An Intelligent Approach To Handling Imperfect Information In Concept-Based Natural Language Queries

VESPER OWEI
University of Oklahoma

Missing information, imprecision, inconsistency, vagueness, uncertainty, and ignorance abound in information systems. Such imperfection is a fact of life in database systems. Although these problems are widely studied in relational database systems, this is not the case in conceptual query systems. And yet, concept-based query languages have been proposed and some are already commercial products. It is therefore imperative to study these problems in concept-based query languages, with a view to prescribing formal approaches to dealing with the problems. In this article, we have done just that for a concept-based natural language query system that we developed. A methodology for handling and resolving each type of imperfection is developed. The proposed approaches are automated as much as possible, with the user mainly serving an assistive function.

Categories and Subject Descriptors: H.2.3 [**Database Management**]: Languages—*query languages*; H.5.2 [**Information Interfaces and Presentation**]: User Interfaces—*natural language, interaction styles, prototyping*; I.2.7 [**Artificial Intelligence**]: Natural Language Processing—*language generation, language parsing and understanding*

General Terms: Design, Human Factors, Languages, Theory

Additional Key Words and Phrases: Natural language interface, natural language query, concept-based query, conceptual query language, imperfect queries, missing information, incomplete information, inconsistency, semantically mismatched query, inexplicit query, elliptical query, anaphoric query, ambiguous query

1. INTRODUCTION

"Imperfect information is ubiquitous—almost all the information that we have about the real world is not certain, complete and precise. Thus to insist on studying just certain information, as has been the case in most work in databases, is to concentrate upon a small part of the whole problem" [Parsons 1996]. An extensive survey of the work done in the database and artificial intelligence communities on imperfect information is given in Parsons [1996]. Surely in an era where there is an increase in the use of databases (DBs) by DB experts and

Author's address: V. Owei, The George Washington University, Management Science Department (Information Systems), Monroe Hall, 2115 G Street, NW, Washington, DC 20052; email: Vesper@gwu.edu.

nonexperts, it is only to be expected that ill-formulated queries would become a very common occurrence. It is therefore imperative that query languages designed to support a broad range of users are functionally powerful enough to aid users in avoiding or resolving queries having imperfect information.

The problem with imperfect queries is that, depending on the nature of the imperfection, they may either not retrieve any data, because they cannot be processed, or retrieve erroneous data without the user realizing it. And where this happens in business applications, it could mean the use of erroneous data for business decisions or the introduction of severe inefficiencies in business processes. In either case, the effect could be disastrous for organizations. In this article, therefore, approaches are developed for handling imperfection in concept-based natural language queries. The approaches are specifically applied to the Conceptual Query Language with Natural Language (CQL/NL) that is presented in Owei [2000]. We examine what imperfection means in the context of concept-based query languages, and then determine and define the types of possible imperfect queries in CQL/NL. We then develop a methodology for handling and resolving each type of imperfection. The proposed approaches are automated wherever and as much as possible, with the user mainly playing an assistive role to the system.

Although techniques for dealing with imperfect information in queries and DBs have been primarily studied in the context of relational database systems, as we discuss in the next section on related research, this is not the case with other DB systems, especially concept-based DB systems, which are introduced and defined in Section 3. The claim that imperfect information in concept-based DB systems is not widely studied is made clearer in Section 4, where we introduce and define the types of query imperfections that are addressed in this article. Section 5 deals with semantically mismatched conceptual queries. Section 6 examines missing information in conceptual queries. Inexplicit conceptual queries are addressed in Section 7. Section 8 deals with ambiguity in concept-based queries. Additional discussion on imperfect information in database systems is given in Section 9. The article concludes in Section 9.

2. LITERATURE REVIEW

Early studies of handling imperfect information in information systems deal with trying to define and classify imperfect information. This is perhaps to be expected, since the nature and ramifications of the problem must be understood and clearly delineated before it can be solved.

Bonnissone and Tong [1985] argue that imperfect information is of three types, namely, uncertainty, incompleteness, and imprecision. In their taxonomy, incompleteness is seen as arising from the absence of a value and imprecision from the existence of a value that cannot be measured with suitable precision. Uncertainty ensues from constructing a subjective opinion about the truth of a fact, the certainty of which is not known. The studies in Bosc and Prade [1993] and Dubois and Prade [1988] go even further by making a finer distinction that includes vagueness and inconsistency as additional categories. Information is considered to be vague if it is fuzzily imprecise. For example, the predicate “tall”

is both fuzzy and imprecise, and thus viewed as vague. Inconsistency results from conflicting values. For example, the propositions "Joe weighs 100 lbs" and "Joe weighs 150 lbs" impart inconsistent information, if the term "Joe" refers to the same real world object.

