG_3. QG_4 has a weight greater than or equal to that of QG_2 because QG_2 is a subgraph of QG_4.

The weights of QG_1 relative to *C_Program*, *Student*, and *Run*1 are, respectively, 0, 0, and 3. The weights of QG_2 relative to *R_Program*, *Student*, and *Run*2 are, respectively, 3, 0, and 1. The weights of QG_3 and QG_4 are both 1, and of QG_1 and QG_2 are both 0.

The interpretations "Which students execute recreational programs" and "Which students administer computer programs" (query graphs QG_3 and QG_4) are least favored because the weights of the corresponding query graphs are high.[2] The heuristic does not help to resolve the ambiguity in the meanings of the words "program" and "run", because the remaining query graphs QG_1 and QG_2 have identical weights.

Suppose that a selectional restriction[3] on the adjective *recreational* lists *R_Program* as one of the possible vertices that the adjective could modify. If the request "List the recreational programs", precedes the request "Which students run programs", then the vertex *R_Program* will be in the current focus, and if there has been no reference during the dialog to computer programs, then the vertex *C_Program* will be absent from the

[2]Usually the interpretations "Which students execute recreational programs" and "Which students administer computer programs" would be excluded by the use of selectional restrictions which in each case state that the verb does not allow the given types of arguments.

[3]Selectional restrictions are labels on the arguments of parts of speech. An example of the use of selectional restrictions is illustrated by the following sentence:

My car drinks gasoline.

If the verb *drink* is restricted to have an *animate* subject and a *liquid* object, then the sense of drink in which a liquid is consumed must be excluded for the about sentence. Selectional restrictions need not be absolute, but may express preferences. Thus, if *drink* prefers an animate object but accepts a machine, then the metaphorical meaning of the above sentence is understood. The reader is directed to [47] for more details.

current focus. The target graph for the request is

$$\{R\_program, Run2, Student\}$$

which determines exactly one query graph.

This example has illustrated the use of context analysis for resolving word sense ambiguities. The domain involved students, computer programs, and recreational programs. The verb "run" is ambiguous as is the noun "program". For the request "Which students run programs" the heuristic does not distinguish the interpretations "Which students run computer programs" and "Which students run recreational programs". Context analysis provides a complementary heuristic for resolving ambiguities. If "recreational programs" have been previously referenced in the dialog, the interpretation "Which students run recreational programs" would be favored.

4.4 Use of Min/Max Values for Resolving Ambiguities

In this section the use of the min/max values for resolving ambiguities in natural language database requests is demonstrated by providing examples from three different domains. The first one is the University domain which we have already seen. We also look at a library circulation domain [34], and a medical domain [84].

4.4.1 The University Domain

The examples presented in this subsection refer to the domain graph of Figure 4.2.

Semantic Ambiguity

Let $TG(Q)$ be the set of vertices referenced by simple request Q. The semantic ambiguity (SA) problem is to select from among the query graphs determined by $TG(Q)$ the one that corresponds with the best interpretation for Q. The approach presented here, like that of Wald and Sorenson but with a different weight measure, is to select the interpretation that corresponds with the query graph of smallest weight where the weight of a query graph is the minimum of the relative weights over all $v \in TG(Q)$.

Example 4.3. There are two interpretations in the university domain that would be acceptable by humans for the phrase "Dr. Lee's Students".[4]

1. "Students taught by Dr. Lee"

2. "Students supervised by Dr. Lee"

Internal representations for the two interpretations follow:

1. [**For some** x:*Student*] [**For some** y:*Course*][**For some** z:*Prof*]

   $(< x, y >$:SC **and** $< y, z >$:CP **and** $< z,$"Dr.Lee"$>$:$PN)$

2. [**For some** x:*Student*] [**For some** y:*Prof*]

   $(< x, y >$:Sup **and** $< y,$"Dr.Lee"$>$:$PN)$

[4]There are three paths between *Student* and *PName*, but only two of them correspond with interpretations that are admitted by humans as possibilities within the context of university administration. A heuristic based on context may be useful here to rule out the interpretation "students in the same department as Dr. Lee".

request: Dr. Lee's students
word assignments
"Dr. Lee" PName
"student" Student

1. Students in courses taught by Dr. Lee

```
Student--->SC<---Course--->CP<---Prof--->PN<---PName
        (0,n)   (0,n)       (1,n)   (0,n)     (1,1)    (0,n)
```

7 nodes
weight rel. Student (7+1+0)=8
weight rel PName (7+7+7)=21
absolute weight minimum(8,21)=8

2. Students supervised by Dr. Lee

```
Student  --->Sup<---Prof--->PN<---Pname
          (1,1)     (0,n)    (1,1)   (0,n)
```

5 nodes
weight rel Student (0+0)=0
weight rel PName (5+5)=10
absolute weight minimum(0,10)=0

*** favored interpretation is 2 ***

Figure 4.3: Application of Semantic Relatedness Measure to "Dr. Lee's students"

$TG(Q) = \{Student, PName\}$ determines two query graphs. The path between *Student* and *PName* through *Course* and *Prof* corresponds with interpretation (1), and the path between *Student* and *PName* through *Sup* corresponds with interpretation (2). Figure 4.3 illustrates the calculations for choosing the best interpretation. Recall that $(0, n)$-edges have a weight equal to the number of nodes in the query graph, and $(0, 1)$ and $(1, n)$-edges have a weight of 1. Only forward edges relative to vertex v are counted when computing the weight relative to v. For computing the weight of the query graph for interpretation (1) relative to *Student*, for example, there are three forward edges relative to *Student* with weights 7 (the number of nodes in the QG), 1, and 0 in left to right order. In future examples, detailed calculations of the relative and absolute weights will not be given, since the calculations are straight-forward.

Word Sense Disambiguation

Each primitive constituent of a request denotes 0, 1, or more vertices in the DG. For request Q a target graph is obtained by selecting exactly one vertex from each nonempty set of vertices denoted by primitive constituents of Q. Word sense disambiguation is the problem of selecting the best target graph $TG(Q)$.

Example 4.4. Consider the request "Jones' courses" and suppose that "Jones" could be the name of either a student or a professor. Furthermore, assume that both *Student* and *Prof* are in the current focus. In the university domain if "Jones" names a student, then the request asks for the courses in which Jones is enrolled. Otherwise, it asks for the courses taught by professor Jones. Figure 4.4 illustrates application of the SRM to choose the best

meaning for the ambiguous word "Jones". The SRM selects *Prof* as the best meaning for "Jones" and, hence, the favored interpretation is "courses taught by professor Jones".

```
request: Jones courses
word assignments
   "Jones"   SName
   "course"   Course

 1. Courses taken by the student Jones

 SName--->SN<---Student--->SC<---Course
     (0,n)    (1,1)          (0,n)    (0,n)

  5 nodes
  weight rel. Course     5
  weight rel SName      10
  absolute weight        5


  2. Courses taught by professor Jones

     Course  --->CP<---Prof--->PN<---Pnam e
          (1,n)     (0,n)     (1,1)    (0,n)

  5 nodes
  weight rel Course     1
  weight rel PName     10
  absolute weight       1

  *** favored interpretation is 2 ***
```

Figure 4.4: Application of Semantic Relatedness Measure to "Jones' Courses"

Modifier Attachment

A modifying phrase or clause tends to be physically close to its head noun in a natural language sentence. However, the linear structure of natural language sentences requires that if a head noun has multiple modifiers, then the head noun must be physically separated in the sentence from all but one of them. The process of attaching a modifying phrase or clause to its head noun is called post noun modifier attachment.

Example 4.5. Consider the sentence fragment "A professor in a large department at a major university with no graduate students". The phrase "with no graduate students" could modify "university", "department", or "professor". The phrase "at a major university" could modify either "department" or "professor".

Notation

The attachment of post noun modifiers is indicated using commas. No comma between a noun and the modifying phrase or clause that immediately follows it indicates that the phrase modifies the noun. The way the sentence fragment of Example 4.5 is written "in a large department" modifies "professor", "at a major university" modifies "department", and "with no graduate students" modifies "university". One comma between a noun and the immediately following modifying phrase indicates that the phrase modifies the noun that appears to the left of it in the sentence separated by one other noun. For the fragment "A professor in a large department, at a major university", "at a major university" modifies "professor", not "department". Two commas between a noun and the modifying phrase indicates that there are two nouns separating the phrase and the noun that it modifies, and

so on.

Example 4.6. Consider the request "a professor for a course with no students". An internal representation (IR) for the interpretation "a professor for a course, with no students" for the corresponding sentence fragment follows:

1. [**For some** x:$Prof$] [**For some** y:$Course$]

   ($< y, x >$:CP **and**

   [**For all** z:$Student$] **not**$< z, x >$:Sup)

An IR for the interpretation "a professor for a course with no students" follows:

2. [**For some** x:$Prof$] [**For some** y:$Course$]

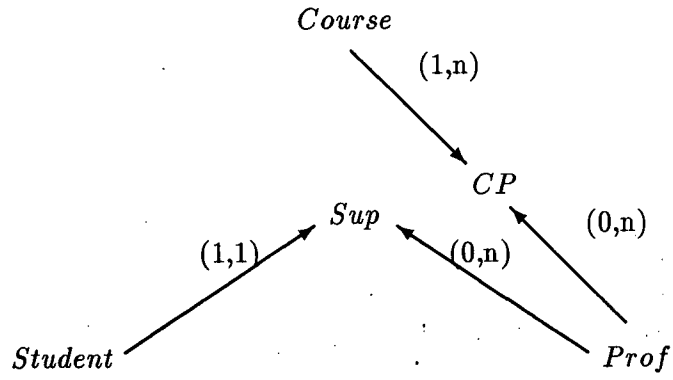
   ($< y, x >$:CP **and**

   [**For all** z:$Student$] **not**$< z, y >$: SC)

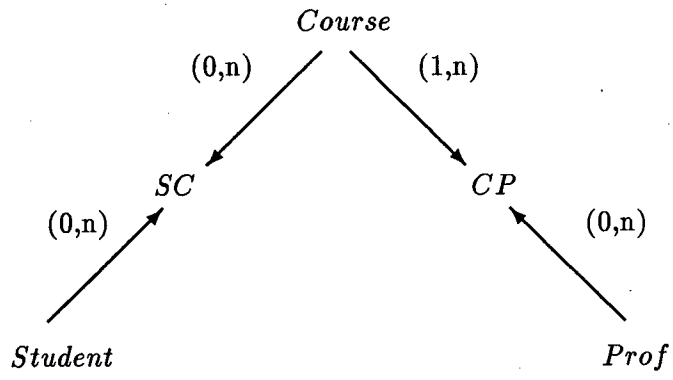
The nouns "student", "course", and "professor" denote, respectively, the vertices $Student$, $Course$, and $Prof$. To determine the best attachments for the modifiers, we again look at the query graph. However, the target graph for the given request will be the same regardless of modifier attachments and therefore will determine the same query graphs for the two interpretations. It is necessary to distinguish the different attachments of the modifiers. We do this by adding vertices to the target graph (TG) to denote modifier attachments.

The TG for the request is $\{Prof, Student, Course\}$. The TG augmented with vertices to denote the attachment of the phrase "with no students" to the noun "professor" is

$$TG_1 = \{Prof, Student, Course, Sup\}.$$



"with no students" modifies "professor"



"with no students" modifies "course"

Figure 4.5: Query Graphs for "a professor for a course with no students"

The TG augmented with vertices to denote the attachment of the phrase "with no students" to the noun "course" is

$$TG_2 = \{Prof, Student, Course, SC\}.$$

A query graph for each target graph is given in Figure 4.5. The query graph for "a professor for a course, with no students" has relative weights 6 on *Student*, 6 on *Course*, 10 on *Prof*, and 5 on *SC*. The query graph for "a professor for a course with no students" has relative weights 6 on *Student*, 6 on *Course*, 10 on *Prof*, and 1 on *SC*. The heuristic favors the attachment of the phrase "with no students" to the noun "course".

It is not always possible to denote the attachment of a modifier to a referent by a vertex that already exists in the DG. In general, a path between the vertices denoted by the referent and the modifier must be added to the TG. The handling of the more complex case is illustrated in Example 4.9 to follow.

4.4.2 The Library Circulation Domain

Our second source of examples is a library circulation domain which has been described by Goldstein in [34]. The entities of interest are libraries, branches of libraries, books, and borrowers. A domain graph for the library domain that is an expansion of an ER diagram given in [34] to include min values for all of the associations is illustrated in Figure 4.6.

The domain contains a collection of libraries each of which consists of one or more branches. A new borrower registers with a library rather than a branch which means that the borrower's library card is valid at any of the branches of the library. There may be several different copies of a book which may be distributed across different branches of

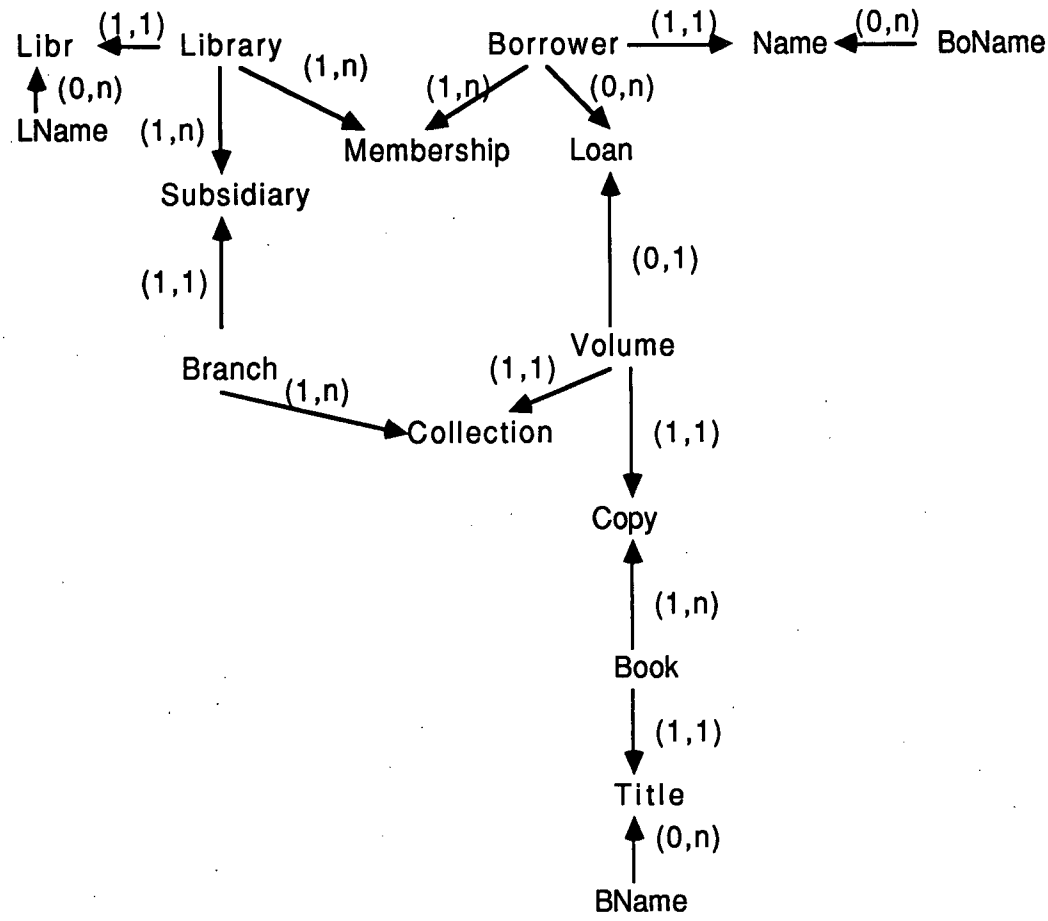


Figure 4.6: Domain Graph for the Library Circulation Domain

different libraries. The set *Volume* contains concrete physical objects called volumes which are copies of a book. A volume belongs to at most one branch. The set *Copy* associates with a book the volumes of the book. A volume is a copy of exactly one book, and every book has at least one volume. The associations *Name*, *Title*, and *Libr* associate borrower names, book titles, and library names with borrowers, books, and libraries, respectively.

Word Sense Ambiguity

Example 4.7. Figure 4.7 illustrates use of the SRM for resolving word sense ambiguity in the request "Does the main library have Joseph Conrad?". The ambiguity arises because the name "Joseph Conrad" could be either the name of a borrower or the title of a book. The SRM favors the interpretation "Does the main library have a borrower named Joseph Conrad?".

A proper noun denotes each value set of which it is a member. Assuming that the possible value sets for the proper noun have been determined (a method for which is given in Chapter 5), the SRM would be useful for ordering the presentation of alternate interpretations to the user. For example, the system would respond "The main library has a borrower Joseph Conrad, but Joseph Conrad is also the name of a book". The reverse order of presentation would be "The main library has a book titled Joseph Conrad, but Joseph Conrad is also the name of a borrower."

request: Does the main library have Joseph Conrad?

1. Does the main library have a borrower named Joseph Conrad?

word assignments
"main" Lname
"library" Library
"Joseph Conrad" BoName

LName--->Libr<---Library--->Membership<---Borrower--->Name<---BoName
7 nodes
weight rel. LName 8
weight rel Library 1
weight rel BoName 8
absolute weight 1

2. Does the main library own a copy of the book titled Joseph Conrad?

word assignments
"main" Lname
"library" Library
"Joseph Conrad" BName

LName --->Libr<---Library--->Subsidiary<---Branch--->Collection<---Volume
BName--->Title <---Book-->Copy <--|

11 nodes
weight rel LName 13
weight rel Library 2
weight rel Bname 12
absolute weight 2

*** favored interpretation is 1 ***

Figure 4.7: Analysis of "Does the main library have Joseph Conrad?

4.4.3 The Medical Domain

Our third source of examples is a medical domain that has been described by Tsichritzis and Lochovsky in [84] and that is a scaled down version of a real application. The entities of interest are hospitals, wards of hospitals, hospital staff, doctors, patients, labs, tests, and diagnoses. The associations of interest are described in Figure 4.8 which also gives the domain graph for the medical domain. Tsichritzis and Lochovsky give an ER diagram for the medical domain and the given domain graph is an expansion of that one to include min values for the associations.

Semantic Ambiguity

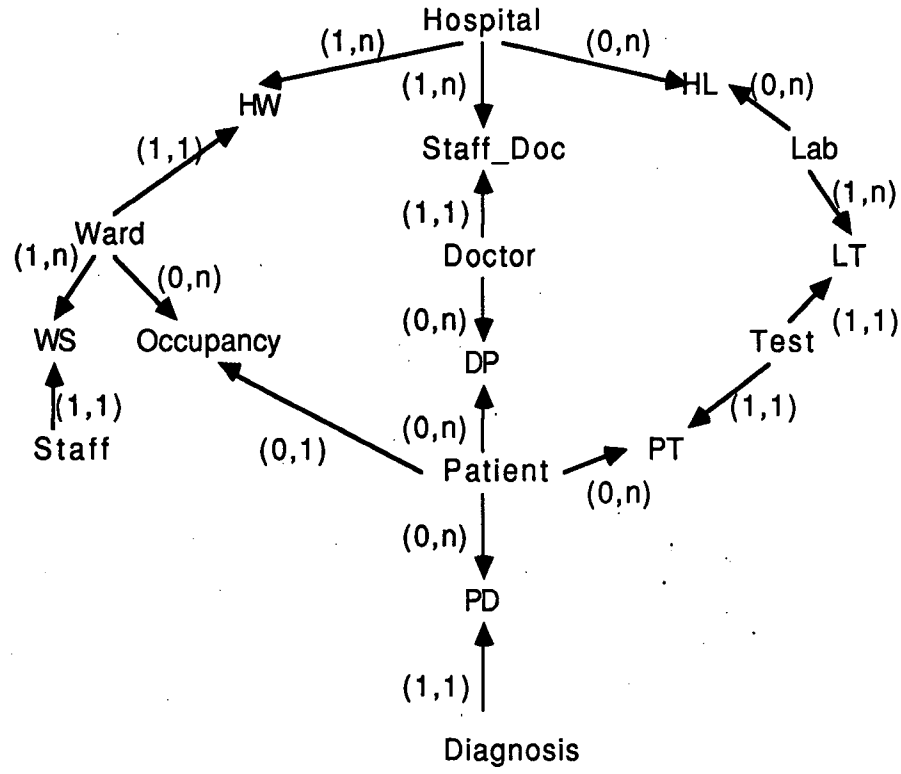
Example 4.8. Resolution of semantic ambiguity in the medical domain is illustrated in Figure 4.9. The request "a patient in a hospital" has three interpretations in the medical domain:

1. a patient who occupies a bed in a ward in a hospital
2. a patient who has a doctor on staff at a hospital
3. a patient who has an order for a test at a lab that does work for the hospital

Interpretation (1) is favored over the other two, and (2) is favored over (3).

Modifier Attachment

Example 4.9. Consider the request "a patient in a hospital with tests". The problem here is to determine whether "with tests" modifies "hospital" or "patient". In either case



HW - associates the wards of a hospital with the hospital

WS - associates employees who work in a ward with the ward

Occupancy - associates with a ward the patients that occupy a bed in the ward

PD - associates with a patient the medical diagnosis(ses) reached for the patient

DP - associates with a patient his or her attending doctor(s)

Staff_Doc - associates with a hospital those doctors that are on staff at the hospital

HL - associates with a hospital those labs that are doing work for the hospital

LT - associates with a lab the medical tests that are to be performed at the lab

PT - associates with a patient those medical test that have been ordered for the patient

Figure 4.8: Domain Graph for the Hospital Domain

request: a patient in a hospital
word assignments
"patient Patient
"hospital Hospital

1. a patient who occupies a ward of a hospital

Hospital--->HW<---Ward--->Occupancy<---Patient

5 nodes
weight rel. Hospital 6
weight rel Patient 1
absolute weight 1

2 a patient who has a doctor on staff at a hospital

Hospital --->Staff_Doc<---Doctor--->DP<---Patient

5 nodes
weight rel Hospital 6
weight rel Patient 6
absolute weight 6

3 a patient who has an order for a test at a lab that does work for a hospital

Hospital --->HL<---Lab--->LT<---Test--->PT<---Patient

7 nodes
weight rel Hospital 8
weight rel Patient 15

absolute weight 8

*** favored interpretation is 1 ***

Figure 4.9: Resolution of Semantic Ambiguity in "a patient in a hospital"

the phrase "in a hospital" modifies the head noun "patient". There is semantic ambiguity in the association between patients and hospitals. For the purpose of the example, we will assume that the ambiguity is resolved in favor of the interpretation "a patient who occupies a bed in a ward of the hospital" based on application of the SRM which has been illustrated in Example 4.8. If the phrase "with tests" modifies "patient", then the request is for a patient who occupies a ward of a hospital and for whom medical tests have been ordered. Otherwise, the request is for a patient who occupies a ward of a hospital which has outstanding orders for tests at some lab. Application of the SRM to this problem is illustrated in Figure 4.10.

The target graph for the interpretation "a patient in a hospital, with tests" is $TG_1 = \{Patient, Hospital, Test, PT\}$. The vertex PT has been added to the target graph to denote the attachment of the phrase "with tests" to the head noun "patient".

The target graph for the interpretation "a patient in a hospital with tests" is $TG_2 = \{Patient, Hospital, Test, Hosp\_Test\}$. The vertex $Hosp\_Test$ denotes the attachment of the phrase "with tests" to the noun "hospital" which is represented by a set defined as follows:

$Hosp\_Test$ **def**
select h:$Hospital$, t:$Test$
where [**For some** l:Lab]($< h, l >$:HL
and $< l, t >$:LT)

$Hosp\_Test$ is the set composition of HL and LT. Min/max values for $Hosp\_Test$ are $(0, n)$ on $Hospital$ and $(0, n)$ on $Test$. These values are computed by taking the product of

request: a patient in a hospital with tests

word assignments

"patient" Patient "hospital" Hospital "test" Test

1. a patient who occupies a bed in a ward of a hospital and for whom tests have been ordered

Hospital--->HW<---Ward--->Occupancy<---Patient--->PT<---Test

7 nodes

weight rel.	Hospital	15	weight rel	Test	1
weight rel	Patient	8	weight rel	PT	1

absolute weight 1

2 a patient who occupies a ward of a hospital which has orders for tests at some lab

nodes 10

```
Hospital  --->HL<---Lab--->LT<---Test--->Hosp_Test
   |
   --->HW<---Ward--->Occupancy<---Patient
```

weight rel Hospital 32 weight rel Patient 22

weight rel Test 31 weight rel HT 31

absolute weight 22

```
Hospital  --->HL<---Lab--->LT<---Test
  |   |
  |   --->Hosp_Test
  |
  --->HW<---Ward--->Occupancy<---Patient
```

weight rel	Hospital	32	weight rel Test 31
weight rel	Patient	21	weight rel Hosp_Test 31
absolute	weight	21	

*** favored interpretation is 1 ***

Figure 4.10: Attachment of Modifiers in "a patient in a hospital with tests"

min/max values on forward edges relative to *Hospital* (to get min/max on *Hospital*) and similarly to get min/max on *Test*, as shown in Chapter 5.

A vertex labeled *Hosp_Test* is introduced to the DG which introduces a cycle. To avoid cyclic query graphs we break the cycle thus generating a collection of minimal connected acyclic subgraphs each of which contains the vertices in the target graph. There are two such graphs for TG_2 which are illustrated in part 2 of Figure 4.10. The query graph determined by a target graph is the minimum weight subgraph among the ones that are obtained by breaking cycles in this way.

The SRM favors the interpretation "a patient in a hospital, with tests" which is the most appropriate one for the given domain based on the following reasoning: The association HL specifies which labs are doing work for which hospitals. However, the work done by a lab need not be for a patient who occupies a ward of the hospital nor for a patient of a doctor on staff at the hospital.[5] The tests carried out by a lab belong to that lab and no other (as stated by the max value of LT on *Test*), however, we cannot determine from the set composition *Hosp_Test* which tests are done for which hospital. If the max value of HL on *Lab* had been equal to 1 (a lab does work for at most one hospital), then it would be possible to determine from *Hosp_Test* which tests are done for which hospital. Our semantic relatedness measure, by giving a lower weight to edges labeled with a max value of 1, favors least the attachment of a modifier to a referent where the two are related in the

[5] A query can be written to retrieve all hospitals h and labs l such that some patient who occupies a ward of h has an order for a test carried out at l. The result of this query is a subset of HL. Therefore, HL is partially derivable from the other sets in the domain. Using Codd's terminology [16] HL would be called a semi-derivable set.

domain by a many-to-many association.

4.4.4 Analysis

We have seen a variety of examples from different domains that illustrate the use of min/max values for resolving ambiguities in natural language requests. In this section the results of the examples are analysed to gain an understanding of why the heuristic works. Here the term *association* will have its mathematical meaning given in Section 2.3.3, and the term *relationship* will be used informally to refer to connections between words in sentences or objects in the domain. The examples have focused on disambiguating prepositional phrase (PP) attachments that use the preposition "with", choosing the most appropriate meaning for pre noun modifiers (in particular, possessives), and using the meaning of a verb to disambiguate the subject and object of the verb.

Our heuristic is intended to be applied simultaneously to the problems MA, WSD, and SA. The different components of the heuristic interact with each other producing better results than if each individual component were to be applied separately. In the words of Jensen and Binot [52] "The cumulative effect of many heuristics, and not the perfection of each one taken separately, has to do the job".

For the purpose of analysis we can separate out the component of the heuristic that is being used to solve post noun modifier attachment (MA). Each possible PP attachment is represented as a set, and the possible attachments are ordered by the extent to which they represent a set that is existence dependent on the set that represents the meaning of the request as a whole. This is to say that the natural language request is conceived as a whole

and the attachment of a modifier as a part of the whole. A measure of the extent to which the attachment of the modifier is *part of* the whole derives from the *part of* relationships that build the whole.

Examples 4.6 and 4.9 address the problem of post noun modifier attachment. For the request "a professor in a course with no students" (Example 4.6) we find that the attachment of "with no students" to "course" is more *part of* the request than the alternate attachment. In the university domain a student is existence dependent on the professor who supervises his or her research, and one might think that the attachment of "with no students" to "professor" would be indicated. However, our heuristic is concerned with the extent to which the PP is *part of* the remainder of the request, and this occurs when the PP modifies "course". In Example 4.9 the request is "a patient in a hospital with tests", and the heuristic indicates that the best attachment of the PP "with tests" is to the noun "patient". In the medical domain a test is existence dependent on the patient being tested, whereas no relationship between hospitals and tests previously exists or can be derived.

The results cannot be understood independently from other heuristics that are useful for choosing the best interpretation for a request. However, an examination of the outcome of the examples is useful for the given requests which exclude many of the linguistic problems for which other heuristics would be needed. A complicating factor is that, since the heuristic relies on global knowledge about the request, it is more useful for lengthy requests than the simple fragments of requests that are illustrated in the examples.

Examples 4.3 and 4.8 illustrate how the heuristic is used to resolve semantic ambiguities. For the request "Dr. Lee's students" the interpretation "students supervised by Dr Lee"

is favored over "students in Dr.Lee's courses" because the students supervised by Dr. Lee are exclusively his, whereas students in Dr. Lee's courses may also be in courses taught by professors other than Dr. Lee.

For the purpose of analysis we can distinguish a component of our heuristic that deals with semantic ambiguity involving the preposition "in" when it is used in a PP that modifies a noun. The second example of semantic ambiguity features the request "patients in hospitals". There were three different possible interpretations in the medical domain for the request "patients in hospitals", and the one "patients who occupy a bed in a ward of the hospital" is favored because a patient occupies at most one ward and a ward is existence dependent on a hospital. Weaker forms of relationships exist between patients and hospitals for the other interpretations.

Two examples were given to illustrate use of the heuristic to resolve word sense ambiguities. Example 4.7 features the request is "Does the main library have Joseph Conrad?". Ambiguity resides in the proper noun "Joseph Conrad". The heuristic does not distinguish among the possible interpretations. A method of testing for membership of proper nouns in value sets is needed as a basis for a complementary heuristic that reduces the number of possible value sets that the proper noun may denote before the SRM is applied.