The taxonomic categories are not disjoint. As discussed in Bosc and Prade [1993], information can be both uncertain and imprecise. The imprecise statement "Joe's weight is between 100 lbs and 150 lbs" may be uncertain, depending on the knowledge of the one making the statement. Information can also exhibit vagueness and inconsistency simultaneously. An example is the claim, "Joe is tall and short."

Much of the work on handling imperfect information in DBs has focused on dealing with incomplete information. The primary concern in this has been devoted to dealing with missing attribute values [Parsons 1996]. Grahne [1991] identifies a set of theoretical problems that must be addressed in resolving the issue of incomplete information in the context of relational DBs. These include:

—the meaning of incomplete information,
—the processing of incomplete information during query processing, and
—the extent to which incomplete information can be tolerated and yet make query processing possible.

Two types of incomplete information are identified in Grahne [1991]: unknown values, and values not applicable in the particular context. Codd [1975] and [1979], on the other hand, proposes a three-level logic ("true," "false," and "unknown") to deal with missing information in relations. This is the null substitution principle in Codd [1979]. Both Grahne and Codd argue that an incomplete relation represents a set of complete relations, since the null, that is, the unknown, can be removed by different substitutions.

A rather elegant solution approach to the problem of querying relations with nulls is discussed in [Imielinski and Lipski [1981] and [1984]: nulls are assumed marked, and nulls with the same mark represent the same unknown value. An entirely different approach is taken in Nijssen and Halpin [1989] by enforcing complete knowledge. In this approach, full information is recorded through the use of frequency constraints that ensure that all possibilities are known to the system. Derivation rules for unstored facts are not allowed. A detailed discussion of the various studies dealing with the treatment of null values in databases is given in Parsons [1996].

Some work has also been done to address the problem of imprecise information in relational DBs. Most of the work in this area uses fuzzy sets [Zadeh 1965] and fuzzy logic [Zadeh 1983a, b]. Different approaches are taken in the literature to applying fuzzy notions to this problem. One approach associates a fuzzy degree of membership with each tuple of a relation [Baldwin and Zhou 1984; and Dubois and Prade 1991]. Such a degree is viewed variously as the degree to which a tuple satisfies the relation to which it belongs [Baldwin and Zhou 1984] or as the degree of association between the elements of a tuple [Dubois and Prade 1991]. In a second approach, fuzzy similarity relations are used to measure the extent to which the elements of an attribute domain are

interchangeable [Parsons 1996; Buckles and Petry 1987, 1982]. Basically, this approach extends relational operations to produce fuzzy answers. The use of fuzzy inference mechanisms is a third approach [Leung et al. 1989]. In this approach, fuzzy production rules are coupled with a relational DB. In the resulting deductive expert DB system, the expert system component uses fuzzy production rules to answer queries. The DB is called only when it is required.

There has been relatively little work done on imperfect information in concept-based query languages. The reason for this may well lie in the fact that the development of concept-based query languages has essentially remained within the confines of research laboratories. Only a small number of them appear to be making an inroad into the commercial arena [Bloesch and Halpin 1997]. Others are still at the prototype level [Owei and Navathe 2001b; Owei 2000; Chan 1989; Berztiss 1993]. We next examine some of these existing concept-based query languages.

ConQuer-II [Bloesch and Halpin 1997] is a commercial concept-based query language based on the object-role modeling (ORM) paradigm [Halpin 1995, 1996; Halpin and orlowska 1992; Halpin and Proper 1995a, b]. ORM models applications in terms of the semantic roles played by objects and entities in relationships. ConQuer-II allows queries to be formulated via paths through the conceptual schema. Query paths are constructed from the semantic roles of objects and entities.