Example 4.4 illustrates the resolution of word sense ambiguity in a pre-noun modifier that indicates possession. The request is "Jones' courses". We have assumed that ambiguity remains in the proper noun "Jones" after application of context analysis. In that case, the interpretation "courses taught by professor Jones" is favored over "courses taken by student Jones" because in the University domain there is an exclusive relationship between courses

and professors, whereas there is no restriction on the relationship between courses and students.

Based on the results of the examples we conclude that the min/max values model the *part of* construction in the English language ("petals of a flower", "handles of a vase", "wards of a hospital", "copies of a book") as well as weaker forms of possessive relationships. In fact, they provide a metric for measuring the strength of the possession (e.g.,"petals of a flower" is stronger than "a professor's students"). Our heuristic reformulates the *part of* relationships that are referenced in a natural language request as a *part of* relationship between the request itself and the ambiguous components of the request. The favored interpretation is the one in which there is the strongest *part of* relationship between the ambiguous components and the request itself.

4.5 Confirmation of the Heuristic

In this section we establish that the heuristic works well for resolving ambiguities. Section 4.5.1 shows that the heuristic is unaffected by the arbitrariness of the design of the SET schema. Section 4.5.2 establishes that the parameters of the heuristic can be varied without affecting its outcome. Finally, Section 4.5.3 shows that previous heuristics which capture knowledge from relational schemas for resolving ambiguities are actually using knowledge expressed by the min/max values.

4.5.1 Sensitivity of the Heuristic to SET Schema Design Alternatives

In this subsection, we investigate the sensitivity of our heuristic to the arbitrariness of the design of the SET schema. Three different ways in which schemas for the same domain may differ while still expressing the same information are considered. The three different ways applied in various combinations any number of times permit most of the usual changes in the schema to be described. We find that the heuristic is very insensitive to the usual ways in which schemas for the same domain may differ.

A pair (a,b) treated either as an entity or a relationship

Some objects in the real world can be treated either as entities or as relationships. An example of such an object is a marriage. Schemas for the two alternatives are illustrated in Figure 4.11.

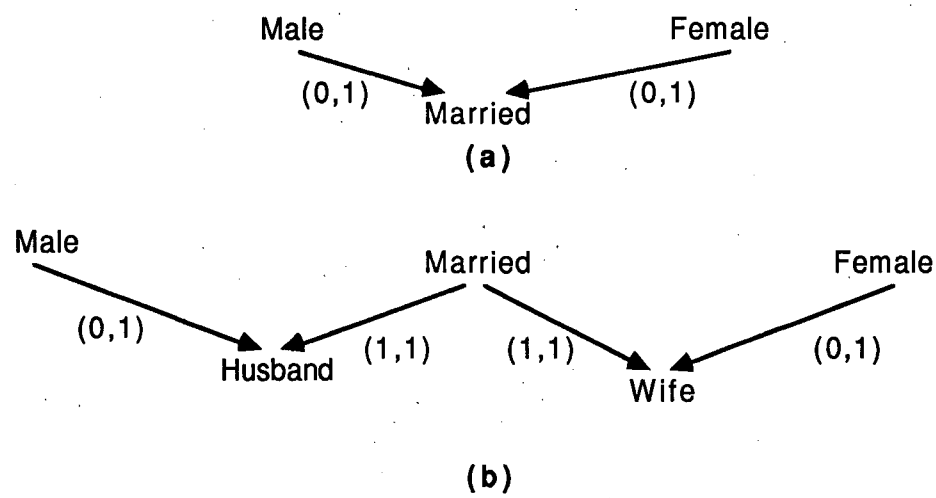


Figure 4.11: Marriage as an Entity and a Relationship

In schema (a), a marriage is considered to be a relationship. Association *Married* is

defined as a subset of the Cartesian product ($Male \times Female$) of two sets $Male$ and $Female$. A male may not be married, but if he is, he is married to at most one female, and the same holds for females. These constraints are expressed by the min/max values of $Married$ on $Male$ and on $Female$.

In schema (b), a marriage is considered to be an entity. If male m and female f are married and that marriage is represented by the entity mf, then the pair (m, mf) is a member of $Husband$ and the pair (mf, f) is a member of $Wife$. Since every marriage has both a husband and a wife, the min values of $Married$ on $Husband$ and on $Wife$ are both 1. Since every marriage has at most one husband and at most one wife, the max values of $Marriage$ on $Husband$ and on $Wife$ are both 1.

Although they look different, the two schemas express the same information. Likewise, the domain graphs (DGs) for the two schemas have the same weight. The two new edges that are introduced in Schema (b) are both (1,1)-edges each of which has a weight of 0. Otherwise, the edges in the two DGs are identical and their weights are identical because the two new edges contribute nothing to the weight.

Schema Equivalence

A useful property of a heuristic that operates in SET schemas is that it gives the same outcome in schemas that have been designed by different database designers for the same domain. Although different schemas for the same domain may look different, they express the same information. In order to determine when two schemas express the same information, a notion of *schema equivalence* is needed. The definition presented here is intended

to provide a narrow definition for a very difficult concept. We show that under this narrow definition, our heuristic in invariant to arbitrary decisions in the design of the SET schema.

For two schemas to be equivalent, they must contain the same sets in the sense that for every set S_1 in one there exists a set S_2 in the other such that S_1 and S_2 are *intensionally equal*.[6] An *implicitly defined set* is a set which is not a declared set of the schema but can be defined in terms of the declared sets.

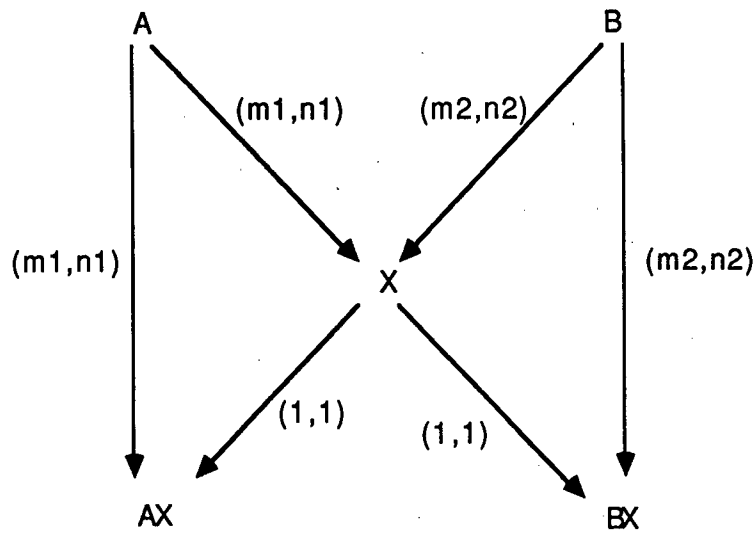
The following conventions will be adopted for naming sets:

1. Sets that are intensionally equal have the same names.

2. Two sets which are related to each other by the entities in one being treated as relationships in the other, have names of the form S and S'.

For illustration, consider the schema of Figure 4.12 which contains an association X whose members are treated as pairs. X is a base set, and AX and BX are defined sets whose definitions are also given in the figure. Consider schemas (a) and (b) of Figure 4.13. Schema (a) contains only the primitive and base sets of the schema of Figure 4.12, while schema (b) contains the sets X', AX' and BX' whose names are intended to convey additional information. In particular, the names indicate that there is a one-to-one correspondence

[6] If S_1 and S_2 are intensionally equal, then at any instance in time the extensions of the two sets are equal. Consider the converse. Suppose that every student enrolled in CPSC 504 must also be enrolled in the lab for the course CPSC 504-LAB, and that no student takes the lab without being enrolled in the course. The two sets "Students enrolled in CPSC 504" and "Students enrolled in CPSC 504-LAB" are extensionally equal, but are they intensionally equal? We adopt the following view of intensional equality: *Two sets are intensionally equal if and only if the database system (or the user of the database system) never allows them to be extensionally unequal.*




```
def      AX
select   x:A, <x,y>:X

def      BX
select   <x,y>:X, y:B
```

Figure 4.12: Association X Treated as a Set of Pairs

between X' and X deriving from the fact that the members of X, which are pairs, are treated as entities of X'. A one-to-one corresondence between AX' and AX derives from the correspondence between X' and X. The right elements of corresponding pairs of AX' and AX are corresponding members of X' and X. Similar correspondences are indicated for BX' and BX.

Definition 4.1. Two schemas are *equivalent* if

1. They contain the same primitive sets.

2. Every declared set in one can be implicitly defined from sets in the other, and vice versa.

Notice that schema (b) of Figure 4.13 includes a primitive set X' which does not appear in schema (a) of the same figure. The two schemas are not equivalent by the above definition. However, our knowledge of the correspondence between the sets X and X' can be used to construct schemas that have the same primitive sets in the following way: An association $P \subseteq X \times X'$ is introduced to schema (a) as illustrated in Figure 4.14. Min/max values for P indicate the one-to-one correspondence. With the introduction of P, we can show that the two schemas are equivalent. In particular, it must be possible to define every declared set in (a) from sets in (b) and every declared set in (b) from sets in (a). When a base association R in one schema is defined from sets in the other, the defined association T will have the same parent sets as R and min/max values identical to those of R.

In Figure 4.15, association X of schema (a) in defined in terms of the sets of schema (b). It follows from the definition that the min/max value of X on A is $(m1, n1)$ and of X on B is $(m2, n2)$. In Figure 4.16 , association AX' of schema (b) is defined in terms

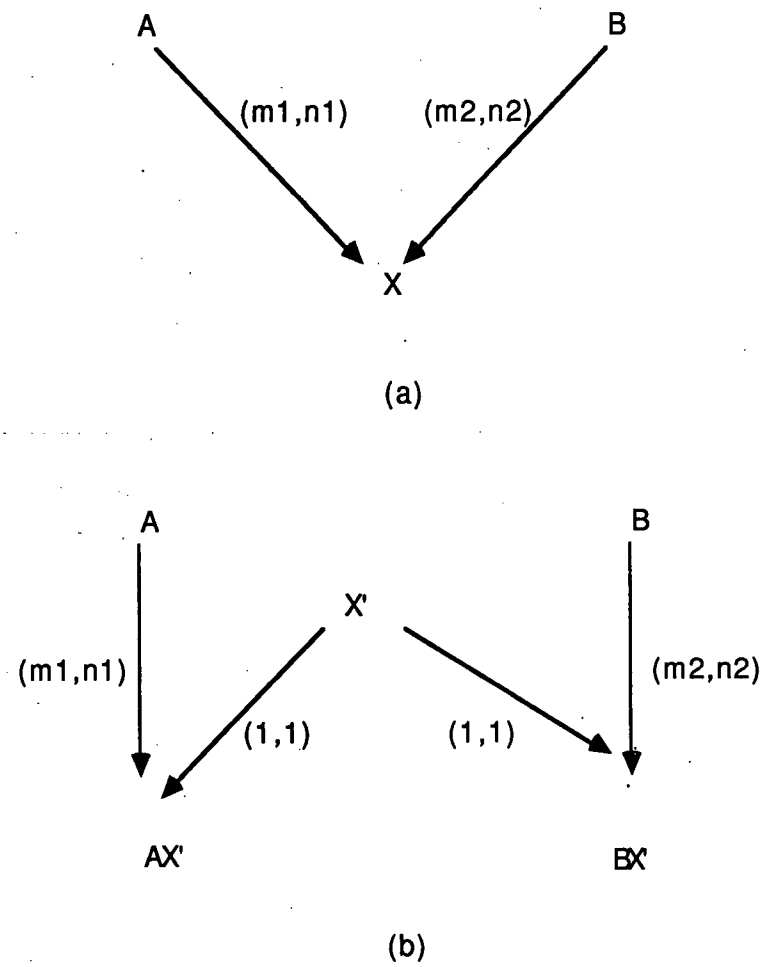


Figure 4.13: Association X Treated as a Set of Pairs and as a Primitive Set

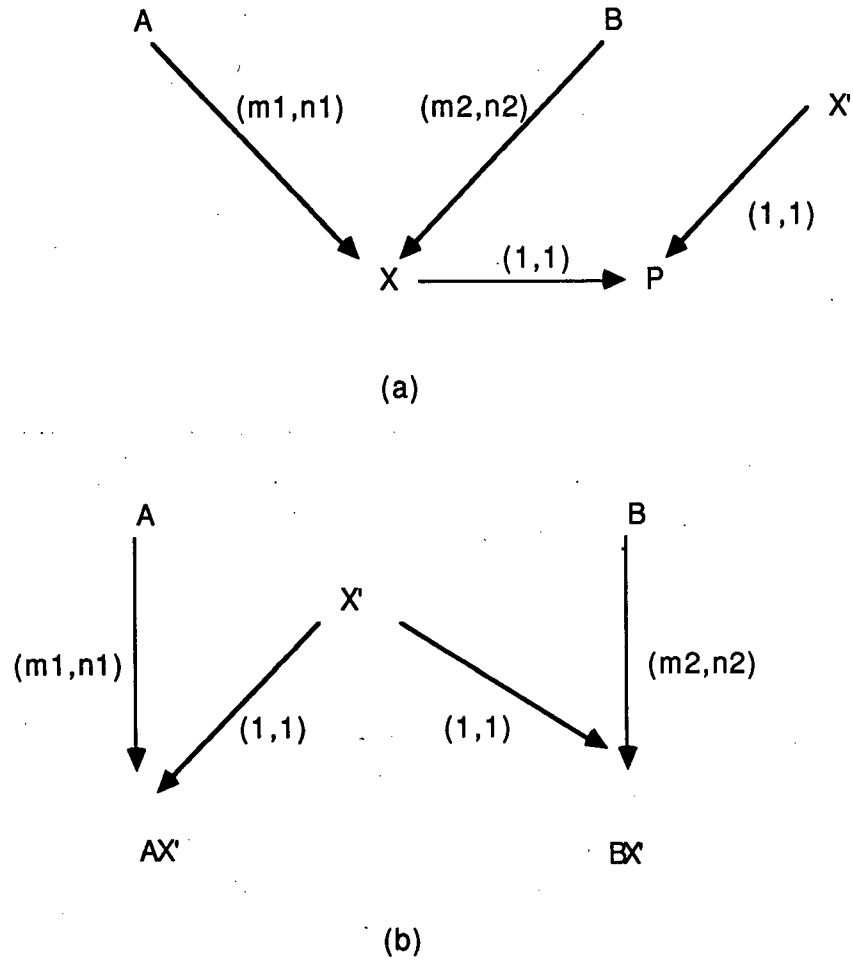


Figure 4.14: Schemas Augmented with Primitive Sets

of the sets of schema (a). A similar definition can be given for BX'. It follows from the definition that the min/max value of AX' on A and on X' are, respectively, $(m1, n1)$ and $(1, 1)$. Similar, correspondences between min/max values of the base set BX' of schema (b) and the implicitly defined set BX' of schema (a) can be easily shown. The two schemas are equivalent because they contain the same primitive sets and for every declared set D in one there exists an implicitly defined set in the other which is intensionally equal to D.

```
def      X
select   x:A, y:B
where    [For some  z:X' ]
             (<x,z>:AX' and  <z,y>:BX')
```

Figure 4.15: Association X Defined from Schema (b)

```
def      AX'
select   a:A, z:X'
where    [For some  b:B ]
             <<a,b>,z>:P
```

Figure 4.16: Association AX' Defined from Schema (a)

We wish to show that the outcome of the heuristic introduced in Section 4.3 for measuring semantic relatedness is the same whether it operates in schema (a) or schema (b). Consider a natural language request with target graph TG_1 whose vertices denote declared

or implicitly defined sets of schema S_1. If two schemas S_1 and S_2 are equivalent, then for every target graph TG_1 in S_1 there exists a target graph TG_2 in S_2 such that for every set in TG_1 there exists an intensionally equal set in TG_2 and the converse. It is important to permit target graphs to include implicitly defined sets. For the schemas of Figure 4.14 the set X is a declared set in schema (a) and an implicitly defined one in schema (b).

Observe that if TG_1 is a target graph containing any subset of the vertices denoting sets of schema (a), and TG_2 is the corresponding target graph in schema (b), then the minimum weight query graph determined by TG_1 in schema (a) and the minimum weight query graph determined by TG_2 in schema (b) have identical weights. For example, the weight of any target graph containing X' is zero in both schemas.

An n-ary association treated as a collection of n binary associations

A ternary association ABC with parent sets A, B, and C is illustrated in Figure 4.17 (a) while (b) illustrates the projections of ABC on each of its parent sets. The set $AABC$ consists of those pairs $< a, < a, b, c >>$ for which $< a, b, c >$ is a member of ABC, and similarly for the sets $BABC$ and $CABC$. It follows that the min/max value of $AABC$ on A is the same as that of ABC on A. Edges with identical min/max values are pointed out in the figure by a corresponding numbers of stars on the edges. Since every triple $< a, b, c >$ which is a member of ABC has exactly one A-component, the min/max value of $AABC$ on A is (1,1) and similarly for the sets $BABC$ and $CABC$.

Consider schemas (a) and (b) of Figure 4.18. Schema (a) includes the sets of schema (a) of Figure 4.17, while in schema (b) each of the defined sets of schema (b) of Figure 4.17

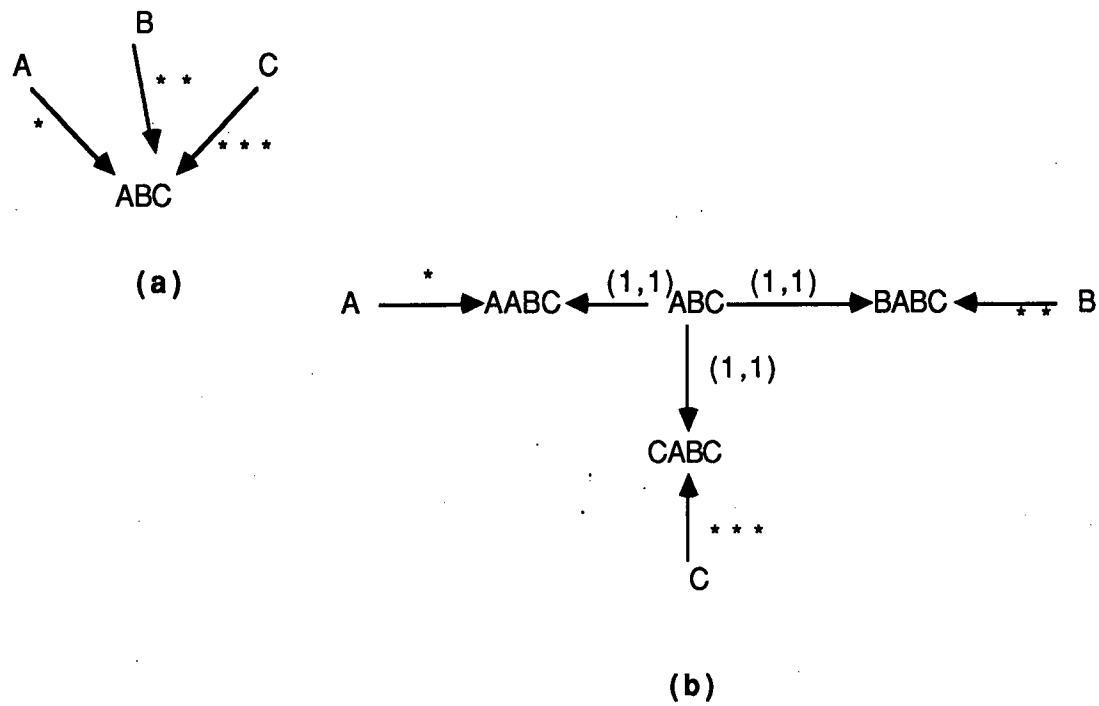


Figure 4.17: Association ABC Treated as a Set of Triples

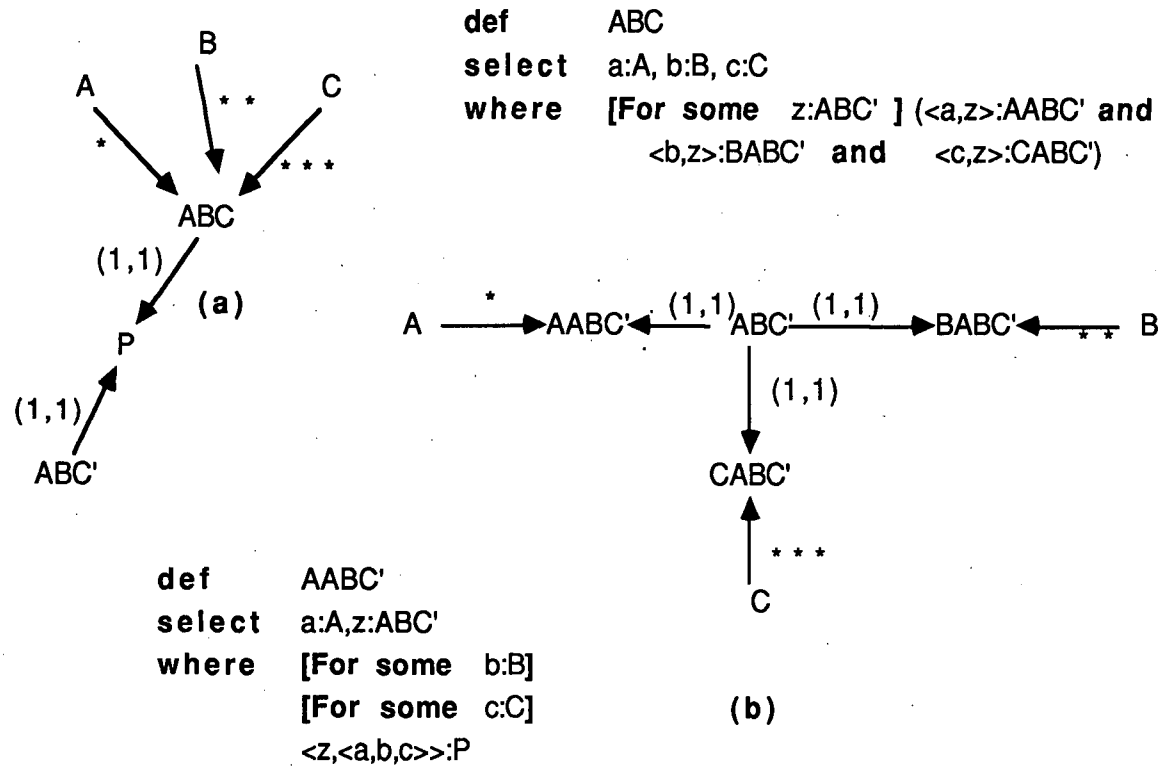


Figure 4.18: A Ternary Association Treated as Three Binary Associations

is represented as a base set. There is a one-to-one correspondence between the sets ABC and ABC' because the members of ABC, which are triples, are treated as entities of ABC'. The association $P \subseteq ABC \times ABC'$ of schema (a) indicates the one-to-one correspondence.

Proof of Equivalence:

(From (b) to (a)) In Figure 4.18, association $AABC'$ of schema (b) is defined in terms of the sets of schema (a). Suppose that the min/max value of ABC on A is (m, n). Thus, every member of A appears as the left element of at least m and at most n triples of ABC. For a given $a \in A$, whenever there exist b and c members of B and C respectively, such that $< a, b, c > \in ABC$, there will exist a $z \in ABC$ such that $< a, z > \in AABC'$. This follows from the min/max values of P and from the definition of $AABC'$. Thus, the min/max value of $AABC'$ on A is (m, n). It also follows from the min/max values of P and the definition of $AABC'$ that the min/max value of $AABC'$ on ABC' is (1,1).

(From (a) to (b)) In Figure 4.18, association ABC of schema (a) is defined in terms of the sets of schema (b). Suppose that the min/max value of $AABC'$ on A is (m, n). We wish to show that for every $a \in A$, whenever there exists a $z \in ABC'$ such that $< a, z > \in AABC'$, there exist $b \in B$ and $c \in C$ such that $< a, b, c > \in ABC$. This follows from the min/max values of $BABC'$ on ABC and $CABC'$ on ABC which are both (1,1) and, therefore, the min/max value of ABC of schema (a) on A is (m, n).

Note that the correspondences hold for the standard as well as general min/max values. Now that it has been established that schemas (a) and (b) of Figure 4.18 are equivalent, let us consider the outcome of our heuristic for measuring semantic relatedness in each schema.

Since the weight of a (1,1) edge is 0, the weights of the edges $(ABC', AABC')$, $(ABC', BABC')$,

and $(ABC', CABC')$ are all zero. If the target graph for a request is $TG = \{A, B, C\}$ then, although the query graphs for TG in the two schemas are different, their weights are the same. The principle is easily generalized to n-ary associations, $n \geq 2$.

An n-ary association treated as a collection of associations of arity less than n

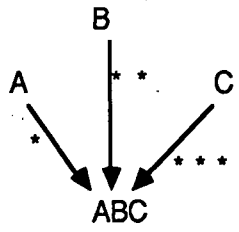
Consider the schemas (a) and (b) of Figure 4.19 . Note that the sets AB and $(AB)C$ of schema (b) are defined in that figure.

A pair $< a, b >$ cannot be a member of AB without also being the left element of a pair $<< a, b >, c >$ of $(AB)C$. Therefore, the min value of $(AB)C$ on AB is 1. The max value of $(AB)C$ on AB may be either 1 or n. If it is 1, then schema (b) expresses different information from schema (a), and hence they are not equivalent schemas.

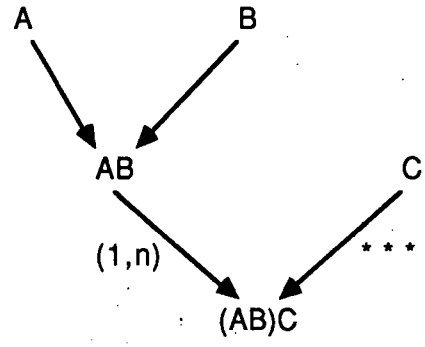
There are three possible alternate schemas for (a), which are illustrated in Figure 4.19 (b), (c) and (d). If any one of the max values of $(AB)C$ on AB, $(AC)B$ on AC, or $(BC)A$ on BC is 1, then the schema of Figure 4.19 (a) will express different information from the schemas of (b), (c), or (d). Therefore, we will be concerned only with the case where all three of the max values are n.

The min/max values of ABC on C and of $(AB)C$ on C are identical since whenever $< a, b, c >$ is a member of ABC, $<< a, b >, c >$ is a member of $(AB)C$, and the converse. The min value of ABC on A is equal to the min value of AB on A since whenever a member a of A participates in a triple $< a, b, c >$ of ABC, it also participates in a pair $< a, b >$ of AB, and the converse.

The max value of ABC on A and of AB on A are not necessarily equal. If the max



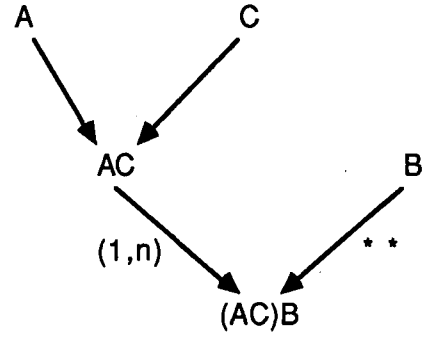
(a)



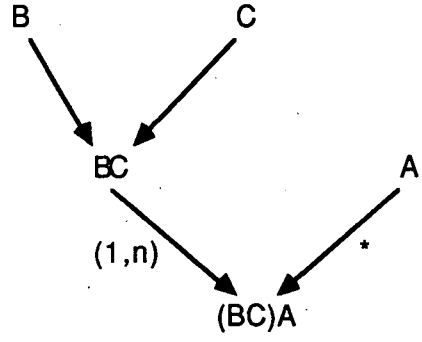
(b)

```
def     AB
select  <a,b>:(A X B)
where   [For some c:C ]
            <a,b,c>:ABC

def     (AB)C
select  <a,b>:AB, c:C
where     <a,b,c>:ABC
```



(c)



(d)

Figure 4.19: A Ternary Association Treated as a Collection of Associations of Arity Less Than Three

value of ABC on A is 1, then the max value of AB on A is also 1, but not the converse. If the max value of ABC on A is n, then the max value of AB on A may be either 1 or n.

The schemas are represented using the domain graph notation. Notice that for each of the graphs (b), (c), and (d) one of the vertices A, B, or C is distinguished as having at most one forward edge relative to it. Vertex C is so distinguished in (b), and the edge $C \longrightarrow (AB)C$ is called a *side edge* starting at A.

If the max value of AB on A is 1, then schema (b) expresses different information from schema (a). We will assume that the max value of AB on A is n. Since (b), (c) and (d) are the same graph but with the vertices labeled differently, and furthermore the graphs themselves are symmetric, a number of max values are being given by the above statement.

```
def      ABC?
select   a:A,b:B,c:C
where    <<a,b>,c>:(AB)C
```

Figure 4.20: Definition of $ABC?$ from Schema (b)

The min/max values of ABC on A, on B, and on C are identical to the min/max values on the side edges of (d), (c), and (b), respectively as indicated by the stars in the figure. It is not possible to define a set on any one of (b), (c), or (d) that is intensionally equal to ABC of schema (a). Consider the definition of $ABC?$ given in Figure 4.20. The max value of $(AB)C$ on C is necessarily equal to the max value of $ABC?$ on C. However, the

max value of AB on A is not necessarily equal to the max value of $ABC?$ on A. A counter example follows: If the max value of AB on A is 1 and of $(AB)C$ on C is n, then the max value of $ABC?$ on A may be either 1 or n. An example where the max value of $ABC?$ on A is n follows: If $a \in A, b \in B, c_1, c_2 \in C$, then a possible extension for AB is $\{< a,b >\}$ and for $(AB)C$ is $\{<< a,b >, c_1 >, << a,b >, c_2 >\}$. By the definition of $ABC?$, the extension of $ABC?$ is $\{< a,b,c_1 >, < a,b,c_2 >\}$ and, hence, the max value of $ABC?$ on A is not equal to 1.