Vizla [Berztiss 1993] is a visual query language interface for the information control prototyping language SF [Berztiss 1986]. In Vizla, a database is abstracted as a collection of sets (entities) and functions that map from this collection of sets to auxiliary sets (attributes). Queries are formulated in Vizla by pointing to representations of functions, their domains and codomains, or subsets of the domains and codomains, and to various operators in a conceptual model of a database. The items selected in this way are displayed and assembled graphically in a workspace, or window, which is used in Vizla to reduce the cognitive burden query formulation imposes on end-users. It achieves this by allowing users to separate querying into sequences of small steps, save intermediate results of such sequences, and combine the intermediate results into final results. Ad hoc queries can therefore be formulated and processed in this manner.

A number of studies have been conducted either to motivate the development of concept-based query languages or to demonstrate their superiority over other query paradigms. A discussion of comparative studies arguing for concept-based query languages can be found in Chan et al. [1993] and Siau et al. [1995]. Both studies compare SQL and the concept-based DBQL Knowledge Query Language (KQL) [Chan 1989]. They show that users of concept-based query languages outperform SQL users: irrespective of time, the KQL users performed better than their SQL counterparts, with respect to query accuracy, query formulation time, and user confidence. Additional empirical studies suggesting the superiority of concept-based data retrieval approaches over other query approaches can be found in Batra [1993], Batra et al. [1990], Batra and Sein [1994], Jarvenpaa and Machesky [1989], and Owei et al. [2002]. All of these studies clearly point to the need for alternative query paradigms. Concept-based approaches are a viable option.

However, none of the concept-based languages cited above addresses imperfect information. An effort along this line is made by Wu and Ichikawa [1992]. They propose a system whereby incomplete query specifications are resolved at the conceptual schema level. A query graph (termed query net) of the specified query is formed and the missing information is handled by complementing the query net with semantic paths generated by a path-finding program. This approach is based on the argument that ill-formed queries and missing information queries at the conceptual level result from end-users' misconceptions about domain concepts. In the cooperative approach taken, the system in Wu and Ichikawa [1992] returns to the end-user all the instantiations of ill-formed subnets of the query graph, that is, the portions that are in need of complementing and correcting. This gives the user a precise view of the domain concepts.

Wald and Sorenson [1990] also deal with ambiguous conceptual queries posed on an entity-relationship involvement data model. In their study, ambiguity is viewed in terms of multiple underlying query paths at the logical level. Therefore, in their case, a query is considered to be ambiguous if it can be mapped to more than one logical query. They address the problem of inferring an unambiguous logical query from a given conceptual query. The approach taken is to use the ER schema to explain in English-like sentences the meaning of the paths of the resulting logical queries. The user is then required to choose which one, if any, is correct. In this article we define the ambiguity of conceptual queries in a broader sense, or more appropriately at a higher level, than is done in Wald and Sorenson [1990], namely, in the conceptual statement of the query itself, thus in the concept-based natural language expression of the query.

From the foregoing, it is clear that the work on imperfect information has been done almost exclusively in the context of relational DB systems. This is quite understandable, given the pervasiveness of relational DBs in society and the strong backing by commercial vendors. But as discussed above, more recently concept-based DB query languages have begun to emerge. Given the significance of imperfect information in information systems, we consider it imperative that this issue be examined in the context of concept-based DB query languages. The important question in this respect is therefore, "What does imperfect information mean in concept-based DBs and conceptual queries?" We examine this question in this article and discuss how we deal with imperfect information in the concept-based controlled natural language query system discussed in Owei [2000] and Owei et al. [1997].

3. PRELIMINARY ON CONCEPT-BASED QUERY LANGUAGES AND CQL/NL

Concept-based or conceptual query interfaces reduce the cognitive load in querying DBs by allowing users to directly use constructs from conceptual schemas [Batra 1993; Batra et al. 1990; Batra and Sein 1994; Chan et al. 1993; Owei 1994; Siau et al. 1995]. As exemplified in Chan et al. [1993], instead of specifying the relational condition, "Where s.sno = sp.sno and sp.pno = p.pno," concept-based interfaces would allow for a more natural specification like