One way to construct a schema that expresses the same information as (a) is to combine schemas (b), (c), and (d). Figure 4.21 gives a definition of the set ABC of schema (a) from schemas (b), (c), and (d). The definition guarantees that the min/max values of the defined set ABC are identical to the corresponding ones of the base set ABC of schema (a).

A schema is a set of names of sets (together with other information about the sets such as their intensions and min/max values). The *union* of two schemas is the set of names that appear in one or other or both of the schemas. Let $(a+b+c)$ denote the schema which is the union of schemas (b), (c), and (d).

```
def      ABC
select   a:A,b:B,c:C
where    <<a,b>,c>:(AB)C
  and    <<a,c>,b>:(AC)B
  and    <<b,c>,a>:(BC)A
```

Figure 4.21: Definition of ABC from Schema $(b + c + d)$

The schemas (a) and $(b + c + d)$ express the same information. Each of (b), (c), and (d) is a query graph in $(a + b + c)$ for a request that references vertices A, B, and C. The following observations leads us to conclude that the weights of the minimum weight query graphs for target graph $TG = \{A, B, C\}$ in (a) and in $(a + b + c)$ are identical.

Observation 1. The minimum weight query graph among (b), (c), and (d) has weight equal to the weight of its side edge.

Proof (by contradiction): Suppose without loss of generality, that (d) is the minimum weight query graph among (b), (c) and (d). Suppose contrary to Observation 1 that the weight of (d) is determined by vertex C. There are two forward edges relative to C one of which is a $(1, n)$ edge of weight 1 and the other a $(0, n)$-edge of weight greater or equal to the weight of any other edge. If the weight of a $(0, n)$-edge is w, then the weight of (d) is $w + 1$.

There is another query graph (b) with side edge starting at C whose weight is determined by the single edge $C \longrightarrow (AB)C$. The weight of $C \longrightarrow (AB)C$ and also of (b) is smaller or equal to w. Since the weight of (d) is $(w + 1)$, the weight of (b) is smaller than the weight of (d), which contradicts the assumption that the minimum weight query graph is (d).

Since (b), (c), and (d) are the same graphs with the vertices relabeled, we could have chosen any one of them as the minimum weight one. □

Observation 2. The minimum weight query graph among (b), (c), and (d) has weight equal to the weight of (a).

Proof ($\Rightarrow$) Suppose that (b) is the minimum weight query graph. The side edge $C \longrightarrow (AB)C$ of (b) is labeled with the same min/max value as the edge $C \longrightarrow ABC$ of (a). If the

weight of any other edge, say $B \longrightarrow ABC$ of (a) is less than the weight of $C \longrightarrow ABC$, then one of the query graphs (c) or (d) would have a side edge with that weight. It follows from the previous observation that the minimum weight query graph would be (c) or (d) but not (b), a contradiction of our initial assumption. Therefore, the weight of edges $A \longrightarrow ABC$ and $B \longrightarrow ABC$ of (a) must be greater than or equal to the weight of $C \longrightarrow ABC$. Since the weight of (a) is equal to the weight of $C \longrightarrow ABC$, and the weight of (b) is equal to the weight of the side edge $C \longrightarrow (AB)C$, and the min/max values of $C \longrightarrow ABC$ and $C \longrightarrow (AB)C$ are identical – it follows that the weights of (a) and (b) are identical.

($\Leftarrow$) If the weight of (a) is determined by the edge $C \longrightarrow ABC$ then the weight of that edge is smaller or equal to the weights of the remaining edges.

For every edge of (a) there is a side edge in one of (b), (c) or (d) such that the min/max values of the two edges are identical. The side edge corresponding to $C \longrightarrow ABC$ is $C \longrightarrow (AB)C$. (b) must be the minimum weight query graph because, otherwise, one of the query graphs (c) or (d) would have a side edge of weight less than the weight of $C \longrightarrow (AB)C$ and also of $C \longrightarrow ABC$. This is a contradiction of our assumption that the weight of $C \longrightarrow ABC$ is smaller or equal to the weights of the remaining edges. Since the weight of (a) is equal to the weight of $C \longrightarrow ABC$, the weight of (b) is equal to the weight of $C \longrightarrow (AB)C$, and the weight of $C \longrightarrow ABC$ is equal to the weight of $C \longrightarrow (AB)C$, the query graphs (a) and (b) have identical weights.

A similar argument permits us to conclude that the weights of (a) and (c) are identical if c is the minimum weight query graph and that the weights of (a) and (d) are identical if d is the minimum weight query graph. □

4.5.2 Varying the Parameters of the Heuristic

In this subsection, the effects of varying the parameters of the heuristic are investigated. The parameters are the weights assigned to the different types of edges. The analyses of Section 4.5.1 provide convincing evidence that the weight of a (1,1)-edge should be 0. Otherwise, the heuristic associates with a request different weights in different schemas for the same domain. For the other types of edges, let their weights be denoted by variables X, Y, and Z as illustrated in the following table:

min/max	weight
$(1,1)$	0
$(0,1)$	Y
$(1,n)$	Z
$(0,n)$	X

Figure 4.22 illustrates the computations of semantic relatedness for two of the interpretations for the phrase "Dr. Lee's Students". The variables X, Y, and Z are assumed to be positive integers including zero.

The computations using specific values for the edge weights have been illustrated in Figure 4.3. To obtain the same relative ordering of the query graphs as with the specific values the following conditions must hold:

1. $X + Z \leq 3X$

2. $X + Z > 0$

3. $2X \geq 0$

Thus, if $X \geq 1$, the same relative ordering of the interpretations is obtained.

request: Dr. Lee's students
word assignments
"Dr. Lee" PName
"student" Student

1. Students in courses taught by Dr. Lee

```
Student--->SC<---Course--->CP<---Prof--->PN<---PName
        (0,n)   (0,n)       (1,n)   (0,n)     (1,1)    (0,n)
```

7 nodes
weight rel. Student (X+Z+0)=X+Z
weight rel PName (X+X+X)=3X
absolute weight minimum(X+Z,3X)=X+Z

2. Students supervised by Dr. Lee

```
Student  --->Sup<---Prof--->PN<---Pname
         (1,1)     (0,n)    (1,1)   (0,n)
```

5 nodes
weight rel Student (0+0)=0
weight rel PName (X+X)=2X
absolute weight minimum(0,2X)=0

Figure 4.22: Varying the Parameters for "Dr. Lee's students"

In the remainder of this subsection, a similar set of inequalities is given for each of the requests that has been studied in Section 4.4. Our objective is to determine whether the original set of edge weights is necessary and, if not, the extent to which the original edge weights can be varied without affecting the results of the heuristic.

Jones' courses

1. $X \leq 2X$

2. $Z \leq 2X$

3. $Z < X$

Conclusion: $X > 0$, $X > Z$.

A professor for a course with no students

1. $X \leq 2X$

2. $X \leq X + Z$

3. $Z \leq 2X$

4. $Z \leq X + Z$

5. $Z < X$

Conclusion: $X > 0$.

Does the main library have Joseph Conrad?

1. $Z \leq X + Z$

2. $2Z \leq X + Z$

3. $2Z \leq X + 2Z$

4. $Z < 2Z$

Conclusion: $X \geq 1$, $Z > 0$.

A patient in a hospital

1. $Y \leq X + Z$

2. $X + Z \leq 2X$

3. $(Y < X + Z) \vee (Y < X)$

4. $X < X + Z$

Conclusion: $Z > 0$, $X > 0$.

A patient in a hospital with tests

1. $Y \leq X + Y$

2. $Y \leq 2X + Z$

3. $2X + 2Y \leq 3X + 2Z$

4. $2X + 2Y \leq 3X + Z$

5. $2X + 2Y \leq 2X + Z$

6. $3X + 2Z \geq 2X + Z + Y$

7. $3X + Z \geq 2X + Z + Y$

8. $2X + 2Z \geq 2X + Z + Y$

9. $(2X + 2Y > Y) \vee (2X + Z + Y > Y)$

Conclusion: $X > 0$; if $Y > 0$ then $Z > 0$.

The results indicate that the specific edge weights may be varied without alternating the outcome of the heuristic. For the sample requests of Section 4.4 the same relative ordering on interpretations would have been obtained if the conditions $(X > 0)$ and $(Z > 0)$ hold.

4.5.3 The Relationship between Distance in Relational and SET Schemas

It is common practice to base the design of a relational schema on an Entity Relationship Diagram [12] that has been previously developed for the domain. SET schemas are a refinement of Entity-Relationship Diagrams. We will assume a schema design methodology introduced by Gilmore in [32].

In this section, a correspondence is drawn between our heuristic and a previous semantic relatedness measure in relational schemas. Specification of the correspondence is simplified by choosing a convenient assignment of values for the different types of edges consistent with the results of sections 4.6.1 and 4.6.2 which indicate that a (1,1)-edge should have weight zero, and the remaining types should have weight greater than zero.

If S is a SET schema and R is the relational schema designed from S, then R and S will be called *corresponding* SET and relational schemas. Distance in a relational schema between objects denoted by sentence constituents is usually measured by the number of links that must be followed to get from one object to the other. Distance between objects in a SET schema is the distance in its domain graph measured by the method described in Section 4.3.(i.e., If the objects are denoted by vertices v_1 and v_2 in the DG, then the distance between the objects is the weight of the minimum weight path between v_1 and v_2. The weight of a path is the minimum of the weights relative to v_1 and relative to v_2, where the weight of an edge in the path depends on its min/max value.) Distance in a relational and the corresponding SET schema is illustrated in Figure 4.23. Here two sentence constituents whose semantic relatedness we wish to determine, denote objects X and Y in the relational schema and objects A and B in the SET schema.

The notion of *distance* in SET schemas, briefly referred to as the *formal measure*, is founded on well understood mathematical concepts (those expressed by the min/max values). We find that the measure of distance in relational schemas usually described in the natural language literature [40, 51, 93, 55] is a restricted version of our formal measure.

The Use of Min/max Values for Relational Schema Design

A *base relation* in the relational model is a relation that cannot be completely derived by means of a relational query from other relations in the database. Within the SET schema methodology, each base relation is viewed as a set defined in terms of sets named in the DG. The problem of designing a database schema is to provide definitions for the sets

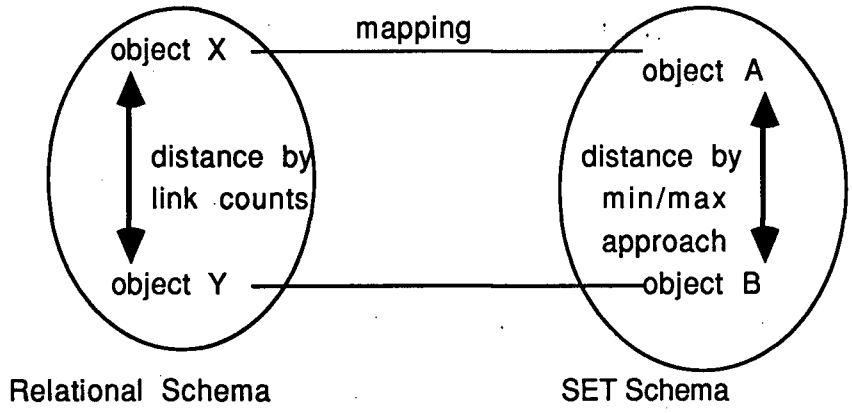


Figure 4.23: The Relationship between Distance in Relational and SET Schemas

corresponding to base relations.

Figure 4.24 illustrates some possible base relations for the university domain, a sample extension for each, and a definition for each expressed in the language DEFINE [32] which is reviewed in Appendix A. A small addition has been made to the language for the purpose of the example. The expression *colname*@x:*setname* occurring as the i^{th} entry in the **select** part of a declaration states that the name of the i^{th} column of the set is *colname*. If *colname* is omitted, then the name of the column is taken to be *setname*.

Example 4.10. In the university domain the min value of SC on *Student* is zero which states that a student may be registered in no courses. If null values are permitted in the extension of the schema, then either the schema of Part (a) or that of Part (b) of Figure 4.24 will permit the sets *Student* and SC to be represented. However, if null values are not permitted, then the schema of Part (a), but not that of Part (b), will permit those sets to be represented. Given the schema of Part (b), it would not be possible without the use of null values to record the identifiers for students who are not registered in any courses. Given the schema of Part (a), identifiers for students who are not registered in any courses can be recorded using the *Student* relation, and student identifier/course identifier pairs for only those students who are taking courses can be recorded using the *Student_Course* relation.

In general, if X is an association with parent set P, and the min value of X on P is at least one, then P can be derived as a projection of X. Therefore, the base relation that represents X will also represent P. If the min value of X on P is zero, and null values are

not permitted, then it is necessary that X and P be represented by different base relations: one from which X can be derived, and another from which P can be derived.

Example 4.11. The max value of SC on *Student* is n which states that a student may take more than one course. Regardless of which schema is used the student identifier will be duplicated in the database once for every course that the student with that identifier is taking. If the relational schema of Part (b) is used, then the name of a student will be duplicated once for every time that the student identifier is duplicated. To avoid propagation of data redundancy the schema of Part (a) is preferred.

In general, if X is an association with parent set P, and the max value of X on P is no greater than one, then no propagation of data redundancy will result if X and P are represented by the same base relation.

The Relationship between Distance in Relational Schemas and Domain Graphs

The above examples illustrate how the min/max values influence the number of links in a relational schema. A (1,1)-edge (X, P) in the DG indicates that X and P can be represented by one base relation without null values and without propagation of data redundancy. If (Y, X) is also a (1,1)-edge then the three sets Y, X, and P can be represented by the same base relation. In general, the sets whose names label the vertices of any (1,1)-connected subgraph of the DG can all be represented by the same base relation. This rule is used in [32] as the basis for a design methodology for relational schemas. The methodolgy is briefly outlined next. Some simplifications have been made for brevity.

Student

SID	*SName*
*S*1	*Jones*
*S*2	*Smith*
*S*3	*White*

Student_Course

SID	*CID*	*Grade*
*S*1	*CS*100	*A*
*S*1	*CS*200	*A*
*S*1	*CS*101	*B*

Student **def**
select $SID@x{:}Student, y{:}SName$
where $< x, y >{:}SN$

Student_Course **def**
select $SID@x{:}Student, CID@y{:}Course, z{:}Grade$
where $< x, y >{:}SC$ **and** $<< x, y >, z >{:}SCG$

(a)

Stud_Course

SID	*CID*	*SName*	*Grade*
*S*1	*CS*100	*Jones*	*A*
*S*1	*CS*200	*Jones*	*A*
*S*1	*CS*101	*Jones*	*B*

Stud_Course **def**
select $SID@x{:}Student, CID@y{:}Course, z{:}Grade, w{:}SName$
where $< x, y >{:}SC$ **and** $<< x, y >, z >{:}SCG$ **and** $< x, w >{:}SN$

(b)

Figure 4.24: Use of Min/Max Values for Designing the Database Schema

1. The methodology describes how to declare a table schema capable of recording the extensions of declared sets of a SET schema.

2. (1,1)-connected subgraphs of the domain graph are formed. We will assume that maximal subgraphs are used. A subgraph is maximal if it cannot be enlarged by adding nodes that are connected by (1,1)-edges.

3. The subgraphs of (2), if undirected-cyclic, are made into trees by removing edges. Restrictions are imposed on which edges may be removed to avoid tables with more columns than necessary.

4. To construct a table from a tree, the tree is extended to ensure that identifiers are included for all sets named in the tree. To provide an identifier for a set which needs one, a one-to-one correspondence is established between the set and a value set. The correspondence is total on the set that needs an identifier, but not in general on the value set.

5. Each tree is used to construct a table. A table is declared as a defined set the intension of which references the sets named in the tree.

Abiding by the restrictions of (3) regarding which edges may be removed to break cycles, there may still be more than one possible tree for a cyclic subgraph and the methodology does not specify which one is best. The important point here is that whichever tree is chosen, all of its edges are labeled (1,1).

A semantic relatedness measure in relational schemas is concerned with links between relations, not with links between columns of relations. The additional columns that are of

concern in (3) add nothing to the link count. For simplicity and without loss of generality, we will assume a methodology for table design in which any edges which leave the subgraph connected may be removed to break cycles.

The sets that are introduced for identification add nothing to the link count in relational schemas. Relationships of interest do not exist between value sets and, hence, there will never be a need to join two relations on that basis. Furthermore, none of the sets that are introduced for identification create a new relation which might introduce additional links. Natural language requests will never reference sets in the SET schema that are introduced solely for identification. Therefore, starting with the SET schema, it is more appropriate to consider not the augmented trees of (3), but trees "stripped" of sets that serve the sole purpose of providing identification. A maximal subgraph has weight zero. Taking away edges will not decrease its weight and certainly will not increase it.

The relationship between a given domain graph and a corresponding relational schema is illustrated pictorially in Figure 4.25. Part (a) of the diagram indicates the relational schema. A relational schema is assumed to be represented as a graph G_r whose nodes denote relations of the schema. An edge $(R1, R2)$ is an edge of G_r if the relations denoted by $R1$ and $R2$ are allowed to be joined. The edges of G_r are undirected and each has a weight of 1. A join path for set of vertices V is a subtree of G_r that contains the vertices in V and each of whose leaf nodes is in V.

Part (b) of the diagram indicates the domain graph from which the relational schema represented by G_r has been derived. Each "blob" denotes a maximal subgraph of the domain graph. The lines connecting blobs indicate paths between maximal subgraphs. A path

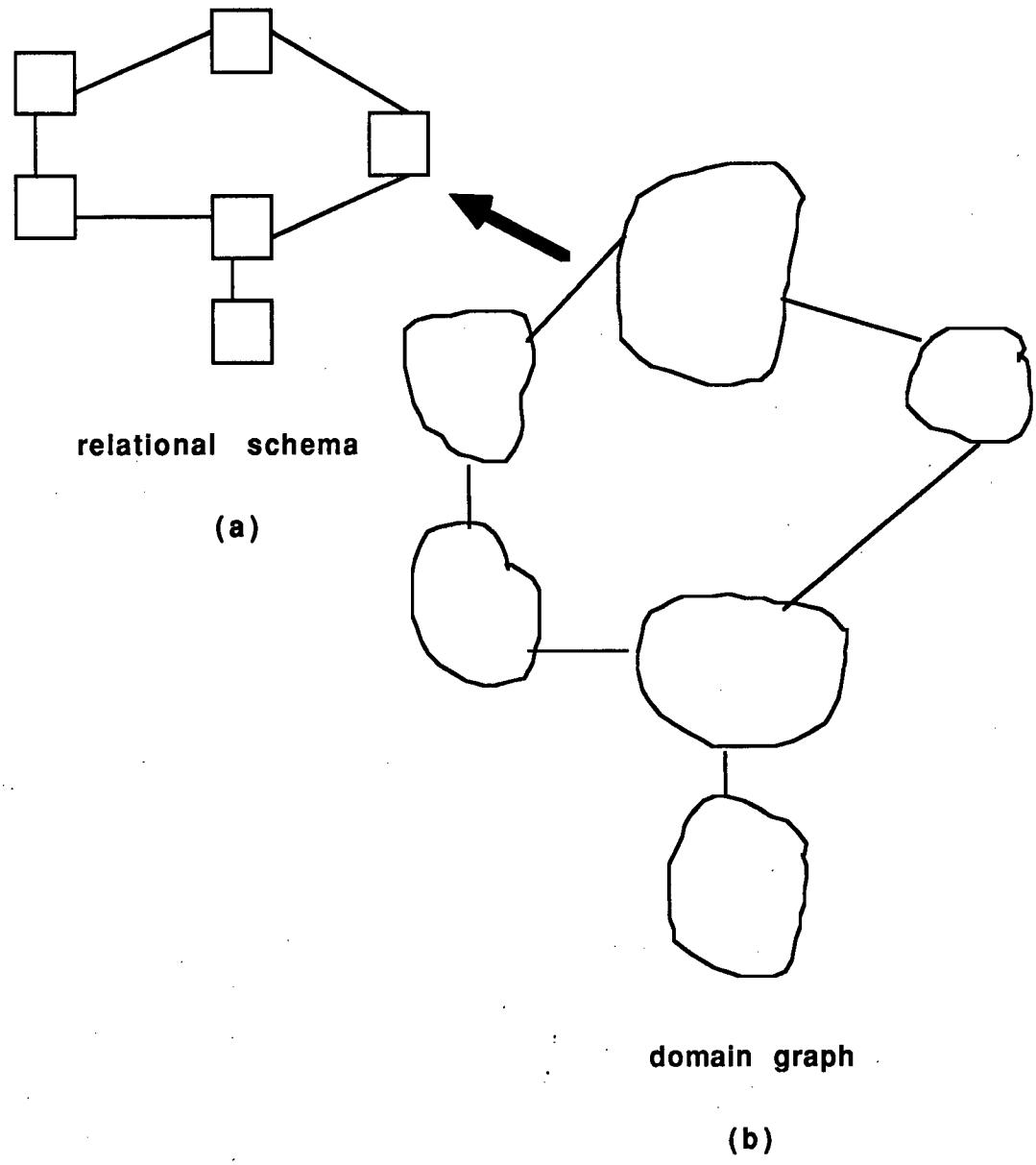


Figure 4.25: The Relationship between the Domain Graph and a Corresponding Relational Schema

between two subgraphs SG_1 and SG_2 is a sequence of edges $(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)$ such that v_1 is a vertex of SG_1 and v_2 is a vertex of SG_2. In the design methodology of [31, 32], each maximal subgraph gives rise to exactly one table in the table schema. The bold arrow in the figure indicates that the tables of the table schema are sets defined from those of the SET schema.

To compare semantic relatedness measures in relational and SET schemas, it is necessary to draw a correspondence between the objects referenced by a natural language request in the corresponding schemas. Using the domain graph method of table design, the objects of both SET and table schemas are sets. In addition, for each set S referenced by the request in one schema, there must be a set T in the corresponding schema such that S and T are intensionally equal, and the converse. Otherwise, the NL request would have a different meaning in the two different schemas. To accomplish this requirement, we adopt the following convention: The primitive constituents of a natural language request denote tables in the table schema and maximal subgraphs in the DG. A maximal subgraph can be viewed as a vertex that denotes a set the definition of which is not important for our analysis. Since there is a one-to-one correspondence between tables and maximal subgraphs, we assume that whenever an NL request references a table in the table schema, it will reference the corresponding maximal subgraph in the DG.

The weight of a (1,1)-connected subtree of the DG relative to any of its vertices is zero, and its absolute weight is also zero. Any subtree of a (1,1)-connected subtree has weight zero relative to each of its vertices and absolute weight zero. If a (1,1)-connected subgraph is cyclic, then each of the trees that can be constructed from it by breaking cycles has

relative and absolute weights equal to zero.

The maximal subgraphs partition the vertices of the domain graph. Therefore, any path between two maximal subgraphs has at least one edge.

It is assumed that all of the declared sets of the SET schema are to be represented in some table. Therefore, every path between two subgraphs must have at most one edge. Suppose that a path has two edges. Both must be non-(1,1)-edges. Otherwise, they would have been included in some maximal subgraph. We must consider the vertex v between the two edges as a maximal subgraph containing only v. Otherwise, there would be a declared set which is not represented by any table.

The above two observations state that every path between two maximal subgraphs contains exactly one edge.

The graphs of Figure 4.25 differ only in that the edges of graph (b) are directed, whereas those of graph (a) are undirected. The edges between tables in graph (a) and between maximal subgraphs in graph (b) all have weight 1.

The heuristic in SET schemas could be redefined in the following way without affecting the results: The weight of a query graph relative to vertex v a member of the target graph is the sum of the weights of backward edges relative to v. The absolute weight is the maximum of the relative weights over all vertices in the target graph.

The following discussion describes the conditions under which query graphs for a given target graph will receive the same relative ordering by our heuristic in a given SET schema as they would using a semantic relatedness measure in the corresponding table schema. Assume first that the weight of a query graph in the SET schema is computed as the

maximum of the relative weights where the relative weight of v is the sum of the weights on backward edges relative to v.

Consider two query graphs QG_1 and QG_2 for target graph TG. For any $v \in TG$, if all of the edges of QG_1 relative to v are backward edges, then the weight of QG_1 in the SET schema is the same as the weight of QG_1 in the relational schema. Furthermore, if there are about the same number of backward edges relative to each vertex in the target graph in both QG_1 and QG_2, then the weight of QG_1 is less than the weight of QG_2 in the SET schema whenever the weight of QG_1 is less than QG_2 in the corresponding relational schema, and the converse.

Now assume that the weight of a query graph in the SET schema is computed as the minimum of the relative weights where the relative weight of $v \in TG$ is the sum of the weights on forward edges relative to v. If all of the edges of QG_1 relative to $v \in TG$ are forward edges, then the weight of QG_1 in the SET schema is the same as its weight in the corresponding relational schema.

In summary, sufficient conditions, for a semantic relatedness measure in relational schemas to have the same outcome as the measure in SET schemas, are as follows: For each subgraph of the relational schema, the corresponding subgraph of the SET schema has about the same number of forward edges relative to each of its vertices, or for each subgraph one of the vertices has all of the forward edges. In short, a notion of direction is missing from the semantic relatedness measure in relational schemas. Its presence in the heuristic that has been presented in this thesis results in a richer descriptive capability for modeling semantic relatedness.

4.6 Related Work

The notion of the min value of an association on a set can be used to express the multivalued dependency constraints (MVDs) of the relational model [67, 5].[7]

Given the domain graph $A \longrightarrow X \longleftarrow B \longrightarrow Y \longleftarrow C$ the statement that both X and Y have a min value of 1 on B is equivalently stated by the pair of MVDs $B \twoheadrightarrow A \mid C$. The notion of the max value of an association on a set can be used directly to express the functional dependency constraints of the relational model.

Maier and Ullman [62] give lossless join conditions based on functional and multivalued dependencies for building *maximal objects* which are intuitively "maximal sets of attributes among which there is significant connection". A relational schema that comprises such maximal objects tends to give more correct answers to queries posed by users who view the database as a universal relation. The universal relation assumption is intended to free users from concerns about the organization of the database. The maximal object approach has immediate relevance for automatic understanding of natural language database requests, because the user who formulates such requests is usually unaware of the organization of the database. In [67] the third normal form, Boyce/Codd normal form, and projection/join normal form schemas of the normalization approach are obtained within a synthetic ap-

[7] Beeri et al [5] show that MVDs can be expressed as lossless join conditions in relational schemas: Given a relational schema R(A,B,C), the MVD $B \twoheadrightarrow A$ holds in R if and only if R is the natural join of its projections R[A,B] and R[B,C]. The MVD $B \twoheadrightarrow C$ also holds in R. Ola [67] proves that a necessary and sufficient condition for a lossless join of relational schemas R1(A,B) and R2(B,C) is that the common attribute B is mapped 'onto' in both R1 and R2.

proach. These results suggest that the lossless join conditions for building maximal objects based on functional and mulitivalued dependency constraints can be equivalently expressed as lossless join conditions based on the min/max values.

4.7 Summary

In this chapter a measure of relatedness between sentence constituents, called semantic relatedness, has been proposed as a basis for resolving ambiguities in natural language database requests. A description of the domain in terms of sets, associations, and min/max values, called the SET schema, is assumed to be available. Our semantic relatedness measure is based on a notion of *distance* between objects in the SET schema.

We have focused on three types of ambiguities in natural language requests: semantic ambiguities, word sense ambiguities, and post noun modifier attachments, in particular, prepositional phrase attachments involving the prepositions 'in' and 'with'.

For the prepositions "in" and "with", our heuristic favors the *part of* meaning which is denoted by an existence dependency association between the referent and the modifier. Similarly, for pre-noun modifiers that indicate possession (genitives) the heuristic favors the *part of* meaning denoted by an existence dependency association between the noun and the modifier.

Favoring the *part of* meaning is motivated by previous work [40, 51, 93, 55] which shows that distance measures in relational schemas are useful for resolving ambiguitiies and the work reported here which shows that there is a close correspondence between distance measured in relational schemas and distance measured in SET schemas. In particular, we

have stated conditions under which our heuristic would give the same results in a SET schema S as a heuristic in relational schemas would give in a relational schema designed from S. A desireable feature of any heuristic that operates in the SET schema is that it should be invariant with arbitrary decisions made by the SET schema designer. We have shown that our heuristic gives the same outcome in SET schemas that look different but express the same information.

We have followed the philosophy of Judea Pearl [68] who pointed out the following properties of good heuristics: 1.) They provide a simple means of discriminating among alternatives 2.) Although they are not guaranteed to identify the best alternative, they do so sufficiently often.

Chapter 5

Design Strategy

We have argued that knowledge for automatically understanding natural language database requests is available from the process used to design the DB schema. A design strategy for natural language interfaces that captures knowledge from this source results in an enhancement of domain portability. In this chapter we discuss some of the issues associated with implementing such a strategy. In addition, the focus will be on providing clean lines of communication between the NLI and the RDB to enhance database portability.

Section 5.1 describes the overall design of our proposed system. In Section 5.2 the problem of obtaining semantic knowledge from the database within the architecture of our proposed system is addressed. Issues associated with portability are discussed in Section 5.3. Finally, in Section 5.4 heuristics that are needed for solving the linguistic problems of word sense disambiguation, semantic ambiguities, and post noun modifier attachment within the proposed architecture are provided.

5.1 Overall Design

An overall design for our proposed system has been illustrated in Figure 2.13 of Chapter 2 where we see that the system, denoted NLI$^+$, is designed as a typical NLI enhanced with a module that implements a data management strategy (DMS) for portability. We will refer to the module *itself* as the DMS. The DMS captures knowledge from the SET schema which we assume to be available as a result of the DB schema design process, and presents it to the natural language interface.

5.1.1 The Natural Language Interface

A more detailed description of the NLI part of the system is illustrated in Figure 5.1. The given design is an expansion of a design proposed by Booth in [7]. The box on the left side of the figure illustrates the kinds of knowledge that are useful for NL processing. The rounded boxes indicate program modules and the square boxes data. The arrows indicate data movement. The middle column illustrates the process of translating an NL request to a DB query. The right most column illustrates the process of formulating an appropriate response to an NL request. All of the NL processing problems that are the focus of the thesis occur in the process of translating an NL request to a DB query (the middle column in the figure), with the exception of one problem in the area of answer presentation which is discussed in Subsection 5.2.3.

The *linguistic core*, which contains the domain independent components of the system, has been proposed by Rosenberg [72] to permit the NLI to be easily adapted to a new domain. The technique of separating the linguistic components from the rest of the system

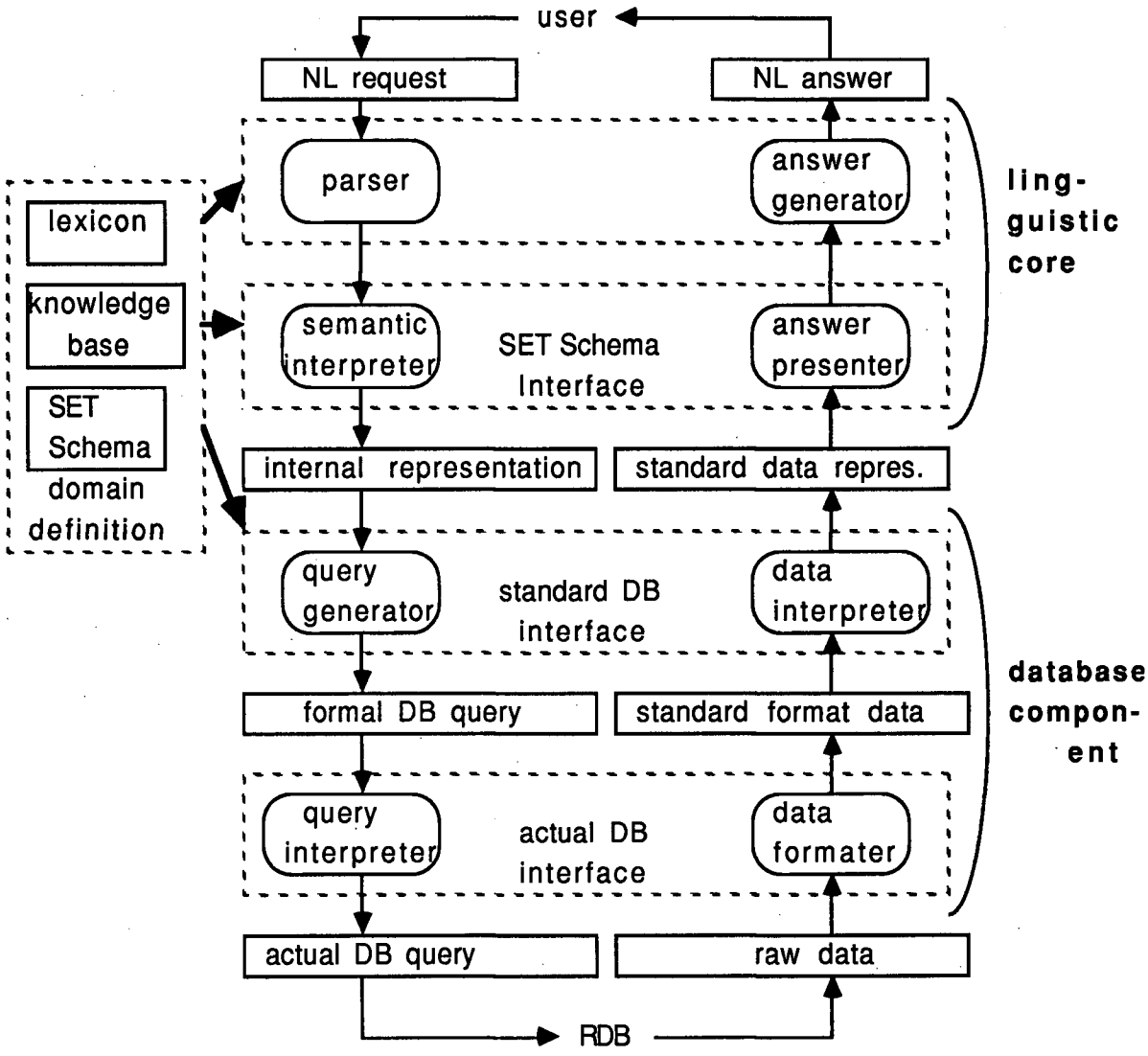


Figure 5.1: Proposed Design for an NL Interface

is important for domain portability because it permits alteration of the domain without requiring significant modifications of the entire natural language interface.

We assume a knowledge base approach in which the components in the underlying structure of the sentence are mapped to simple and composite objects in the knowledge base, and further that the semantic interpretation of a sentence constituent is built from the semantic interpretations of its constituents. The interpretation of a primitive constituent derives independently from the interpretations of other constituents.

The internal representation for an NL request is expressed in the context of a SET schema. The language DEFINE is used as the target language of a set of semantic rules which specify interpretations for fragments of a parse tree in terms of interpretations for the constituents of the fragment.(See Appendix B for the semantic rules for part of the University domain).

5.1.2 Data Management Strategy

A layering of schemas each one based on a different data model facilitates the provision of knowledge to the NLI for resolving ambiguities (as discussed in Chapter 4), for formulating the internal representation of a request, and for translating the internal representation to a formal database query. The top level schema (the SET schema) provides knowledge for formulating the internal representation (IR) and for resolving ambiguities. Knowledge about the *correspondence* between objects in different schemas assists the process of automatically translating the IR of a request to a formal DB query.

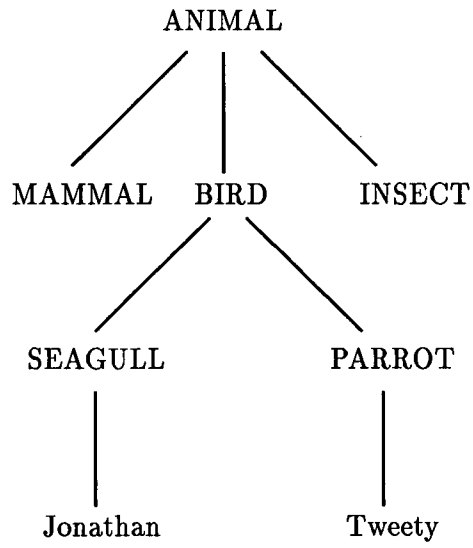
The use of several schemas each one based on a different data model does not imply

that the database system must support a query language for each different data model. Intermediate forms of natural language requests (internal representation, formal DB query) are not actually executed during processing of a request. Hence, the implementation of the data management strategy is a much simpler problem than that of building a database system that supports each of the different data models.

Part of the process of translating a particular SET schema to a corresponding relational schema can be automated. (This subject has been researched by Gilmore [31, 32] and Storey and Goldstein [80].) Knowledge about how the RDB schema has been derived from the SET schema would be useful for translating the internal representation of a request to a formal DB query. It is conjectured that the required knowledge can be captured from the process used to design the RDB schema. Therefore, a new feature for database systems that would be useful for portable natural language interfaces, is a component that automatically designs the RDB schema and records knowledge about its derivation from the SET schema.

One of the schemas in the layering is based on a model, the NF^2 object model, which we have defined specifically as an intermediate model between the SET model and the relational model. The NF^2 object model is introduced in Chapter 6. A useful feature of the model for portable natural language interfaces is that it supports multiple representations of objects. This feature facilitates the logical structuring of the DMS as a layering of schemas in which objects in one schema are represented redundantly in another. Furthermore, the NF^2 object model supports the capability of recognizing different representations of the same object, a facility that is useful for representing mappings between different schemas.

Each schema is assumed to be represented as a collection of first normal form relations



SUB	SUP
Jonathan	SEAGULL
Tweety	PARROT
SEAGULL	BIRD
PARROT	BIRD
BIRD	ANIMAL
MAMMAL	ANIMAL
INSECT	ANIMAL

Figure 5.2: Relational Representations for an Animal Taxonomy

using Codd's approach [16] for describing knowledge representation strategies, other than the relational one, using relations. The technique is illustrated in Figure 5.2 where a generalization hierachy is represented using one relation. The arcs denote two different kinds of relationships. The arcs between nonterminal nodes denote set inclusion (Seagulls are Birds) and arcs between a nonterminal and a terminal node set membership (Jonathan is a Seagull).

It is desirable to permit integrity checking on objects represented in this way. However, a standard RDB system [63] automatically enforces constraints only on objects of the relational model. Integrity checks on more general objects can be automatically enforced by extending the operations that are provided by the RDB system for updating the metadatabase as illustrated in Chapter 6.

The data management strategy has two parts, a *data* part which comprises a collection

of relations for representing NF^2 data, NF^2 object, and SET schemas, and an operation part which comprises expansions of existing operations in the metaschema to enforce integrity checks on objects (e.g., non-first normal form relation schemes, sets, associations).[1] Logically, the DMS can be separated into those expansions needed to describe NF^2 data relations, further expansions to describe NF^2 object schemas, and still further expansions to describe SET schemas.

In Chapter 6 the metaschema of a standard RDB system is extended to describe NF^2 object schemas and enforce constraints on objects (e.g., non-first normal form entity-relation schemes) of NF^2 object schemas. Additional expansions would be needed to permit *SET* schemas to be described, and integrity contraints to be enforced on objects of SET schemas. The extended metaschema has been designed to express knowledge about the mapping between a particular NF^2 object schema and the RDB schema designed from it. We conjecture that general algorithms can be written for using the mapping information to assist the process of translating the internal representation to a formal RDB query.

5.1.3 Interfaces between the NLI and the DMS

To keep the interface between the NLI and the RDB clean and simple, all access to the RDB must occur through the DMS. Figure 5.3 illustrates the lines of communication between the NLI and the DMS. On the left side of the figure appears a more abstract view of the NLI that has previously been depicted in Figure 5.1. The DMS is depicted on the right side of

[1] Here, we are referring to objects in the schema rather than those in the raw data interpreted by the schema.

the figure where its logical structure as a layering of schemas is highlighted. The double headed short verticle arrows (the diamonds) indicate information about mappings between different schemas. The thick arrows indicate flow of knowledge from the data management strategy to the component of the NLI that uses it. The SET schema provides knowledge to the linguistic core for constructing a unique internal representation of the request. The DMS provides mapping information to the database component for translating the internal representation to a formal DB query.

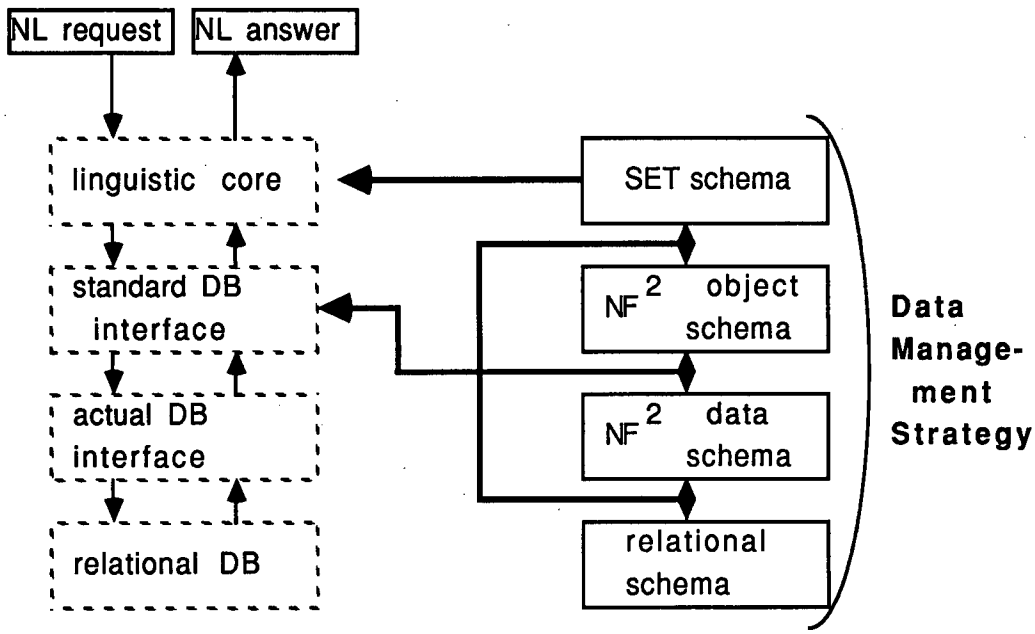


Figure 5.3: Interfaces between the NLI and DMS

5.1.4 Formulating the Internal Representation

Knowledge about which objects in the domain are related is needed for automatically formulating the internal representation of a simple request (See definition Subsection 4.3.2), however, relationships between objects are not explicitly stated in relational schemas. The RDB system automatically enforces the requirement that only fields defined on the same value set may be used as the basis for a join but this technique is not, in general, sufficient to represent relationships between objects. This problem is known as referential ambiguity. If it is not known which objects in the domain are related, it is not possible to automatically formulate a precise statement of the meaning of a natural language request.

To illustrate the problem of referential ambiguities in relational schemas, consider the following database: We wish to record the relationship between managers and their employees. This situation is typically represented within the relational model by a relation, say $MANAGER$, with columns MGR and EMP. An entry in the MGR column denotes a manager and the corresponding entry in the EMP column, an employee of that manager. Even though the values for both columns may be drawn from the same value set, say 'strings of characters length 8', we cannot conclude that the entries in the two columns denote semantically related objects (e.g., employees). To do so, we would also have to conclude that the values of all other columns defined on 'strings of characters length 8' denote employees. The latter conclusion is erroneous because, for example, column $CITY$ might be defined on 'strings of characters length 8', but the values in that column denote cities, not employees.

The SET schema is free of referential ambiguities. Graphically the relationship between

managers and employees is described by two vertices *Manager* and *Employee* and two edges each directed from *Employee* to *Manager*. This creates an undirected cycle but not a directed one. There can be no ambiguity about the parent sets of the *Manager* association. The left and right parent sets must be the same set because they are denoted by the same vertex.

If the internal representation is expressed within the context of a relational schema, then the NLI must be provided with knowledge about which relations may be joined and about which columns may be used as the basis for the join. This knowledge is usually referred to in the natural language literature as join path information. A conclusion resulting from normalization theory [22, 25, 24] as well as the synthetic methods of relational schema design [32, 57, 67] is that a relation describes an entity set. This is to say that in a domain of entities, viewed as a collection of disjoint entity sets, a relation describes entities from at most one of them. From this result, we can conclude that join path information gives knowledge about which sets in the domain are related.

If the internal representation is expressed in the context of a SET schema, then join path information can be captured from the SET schema. An example of a strategy for providing knowledge about legitimate join paths is illustrated in Appendix B. Semantic rules specify how the internal representation is built from the parse tree for a request. The right hand side of a rule gives a partial query expressed in the context of a particular SET schema. The semantic rules express the following types of information: 1.) join paths, which for the given rules are very short, (In general, they would consist of a sequence of join conditions.)[2],

[2]If X and Y are associations, X with parent sets A and B, and Y with parent sets B and C, then a join

2.) associations between words, in particular nouns and verbs, in the request and objects (sets and associations) in the SET schema, 3.) associations between determiners (some, a, any, all, every) in the request and quantifiers (**For some, For all**). The rules would have to be provided to the NLI as part of the customization process, however, if the internal representation is expressed in the context of a particular SET schema, then knowledge of type (1) is automatically provided.

5.2 Using the Database as Part of the Lexicon

Harris proposed the idea of obtaining, from the relational database, knowledge about membership in columns which is useful for automatically translating the internal representation of a request to a formal DB query.

To translate the request 'Tell me about green ford cars' to a formal DB query we need to know which database objects the primitive constituents 'green', 'ford', and 'car' denote. In a relational database 'car' might denote the CAR relation, and 'green' and 'ford' the COLOR and MANUFACTURER columns, respectively.[3] Harris' algorithm for determining column information is as follows: To determine if a primitive constituent is a member of a column, search the current extension of the column. A gain in efficiency is realized by searching the database indices on columns, rather than the columns themselves.

---

condition is a statement [**For some** b:B] ($< a, b >$:X **and** $< b, c >$:Y))

[3]In a more practical situation, 'green' may refer to either the color or the owner of a car and 'ford' to either the owner or manufacturer. Before addressing the problem of how to formulate queries when a primitive constituent denotes more than one DB object, let us restrict our attention to the case where it denotes at most one.

In the database system to which Harris' INTELLECT system is attached (ADABASE), the indices are part of the user's conceptual view of the database. In more current database systems the indices are hidden from the user and automatically applied as needed for efficient retrieval. This new feature of database systems implies that Harris' solution for obtaining column information is not *directly* applicable to current database systems. In this section, we show how column information can be obtained from a standard current RDB system, thus providing an update of Harris' approach to apply to current technology, and filling in many of the details that are missing in the literature.

A related problem which has not been resolved by Harris' results is the one of providing an appropriate answer to a request when primitive constituents referenced in the request are neither in the DB nor the lexicon. We propose new features for DB systems that could be used to solve it. The solution proposed is with a view to the future because the current technology does not yet allow an efficient implementation.

5.2.1 The Requirements of an NLI for Membership Information

For primitive constituent t, let A and B be lists of names of columns with the following properties: The A-list identifies columns to which t belongs, the B-list columns to which t could possibly belong. Primitive constituent t belongs to column C if t is a member of C in the current extension of the database. Primitive constituent t possibly belongs to column C if t is a member of the value set of C but not a member of the current extension of C.

If $|A\text{-list} \cup B\text{-list}| > 1$ then there is more than one interpretation for the request.

If the A-list is empty and the B-list is non-empty then the answer to the question is

'none'. For example, given the request 'Tell me about green cars', if 'green' is a member of the COLOR value set but there are no green cars described in the database, then the answer is 'There are no green cars'.

If both the A-list and the B-list are empty then the natural language system must report on the presuppositions of the request. Given the above request if there is no value set to which the primitive constituent 'green' belongs then the answer is 'I don't know what green means'.

In the INTELLECT system, the answer 'none' is given for requests for which column information cannot be obtained from either the lexicon or the database. Such a solution does not permit the distinction between the answer 'No' and an answer which reports on the system's lack of knowledge. The B-list has been introduced to solve this problem.

5.2.2 A Construction for Providing Membership Information

In this section and the next, we show how the inverted indices and the metadatabase can be used to construct the A-list and B-list defined in Subsection 5.2.1. For illustration, we will assume that the particular RDB system is ORACLE which supports the SQL query language.

An inverted index on a database column C is a function $f_c : V \rightarrow \mathcal{P}(addresses)$ with domain[4] V, the set of all database values, and range the power set of database addresses. $f_c(v)$ is the set of addresses of tuples with value v in column C. If there are no tuples with

[4]Here the term *domain* is being used as in mathematics, as opposed to referring to the domain of discourse for a natural language interface.

value v in column C, then $f_c(v) = \{\}$ where $\{\}$ is the empty set.

The following rule is used to obtain column information for primitive constituents in natural language requests. If v is a primitive constituent and $f_c(v)$ returns a non-empty set, then v is a member of column C.

An inverted index is created in ORACLE using the INDEX command. The following command creates an index on the ADDRESS field of the EMPLOYEE relation:

index I-ADDRESS on EMPLOYEE(ADDRESS)

Once a column of a relation has been indexed ORACLE will use the index to locate tuples that satisfy a search condition on that column. Consider the following query:

```
select   *
from     EMPLOYEE
where    ADDRESS = 'Vancouver'
```

The index on the ADDRESS column is I-ADDRESS. The database system will use the index to locate the tuples from the EMPLOYEE relation satisfying the search condition ADDRESS = 'Vancouver'.

A table, COL, of the metadatabase describes database columns. The following query selects a subset of the fields of COL: The result of the query is given in Table 5.1

```
select   TNAME, CNAME, COLTYPE, WIDTH, NULLS
from     COL
```

Suppose that t is a primitive constituent in the input request, and we wish to determine which columns have t as a member.

TNAME	CNAME	COLTYPE	WIDTH	NULLS
EMP	EMP#	CHAR	6	not null
EMP	NAME	CHAR	20	null
EMP	DEPT	CHAR	10	null
EMP	SALARY	NUMBER	10	null
EMP	ADDRESS	CHAR	20	null

Table 5.1: A Fragment of the ORACLE Metadatabase

For each table-(character)column pair (*table*, *column*) returned by the previous query, the following query is executed:

select COUNT(*)
from *table*
where *column* = 't'

The COUNT is used as a convenience so that a numerical result will be returned. If the query returns a non-zero result, then $t \in column$.

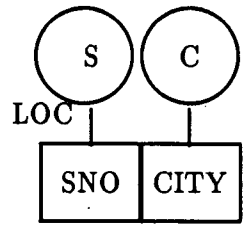
5.2.3 Distinguishing between 'No' and 'Null' Answers

A useful facility for providing cooperative answers to NL requests would be the ability to find the *complement* of a request. Consider the database schema of Figure 5.4 and the request "Print supplier numbers of suppliers in Paris". The SQL query in the figure results in a possibly empty set of such numbers, say PS.

If PS is empty then the system can answer either "There are no suppliers in Paris" or "There are no suppliers in Paris and there never will be (short of restructuring the database). The database has no knowledge of Paris suppliers."

The complement of PS is

$$\{s : s \in S \;\&\; (\exists c)(c \in C \;\&\; c = Paris \;\&\; (s,c) \notin LOC)\}$$




```
select  LOC.SNO
from    LOC
where (LOC.CITY = 'Paris')
```

Figure 5.4: Schema for a Suppliers Database

which is the set of all supplier-Paris pairs that are members of $(S \times C)$ and not members of LOC.

Assuming that value sets S and C are represented by relations the following SQL query computes the complement of PS.

```
select  S.SNO
from    S, C
where C.CITY = 'Paris'
and     S.SNO
not in select  LOC.SNO
       from    LOC
       where LOC.CITY = 'Paris'
```

If the original request results in an empty set and the complement is non-empty then the answer to the request is 'No'. Previous systems give the answer 'No' even when the complement is empty (INTELLECT [42]).

An algorithm for transforming a query into another query that computes the complement of the original query is given in [31]. The source and target queries are both written in the language DEFINE in which a complementation operator is available. A DB system that supports such a language would be useful for a new generation of natural language interfaces that provide more cooperative answers.

The architecture of the natural language system given in Figure 5.1 requires the following modifications to take advantage of the facility of finding the complement of a request: The semantic interpreter finds the complement of the natural language request expressed in its internal representation (IR). Both the IR and its complement are processed by the query generator to produce formal database queries. The query interpreter receives as input both the formal query and its complement and produces a cooperative response.

5.3 Portability Issues

Portability is measured by the extent to which the NLI can be adapted to a new domain and database, by the DBA, as opposed to a linguistic expert. In this section, we examine how our design strategy enhances domain and database portability.

5.3.1 Domain Portability

If any of the semantic knowledge needed by the NLI for understanding natural language requests can be easily obtained from any source, then domain portability will be improved. Our design strategy is based on the key concept that semantic knowledge can be obtained from the process used to design the DB schema. The DBA has already undertaken the work of gathering semantic knowledge that is useful to the NLI when he or she designed the DB schema. Our design strategy enhances portability of the resulting NLI because the DBA, not a linguistic expert is called upon to gather the knowledge and, furthermore, the DBA has already gathered the knowledge for designing the DB schema.

5.3.2 Database Portability

This section discusses the way in which our design strategy provides each of the three different types of database portabilities that we introduced in Chapter 3.

Data Model Portability

Data model portability is the ease with which an NLI can be adapted to a DB system that supports a new data model. The standard solution to obtaining data model portability (i.e., separating the linguistic and database components) entails modifications of the programs that implement the query generator and query interpreter when the data model is changed.

In an NLI$^+$ the DB system itself is used to enhance data model portability. The DMS provides an expansion of the available DB capabilities to support at least some of the capabilities of the data model expected by the query generator and query interpreter. In this way these components of the system will require fewer modifications when the NLI is to be attached to a DB system that supports a different data model.

The DMS must be modified each time that the NLI is adapted to a different data model. However, if the database system is extensible, then modifiying the DMS will be easier than modifying the query generator and query interpreter.

Within our design strategy the enhancement in data model portability is dependent on the extensibility of the particular database system. Using the standard relational architecture of Mark and Roussopoulos [63], the DMS is easily modified due to the fact that the database system has been designed to be easily extended. However, the standard relational architecture does not permit the relational query language to be extended, which limits the

ease with which an NLI can be attached to it.

A current hot topic in database research is the design of easily extensible database system architectures [74, 59]. The design of our system takes full advantage of the extensibility of the database system to enhance data model portability, and has been motivated with a view to future architectures for DB systems.

Schema Portability

Schema portability of a natural language interface is the ease with which the interface can be adapted to a new *relational* database schema (i.e., the schema in which the *formal* database query is formulated). Schema portability is concerned with independence of the NLI from changes in the structures used for representing the knowledge in the domain. The term "logical data independence" refers to a database system capability which permits the database schema to be changed without requiring modification of existing application programs. This capability which is widely provided by current DB systems is useful for enhancing the schema portability of natural language interfaces.

The programmer of a database application has knowledge of the database schema that he or she obtains by studying the metadatabase. The knowledge obtained in this way is used to formulate database queries in the context of the given DB schema.

In a natural language interface it is not known in advance of writing the programs that implement the interface, which particular database requests will be presented to the system. Therefore, the NLI must itself obtain (from somewhere) sufficient information about the given schema to formulate database queries.

If knowledge of a particular schema is built into the NL interface, then program modifications will be required to adapt it to a new database schema which makes the system not very portable. Program modifications are avoided, and schema portability improved, by providing a mapping between objects that are referenced in the internal representations of requests and objects in the relational DB schema. This approach is not new. It is used in the TEAM system [36] and elsewhere. The mapping must be provided whenever the natural language interface is attached to a new database, but it is easier to provide the mapping which is specified in the form of data than to modify the NLI to adapt it to a new DB schema.

Our proposed system enjoys an enhancement in schema portability in addition to that provided by other systems by taking advantage of the results of a large body of research into the problem of automatically designing relational DB schemas. The internal representation (IR) for a request is formulated in the context of a SET schema. The process of translating a SET schema to a relational schema is not entirely automatic; however, a specification of the translation (for those parts that can be specified) provides the mapping between objects that are referenced by the (IR) and the those that are referenced by the formal DB query. This mapping would otherwise have to be provided as part of the customization process.

Database System Portability

The part of the customization programs of ASK and LDC-1, for example, that assist in acquiring knowledge about the physical files have a direct counterpart in database systems. The customization program provides a language for describing the correspondence between

conceptual objects of the domain and fields of the physical files. The counterpart in database systems is the language for mapping between the conceptual schema and the internal schema as proposed by the ANSI/X3/SPARC Study Group [83].

The separation between conceptual and internal schemas is made in the ANSI/X3/SPARC framework to afford application programs protection from changes in their physical files. When this level of protection is provided the application programs are said to be physically independent from their data.

A natural language interface to a database system is an application supported by the database system. Physical data independence contributes to the database portability of the natural language interface because it reduces the need to modify the interface to adapt it to the physical files of a new domain. Since the database system provides data independence, there is no need for the natural language interface to provide it.

5.4 Heuristics for Linguistic Applications

Figures 5.5 illustrates the relationships between ambiguities that may arise in transforming an NL request to an internal representation (IR) and the IR to a DB query. The root of the tree denotes the NL request. Each level in the tree identifies a particular intermediate form of the request, with the leaf level vertices denoting alternate possible DB requests for the given NL request. The branching illustrates ambiguities that may occur at each stage. The sources of ambiguity for each stage are identified along the right margin of the figure. For example, the NL request (assumed to be an ellipsis) has three immediate children in the tree each of which denotes a possible completion of the ellipsis. For each

of the resulting non-elliptical requests there may be a number of parse trees resulting from different attachments of the modifiers and conjunction scopings. The semantic relatedness

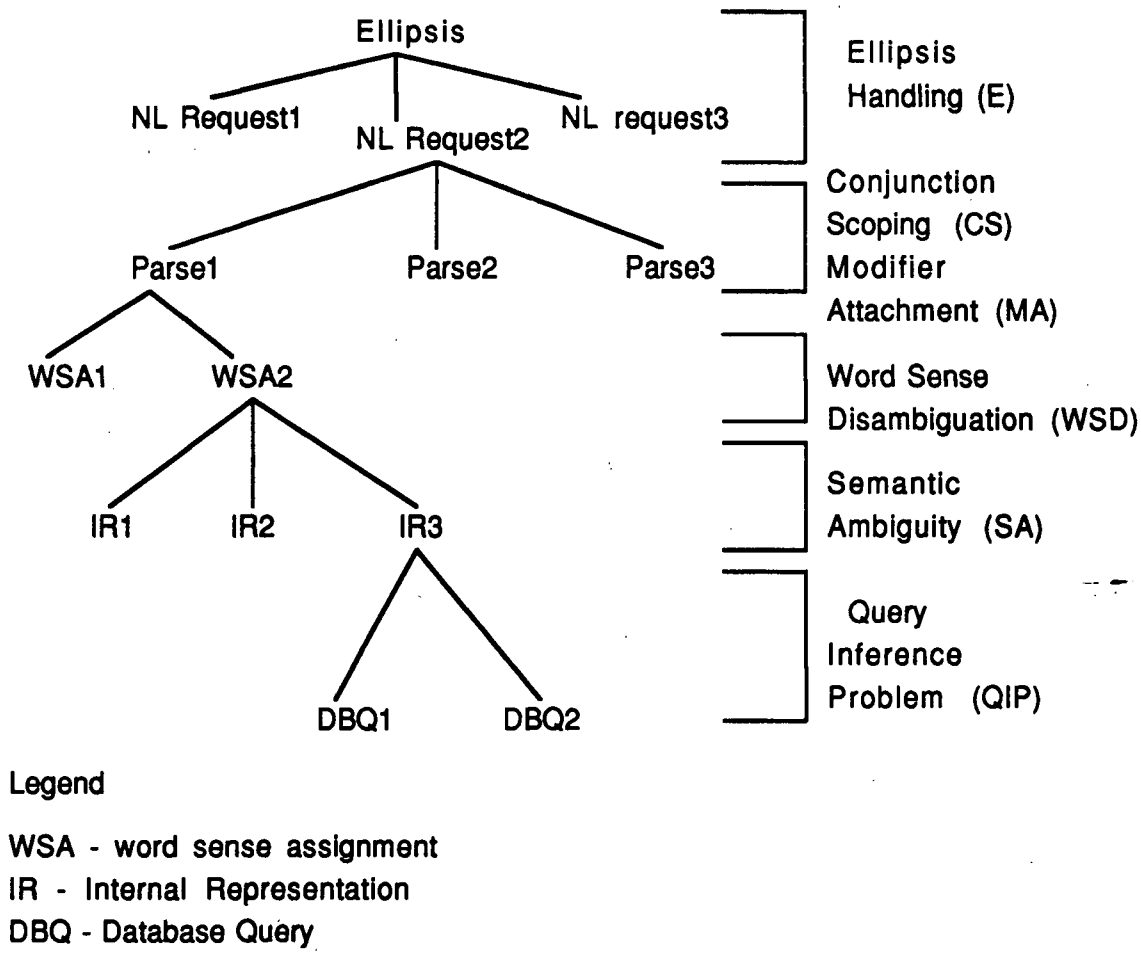


Figure 5.5: Sources of Ambiguities in Translating an NL Request to a DB query

measure in SET schemas is applied for the following purposes:

1. For determining the type of ellipsis that has occurred it is used to compare IRs corresponding to different completions of the elliptical fragment.