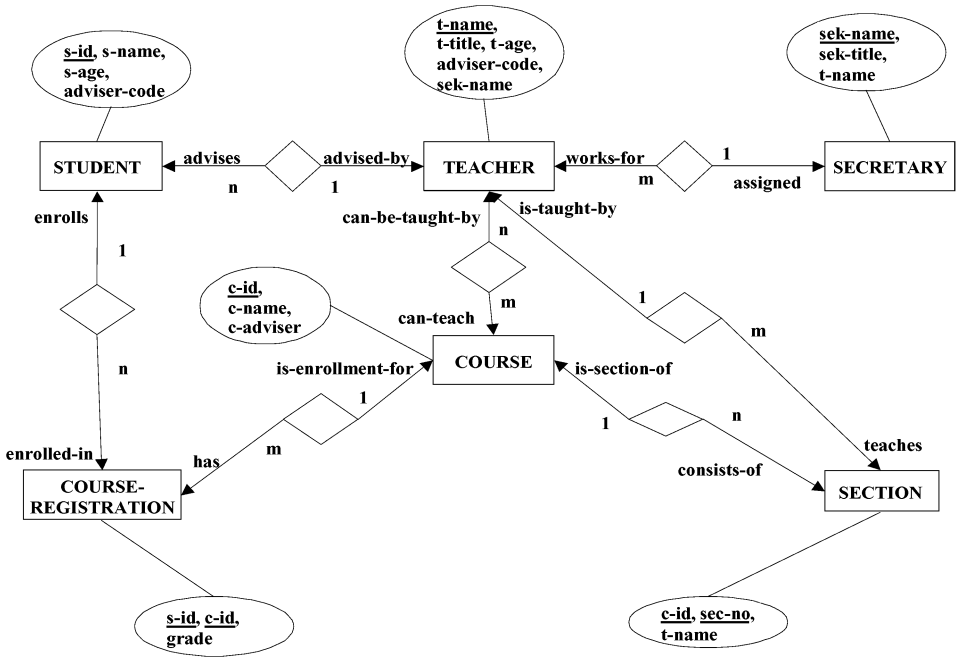

Fig. 1. Semantically constrained E-R diagram of a university department schema.

"Where Supplier supplies Parts." Conceptual query formulation can be illustrated with the Conceptual Query Language (CQL) [Owei et al., 2001b, and Navathe 2001a].

Figure 1 is a semantically constrained entity-relationship diagram[1] (SCERD) of a university department. In SCERD, entity types in the schema bear explicitly named relationships, or associations, among themselves. Each relationship has a semantic meaning. Double-headed arrows are used in SCERD to indicate that the entities at both heads of the arrows have a direct semantic relationship, and the arrowheads are labeled with the roles played, for example, consists-of and advises, by the entities in specific relationships. The association semantics of the relationships involving entities are constrained by the roles the entities play in the particular relationship. In SCERD, the semantics of the links between entities, therefore, lie in the form of roles. CQL supports the direct use of SCERD constructs in query formulation.

Given the schema in Figure 1, suppose that Query 1 below is posed.

*Query*1: What course(s) is the student *named* Marshall *taking* from associate professor Jones?

An abbreviated CQL formulation of this query requires the user to specify only the stated entities: Student, Teacher, and Course, along with a set of selection predicates on these entities. The system then charts one or more paths from

[1]SCERD contains other constructs that are used for updates. These have been left out of Figure 1, since they are not pertinent to the discussion here.