2. For resolving MA and CS it is used to compare IRs corresponding to different parse trees.

3. For WSD it is used to compare IRs corresponding to the same parse tree, but with different assignments of meanings to the words.

4. For resolving SA it is used to compare different IRs corresponding to the same parse tree and the same assignment of meanings to the words.

In the following subsections, heuristics are given for making use of the knowledge captured from the SET schema for solving the problems of WSD, SA, and MA.

5.4.1 Preliminaries

The union $G_1 \bigcup G_2$ of two graphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ where V_1 and V_2 are the sets of vertices for G_1 and G_2 and E_1 and E_2 the sets of edges is defined as follows:

$$G_1 \bigcup G_2 = (V_1 \bigcup V_2, E_1 \bigcup E_2)$$

Let $forward\_edges(G, v)$ denote the forward edges of graph $G = (V, E)$ relative to vertice $v \in V$. Let $edge - weight(e)$ where e is an edge denote the weight of e as defined in Subsection 4.3.2. The weight of G relative to v denoted $r - weight(G, v)$ is defined as follows:

$$r - weight(G, v) = \sum_{e \in forward\_edges(G,v)} edge - weight(e)$$

If $V = E = \{\}$ (the empty set) then $r - weight(G, v) = 0$. If $T \subseteq V$ and $T = \{v_1, \ldots, v_r\}$,

then the weight of G relative to T is

$$weight(G,T) = MIN(r - weight(G,v_1),\ldots,r - weight(G,v_r)).$$

$MIN(a_1,\ldots,a_p)$ where $a_1,\ldots,a_p$ are integers evaluates to the minimum of $a_1,\ldots,a_p$. It should not be confused with the min value of an association.

5.4.2 Word Sense Disambiguation

For each primitive constituent of a sentence there is a set of possible vertices that the constituent could denote. If the set is a singleton set, then the constituent is unambiguous. If the set is nonempty, and it is not a singleton set, then the constituent is ambiguous. This subsection describes a heuristic that uses knowledge expressed by the min/max values to resolve word sense ambiguities in natural language sentences.

Primitive constituents of a sentence may denote either vertices or edges in the DG. Figure 5.6 illustrates two different ways of describing the marriage relationship. Primitive constituents 'husband' and 'wife' would denote the edges *husband* and $wife$ in the domain graph of part (a), and the vertices *husband* and $wife$ in the DG of part (b). It is convenient to restrict primitive constituents to denote only vertices. Let S be an n-ary association with parent set S_i. The edge $(S_i \longrightarrow S)$, is defined as a vertex $P^S_{S_i}$ in the following way. Let $\pi^n_i[t]$ denote the projection of n-ary association entity t on the i^{th} component. For example, $\pi^3_2[(a,b,c)] = b$. $P^S_{S_i}$ is defined as follows:

$$P^S_{S_i} = \{(x_i,t) : t \in S \;\&\; \pi_i[t] = x_i\}.$$

$P^S_{S_i}$ is a binary association with parent sets S_i and S. Such a set is called a P-set. An

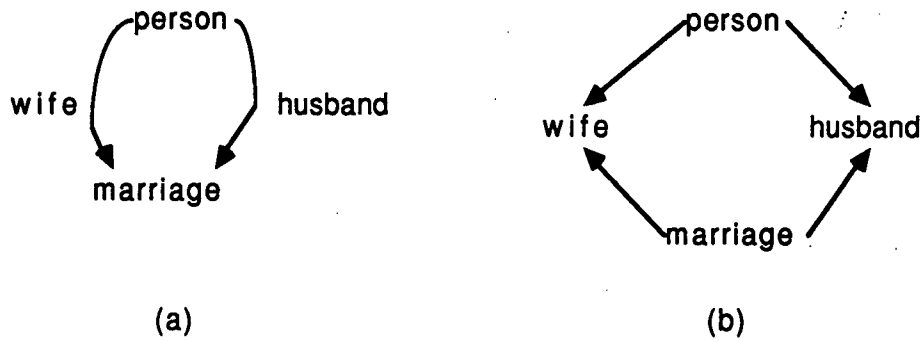
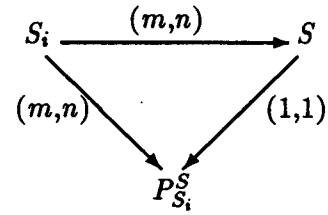


Figure 5.6: Domain Graph Descriptions of the Marriage Relationship

edge $S_i \longrightarrow S$ can be represented by the P-set $P^S_{S_i}$ and a pair of edges $S \longrightarrow P^S_{S_i}$ and $S_i \longrightarrow P^S_{S_i}$ as illustrated in the following figure:



If the min/max value of S on S_i is (m, n) then the min/max value of $P^S_{S_i}$ on S_i will also be (m, n). The min/max value of $P^S_{S_i}$ on S is (1,1) independent of the min/max values of S. These two results are proved next.

No particular ordering of the parent sets of a P-set is intended. In order to state the definition S_i was designated as the left parent and S as the right parent. The definition could equally well have been stated by making the opposite designation.

Observation 1. If the min/max value of S on S_i is (m, n) then the min/max value of $P^S_{S_i}$ on S_i is also (m, n), and the converse.

Proof. For all $x_i \in S_i$ and all $t \in S, x_i$ participates in t if and only if $(x_i, t) \in P^S_{S_i}$, by definition of $P^S_{S_i}$. By definition of min and max values it follows that the min and max values of S on S_i and $P^S_{S_i}$ on S_i are the same. □

Observation 2. The min/max value of $P^S_{S_i}$ on S is (1,1) for all i.

Proof. Since π_i is a function with domain S and range S_i, the maximum number of pairs $(x_i, t) \in P^S_{S_i}$ in which any given $t \in S$ occurs is 1, and since π_i is total on its domain, the minimum number of pairs $(x_i, t) \in P^S_{S_i}$ in which any given $t \in S$ participates is 1. □

Denotations for primitive constituents are specified as vertices in the domain graph (as opposed to edges and vertices). To accomplish this goal, new vertices are introduced to the DG to denote P-sets. Since min/max values are easily determined for P-sets, the requirement that min/max values are available for labeling all edges of the DG is met.

The name of the heuristic for resolving WSD is *BTG* for *best target graph*. Input is a sentence constituent represented as a parse tree. Output is a set of vertices called the *target graph* which is obtained by selecting exactly one vertex for each primitive constituent that denotes a nonempty set of vertices. Furthermore, BTG outputs the "best" target graph which is the target graph among the possible ones that determines a query tree of smallest weight. Input and output of BTG is illustrated by the following pictorial:

$$\text{sentence constituent} \Longrightarrow BTG \Longrightarrow \text{the best target graph}$$

Let $PRIM$ denote the set of primitive constituents in the domain, and V the set of vertices in the domain graph. Let $DENOTE \subseteq (PRIM \times V)$ denote a set of pairs of the form (p, u) where primitive constituent p denotes vertex v. $DENOTE$ is usually recorded in

the lexicon. The min/max value of $DENOTE$ on V is $(0, n)$ (different primitive constituents may have the same meaning (synonyms), and not every vertex need serve as the meaning for some primitive constituent). The min/max value of $DENOTE$ on $PRIM$ is also $(0, n)$ (a primitive constituent may not have a meaning such as noise words "please" and "quickly", or may have more than one such as "orange" which may be either a fruit or a color).

Let $p_1, \ldots, p_l$ denote the primitive constituents of a sentence constituent SC that are each associated in $DENOTE$ with one or more vertices. For $i = 1, \ldots, l$ let $DENOTE(P_i)$ be the set of vertices denoted by p_i. A target graph $TG(SC)$ for SC is a collection of vertices $v_1, \ldots, v_n$ where for $i = 1, \ldots, l, v_i \in DENOTE(p_i)$.

Let $TG_1(SC), \ldots, TG_s(SC)$ denote the target graphs for sentence constituent SC. The word sense disambiguation (WSD) problem is to choose the best target graph $BTG(SC)$ for SC. The heuristic for computing $BTG(SC)$ involves generating one or more query graphs for each possible target graph and choosing the target graph that is associated with the query graph of smallest weight.

Let $TG_j(X)$ where X is a sentence constituent denote a target graph of X. Let $BQG(Y)$ where Y is a target graph denote the query graph of smallest weight among the possible ones for Y. The best target graph $BTG(SC)$ is $TG_i(SC)$ such that

$$weight(BQG(TG_i(SC))) \leq weight(BQG(TG_j(SC))) \text{ for } j = 1, \ldots, s.$$

The function *weight* has been defined in the preliminaries, and BQG is defined in the next section.

5.4.3 Semantic Ambiguity

The semantic ambiguity (SA) problem is to find the "best" query graph among the possible ones for a given target graph. In this section we will give a definition for "best"query graph. The heuristic for this problem is named BQG for *best query graph.* Input is a target graph, and output is the best query graph as illustrated by the following pictorial:

$$\text{target graph} \Longrightarrow BQG \Longrightarrow \text{the best query graph}$$

A query graph for a target graph $TG, QG(TG)$, is any minimal connected undirected-acyclic subgraph of the domain graph that contains the vertices in TG. If a query graph does not exist for TG, then $QG(TG) = (V, E)$ where $V = \{\}$ and $E = \{\}$. The best query graph for TG is the one among the possible ones of smallest weight with respect to TG. If $QG_1(TG), \ldots, QG_g(TG)$ are query graphs for TG, then the best query graph $BQG(TG)$ is defined as follows: It is the one $QG_i(TG)$, $1 \leq i \leq g$, for which

$$weight(QG_i(TG)) \leq weight(QG_j(TG)) \text{ for } j = 1, \ldots, g.$$

5.4.4 Modifier Attachment

For the modifier attachment (MA) problem, we restrict the modifiers to adjectives and prepositional phrases, and the referents to nouns. The sentence constituents for which the MA problem is addressed are restricted to noun phrases. A grammar for the NPs under consideration follows:

$$NP \Longrightarrow \{DET\}\{ADJ\}^*N\{PP\}^*$$

$$PP \Longrightarrow PREP \ \ NP$$

The '*' is the Kleene closure operator meaning "any number of". The brackets { and } enclose optional items. Thus an NP is an optional determiner followed by any number (including zero) of adjectives followed by a noun followed by any number (including zero) of prepositional phrases.

Nouns and adjectives are assumed to denote vertices in the DG. A prepositional phrase (PP) denotes the vertex denoted by the head noun of its noun phrase. For prepositional phrases the meaning is *implicit* in the sense that it is derived from the meanings of constituents of the PP.

Ambiguous Parse Trees

An ambiguous parse tree is a parse tree in which modifiers, that have more than one possible referent, are left unattached. However, the possible referents for a modifier are indicated by marking the modifier and each of the possible referents with a common symbol. For example, for the parse tree of Figure 5.7, the PP "with a red cover" is unattached and marked with a '*'. Possible referents for the PP are also marked with a '*'. The unambiguous parse trees in which the modifier has been attached are illustrated in Figure 2.1.

Unambiguous Parse Trees

Let $m_1, \ldots, m_l$ be the modifiers of a parse tree PT. For $i = 1, \ldots, l$ let $\{m_i : MODIFY :\}$ denote the set of possible referents for m_i. An unambiguous parse tree $U(PT)$ is PT together with a set of markers $r_1, \ldots, r_q$ such that for every m_i there exists a unique r_θ such that $r_\theta \in \{m_i : MODIFY :\}$. That is, exactly one referent has been selected for each modifier.

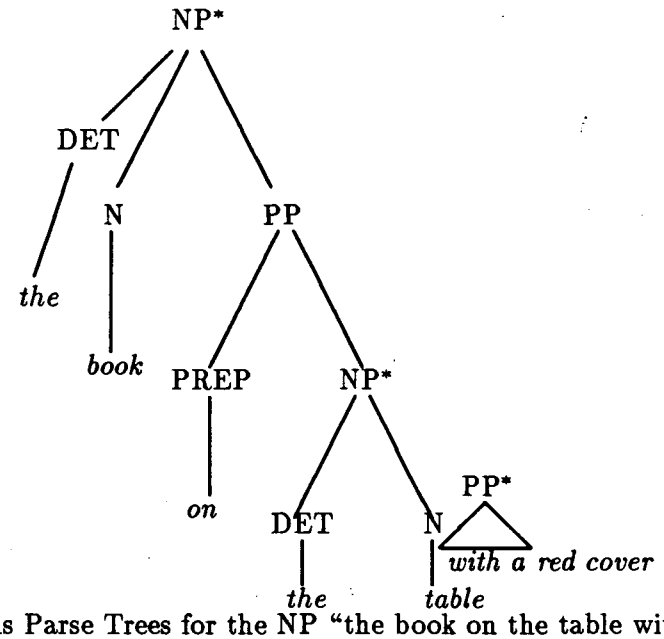


Figure 5.7: Ambiguous Parse Trees for the NP "the book on the table with a red cover"

Representing the Attachment of a Modifier to a Referent

Definition 5.1 that follows defines a particular kind of set, called a P-composition, as a means of representing the attachment of a modifier to a referent. Given an arc $S_i \longrightarrow S$ in the DG the *corresponding* P-set is $P^S_{S_i}$. Let R denote a referent and M a modifier of R. Let v_1 and v_2 denote vertices in the DG denoted by R and M, respectively. The P-composition for the attachment of M to R is the set composition of the P-sets corresponding to arcs in the minimum weight path connecting v_1 and v_2. Given two binary sets $R_1 \subseteq A \times B$ and $R_2 \subseteq B \times C$, the set composition of R_1 and R_2, denoted $R_1 \circ R_2$, is the set

$$R_1 \circ R_2 = \{(a,c) : \exists b(aR_1b \ \& \ bR_2c)\}.$$

Definition 5.1. Given a simple directed path between R_0 and R_k with vertices $v_1, \ldots, v_{k-1}$ labeled $R_1, \ldots, R_{k-1}$, respectively, the P-composition of sets R_0 and R_k on the given path is the set $R_0 \Delta R_k$ defined as follows: There are three cases.

1. The path R_0 is of length 0.

$$R_0 \Delta R_0 = \{\} \text{ (the empty set)}$$

2. The path $R_0 \rightarrow \ldots \rightarrow R_{k-1} \rightarrow R_k$ is of length $k > 1$ and (R_{k-1}, R_k) is a forward arc from R_0 to R_k.

$$R_0 \Delta R_k = (R_0 \Delta R_{k-1}) \circ P^{R_k}_{R_{k-1}}$$

3. The path $R_0 \rightarrow \ldots \rightarrow R_{k-1} \leftarrow R_k$ is of length $k > 1$ and (R_{k-1}, R_k) is a backward arc from R_0 to R_k.

$$R_0 \Delta R_k = (R_0 \Delta R_{k-1}) \circ P^{R_{k-1}}_{R_k}$$

The parent sets of $R_0 \Delta R_k$ are the sets R_0 and R_k.

For example, consider the path between *Student* and *Prof* through *Course* in the University DG:

$$Student \longrightarrow SC \longleftarrow Course \longrightarrow CP \longleftarrow Prof$$

The P-composition of *Student* and *Prof* on the given path $Student \Delta Prof$ is the set

$$((P^{SC}_{Student} \circ P^{SC}_{Course}) \circ P^{CP}_{Course}) \circ P^{CP}_{Prof}.$$

The P-composition permits the attachment of a modifier to a referent to denote a vertex in the DG just as primitive constituents denote vertices. The ability to denote the attachment of a modifier by a vertex makes it possible to use the heuristic BTG given previously (for resolving the WSD problem) to also resolve the MA problem. It is important to note that no extra work is required to provide denotations for modifier attachments

because, as we have shown in this subsection, the vertices denoted by modifier attachments can be computed automatically based on knowledge in the parse tree and the domain graph.

Computing Min/Max Values for Sets Denoted by Modifier Attachments

This section shows how to compute min/max values for sets that are defined using only the relational *join* and *projection* operators. P-sets fall into this category. A method for computing min/max values for P-sets is needed as part of our heuristic for resolving ambiguous modifier attachments.

Let $TG(Q)$ be a target graph for request Q. The vertices in $TG(Q)$ are called *distinguished vertices*. A query graph G determined by $TG(Q)$ defines a set Γ which is the join of sets denoted by adjacent vertices in G projected on the sets denoted by vertices in $TG(Q)$. The parent sets of Γ are those that label the vertices in $TG(Q)$. If S denotes the set that labels vertex $v \in TG(Q)$, then the min/max value of G relative to v gives the min/max value of Γ on S.

Definition 5.2. Given a query graph G with distinguished vertex v, the min/max value of a forward edge in G relative to v is the min/max value that labels the corresponding edge in the domain graph. The min/max value of a backward edge in G relative to v is $(1,1)$.

Definition 5.3. Let the product of min/max pairs (a,b) and (c,d) be the pair $(a \times c, b \times d)$. The min/max value of G relative to v is the product of the min/max values of all edges in G relative to v.

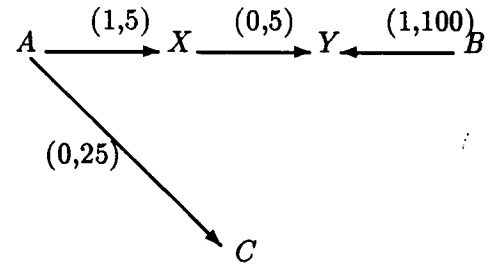


Figure 5.8: A Query Graph Determined by $\{A, B, C\}$

The min/max value $(1, 1)$ acts as an identity for multiplication.

$$(p, q) \times (1, 1) = (p, q) \text{ for all } m \text{ and } n$$

Therefore, the backward edges can be ignored when computing the min/max value of G relative to v, unless all of the edges are backward edges in which case the min/max value is $(1, 1)$.

Example 5.1. Consider the domain graph of Figure 5.8.[5] The target graph $TG(Q) = \{A, B, C\}$ determines exactly one query graph which is the entire domain graph. The forward edges relative to A are (A, X), (X, Y), and (A, C). There is only one backward edge relative to A which is (Y, B). The min/max value of G relative to A is $(1, 5) \times (0, 5) \times (0, 25) \times (1, 1) = (0, 625)$. By similar computations, the min/max value of G relative to B is (0,2500) and of G relative to C is (0,25).

[5]Here we are using non-standard min/max values which were introduced in Chapter 4.

Simultaneous Resolution of Word Sense Ambiguities and Ambiguous Modifier Attachments

This subsection describes a heuristic which makes use of knowledge expressed by the min/max values for resolving MA and WSD. It is well known by researchers in the area that the two problems cannot be resolved independently from each other. A brief introduction to the heuristic is given first followed by a more detailed description.

Brief Introduction

Given an unambiguous parse tree UPT, the assignment of a unique meaning to each primitive constituent results in a target graph $TG(UPT)$ which gives the meaning either explicitly or implicitly for each modifier and for each referent. However, information about the attachments of modifiers to referents is missing.

For a given modifier M denoting vertex v_1 and referent R denoting v_2, we are not guaranteed to find a path in the DG connecting v_1 and v_2. Many of the ambiguous parse trees can be eliminated on those grounds. For each of the remaining ones, an augmented target graph is constructed by adding one vertex for each ambiguous attachment of a modifier to a referent. The new vertex denotes the P-composition of the vertices denoted by the referent and its modifier.

Min/max values are computed for P-compositions and the minimum weight query graph for each augmented target graph is determined. The best attachment of the modifiers (and assignment of meanings to the words) is that specified by the augmented target graph which determines the query graph of smallest weight over all possible augmented target graphs.

The Details

Let $U_1(PT),\ldots,U_r(PT)$ be the unambiguous parse trees for PT. Associated with each of them $U_i(PT), 1 \leq i \leq r$ is a collection of target graphs, each of which assigns exactly one vertex in the DG to each primitive constituent in PT. Let the target graphs for $U_i(PT)$ be $T_{i1}(PT),\ldots,T_{it}(PT)$. Each unambiguous parse tree $U_i(PT)$ has exactly l modifiers $m_1,\ldots,m_l$. Let $\mathcal{V}(r_j)$ and $\mathcal{V}(m_j)$ denote the vertices denoted by the referent r_j and the modifier m_j, respectively. With respect to a given target graph $T_{i\theta}(PT)$ there will be only one such vertex for each referent and modifier. The modifier graph for $U_i(PT)$ in the context of the target graph $T_{i\theta}(PT), 1 \leq \theta \leq t$, denoted $MG_{i\theta}(PT)$ is defined as follows:

1. If a path does not exist between $\mathcal{V}(r_j)$ and $\mathcal{V}(m_j)$ for any $1 \leq j \leq l$ then the modifier graph $MG_{i\theta}(PT)$ is undefined.[6]

2. Otherwise

$$MG_{i\theta}(PT) = \bigcup_{j=1}^{l} \mathcal{V}(r_j)\Delta\mathcal{V}(m_j)$$

The augmented target graph for $TG_{i\theta}(PT)$ denoted $TG^*_{i\theta}(PT)$ is defined as follows:

$$TG^*_{i\theta}(PT) = TG_{i\theta}(PT) \bigcup MG_{i\theta}(PT)$$

Given the augmented target graph and assuming that the modifier graph is defined, the MA problem can be handled like the WSD problem as illustrated by the following pictorial.

$$\text{sentence constituent} \Longrightarrow BTG \Longrightarrow \text{the best augmented target graph}$$

[6]The modifier and the referent are not related above the minimum threshold needed to consider them related at all.

The best augmented target graph for PT assigns exactly one vertex to each primitive constituent of PT and exactly one referent to each modifier.

The above specification assumes that all of the ambiguous parse trees are available at one time for analysis by the heuristic. However, it is inefficient for the parser to generate all possible unambiguous parse trees. Our heuristic is assumed to be applied to fragments of larger requests. Query graphs are generated for the fragments and analysed by the heuristic as a basis for early elimination of some of the unambiguous parse trees for the complete request.

5.5 Summary

In the descriptions of natural language interfaces that appear in the literature, little attention has been paid to the mechanism by which the interface to a database system will be accomplished. This is due to the fact that most of the existing natural language interfaces are experimental, and the database is implemented using LISP or PROLOG (Datalog, CHAT-80 [91]). Some systems are interfaced with a file system rather than a database system (ASK, LDC-1). Therefore the data management capabilities that could be exploited to improve the portability of the interface are not available. For the commercially available systems (THEMIS [29], INTELLECT) details of the implementation are not reported because this information is proprietary to the company that markets the product.

In this chapter we have focused on the mechanism by which the interface can be accomplished with the added requirement that the natural language interface (NLI) take full advantage of capabilities associated with the database system to permit it to be portable

from one domain and database to another. An architecture for natural language interfaces has been proposed which comprises a typical NLI enhanced with a module called the data management strategy (DMS) for portability.

The data management strategy is designed as a layering of schemas each one based on a different data model. The top level schema (the SET schema) provides knowledge for formulating the internal representation, and for dealing with ambiguities in requests (semantic ambiguities, word sense ambiguities, and modifier attachments). Each schema in the layering is assumed to be derived from the previous one, by methods which are at present partially automated [31, 80, 61, 73]. Knowledge, about how the objects in one schema relate to those in the schema from which it is derived, is useful for automatically translating the internal representation of a request to a formal DB query. The NF^2 object model has been introduced specifically to facilitate representation of that knowledge.

The purpose of the NF^2 object model in our design strategy is two fold: It permits a natural structuring of the knowledge about how the RDB schema is derived from the SET schema. 2.) It facilitates representation of the relationships between a SET schema and the corresponding relational schema due to its capability of multiply representing objects. The particular organization of the knowledge that is imposed by the NF^2 object model is useful as a basis for providing general algorithms for translating the internal representation of a request to a formal DB query. Schema portability of the NLI is enhanced because the knowledge needed to construct the formal query is already available as a result of designing the RDB schema. Otherwise, it would have to be provided each time the NLI is adapted to a new RDB schema.

The internal representation is expressed in the context of the SET schema. If the internal representation is expressed within the context of a relational schema, then the NLI must be provided with knowledge, about which relations may be joined and about which columns may be used as the basis for the join. No such supplementary information is needed when the internal representation is expressed in the context of the SET schema.

The DB system itself is used to enhance data model portability. Extensible database systems permit themselves to be easily extended to support new objects. Additions to the existing DB system provide a *virtual* data model for the query generator and query interpreter. When the NLI is to be attached to a new data model, the DMS must be modified, but this will be easier than modifying the query generator and query interpreter. This feature of our proposed design is particularly useful for future generations of DB systems which are expected to be highly extensible [74, 59].

For a request such as "List all the red cogs", the information is not available in the request that *red* is a color and *cog* is the name of a part. This knowledge, which we have referred to as column information, is needed for automatically formulating the formal DB query. We have shown how column information can be obtained from a standard current RDB system, thus providing an update of Harris' results [41] to apply to more current database systems.

We have examined a related problem which has not been resolved by Harris' approach, that of providing an appropriate answer to a request when primitive constituents referenced in the request are neither in the DB nor in the lexicon, and proposed new features for DB systems that would be useful for dealing with this problem.

Finally, heuristics have been specified for making use of the knowledge available from the SET schema for resolving the problems of semantic ambiguities, word sense ambiguities, and modifier attachments. It is well understood by researchers in the area that, in general, ambiguous modifier attachments in a request cannot be resolved independently from each other, or from word sense ambiguities. It is expected that our heuristic would, in practice, be applied to fragments of a request. For those fragments, ambiguous attachments of the modifiers and ambiguous meanings for words are handled simultaneously.

Chapter 6

Using the Metadatabase to Provide Semantic Knowledge

6.1 Introduction

A strategy for logically structuring the knowledge needed by a natural language interface has been described in Chapter 5. The strategy involves a layering of schemas each one based on a different data model. The purpose of the layering is to take advantage of previous work which specifies the mapping between different types of schemas. Knowledge of the mapping is useful to the natural language interface for translating the internal representation of a request to a DB query.

One of the types of schemas in the layering is based on a model called the NF^2 object model which was introduced in Section 2.3.3 and which will be described more fully in this chapter. In addition, in this chapter we provide an expansion of the metaschema

of a standard relational database system that would be needed to describe NF^2 object schemas. In addition, the expanded metaschema is capable of expressing knowledge about the derivation of an RDB schema from a particular NF^2 object schema. A significant feature of the extended system is that operations in the metaschema were extended by adding assertions just as the data of the metaschema were extended by adding relations.

A metaschema comes with knowledge of certain classes of objects built into it. For example, a metaschema for a relational database has knowledge of value sets and base relations. For illustration assume that objects are described by a generalization hierarchy.[1]

A metaschema "breaks down" when there is not an exact correspondence between the classes of objects that it can describe and the classes that need to be described. That is, if the metaschema does not distinguish a subclass of objects that need to be described, or if a more general class of objects must be described, but the metaschema is capable of describing only a subclass, then the metaschema breaks down. For example, the SET model distinguishes between *primitive value* and *primitive non-value* sets, and corresponding concepts exist in the standard metaschema, proposed by Mark and Roussopoulos [63], which are known by the names *lexical* and *non-lexical set.* Therefore, the metaschema does not break down on primitive sets. In the SET model the parent sets of a base set may be any previously declared sets, primitive, base or defined. The metaschema only has knowledge of a subclass of base sets, those each of whose parent sets is a primitive set. Therefore, the

[1] Tsichritzis and Lochovsky [84] point out that this is a good assumption. Generalization [77] has been used informally in data management for a long time. A metaschema, for a database organized as a generalization heirarchy, describes a wide range of schemas.

metaschema breaks down on base sets. At the point where a metaschema breaks down, it is necessary to describe additional general properties of objects by data. The metaschema must be expanded by defining additional relations to hold the data, and operations on the new relations must be defined which enforce constraints on the data.

The remainder of the chapter is organized as follows: In Section 6.2 the NF^2 object model is presented. Deficiencies of the metaschema for describing NF^2 object schemas are pointed out in Section 6.3. In Section 6.4 additions to the metaschema that would be needed to describe NF^2 object schemas are given. In Section 6.5 the use of the extended metaschema to describe a particular NF^2 object schema is illustrated.

6.2 The NF^2 Object Model

Knowledge representation strategies for current applications such as natural language understanding, computer aided design, and full-text and mixed media databases are typically *not* first normal form and often include the concept of an entity. The concept of an entity is introduced to the NF^2 data model in the following way: Associated with every column name is a value set name which identifies either a set of values or a set of surrogates used to identify entities. A set of values is called a ***value*** set and a set of entity surrogates a ***non-value*** set. Leaf column names are associated with value sets and non leaf names with non-value sets. The NF^2 data model extended in this way is called the NF^2 object model.

A tuple of a column is defined as follows: For leaf columns a tuple of a column is a value from the value set of the column. For non-leaf columns a tuple of a column is a value from the value set of the column together with one or more tuples from each of the immediately

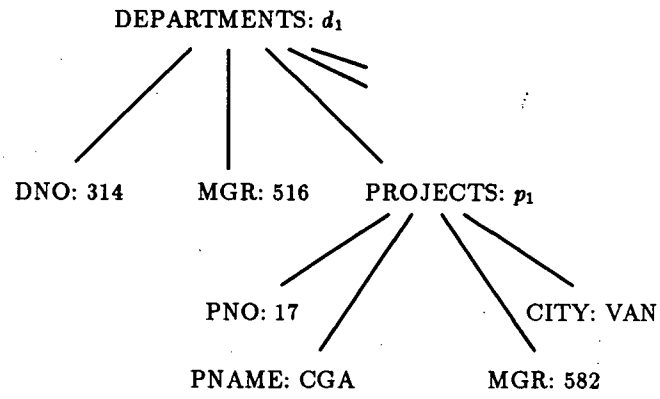


Figure 6.1: A Tuple of Column DEPARTMENTS

subordinate columns.

The tree structure of a tuple is illustrated in Figure 6.1. The nodes of the tree are labelled $c : v$ where c is a column name and v is a value from the value set of c. In the figure d_1 and p_1 are non-lexical values. The entire tree rooted at (DEPARTMENTS: d_1) is a tuple of DEPARTMENTS. The subtree rooted at (PROJECTS: p_1) is a tuple of PROJECTS. Only one of the three PROJECTS tuples subordinate to d_1 is illustrated.

The next several definitions are needed so that ultimately we can give a precise definition for the term NF^2 *object schema.* A tuple of a leaf column is called a *leaf value.* The value that occurs at the root of a tuple is called a *root value.* Tuples of leaf columns have only one value which is both the leaf value and the root value. The *extension* of a column is a collection of tuples of the column. The *root extension* of a column is the set of all values from the value set of the column that partake as root values in tuples of the column.

An *inclusion constraint* is a statement of the form $a \subseteq b$ where a and b are column names. The statement $a = b$ is an abbreviation for the pair of constraints $a \subseteq b$ and $b \subseteq a$.

In this chapter all columns are assumed to have unique names which are used rather than the full path names for identifying columns. An NF^2 *object scheme* is a collection of column names partitioned into one root partition and zero or more non root partitions. The root partition contains exactly one name and each non root partition is itself an NF^2 scheme. An NF^2 *object schema* is a collection of NF^2 schemes together with a set of inclusion constraints. For each inclusion constraint $a \subseteq b$ in a schema the root extensions r_a and r_b of a and b obey the rule that every object in r_a occurs also in r_b.

Multiple representations for objects is supported within the NF^2 object model: Given a tuple rooted at e, let $r(e)$ denote the representation of e. $r(e)$ is defined recursively as follows:

1. If e is a leaf value, then $r(e) = e$.

2. If e is a non-leaf value with immediately subordinate values $e_1, \ldots, e_n$, then $r(e) = \{r(e_1), \ldots, r(e_n)\}$.

If a given entity is a member of the root extensions of two different columns, then it may have two different representations. A leaf value has identical representations in every leaf column in which it appears.

6.3 Deficiencies of the Metaschema

We now consider how NF^2 object schemas can be described by the core metaschema and additions to it. The following constraints (among others) are imposed by the core metaschema, and any NF^2 object schema that is to be described must abide by them:

1. Every existing column belongs to some relation.

2. Every existing relation has at least one column.

By constraint (1) a root relation cannot be considered to be a column, and by constraint (2) a leaf column cannot be considered to be a relation.

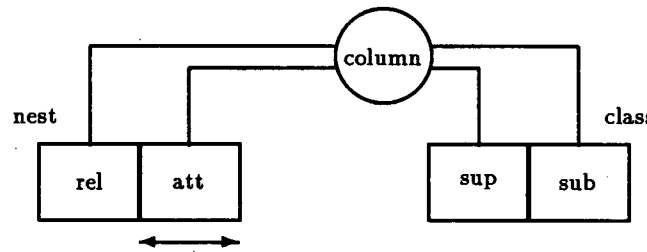
To ensure that NF^2 object schemas abide by constraint (1) we make the assumption that a root relation is a column of itself. For illustration, consider the Projects schema of Figure 6.9 and the description of the Projects Schema of Figure 6.10. The root relation DEPARTMENTS is denoted by entity surrogate r_5 in relation *reln* and its role as a column by entity surrogate a_{18} in relation *coln*. The assertion $rdas(r_5, d_{11}, a_{18})$ states that a_{18} is a column of r_5. It is important that a root relation be considered as a column because every root relation has a value set and the relation *rdas* associates columns (not relations) with value sets.

This trick does not work for leaf relations. Let us assume that every leaf relation has itself as a column. More precisely, any leaf relation l in its role as a column a has a value set d such that the assertion $rdas(l, d, a)$ is true. Unfortunately, this assumption leads to a violation of one of the key constraints in the core metaschema. A leaf relation of an NF^2 relation plays a role as a column of the relation that is its immediate parent. If we assume that a leaf relation is also a column of itself, then the constraint that a column belongs to at most one relation is violated. This constraint is expressed by the condition in the core metaschema that *att* is the key for relation *rdas*.

To ensure that NF^2 object schemas abide by constraint (2) we make the assumption that leaf columns are not also relations. Thus, every relation is considered to be a column, but not every column is considered to be a relation. For the Projects schema (Figure 6.9) and its description (Figure 6.10) observe that names of leaf relations do not appear in the extension of relation *reln*.

6.4 The Extended Metaschema

To capture the nested structure of NF^2 relations a new relation *nest* is introduced. In addition a new relation *class* is introduced to represent inclusion constraints.

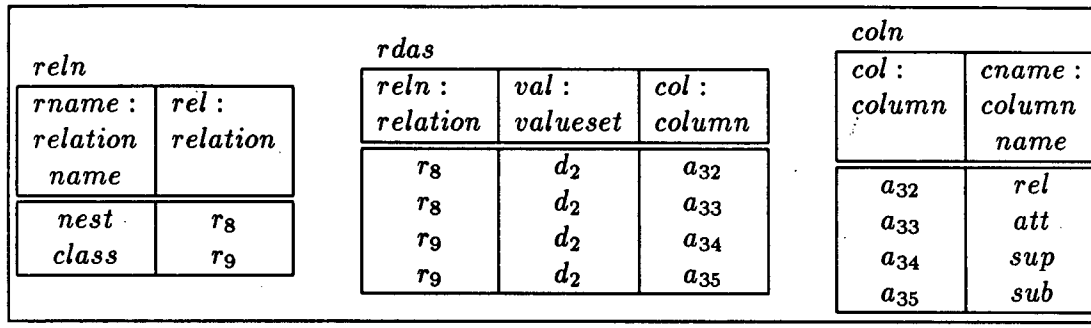


$$\begin{aligned} \text{column[att]} &= \text{nest[att]} \\ \text{nest[rel]} &\subseteq \text{column[att]} \\ \text{class[sup]} &\subseteq \text{column[att]} \\ \text{class[sub]} &\subseteq \text{column[att]} \end{aligned}$$

Figure 6.2: The Extended Metaschema

The data structure diagram and inclusion constraints for the additional relations are given in Figure 6.2. *nest*(a,b) states that column a in its role as a relation has b as a column. *class*(a,b) represents the constraint $b \subseteq a$. Only immediate subset relationships are explicitly represented in relation *class*.

The relations *nest* and *class* are described by entering tuples in relations of the core



reln

rname : *relation name*	*rel* : *relation*
nest class	r_8
	r_9

rdas

reln : *relation*	*val* : *valueset*	*col* : *column*
r_8	d_2	a_{32}
r_8	d_2	a_{33}
r_9	d_2	a_{34}
r_9	d_2	a_{35}

coln

col : *column*	*cname* : *column name*
a_{32}	*rel*
a_{33}	*att*
a_{34}	*sup*
a_{35}	*sub*

Figure 6.3: Extended Metaschema Description Stored in Metaschema Extensions

class

sup : *column*	*sub* : *column*
a_{33}	a_{36}
a_{36}	a_{33}
a_{36}	a_{32}
a_{36}	a_{34}
a_{36}	a_{35}

Figure 6.4: Inclusion Constraints Stored in *Class* Extension

metaschema as illustrated in Figure 6.3. Inclusion constraints associated with the new relations are stated by entering tuples in the relation *class* of the extended metaschema as illustrated in Figure 6.4. In the figure the entity surrogate for column *column*[*att*] of the core metaschema is assumed to be a_{36}.

The *class* relation is provided for use by the natural language interface. In Section 5.2, we have considered how column information (knowledge about membership of primitive constituents in database columns) can be obtained from the database. This information is useful for automatically formulating relational queries for natural language requests which do not, in general, reference database columns. From the *class* relation and the standard

```
insert(column(A))
→   var(A),
    new(column(A)),
    insert(column(A)).

→   nonvar(A) ∧ column(A) ∧ (rdas(_,D,A)) ∧ (vset(D,_,value)).

→   nonvar(A) ∧ column(A) ∧ (rdas(_,D,A)) ∧ (vset(D,_,non-value)),
    insert(nest(A,_)).

→   nonvar(A) ∧ ¬(column(A)),
    assert(column(A)),
    insert(coln(A,_),
    insert(rdas(_,_,A),
    insert(column(A)).
```

Figure 6.5: Insertion into *Column*

metaschema relations ***reln***, ***rdas***, and ***coln***, the columns that are not included in any of the others can be automatically determined. Only these columns will be searched for a given primitive constituent to determine the columns of which it is a member. Knowledge about membership of the given primitive constituent in other columns can be deduced from relation *class*.

6.4.1 Operations in the Extended Metaschema

Just as additional relations are provided without disturbing the existing metaschema relations, additional update dependencies are provided without disturbing the existing ones. Expanded versions of the existing operations *insert*(*column*(*A*)) and *delete*(*column*(*A*)) and definitions of insert and delete operations for the *nest* relation are provided in Figures 6.5, 6.6, 6.7, and 6.8.

```
delete(column(A))
→   ¬(column(A)).

→   nonvar(A) ∧ column(A) ∧ (rdas(_,D,A)) ∧ (vset(D,_,value)),
    retract(column(A)),
    delete(rdas(_,_,A)),
    delete(coln(A,_)).

→   nonvar(A) ∧ column(A) ∧ (rdas(_,D,A)) ∧ (vset(D,_,non-value)),
    retract(column(A)),
    delete(nest(A,_)),
    delete(rdas(_,_,A)),
    delete(coln(A,_)).

→   var(A),
    write("column surrogate?"),
    break, read(A),
    delete(column(A)).
```

Figure 6.6: Deletion from *Column*

The operation *insert*(*column*(*A*)) is modified by adding one update dependency as illustrated in Figure 6.5. A new column surrogate is created if needed. If the column surrogate identifies a simple column the operation succeeds with the database unchanged. The third update dependency is new. If the column surrogate identifies a non-leaf column the insertion is propagated to the *nest* relation. The fourth update dependency inserts the column surrogate and propagates the insertion to relations *coln* and *rdas*.

In the operation *delete*(*column*(*A*)) if *A* is defined on a non-value set then *A* has columns that should also be deleted. The definition of operation *delete*(*column*(*A*)) is given in Figure 6.6. If *A* is not a column then the operation succeeds with no change to the database. If *A* is defined on a value set then *A* is deleted and the deletion is propagated to relation *rdas* and *coln*. If *A* is defined on a non-value set then *A* is deleted and the deletion is propagated to

```
insert(nest(A,B))
→   nonvar(B) ∧ nest(_,B).

→   var(A),
    write("parent surrogate?"),
    break, read(A),
    insert(nest(A,B)).

→   var(B),
    new(column(B)),
    insert(nest(A,B)).

→   nonvar(A) ∧ nonvar(B) ∧ ¬(nest(_,B)),
    assert(nest(A,B)),
    insert(column(A)),
    insert(column(B)).
```

Figure 6.7: Insertion into *Nest*

relations *rdas*, *coln*, and *nest*. The fourth update dependency requests a column surrogate if A is uninstantiated. In the operation $delete(column(A))$ the third update dependency is new (not part of the core metaschema).

In the operation $insert(nest(A, B))$ (See Figure 6.7.) let us call A the parent surrogate and B the child surrogate. If variable B is instantiated and the surrogate occurs in the nest relation as a child then the operation succeeds with the database unchanged. If the parent surrogate is not provided it is requested. If the child surrogate is not provided one is created. If both variables are instantiated and the child surrogate is not already in the nest relation then the tuple is inserted. Update dependencies one and four guarantee that a child surrogate has at most one parent.

The variables A and B in operation $delete(nest(A, B))$ (Figure 6.8) bind to parent and child column surrogates, respectively. If both variables are uninstantiated or if no tuples

```
delete(nest(A,B))
→   var(A) ∧ var(B),
    write("nothing done").

→   ¬(nest(A,B)).

→   nonvar(A) ∧ var(B) ∧ nest(A,B),
    retract(nest(A,B)),
    delete(nest(B,_)),
    delete(column(B)),
    delete(nest(A,_)).

→   nonvar(B) ∧ (nest(A,B)),
    retract(nest(A,B)),
    delete(nest(B,_)),
    delete(column(B)).
```

Figure 6.8: Deletion from *Nest*

match the instantiated values then the operation succeeds with no change to the database. If the parent variable is instantiated and the child variable is uninstantiated then all child surrogates and their descendents are deleted. If the child variable is instantiated then only that child surrogate and its descendents are deleted.

The operations *insert*(*nest*(A, B)) and *delete*(*nest*(A, B)) and the new update dependencies of operations *insert*(*column*(A)) and *delete*(*column*(A)) are part of the extended metaschema. We were pleased to find that none of the existing update dependencies had to be deleted or modified.

6.5 Example

This section provides an example of the use of the core metaschema and additions to it to describe NF^2 object schemas. A pictorial description of an NF^2 object schema for a database that records information about projects is given in Figure 6.9. In Figure 6.10 part of the Projects schema of Figure 6.9 is described in the extensions of the relations *reln*, *vset*, *rdas*, and *coln* of the core metaschema. Only that part of the extensions that describes the Projects schema is illustrated. In a fully operational database system other schemas as well as the core metaschema itself would be described in the extensions of the core metaschema relations. In Figure 6.11 the relations *nest* and *class* of the extended metaschema are used to describe the remaining part of the Projects schema. In a fully operational database system additional NF^2 object schemas might be described in the extensions of the relations of the extended metaschema.

6.6 Summary

We have introduced a model, called the NF^2 object model by extending the NF^2 data model to include the concept of an entity. The NF^2 object model is capable of multiply representing an object and permits different representations of the same object to be recognized.

The NF^2 object schema figures in our design strategy for portable natural language interfaces as an intermediate step between the SET schema and the relational schema. Mappings between the SET schema and the NF^2 object schema and between the NF^2

DEPARTMENTS					
DNO	MGR	PROJECTS			
		PNO	PNAME	MGR	CITY

COMPANIES					
CNO	CNAME	PROJECTS			
		PNO	PNAME	MGR	CITY

PROJECTS			
PNO	PNAME	MGR	CITY

$$\text{DEPARTMENTS[PROJECTS]} \subseteq \text{PROJECTS}$$
$$\text{COMPANIES[PROJECTS]} \subseteq \text{PROJECTS}$$

Figure 6.9: NF^2 Schema for the Projects Database

object schema and the relational schema provide useful knowledge for translating the internal representation of a request to a formal DB query. The improvement in portability over previous systems results from the ability to automatically generate information about mappings between the different schemas.

To represent the mapping information, there is a need to maintain multiple representations of objects and to recognize different representations of the same object. The concept of multiple representations is missing from the NF^2 data model as previously described[18]. We have specifically defined the NF^2 object model to ensure that extensional objects can be multiply represented.

Current database systems are capable of supporting a variety of applications each of

reln

rname : *relation name*	*rel* : *relation*
DEPARTMENTS	r_5
COMPANIES	r_6
PROJECTS	r_7
DEPARTMENTS[PROJECTS]	r_8
COMPANIES[PROJECTS]	r_9

vset

val : *valueset*	*vname* : *valueset name*	*lex* : *lexicality*
d_8	deptno	value
d_9	empno	value
d_{10}	project	non-value
d_{11}	department	non-value
d_{12}	projno	value
d_{13}	projname	value
d_{14}	cities	value
d_{15}	company	non-value
d_{16}	compno	value
d_{17}	compname	value

rdas

rel : *relation*	*val* : *valueset*	*col* : *column*
r_5	d_8	a_{11}
r_5	d_9	a_{12}
r_5	d_{10}	a_{13}
r_8	d_{12}	a_{14}
r_8	d_{13}	a_{15}
r_8	d_9	a_{16}
r_8	d_{14}	a_{17}
r_5	d_{11}	a_{18}
r_6	d_{16}	a_{19}
r_6	d_{17}	a_{20}
r_6	d_{10}	a_{21}
r_9	d_{12}	a_{22}
r_9	d_{13}	a_{23}
r_9	d_9	a_{24}
r_9	d_{14}	a_{25}
r_6	d_{15}	a_{26}
r_7	d_{12}	a_{27}
r_7	d_{13}	a_{28}
r_7	d_9	a_{29}
r_7	d_{14}	a_{30}
r_7	d_{10}	a_{31}

coln

col : *column*	*cname* : *column name*
a_{11}	DNO
a_{12}	MGR
a_{13}	PROJECTS
a_{14}	PNO
a_{15}	PNAME
a_{16}	MGR
a_{17}	CITY
a_{18}	DEPARTMENTS
a_{19}	CNO
a_{20}	CNAME
a_{21}	PROJECTS
a_{22}	PNO
a_{23}	PNAME
a_{24}	MGR
a_{25}	CITY
a_{26}	COMPANIES
a_{27}	PNO
a_{28}	PNAME
a_{29}	MGR
a_{30}	CITY
a_{31}	PROJECTS

Figure 6.10: Projects Schema Description Stored in Metaschema Extensions

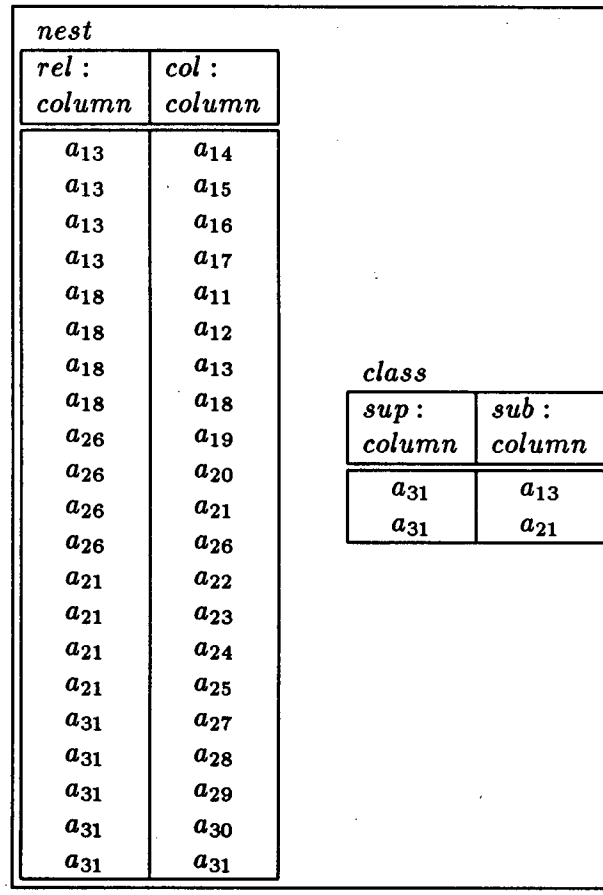


Figure 6.11: Projects Schema Description Stored in Extended Metaschema Extensions

which may use a different knowledge representation strategy. Standard relational database systems provide this capability by supplying one core metaschema (including core operations) for use by all applications, and permitting additional relations to be defined for each application to describe additions to the core metaschema that are needed to describe that application's knowledge representation strategy.

The metaschema of a standard relational database system has been expanded to describe NF^2 object schemas. Expansion of the metaschema results in a collection of relations suitable for recording information about the mapping between the RDB schema and the NF^2 object schema. Further research is needed to define a collection of relations that is suitable for recording information about mapping between the NF^2 object schema and the SET schema.

Chapter 7

Conclusions and Further Research

This thesis has focused on the problem of designing a highly portable natural language interface for relational database systems. The issues considered can be grouped into two parts: those concerned with identifying knowledge associated with the DB that is useful for automatically understanding natural language, and those dealing with how the NLI should be designed to be portable between domains and databases. The two topics are related because, if knowledge associated with the DB that is useful for natural language understanding can be automatically provided to the NLI, then portability of the NLI will be enhanced. This chapter summarizes the issues considered in each of the main areas, presents our conclusions, and suggests areas for further research.

7.1 Knowledge Associated with the DB that is Useful for Natural Language Understanding

The SET schema expresses knowledge that is useful for automatically understanding natural language. Previous systems have captured knowledge for natural language understanding from the relational database schema. The SET schema is a better source of knowledge for the following reasons:

1. Natural language interfaces that obtain knowledge from the relational schema require supplementary knowledge to alleviate the problem of referential ambiguity. Any ER based model (such as the SET model) and even the Network model [17] uses the notion of a link between objects to indicate a relationship between them, and, therefore, the problem of referential ambiguity does not arise. If the natural language interface obtains knowledge from the SET schema, supplementary knowledge to resolve the problem of referential ambiguity is not needed, and therefore the domain portability of the NLI is enhanced.

2. Knowledge about which objects in the domain are related is useful for natural language understanding as pointed out in (1). In addition, knowledge about the *nature* of the relationship is useful, and such knowledge is called *metaknowledge*. The SET model permits metaknowledge to be stated explicitly in the SET schema. The relational model does not, but metaknowledge may be obtained indirectly from the schema by inspecting the structures that are described therein. However, not all of the metaknowledge available from SET schemas can be captured from relational database structures. Thus,

capturing knowledge for natural language understanding from the SET schema is a better approach than obtaining it from the relational schema.

7.1.1 What is it Useful for?

The metaknowledge is knowledge about the relationships in the domain, in particular, whether the relationships are one-to-one, one-to-many, or many-to-many, and whether they are total or partial, and into or onto. We considered two main constructs in English.

1. genitive (possessive) noun modifiers (e.g., Jones' courses)

2. prepositional phrases that modify a noun (e.g., patients in hospitals)

Heuristics were provided to resolve three different types of ambiguities (word sense ambiguities, semantic ambiguities, and post noun modifier attachments) in requests that contain these two different language constructions. The heuristics give an ordering on the different possible interpretations according to their likelihood of being the interpretation intended by the user.

For the possessive case the heuristic favors the interpretation in which the objects possessed belong to the possessor and no one else (a many to one relationship) as well as the case in which every possessor possesses at least one object. The prepositions considered were "in" and "with", and, in particular, the meaning *part of*. For such relationships the heuristic favors the case in which the modifier is associated with at least one and at most one referent.

7.2 The Design of a Portable Natural Language Interface

The SET schema has been used as the basis for our design of a portable natural language interface. Supplying the natural language interface with knowledge captured from the SET schema enhances domain portability, that is the ease with which the NLI can be adapted to a new domain. We have focused on resolving ambiguities in natural language database requests. Little further work is required to provide the NLI with knowledge for resolving ambiguities, because the knowledge has previously been gathered for the purpose of designing the RDB schema. A major advantage of our approach is that the duplication of effort between the processes of designing the DB schema and adapting the NLI to a new domain is eliminated.

We have also addressed the problem of adapting the NLI to a new database. Our proposed system takes advantage of the extensibility features of the database system to permit it to be easily adapted to a new data model. To permit the NLI to be easily adapted to a new DB schema, the internal representation of the NL request must be independent from the DB schema. In our proposed system, the internal representation is expressed within the context of the SET schema, a choice that has been motivated by the fact that the SET schema is independent of any particular DB schema and, moreover, independent of any particular data model. Therefore, knowledge about the particular DB schema need not be built into the linguistic components of the system. In addition, knowledge needed for translating the internal representation to a formal database query is provided as data, as opposed to being built into the database components of the natural language interface.

Both of these features contribute to the ease with which the natural language interface can be adapted to a new database schema. A technique in widespread use for designing the RDB schema is to begin with a SET schema for the domain and translate that description to a relational schema. We have investigated whether the knowledge generated by this process could be used for automatically translating the internal representation to a formal database query, and conclude the afirmative. Portability of the NLI is further enhanced because the knowledge is already available as a result of designing the RDB schema.

An intermediate model, called the NF^2 object model, has been proposed as a basis for specifying the relationship between a particular SET schema and the corresponding relational schema. The NF^2 object model is a conceptual tool which aids our understanding of the correspondence between the SET model and the relational model. It is easier to understand the correspondence in pieces, and the NF^2 object model provides a natural division. The subproblems are the correspondence between the SET model and the NF^2 object model and between the NF^2 object model and the relational model.

The metaschema of a standard relational database has been expanded to describe NF^2 object schemas, and we have conjectured that general algorithms can be written to capture, from the expanded metaschema, knowledge about the mapping between a particular NF^2 object schema and the corresponding relational schema. The expansion of the metaschema to capture knowledge about the mapping between a particular SET schema and the corresponding NF^2 object schema is left for future research.

7.3 Suggestions for Further Research

This section describes directions for further research emerging from our results.

7.3.1 Temporal Knowledge for NL Understanding

The notion of time is important for NL processing, and the foundations for it are provided within the SET model. The extension of a set changes with time. A notion of time would be useful for resolving conjunction scoping. For the requests

1. Which presidents visited New York and New Jersey after leaving office?

2. Which presidents died in New York and New Jersey after leaving office?

our knowledge that two different cities may be visited by a president at two different times leads use to admit the narrow scope reading as a possible meaning for (1). Similarly, our knowledge that presidents die at most once and that they cannot be in two different places at the same time leads us to exclude the narrow scope reading of (2) in favor of the wide scope reading.

7.3.2 Conjunctions of Database Values

A conjunction in natural language is frequently used to mean disjunction. For example, the request "red and blue books" has a conjunctive meaning "books that are both red and blue" as well as a "disjunctive" one "books that are red or blue". Janis [51] has shown how to generate subgraphs in relational schemas to represent the meaning of requests that contain database values separated by the word "and". In this subsection we will consider how

our semantic relatedness measure can be extended to resolve ambiguities in such requests. Janis' results are useful for our purpose because the subgraph in relational schemas for representing the meaning of a request is easily adapted to the domain graph.

For illustration, consider the request "Which libraries have borrowers named Smith and Jones?" which has the following possible meanings:

1. Which libraries have borrowers named both Smith and Jones?

2. Which libraries have borrowers named Smith or borrowers named Jones?

To determine the best scoping for the conjunction we would like to compare the absolute weights of the query graphs for the different interpretations. However, the target graph will be the same regardless of which scoping we choose, and therefore, the query graph will also be the same. To represent the distinction between the different conjunction scopings, a technique similar to the one we used for representing different prepositional phrase attachments is useful. Following Janis' terminology we will refer to the association to which the DB values are *attached* as the *branching node*. The branching node for interpretation (1) is the vertex that denotes the verb "name". In the library database introduced in Chapter 4, the branching node would be *Name* which associates borrowers with their names. The branching node for interpretation (2) is the vertex that denotes the verb "have" which in the library database would be *Membership* which associates libraries with borrowers.

The query graph for a given interpretation is constructed as follows: 1.) A vertex *newnode* is added to the domain graph to denote a relationship between the set from which the DB values are drawn and the branching node. 2.) The vertex *newnode* is added to

the target graph (TG) for the request. 3.) The possible interpretations are ordered by the weight of their query graphs (QGs) with the minimum weight QG corresponding to the most likely interpretation.

When the branching node is *Name*, *newnode* denotes an association which consists of those pairs $<< borr, boname >, boname >$ for which $< borr, boname >$ is a member of *Name*, and *boname* is a member of *BoName*. The min/max values of the association denoted by *newnode* are easily calculated. When the branching node is *Borrower*, the association denoted by *newnode* is intensionally equal to the association denoted by *Name*. Interpretations (1) and (2) above for the request "Which libraries have borrowers named Smith and Jones?" correspond with QGs of equal weight. Therefore, they are considered to be equally likely interpretations.

A conjunction of DB values can be viewed as a *complex* verb constructed from more simple verbs. For example, in interpretation (2) above, a complex verb, say *have_borrowers_named*, is constructed from the verbs *have* and *named*. Our heuristic measures the *part of* relationship between the verb so constructed and the request itself.

7.3.3 Ellipses

In this subsection, we consider how our semantic relatedness measure can be extended to apply to a class of ellipses that has been distinguished by Janis [51] called *qualification ellipses*. An ellipsis is an utterance in which constituents have been omitted. In a qualification ellipsis, moreover, the omitted constituents define qualifications on the entities referred to in the utterance. An approach to automatically understanding the meaning of an ellipses

is to determine the missing qualifications from a previous request in the dialog, and it is within this approach that our semantic relatedness measure is to be applied. Consider the pair of requests

1. students in the Computer Science department
2. supervised by Dr. Lee

The second utterance has at least the following possible completions:

1. students supervised by Dr. Lee in the Computer Science department
2. students supervised by Dr. Lee

The semantic relatedness measure orders possible completions of an ellipsis according to their likelihood of being the interpretation intended by the user with the possibility of some completions being ruled out. For the problem of ellipses our SRM is applied differently than for the other linguistic problems (semantic ambiguity, word sense disambiguation, etc.) The query graphs based on the target graphs for the different possible completions of the ellipsis are determined, and ordered on the basis of their weight relative to their respective target graphs. The notion of *relative weight* introduced in Chapter 4 needs to be generalized to apply to a set of vertices rather than just one. This is an easy task. The weight of a query graph $G = (V, E)$ relative to $T \subseteq V$ is the minimum of the relative weights over all $v \in T$.

7.3.4 Violations of the Min/max Values

It is useful to determine whether any of the interpretations of a request violates a condition in the domain that should always be true (a constraint). Such a request needs special

processing to determine the false presuppositions that underly it. A direct expansion of our work would be to develop a heuristic based on min/max values for this problem.

In this subsection some ideas on developing a heuristic that identifies parse trees that violate a constraint expressed by the min/max values are presented. The focus will be on violations due to prepositional phrase attachments, although violations due to other properties of the parse tree such as conjunction scoping are equally important. The method is based on two different types of violations that may occur as illustrated by the following examples based on the university domain graph of Figure 4.2.

1. A prepositional phrase attachment may contradict a constraint expressed by the min/max values. For example, the max value of CP on *Course* states that every course has a professor assigned to it. The interpretation "a student for a course without a professor" for the corresponding sentence fragment violates the condition expressed by the max value. If the noun *professor* had been itself further modified ("a student for a course without a good professor"), then it cannot be determined from the parse tree whether the max value is violated.

2. A prepositional phrase attachment may express no information in addition to that expressed by the min/max values. For example, if every course is taught by a professor, then the prepositional phrase attachment in the interpretation "a course taught by a professor, with no students" expresses no new information. If the noun "professor" is itself further modified then it is not possible to determine from the parse tree whether the attachment expresses no new information.

A *positive violation* is committed when information expressed by the min/max values is restated in the request. A *negative violation* is committed when the request is inconsistent with the min/max values. A positive violation of either min or max and a negative violation of min can be detected only if the head noun is not further modified. To clarify these ideas, consider the following cases:

1. min violated/positive violation:

   The request is "a course taught by a professor". The min value of CP on $Course$ states that every course is taught by a professor. The attachment of the modifying phrase "taught by a professor" to the head noun "course" gives no new information. However, if the head noun "course" is further modified (e.g., "a course taught by a good professor"), then new information is requested.

2. min violated/negative violation:

   The request is "a course without a professor". Since every course is taught by a professor and the noun "professor" is not itself modified, the attachment of the phrase "without a professor" to the head noun "course" violates the constraint expressed by the min value. If the request had read "a course without a good professor", then no violation occurs because even though every course is assigned a professor, not every course may be assigned a good professor.

3. max violated/negative violation:

   The request is "a student in several departments". According to the max value, a student belongs to at most one department. The attachment of the phrase "in several

departments" violates the information expressed by the max value. The violation occurs regardless of whether the noun "departments" is further modified.

4. max violated/positive violation:

   The request is "a student in at most one department". Since "department" is not itself modified, the attachment of "in at most one department" to the head noun "student" expresses no new information. No violation of the max value is committed by the request "a student in at most one small department".

The transitivity property of min/max values can be employed to gain more power out of these simple ideas. According to the min value every student belongs to at least one department and every department has at least one professor. By the transitivity property, every student has at least one professor in his or her department.

The interpretation "a student in a department with professors" denotes an association, say SP, defined as follows:

(**def** SP

select x:*Student*, y:*Prof*

where [**For some** z:*Dept*]

($< x, z >$:SD **and** $< z, y >$:PD))

SP is the join of SD and PD on *Dept*. A method for computing min/max values for n-ary sets, $n \geq 2$, that are defined on any connected directed-acyclic subgraph of the domain graph using only joins has been given in Section 5.4.4. By applying our method, the min value of SP on *Student* is computed to be 1. Therefore, the interpretation "a student in

a department with no professors" commits a negative violation of the min value of SP on *Student*.

We see from the above example that it will, in general, be necessary to compute min/max values to determine violations of min/max constraints. A method of computing min/max values for n-ary sets, $n \geq 2$, is necessary, because a prepositional phrase attachment is not always represented in the internal representation as a binary set. For example, if a noun has more than one modifier, the relationships between the noun and each modifier may be represented altogether by one n-ary association.

The remainder of this subsection describes how the internal representations of sentences can be analysed to determine which ones correspond with parse trees that commit a negative or positive violation. A *term* is a constant, a variable, or a tuple of terms. An *elementary assertion* is an assertion of the form $term$:*setname* or $term1 = term2$ where $term, term1$, and $term2$ are terms and *setname* names either a declared or embedded set. Let us refer to a variable that occurs in exactly one elementary assertion in the *assertion* part of an internal representation an ***underconstrained variable.*** The condition for identifying a positive violation derives from the following observation: If x_j is a universally quantified underconstrained variable, and the j^{th} min value of *assoc* is 1, then the assertion

$$not < x_1, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_n > : assoc$$

is necessarily false. If in the query form of the internal representation any of the variables $x_1, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n$ occur free, then the set defined by the query is necessarily empty.

Example 7.1. For the interpretation "a professor for a course, with no students", the

assertion form of the internal representation is

[**For some** x:$Prof$][**For some** y:$Student$] [**For some** w:$Course$]

(**not** $< x, y >$:PS **and** $< y, z >$:SC)

(PS **def**

select x:$Prof$, y:$Student$

where [**For some** z:$Course$]

($< z, x >$:CP **and** $< y, z >$:SC))

The min value of PS on $Prof$ is 1 which states that every professor has at least one student. Knowledge expressed by the min value is expressed by the following assertion:

(1) [**For all** x:$Prof$][**For some** z:$Student$]$< z, x >$:PS

Its negation:

(2) [**For some** x:$Prof$][**For all** z:$Student$]**not**$< z, x >$:PS

is necessarily false in the given domain. In addition, the assertion

[**For all** x:$Prof$][**For all** z:$Student$]**not**$< z, x >$:PS

is necessarily false. Any query that contains the assertion [**For all** z:$Student$]**not**$< z, x >$:PS where z is underconstrained and x occurs free defines a set that is necessarily empty in the given domain.

The internal representation for "a professor for a course, with no students" contains the elementary assertion **not**$< z, x >$:PS where z is a universally quantified underconstrained

variable and x occurs free. The set defined by the internal representation is necessarily empty and will therefore be excluded as a possible interpretation.

If z is not underconstrained, then an assertion which contains (2) is not necessarily false. For example, consider the query corresponding to the request "a professor with no graduate students".

$$[\textbf{For some } x{:}Prof][\textbf{For all } z{:}Student](\textbf{not}< z, x >{:}PS \textbf{ and } z{:}Graduate)$$

The set *Graduate* has been declared in Appendix A. z is not an underconstrained variable because (after application of DeMorgan's law) it occurs in two elementary assertions **not**$< z, x >{:}PS$ and **not**$z{:}Graduate$. The set defined by the query is not necessarily empty. Although there do not exist professors with no students, there may exist professors with no graduate students.

The condition for identifying a positive violation derives from the following observation: If x_j is an existentially quantified underconstrained variable, and the j^{th} min value of *assoc* is 1, then the assertion

$$not < x_1, \ldots, x_{j-1}, x_j, x_{j+1}, \ldots, x_n >{:}assoc$$

expresses no information in addition to that expressed by the min value.

Example 7.2. The min value of CP on $COURSE$ is 1 which states that every course is taught by at least one professor. The interpretation "a course taught by a professor, with no graduate students", could be more succinctly stated "a course with no graduate students" given the min value of CP on *Course*. A possible internal representation is

select x:*Course*

where [**For some** y:*Prof*] $< x, y >$:*CP* **and**

[**For all** z:*Student*] **not**$< z, x >$:*SC* **and** z:*Graduate*.

The part of the query [**For some** y:*Prof*] $< x, y >$:*CP* could be eliminated without changing its meaning. A sufficient condition for elimination of that portion without affecting the meaning of the query is that y is underconstrained.

In this subsection, we have considered some aspects of the problem of developing a heuristic to identify violations of min/max values due to prepositional phrase attachments. Further research is needed to determine how the system should respond when it has detected a violation.

7.3.5 Automatically Generating the Formal Database Query

Our design for a portable natural language interface is based on two conjectures each of which needs to be further explored. First, we have conjectured that knowledge generated in the process of translating a SET schema for the domain to a relational schema is useful for automatically translating the internal representation of a request to a formal DB query. Second, we have conjectured that general algorithms can be written to automatically generate the DB query based on an intermediate model, the NF^2 object model, between the SET model and relational model. A design strategy based on the availability of such algorithms results in a natural language interface that is easily portable between databases.

7.4 Main Contributions of the Thesis

This thesis has focused on the problem of transporting a natural interface to a new domain and database, and it reflects the excitement that prevails in both the areas of natural language understanding and database systems. In particular, we have applied current database theory and database system capabilities to solve problems in the area of natural language understanding.

The main contributions of the thesis to ongoing research are as follows:

1. The discovery that the process used for adapting the natural language interface to a new domain and database overlaps considerably with the process of designing the DB schema.

2. The design of an enhanced natural language interface based on sharing of knowledge, about the relationships in the domain, for DB schema design and NL understanding.

3. The development of heuristics based on the knowledge referred to in 2) for resolving ambiguities in natural language database requests, in particular, semantic ambiguity, word sense ambiguity, and post noun modifier attachment.

4. Specification of the relationship between the mathematical notions of mapping type and total/partial mapping and specific English language constructions.

Further research is expected, particularly with regard to (4). The richness of description that the notions of mapping type and total/partial mapping provide for modeling language phenomena has not been fully tapped by our approach. We have provided a general heuristic

that applies to several linguistic problems and several English language constructions. A challenging problem for the near future is to provide heuristics based on the notions of mapping type and total/partial mapping, but specialized to each linguistic problem and language construction.

Appendix A

Internal Representation of Sentences and Noun Phrases

The semantics of a sentence S is represented by an assertion (called the internal representation of S) that references vertices in the domain graph. The language for expressing internal representations is DEFINE [32]. In this appendix the language DEFINE is reviewed and internal representations for sentences and noun phrases are illustrated by example.

Review of the Language DEFINE

A set is defined in the language in the context of a SET schema such as the one illustrated by the domain graph for the university domain. The format of a definition is

< *setname* > **def**

select < *parent set* + *variable declaration* >

where < *assertion* >

Each occurrence of a free variable in $<$ *assertion* $>$ must be declared in $<$ *parent set + variable declaration* $>$. The format of $<$ *parent set + variable declaration* $>$ is $< V1 : S1, V2 : S2, \ldots, Vn : Sn >$ where $V1, \ldots, Vn$ are terms and $S1, \ldots, Sn$ are set names. A *term* is a constant, a variable, or a tuple of terms. The ":" stands for set membership. $V1 : S1$ means $V1$ is a member is $S1$. The set defined by such a $<$ *parent set + variable declaration* $>$ is a subset of the Cartesian product of $(S1 \times S2 \times \ldots \times Sn)$. Elementary assertions are of the form $V : S$ or $V1 = V2$ where $V, V1$ and $V2$ are terms and S is a set name. Quantified assertions are of the form $[QT : S] <$ *assertion* $>$ where Q is a quantifier 'For some' or 'For all', T is a term, and S is a set name.

Example:

Comp-Sci-Stud **def**

(**select** x: *Student*

where [For some y: *Course*] ($< x, y >$:SC and $< y$,'CPSC101'$>$:CN))

Comp-Sci-Stud is the set of students who take the course CPSC101. Entities are denoted by "surrogates" in a database. Comp-Sci-Stud is actually the set of surrogates that denote students who take at least one computer science course.

The language permits a set declaration to be embedded within a query. The embedded defined set is given a name which may be used within the query to refer to the embedded set, but the name is meaningful only during execution of the query. Embedded set declarations appear as a list of parenthesized set declarations following the query each of the form

(< *setname* > **def**

select < *parent set* + *variable declaration* >

where < *assertion* >)

as before. *setname* is the name of the embedded defined set. An embedded declaration may itself contain embedded declarations.

Functions in the language DEFINE

Given an association (for example SC in the university domain) and a variable y which is bound to a member of *Student*, the expression $< y : SC :>$ refers to the set of surrogates of courses taken by the student denoted by the surrogate to which y is bound. If y is bound to a member of *Course* then $<: SC : y >$ is the set of (surrogates of) students in the course denoted by the surrogate to which y is bound.

Examples

In this section, sample sentences and noun phrases and their internal representations are illustrated.

The internal representations for NL requests are represented using an assertion form of a query. The **select** part of a query is omitted in its assertion form and explicit existential quantification is given for all of the variables which occur free in *assertion*. Embedded set declarations in a query are not altered in the assertion form.

Suppose that a graduate student is a student who takes a course numbered 500 or greater.

Graduate **def**

select x:*Student*

where [**For some** y:*Course*](< x, y >:*SC* **and**

(<< y:*NUM*: >:*GREATER*:500 >**or** << y:*NUM*:>:*EQUAL*:500 >))

GREATER, *EQUAL*, and *NUM* are functions. When y is bound to a member of *Course*, < y:*NUM*:> is the number of the course. << y:*NUM*:>:*GREATER*:500 > is true if < y:*NUM*:> is greater than 500.

The assertion form for the set *Graduate* which has been defined in Chapter 4 is:

[**For some** x:*Student*][**For some** y:*Course*] (< x, y >:*SC* **and**

(<< y:*NUM*:>:*GREATER*:500 >**or** << y:*NUM*:>:*EQUAL*:500 >))

The internal representation of a sentence has the form

assertion (set_1 **def** ...) ... (set_n **def** ...).

where the set_i denote the arguments of the verb. A possible internal representation for the sentence "professors teach courses" is

[**For some** x:*Prof*][**For some** y:*Student*][**For some** z:*Course*]

(< y, z >:*SC* **and** < z, x >:*CP*).

The internal representation states that a professor teaches a course if the professor is assigned to teach the course and there exists at least one student enrolled in the course. The arguments of the verb are simple NPs in that they do not themselves contain embedded NPs or other modifiers. The sets *Prof* and *Student* that denote the arguments of the verb

are primitive sets declared as part of the SET schema. In general, previously declared sets will not be available to represent the meaning of the arguments of the verb.

The internal representation of an NP is an assertion of the same form as the internal representation for a sentence. The following examples illustrate internal representations for different types of noun modifiers:

Adjectives

A possible internal representation for the phrase "a graduate student" is

[For some x:*Student*](*x*:*Graduate*).

Since the declaration is previously given as part of the set schema, it is not given as part of the internal representation.

Genitives

An internal representation for the phrase "Dr. Lee's Students" has been given in Chapter 4, page 104.

Modifying Phrase

A possible internal representation for the NP "students in the Computer Science department" is:

[For some x:*Student*][**For some** y:*Dept*] ($< x, y >$:SD **and** y:CS_Dept)

(*CS_Dept* **def**

select x:*Dept*

where (< x,'Computer Science'>:DN))

Ordinals

The following example illustrates the internal representation for another type of modifier, the ordinal modifier, which defines an objects position in a series (e.g., first, tenth, hundredth). If the NP "the first CPSC 101 student" means the student who received the highest grade in CPSC 101, then an internal representation for the NP might be:

[**For some** x:$CPSC\_Stud$][**For some** y:$Course$] (< x, y >:$First\_Stud$)

($CPSC\_Stud$ **def**

select x:$Student$

where [**For some** y:$Course$]

(< x, y >:SC **and** < y,'CPSC101'>:CN)

($First\_Stud$ **def**

select < st, cr >:$Student \times Course$

where [**For some** gr:$Grade$] (< gr, < st, cr >>:SCG **and**

[**For all** gr':$Grade$] [**For all** st':$Student$]

(< gr', < st', cr >>:$SCG \supset gr' \leq gr$)))

Appendix B

Procedural Semantics for the University Domain

Semantic rules specify how the internal representation is built from the parse tree for a sentence. In this appendix a collection of rules is given for generating internal representations for some of the NL requests that arise in the university domain. The rules are based on Woods' procedural semantics approach [95]. The action of the semantic interpreter is also illustrated by showing how it applies the rules to a particular sentence to produce an internal representation. The rules given in this appendix are intended for illustration. In a working system many more rules would be needed to handle the variety of requests that would be presented to the system.

B.1 Defined Sets Referenced by Semantic Rules

The rules reference base sets of the University domain as well as a number of sets defined in terms of the base sets. The defined sets are *Avg_Student* which is the set of average students, *Graduate* which is the set of graduate students, and *CourseAvg* which is an association that gives for each student, the average in his or her courses.

The set Graduate has been defined in Appendix A. Definitions for the others are given here. For definition of *Avg_Student* we assume that an average student is one who has an average of B in his or her courses:

Avg_Student **def**

select Y:Student

where <Y,'B'>:StudAvg

(StudAvg **def**

select X:Student, Z:Grade

where <X:SG:>:AVERAGE:Z)

(SG **def**

select X: Student, Z:Grade

where [**For some** Y:Course]

<<X,Y>,Z>:SCG))

The association StudAvg associates with a student the average of grades obtained in his or her courses. When X is bound to a member of Student, <X:SG:> is a list of the grades

obtained by X in all of X's courses. The list may contain duplicates since a student may receive the same grade in more than one course. AVERAGE is a system declared function. It is a subset of (GradeSet X Grade) where the members of Grade are 'A', 'B', 'C', 'D', ... and the members of GradeSet are sets of grades.

The association CourseAvg associates with a course the average of grades obtained by students in the course.

CourseAvg **def**

select X:Course, Z:Grade

where <X:CG:>:Average:Z

(CG **def**

select X:Course, Z:Grade

where [**For some** Y:Course] <<Y,X>,Z>:SCG)

B.2 Semantic Rules for the University Domain

The rules refer to partial tree structures (as illustrated in Figures B.1 and B.2) which are matched to subtrees of the parse tree. An example of a rule follows:

1 - (G8: (1) = Student) and

2 - (G10: (1) = in and Dept((2)))

$\Longrightarrow$ [For some <2-2>:Dept] <X,2-2>:SD

The numbers in parentheses in the semantic rules match the numbers in parentheses in the partial tree structures. Arguments for the predicates are specified using a pair of

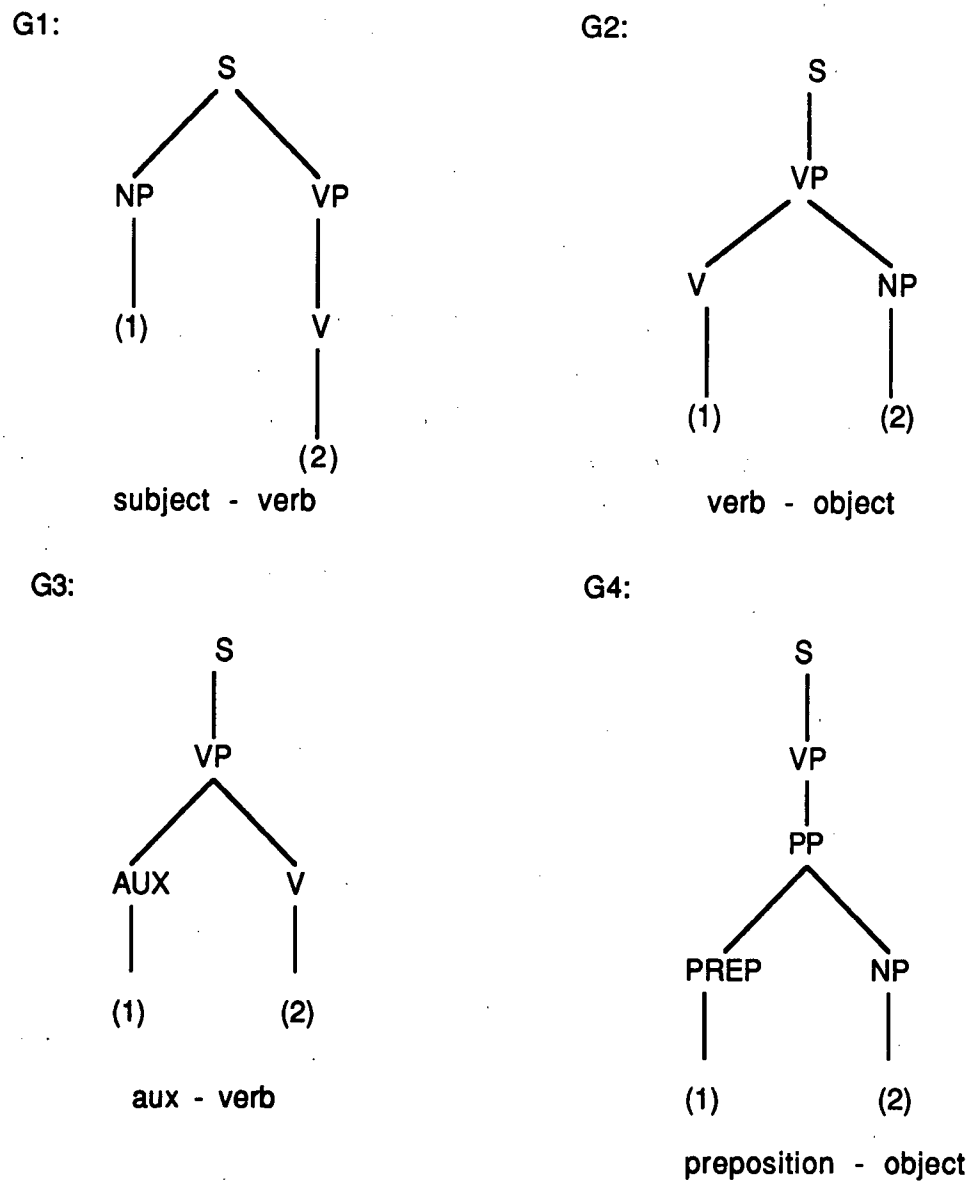


Figure B.1: Partial Tree Structures for Use by S-rules

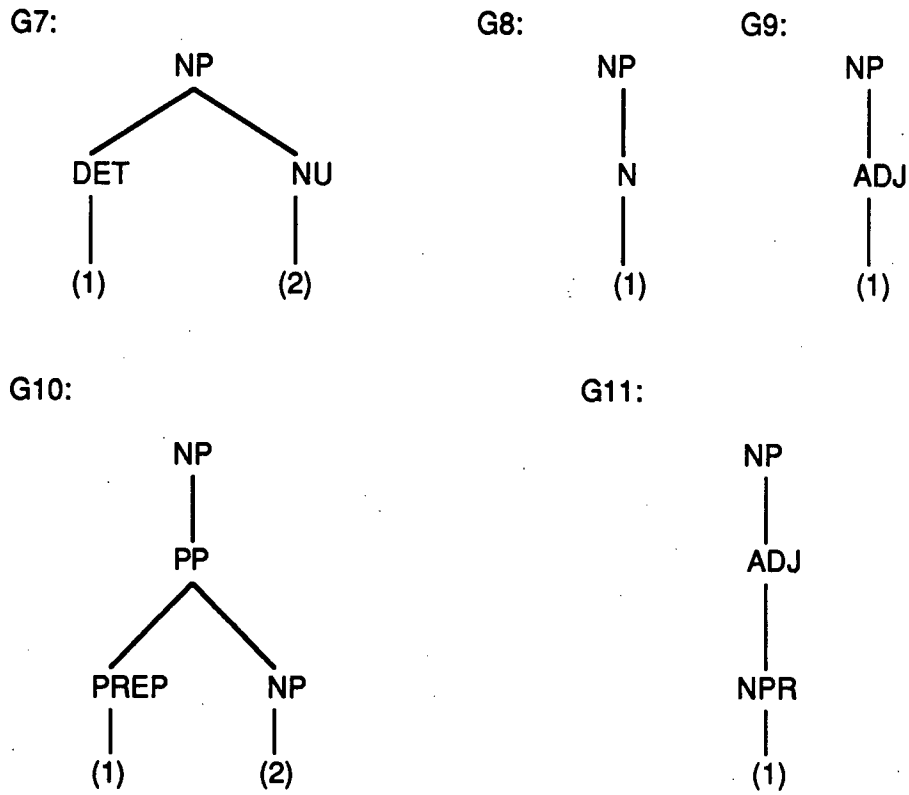


Figure B.2: Partial Tree Structures for Use by D, N, and R-rules

numbers <n1,n2> where n1 refers to a numbered part of the left side of the rule and n2 is one of the parenthesized numbers in that part. A number of type checking predicates are assumed to be available. Examples include Student(x), Course(x), Dept(x), Dname(x). There is one type checking predicate for each primitive set in the domain.

Conditions in the semantic rules are expressed in terms of the type checking predicates and the relation '='. The condition "(n)=W" is true if string W is identical to the terminal string of the subtree of the parse tree rooted at node n. This is to say that node n dominates the string W. The condition P((n)) is true if n denotes an object of type P.

The conditions on the left-hand side of a rule are numbered, and each refers to exactly one partial tree structure. At any given moment the semantic interpreter is working on a given node in the parse tree which is called the root. For a condition to be true, the partial tree structure identified by it must match the subtree of the parse tree starting at the root, and its subconditions must be true of the matched subtree of the parse tree.

Condition 1 of the given rule refers to the partial tree structure labeled G8. It is satisfied if G8 matches the parse tree starting at the root, and the node in the parse that matches the node labeled (1) in G8 dominates the string "student". A rule is applicable if all of its conditions are satisfied.

The semantic rules used here differ from Woods' in the right hand side which contains a query fragment expressed in a set notation (the language DEFINE) rather than a functional calculus. The left-hand part of our rules is still expressed in a predicate calculus which may cause confusion. For example, Student((1)) on the left and <1>: Student on the right have the same meaning but serve different purposes.

S-Rules for the University Domain

S1

1- (G1: Student((1)) and ((2)=enroll or (2) = register)) and

2- (G3:(1) = be and ((2)=enroll or (2)=register)) and

3- (G4:(1) = in and Course((2)))

$\Longrightarrow$ <1-1, 3-2>: SC

e.g.s.

"Jones is enrolled in CPSC 101"

"Jones is registered in CPSC 101"

S2

1- (G1: Prof((1)) and ((2) = teach or (2) = instruct) and

2- (G2: ((1) = teach or (1) = instruct) and Course((2)))

$\Longrightarrow$ <2-2, 1-1>: CP

e.g. "Dr. Lee teaches CPSC 101"

S3

1-(G1: Course((1)) and ((2)) = teach or (2) = instruct and

2-(G3: ((2) = teach or (2) = instruct) and (1) = be) and

3-(G4: (1) = by and Prof((2))

$\Longrightarrow$ <1-1, 3-2>: CP

e.g.s. "CPSC 101 is taught by Dr. Lee"

S4

1-(G1: Student((1)) and ((2) = receive or (2) = earn or (2) = obtain or (2) = get)) and

2-(G2: (4) = receive or (1) = earn or (1) = obtain or (1) = get) and Grade ((2))) and

3-(G4: (1) = in and Course((2)))

$\Longrightarrow$ <2-2, <1-1,3-2>>: SCG

e.g.s.

"Smith received A in CPSC 101"

"Smith in CPSC 101 received A"

Note: An order on the prepositional and verb phrases is not specified by rule S4. To impose an order a different tree structure from G4 would be needed which includes a PP for each of the direct and indirect objects.

S5

1-(G1: Student((1)) and ((2) = take or (2) = study) and

2-(G2: ((1) = take or (1) = study) and Course((2)))

$\Longrightarrow$ <1-1,2-2>:SC

e.g. "Jones takes CPSC 101"

Determiner Rules (D-Rules)

D1

1-(G7: ((1) = some or (1) = a or (1) = any) and (2) = SG)

$\Longrightarrow$ ([**For some** X: $\triangle$] $\triangledown$)

e.g.s. "any student enrolled in CPSC 101"

"some professor in the computer science department"

D2

1-(G7: ((1) = each or (1) = every or (1) = all) and (2) = SG)

$\Longrightarrow$ ([**For all** X:$\bigtriangledown$] $\bigtriangleup$)

e.g.s.

"all students enrolled in CPSC 101"

"every professor in computer science"

D3

1-(G7: (1) = no and (2) = SG)

$\Longrightarrow$ (**not** [**For some** X: $\bigtriangledown$] $\bigtriangleup$)

e.g. "no student in CPSC 101"

D4

1-(G7: (1) = **not** every and (2) = SG)

$\Longrightarrow$ (**not** [**For all** X: $\bigtriangledown$] $\bigtriangleup$)

e.g. "Not every student taught by Dr. Lee"

The symbol $\bigtriangledown$ in the D-rules will be replaced by a set name resulting from an N-rule. The symbol $\bigtriangleup$ will be replaced by an assertion resulting from R-rules (and applications of D-rules, N-rules, and R-rules to NPs lower down in the parse tree).

Noun Rules (N-Rules)

N1

1-(G8: (1) = student or (1) = pupil)

$\Longrightarrow$ Student

e.g. "a student"

N2

1-(G8: (1) = graduate)

$\Longrightarrow$ Graduate

e.g. " a graduate"

N3

1-(G8:(1) = professor or (1) = instructor)

$\Longrightarrow$ Prof

e.g. "a professor"

N4

1-(G8:(1) = course)

$\Longrightarrow$ Course

e.g. "a course"

N5

1-(G8: (1) = department)

$\Longrightarrow$ Dept

e.g. "a department"

Additions to Wood's Framework

The N-rules are applied differently depending on whether the head noun is universally of existentially quantified. For existential quantification, the symbol $\bigtriangledown$ is replaced by the right-hand side of the applicable N-rule. For universal quantification the NP-processor

generates the name of an embedded set which replaces ∇ in the working string, and the initial part of an embedded set declaration replaces the symbol Δ.

Example:

If the working string generated by the D-rules is [**For some** X: ∇] Δ where X is a variable that denotes the NP, then application of an N-rule will replace ∇ with the name of a set appearing on the right-hand side of the N-rule. If the working string is

[**For all** X: ∇] Δ

then the N-rules will produce a string such as

[**For all** X:S]

(S **def**

select X:T

where Δ)

where X is a variable generated by the NP-processor to represent the NP, S is the name of an embedded set declaration also generated by the NP-processor, and T is the name of a set which appears on the right-hand side of the applicable N-rule.

Restriction Rules (R-rules)

R1

1-(G8: (1) = student) and

2-(G9: (1) = average) and

3-(G10:(1) = in and Course((2)))

$\Longrightarrow$ <<<3-2>: CourseAvg:>, <X,3-2>>: SCG and Δ

e.g. "an average student in CPSC 101"

R2

1-(G8: (1) = student) and

2-(G9: (1) = average)

$\Longrightarrow$ X:Avg_Student

e.g. "an average student"

R3

1-(G8: (1) = student) and

2-(G9: (1) = graduate)

$\Longrightarrow$ X: Graduate

e.g. "a graduate student"

R4

1-(G8: (1) = student) and

2-(G10: (1) = in and Dept((2)))

$\Longrightarrow$ [**For some** <2-2>: Dept] <X,2-2>: SD and Δ

e.g. "a student in the computer science department"

R5

1-(G8: (1) = department) and

2-(G9: DName ((1)))

$\Longrightarrow$ <X,2-1>:DN

e.g. "the computer science department"

R6

1-(G8: (1) = professor) and

2-(G10: (1) = in and Dept((2)))

$\Longrightarrow$ [**For some** <2-2>: Dept] <X,2-2>: PD and $\triangle$

e.g. "a professor in the computer science department"

R7

1-(G8: (1) = computer science course

2-(G9: CName ((1)))

$\Longrightarrow$ X:CPSC_course

e.g. "a computer science course"

The symbol X in the R-rules stands for the variable that the NP-processor has created for the noun phrase which it is looking at when it applies the R-rule.

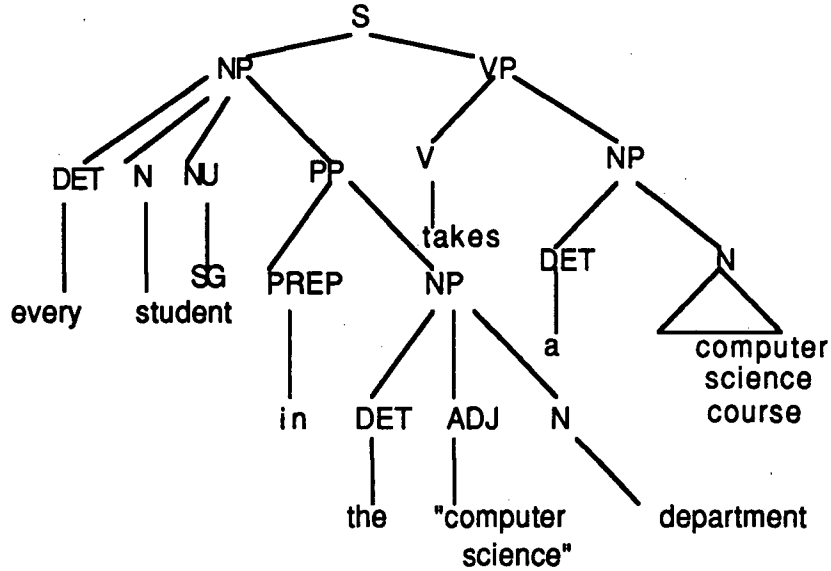


Figure B.3: Parse Tree for the Sentence "Every student in the computer science department takes a computer science course."

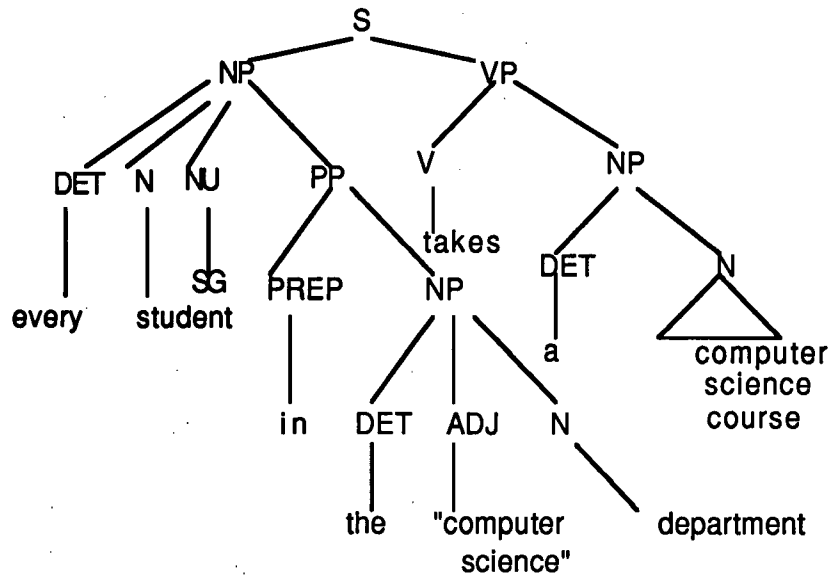


Figure B.3: Parse Tree for the Sentence "Every student in the computer science department takes a computer science course."

given in Figure B.3.

The S-processor begins with a working string $\triangle$. The NP-processor is called to process the quantified noun phrase "every student in the computer science department". The NP-processor applies the D-rules, N-rules and R-rules in succession. $\triangle$ in the working string is replaced by rule D2 resulting in the new working string ([**For all** X1: $\bigtriangledown$] $\triangle$). X1 is a new variable created by the NP-processor for the noun phrase. Next the N-rules are applied. The result of applying rule N1 is the string

[For all X:S]

(S def

select X: Student

where $\triangle$)

Finally, the R-rules are applied. Rule R4 substitutes $\triangle$ in the working string producing

The R-rules (R5) substitute Δ in the working string producing

[**For all** X:S]

(S **def**

select X: Student

where [**For some** Y:Dept] <X,Y>: SD and <Y, 'Computer Science'>:DN)

The NP-processor is called on the NP "a computer. science course" producing

[**For some** Z: Course] Z:CPSC-Course

by applying D1, N4. and R7.

The S-processor applies S5 substituting Δ in the working string to produce

[**For all** X:S] [**For some** Z:Course]

(<X,Z> SC and Z:CPSC_Course)

(S **def**

select X: Student

where [**For some** Y: Dept] <X,Y>:SD and <Y,'Computer Science'>:DN)

Appendix C

A Transportable Natural Language Interface

A natural language interface based on the architecture illustrated in Figure 2.5 is available at UBC, and it has been designed to be portable. The specific architecture of this system is assumed for interfacing an NLI with a database system. It was designed and implemented by Steve White in 1985 as part of his master's thesis work [93]. The system is named ALPS (automatic language processing system). This appendix describes ALPS and compares it with the TEAM system [36] which is another transportable NL interface that has been recently described in the literature.

Domain portability is achieved in ALPS by separating domain independent and domain dependent information. There are two sources of domain independent information: the general dictionary, and the global dictionary. Both dictionaries are provided as part of the initial system before it is adapted to any particular domain. The general dictionary contains

syntactic descriptions of words that appear frequently in database requests independent of the domain. These include pronouns such as who, what, me, you, that, which and command verbs such as show, print, and count. Grosz et al. [36] refer to these types of lexical entries as *closed class*. Semantic information for some nouns, question pronouns (who, what), and verbs is also provided in the general dictionary. The global dictionary contains syntactic information for words that are widely applicable to many different domains (e.g.,words for units of measure - foot, pound, kilogram). Grosz et al. refer to such words and domain dependent words as *open class*. The meaning of open class words is usually dependent on the domain. However, since some open class words occur so frequently it is worth providing information for them in the initial lexicon. Syntactic definitions for those words that occur in the given domain are automatically extracted from the global dictionary when the system is started up. Thereafter, the global dictionary is not referenced.

Domain dependent information includes a domain schema which provides syntactic, semantic, and linguistic information about the domain, an inverted index which is a table of most of the words in the database together with their syntactic category and the database fields to which they belong, and the database schema library which includes syntactic and semantic definitions of words that are commonly used in requests against databases.

A dictionary (called the active domain dictionary) is compiled from the domain independent and domain dependent information when the system is started up. The active domain dictionary serves as the lexical dictionary during operation of the system. The general dictionary, inverted index, and database schema are not referenced during operation of the system.

All information required to adapt the system to a new domain is specified as data. No programming changes are required. Data model portability is achieved in the standard way by using a logical form that is data model independent. No changes to the syntactic or semantic components are required to adapt the system to a new data model.

The parser is based on Wood's specification [96], and it uses an ATN grammar for English developed by Winograd [94]. The grammar is a syntactic grammar enhanced with semantic routines for resolving syntactic ambiguity. The parser reads the global dictionary and the active domain dictionary during the parse.

ALPS is particularily good at handling compound proper nouns. When a proper noun is parsed the system will parse as many subsequent words as possible which might be part of a compound proper noun. These words include abbreviations, initials, proper nouns, and the words 'of' and 'and'. This string of words is then matched against proper nouns that occur in the database. This is the reason that the active domain dictionary is accessed during the parse. If an exact match is found then the string of words is parsed as a compound proper noun. If a match is not found then the string is checked against entries in the global dictionary to see if it (or its components) occur there as proper nouns. If they do then the string will be assumed to be an unknown proper noun, and an appropriate response will be given to the user.

The semantic interpreter uses verb frames which permit the identification of the semantic roles that the syntactic constituents of the parse play with respect to the main verb. The definition of a verb consists of a *verb frame* and optionally some information on how relations are to be joined together. A verb frame specifies which relations and columns can serve

as argument types of the verb. For example, the direct object of the verb 'teach' may be specified to be an entity of the *Student* relation, and modifiers for the verb may be specified to come from the *Rating* field of the *Teacher* relation.

When a verb frame specifies that different relations can provide values for the verb's arguments, it must also specify how the relations are to be joined together. This information is expressed as a collection of join conditions. For example, if the subject of the verb 'teach' is an entity from the *Teacher* relation and the direct object is as given previously then the join information may be *Teacher.Name=Student.Teach* where $Name$ and $Teach$ are column names. The given statement indicates a join of the $Teacher$ relation on $Name$ with the *Student* relation on $Teach$.

The structure of verb frames is similar to Fillmore's case grammar [27, 26] except that the argument types are syntactic categories (subject, direct object) rather that Filmore's cases (agent, beneficiary). The motivation for choosing syntactic categories over the conventional cases is to permit a DBA without a strong background in linguistics to be able to adapt the system to a new domain.

The language for expressing the formal database query is the relational query language SQL [78]. Database portability is improved by the decision to use SQL.

The system does not have a query interpreter. Natural language requests are accepted by the system and the corresponding formal database queries are logged in a file. For testing the system each query in the log file was executed by hand using the terminal interface to the relational database system ORACLE which supports the SQL query language. In ALPS the user performs the job of the query interpreter. Note that the formal and actual database

queries are identical in this framework.

The main features of the ALPS system especially those that contribute to the transportability of the system have been described in this section. The section is concluded with a comparison of ALPS and the TEAM system which has been described in Section 3.7.

TEAM provides a knowledge acquisition module that engages the DBA in a dialog to obtain knowledge that it needs for answering questions. TEAM knows when it has acquired a sufficient amount of knowledge. In ALPS semantic information is represented and presented to the system as a collection of LISP forms (not very user friendly). The DBA decides when the system has acquired enough knowledge.

TEAM is particularly good at handling quantifiers. The language for expressing the logical form includes special quantifiers for definite determiners (all, the) and question determiners (what, which). Six different heuristics are used to determine the scope of quantifiers. ALPS recognizes question determiners but not definite determiners. For a sentence such as the following

Show me a student with an A average in Computer Science

the system will print out all students with an A average in computer science. The definite determiner *a* is not being recognized. Complicated quantifiers including *not* are not handled in ALPS, and there are no heuristics for determining the scope of quantifiers.

ALPS is particularly good at recognizing compound proper nouns in the input and responding appropriately. The handling of compound proper nouns does not appear to be a strong point of TEAM.

In both ALPS and TEAM the interpretation of a natural language request is independent of the context of the dialog. Both systems handle only a limited range of conjunction. Both implement the syntactic and semantic components as separate modules that run in parallel.

Both systems employ the technique of separating domain independent from domain dependent information to improve domain portability. Both systems use a logical form that is data model independent to achieve data model portability.

TEAM uses an elaborate sort hierarchy which represents set-subset relationships between monadic sort predicates. A large amount of the sort hierarchy is built into the natural language interface, but the DBA may also add new nodes to the sort hierarchy. A simple two level sort hierarchy is built into ALPS, and it cannot be expanded by the DBA. TEAM separates both logically and physically (using separate modules) between the conceptual schema (which includes the sort hierarchy, information about non-sort predicates, and pragmatic information) and the database schema. ALPS makes a logical separation between the two components but not a physical separation.

The two systems appear to be more similar than different with respect to their basic paradigms for natural language processing. The designers of TEAM state that the processes that TEAM uses for reponding to natural language requests are similar to those of many other systems. What makes TEAM different from the others is its focus on a careful modular design that permits maximum generality. The use of a semantic grammar is therefore ruled out. A semantic grammar has also been ruled out by the designer of ALPS. Also ruled out is the use of the database query language for expressing the logical form. The designers of TEAM see this as a mixing together of the meaning of a request with a procedure for

retrieving the answer from the database. In ALPS a separate language is used for expressing the meaning of a request, but the language is similar to the database query language SQL. In ALPS natural language requests are mapped onto a rather small subset of SQL in which there are few procedural components. Even though the language for the logical form in ALPS is similar to a subset of the database query language, there is a recognition of the need to separate the meaning of a request from the procedure that answers the request. Different modules implement the transformation from the natural language request to the logical form and from the logical form to the database query. If the system were to be modified to handle a broader range of natural language constructions then the language for the logical form and the database query language would begin to diverge. It has been noted in the introduction to this appendix that the language for the logical form in ALPS is not independent of the database schema as it is in TEAM.

Many of the ideas implemented in TEAM for improving transportability by using a careful modular design have been implemented in whole or in part in ALPS. The proposed extensions to ALPS will result in a system that is distinguished from TEAM and other systems by its focus on the use of database system capabilities to improve the transportability of the system. The approach of using modularity to improve transportability and that of using the database system to improve transportability are complementary rather than competing approaches.

Bibliography

[1] Hiyan Alshawi and Robert C. Moore. *Feasibility Study for a Research Programme in Natural-Language Processing.* SRI International Cambridge Computer Science Research Center, Cambridge, England, August 1986. SRI Project ECC-1437. Contract No. ALV/CONS/IKBS/026.

[2] Douglas E. Appelt. *Planning English Sentences.* Computational Linguistics, 1985.

[3] Bruce W. Ballard, John C. Lusth, and Nancy L. Tinkham. Ldc-1: A transportable, knowledge-based natural language processor for office environments. *ACM Transactions on Office Information Systems*, 2(1):1–25, 1984.

[4] D.S. Batory, T.Y. Leung, and T.E. Wise. Implementation concepts for an extensible data model and data language. *ACM Transactions on Database Systems*, 13(3):231–262, 1988.

[5] C. Beeri, R. Fagin, and J.H. Howard. A complete axiomatization for functional and multivalued dependencies. In *Proceedings ACM SIGMOD Conference*, 1977.

[6] P.A. Bernstein and C.W. Chiu. Using semijoins to solve relational queries. *J. ACM*, 28(1):25–40, 1981.

[7] Allan David Booth. Designing a portable natural language database interface. Master's thesis, University of British Columbia, Department of Computer Science, Vancouver, B.C., Canada, 1983.

[8] Roger A. Browse. A knowledge identification phase of natural language analysis. Master's thesis, University of British Columbia, Department of Computer Science, Vancouver, B.C., Canada, 1977.

[9] Thomas Burns, Elizabeth Fong, David Jefferson, Richard Knox, Leo Mark, Christopher Reedy, Louis Reich, Nick Roussopoulos, and Walter Truszkowski. Reference model for dbms standardization. report from the database architecture framework task group of the ansi/x3/sparc database system study group. *SIGMOD Records*, March 1986.

[10] N. Cercone and G. McCalla. Accessing knowledge through natural language. *Advances in Computers*, 25:1–99, 1986.

[11] Eugene Charniak and Drew McDermott. *Introduction to Artificial Intelligence.* Addison-Wesley, 1985.

[12] P.P. Chen. The entity-relationship model - toward a unified view of data. *ACM Transactions on Database Systems*, 1(1):9–36, 1976.

[13] Noam Chomsky. *Syntactic Structures.* Mouton and Co, The Hague, 1957.

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

[15] E.F. Codd. A relational model of data for large shared data banks. *Communications of the ACM*, 13(6):377–387, 1970.

[16] E.F. Codd. Extending the database relational model to capture more meaning. *ACM Transactions on Database Systems*, 4(4):397–434, 1979.

[17] COBOL Committee. Codasyl. *Journal of Development*, 1978. Department of Supply and Services. Canadian Federal Government. Hull, Quebec, Canada.

[18] P. Dadam, K.Kuespert, F. Andersen, H. Blanken, R. Erbe, J. Guenauer, V. Lum, P.Pistor, and G. Walch. A dbms prototype to support extended nf^2 relations: an integrated view on flat tables and hierarchies. In Carlo Zaniolo, editor, *Proceedings of the International Conference on Management of Data*, pages 356–367. ACM SIGMOD, 1986.

[19] Fred J. Damerau. Problems and some solutions in customization of natural language database front ends. *ACM Transactions on Office Information Systems*, 3(2):165–184, 1985.

[20] James Edward Davidson. Interpreting natural language database updates. Technical Report STAN-CS-87-1152, Department of Computer Science, Stanford University, Stanford, CA, 1987.

[21] Bipin C. Desai, Richard J. Pollock, and Philip J. Vincent. A natural language interface to a multiple databased office information system. *SIGOIS Bulletin*, 9(4):19–33, 1988.

[22] Jim Diederich and Jack Milton. New methods and fast algorithms for database normalization. *ACM Transactions on Database Systems*, 13(3):339–365, 1988.

[23] Samuel S. Epstein. Transportable natural language processing through simplicity - the pre system. *ACM Transactions on Office Information Systems*, 3(2):107–120, 1985.

[24] R. Fagin. Normal forms and relational database operators. *Proc. ACM SIGMOD Int. Conf. on Management of Data*, pages 153–160, 1979.

[25] R. Fagin. A normal form for relational databases that is based on domains and keys. *ACM Transactions on Database Systems*, 6(3), 1981.

[26] Charles J. Fillmore. The case for case. In E. Bach and R.T. Harms, editors, *Universals in Linguistics*, pages 1–88, New York, 1968. Holt, Rinehart, and Winston.

[27] Charles J. Fillmore. The case for case reopened. In P. Cole and J.M. Sadock, editors, *Syntax and Semantics 8: Gramatical Relations*, pages 59–81, New York, 1977. Academic Press.

[28] J.P. Fournier, P. Herman, G. Sabah, and A. Vilnat. Processing of unknown words in natural language question answering. *Computational Linguistics*, 4(2):205–211, 1988.

[29] Frey Associates, Inc., Chestnut Hill Road, Amherst, NH. *THEMIS Management Information System*, 1984.

[30] Paul C. Gilmore. Natural deduction based set theories: A new resolution of the old paradoxes. *J. Symb. Logic*, 51(2), 1986.

[31] Paul C. Gilmore. Concepts and methods for database design. Technical Report TR87-31, Department of Computer Science, University of British Columbia, 1987.

[32] Paul C. Gilmore. A foundation for the entity relationship approach: How and why? In *Proceedings of the 7^{th} International Conference on Entity-Relationship Approach*, New York, 1987.

[33] Paul C. Gilmore. Personal correspondence. Department of Computer Science, University of British Columbia, 1987.

[34] Robert C. Goldstein. *Database Technology and Management.* John Wiley and Sons, 1985.

[35] Ralph Grishman. *Computational Linguistics.* Cambridge University Press, 1986.

[36] Barbara J. Grosz, Douglas E. Appelt, Paul A. Martin, and Fernando C.N. Pereira. Team: An experiment in the design of transportable natural-language interfaces. In *Artificial Intelligence*, volume 32, pages 173–243. Elsevier Science Publishers B.V. (North-Holland), 1987.

[37] Barbara Jean Grosz. The representation and use of focus in a system for understanding dialogs. In *Proc. of the Fifth International Joint Conference on Artificial Intelligence*, pages 67–76, Cambridge, Mass., 1977.

[38] Barbara Jean Grosz. Focusing in dialog. In David L. Waltz, editor, *TINLAP-2: Theoretical Issues in Natural Language Processing*, pages 96–103, University of Illinois at Urbana-Champaign, 1978.

[39] Carole D. Hafner and Kurt Godden. Portability of syntax and semantics in datalog. *ACM Transactions on Office Information Systems*, 3(2):141–164, 1985.

[40] Gary Hall, WoShun Luk, and Nick Cercone. Disambiguating queries using dependency graphs. Technical Report LCCR TR 87-7, School of Computing Science, Simon Fraser University, Burnaby, British Columbia, Canada, 1987.

[41] L. Harris. Natural language data base query: Using the data base itself as the definition of world knowledge and as an extension of the dictionary. Technical Report TR77-2, Department of Mathematics, Dartmouth College, 1977.

[42] L. Harris. User-oriented database query with the robot natural language system. *Int. J. Man-Mach. Stud.*, 9:697–713, 1977.

[43] L. Harris. The robot system: natural language processing applied to database query. In *Proc ACM Annual Conference*, pages 165–172, New York, 1978. ACM.

[44] Philip J. Hayes, Peggy M. Anderson, and Scott Safier. Semantic caseframe parsing and syntactic generality. In 23^{rd} *Annual Meeting of the Association for Compuational Linguistics*, pages 153–160, 1985.

[45] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum. Developing a natural language interface to complex data. *ACM Transactions on Database Systems*, 3(2):105–147, 1978.

[46] Graeme Hirst. *Anaphora in Natural Language Understanding: A Survey.* Berlin Heidelberg, 1981. Lecture Notes in Computer Science. Springer-Verlag.

[47] Graeme Hirst. *Semantic Interpretation and the Resolution of Ambiguity.* Cambridge University Press, 1987.

[48] Richard Hull and Roger King. Semantic database modeling: Survey, applications, and research issues. *ACM Computing Surveys*, 19(3):200–260, 1987.

[49] G. Jaeschke. Nonrecursive algebra for relations with relation-valued attributes. Technical Report TR 85.03.001, IBM Scientific Center, Heidelberg, West Germany, 1985.

[50] G. Jaeschke and H.J. Schek. Remarks on the algebra of non first normal form relations. In *Proc. ACM SIGART-SIGMOD Symp. on Principles of Database Systems*, pages 124–138, Los Angeles, Cal., 1982.

[51] Jurgen M. Janas. The semantics-based natural language interface to relational databases. In L. Bolc and M. Jarke, editors, *Topics in Information Systems, Cooperative Interfaces to Information Systems*, pages 143–188. Springer-Verlag, 1986.

[52] Karen Jensen and Jean-Louis Binot. Disambiguating prepositional phrase attachments by using on-line dictionary definitions. *Computational Linguistics*, 13(3-4):251–260, 1987.

[53] Candace E. Kalish and Matthew B. Cox. Porting an extensible natural language interface: A case history. In *Proceedings of AAAI*, pages 556–560, 1987.

[54] M. Kao, N. Cercone, and W. Luk. Providing quality responses with natural language interfaces: the null value problem. *IEEE Transactions on Software Engineering*, 14(7):959–984, 1988. also 3^{rd} IEEE International Data Engineering Conference, 1986.

[55] S. Jerrold Kaplan. Designing a portable natural language database query system. *ACM Transactions on Database Systems*, 9(1):1–19, March 1984.

[56] Samuel Jerrold Kaplan. *Cooperative Responses from a Portable Natural Language Data Base Query System.* PhD thesis, University of Pennsylvania, The Moore School of Electrical Engineering, Philadelphia, Pennsylvania, 1979.

[57] W. Kent. Fact-based data analysis and design. In C.G. Davis, S. Jajodia, P.A. Ng, and R.T.Yeh, editors, *Entity-Relationship Approach to Software Engineering*, pages 3–53. Elsevier Science Publishers B.V. North-Holland, 1983.

[58] A. Kumar and M. Stonebraker. The effect of join selectivities on optimal nesting order. *SIGMOD Record*, 16(1):28–41, 1987.

[59] B. Lindsay, J. McPherson, and H. Pirahesh. A data management extension architecture. In Umeshwar Dayal, editor, *Proceedings of the International Conference on Management of Data*, pages 220–226. ACM SIGMOD, 1987.

[60] J.W. Lloyd. *Foundations of Logic Programming.* Springer-Verlag, 1987.

[61] P. Lyngbeak and V. Vianu. Mapping a semantic data model to the relational model. In Umeshwar Dayal, editor, *Proceedings of the International Conference on Management of Data*, pages 132–142. ACM SIGMOD, 1987.

[62] David Maier and Jeffrey Ullman. Maximal objects and the semantics of universal relation databases. *ACM Transactions on Database Systems*, 8(1):1–14, March 1983.

[63] Leo Mark and Nick Roussopoulos. Metadata management. *IEEE-Computer*, pages 26–36, December 1986.

[64] Elaine Marsh and Carol Friedman. Transporting the linguistic string project system from a medical to a navy domain. *ACM Transactions on Office Information Systems*, 3(2):121–140, 1985.

[65] Kathleen R. McKeown. *Text Generation - Using Discourse Strategies and Focus Constraints to Generate Natural Language Text.* Computational Linguistics, 1985.

[66] Robert Mercer. *A Default Logic Approach to the Derivation of Natural Language Presuppositions.* PhD thesis, University of British Columbia, Department of Computer Science, Vancouver, B.C., Canada, 1987.

[67] A. Ola. Design of relational database schemas: The traditional dependencies are not enough. Master's thesis, University of British Columbia, Department of Computer Science, Vancouver, B.C., Canada, 1982.

[68] Judea Pearl. *Heuristics - Intelligent Search Strategies for Computer Problem Solving.* Addison-Wesley Publishing Company, 1984.

[69] Joan Peckham and Fred Maryanski. Semantic data models. *ACM Computing Surveys*, 20(3):153–189, 1988.

[70] Brian Philips, Michael J. Freiling, James H. Alexander, Steven L. Messick, Steve Rehfuss, and Sheldon Nicholl. An eclectic approach to building natural language interfaces. In 23^{rd} *Annual Meeting of the Association for Compuational Linguistics*, pages 254–261, 1985.

[71] Van Riemsdijk and Edwin Williams. *Introduction to the Theory of Grammar.* M.I.T. Press, 1986.

[72] Richard S. Rosenberg. *Approaching Discourse Computationally: A Review*, pages 10–83. Carl Hanser Verlag, 1980.

[73] M.A. Roth and K.A. Korth. The design of 1nf relational databases into nested normal form. In Umeshwar Dayal, editor, *Proceedings of the International Conference on Management of Data*, pages 143–159. ACM SIGMOD, 1987.

[74] H.J. Schek. A data model and query language for exodus. In *Proceedings of the International Conference on Management of Data*, pages 413–423. ACM SIGMOD, 1988.

[75] H.J. Schek and M. Scholl. An algebra for the relational model with relation -valued attributes. *Information Systems*, 11(2), 1986.

[76] Peter Sells. *Lectures on Contemporary Syntactic Theories.* Center for the Study of Language and Information, 1985.

[77] J.M. Smith and D.C.P. Smith. Data abstractions: Aggregation and generalization. *ACM Transactions on Database Systems*, 2(2):105–133, 1977.

[78] Standards Council of Canada, Mississauga, Ontario. *ANSI X3.135-1986 SQL Database Query Language*, 1986.

[79] Veda C. Storey. *View Creation: An Expert System for Database Design.* PhD thesis, University of British Columbia, Faculty of Commerce and Business Admin., 1986. ICIT Press.

[80] Veda C. Storey and Robert C. Goldstein. A methodology for creating user views in database design. *ACM Transactions on Database Systems*, 13(3):305–338, 1988.

[81] J.B. Sykes. *The Concise Oxford Dictionary of Current English.* Oxford University Press, 1987. First Edition 1911.

[82] Bozena Henisz Thompson and Frederick B. Thompson. Ask is transportable in half a dozen ways. *ACM Transactions on Office Information Systems*, 3(2):185–203, 1985.

[83] D.C. Tsichritzis and A. Klug. The ansi/x3/sparc dbms framework: Report of the study group on data base management system. *Information Systems*, 3:173–191, 1978.

[84] D.C. Tsichritzis and F.H. Lochovsky. *Data Models.* Prentice-Hall, Inc., Englewood Cliffs, N.J., 1982.

[85] Jeffrey D. Ullman. *Principles of Database Systems.* Computer Science Press, 1982.

[86] J.J. van Griethuysen, editor. *Concepts and Terminology for the Conceptual Schema and Information Base.* ISO/TC97/SC5/WG3-N695, 1982.

[87] J.A. Wald. *Problems in Query Inference.* PhD thesis, University of Saskatchewan, Dept. of Computational Science, Saskatoon, Saskatchewan, Canada, 1985.

[88] Joseph A. Wald and Paul G. Sorenson. Resolving the query inference problem using steiner trees. *ACM Transactions on Database Systems*, 9(3):348–368, September 1984.

[89] Joseph A. Wald and Paul G. Sorenson. Explaining ambiguity in a formal query language. *ACM Transactions on Database Systems*, 1989. To appear.

[90] D. L. Waltz. An english language question answering system for a large relational database. *Communications of the ACM*, 3(2):526–539, 1978.

[91] David H.D. Warren and Fernando C.N. Pereira. An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8(3-4):110–119, July-December 1982.

[92] W. Watt. Habitability. *Am. Documentation*, pages 338–351, 1968.

[93] Steven John White. A portable natural language database query system. Master's thesis, University of British Columbia, Department of Computer Science, Vancouver, B.C., Canada, 1985.

[94] Terry Winograd. *Language as a Cognitive Process*, volume 1. Addison-Wesley, 1983.

[95] W. Woods. Procedural semantics for a question-answering machine. In *Fall Joint Computer Conference*, pages 458–471, 1968.

[96] W. Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, 1970.

[97] W. Woods. Cascaded atn grammars. *Am.J. Comput. Linguist*, 6(1):1–12, 1980.

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

[99] William A. Woods. Knowledge representation. In Thomas C. Bartee, editor, *Expert Systems and Artificial Intelligence - Applications and Management*, pages 147–176. Howard W. Sams and Company, 1988.