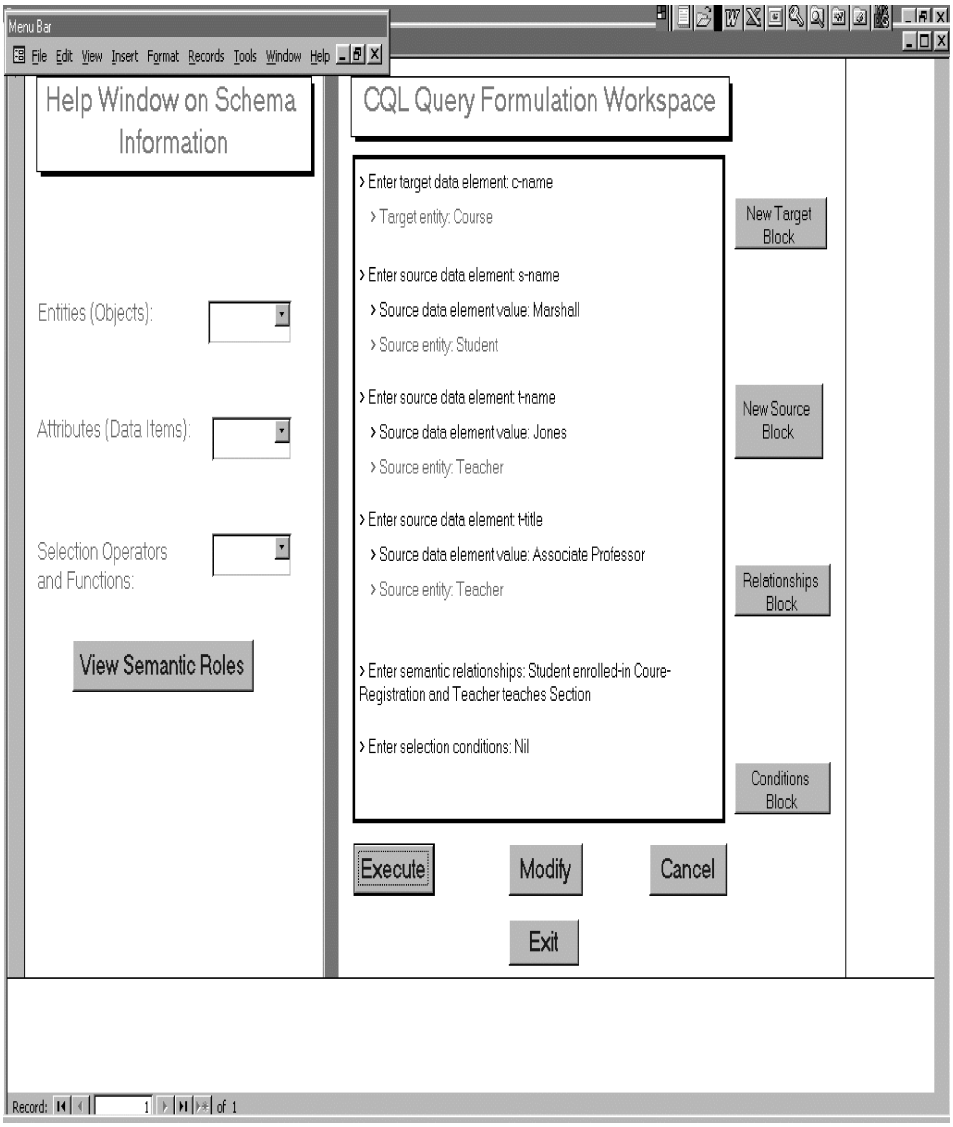

Fig. 2. CQL query input form.

Student and Teacher to Course. Unlike other concept-based query languages which require users to specify each query-path in its entirety, CQL requires users to specify only the endpoints, that is, the starting and terminating entities and relationship roles, of query-paths. The CQL system automatically deduces the correct intermediate entities to use on a given query-path. In addition, the system performs needed logical operations such as conjunction or disjunction on the derived paths.

Figure 2 shows the formulation of this query on an experimental CQL query input form. The help window on the left of the figure aids the user in formulating

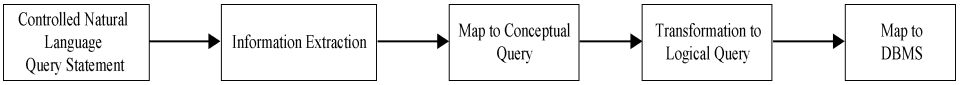

Fig. 3. Overall methodology of the CQL/NL approach.

queries by providing information on the schema to the user, and is part of CQL's computer-supported query formulation system [Owei 2000]. The user can request as many target and source blocks as desired by clicking the appropriate button on the right of the window.

With a natural language frontend to CQL, the input form in Figure 2 is overlaid with an NL interactive interface and is hidden from the user. Clearly, all that is required of the NL frontend is the ability to extract the terms Student, enrolled-in, Course-Registration, Teacher, teaches, Section, Course, and the other search terms from the NL query statement and write them to the semantically correct fields of the CQL input form. A form of controlled natural language (CNL) [Lewis and Jones 1996] is developed and used in this study. The use of CNL obviates the need for full syntactic parsing, and is not as restrictive as pseudonatural language. CNL is therefore able to support a wide range of linguistic styles and preferences.

The CNL query is submitted to an information extraction (IE) system. IE systems find and link relevant information while ignoring what is extraneous and irrelevant. Our goal in IE is to transform CNL queries into semantic grammatical patterns from which predefined templates of a concept-based query language, in our case CQL, can be filled with relevant information. The CQL/NL system combines natural language interface methods with the CQL querying approach into the single approach that we refer to as the semantic grammar (SG) approach. In the SG approach, the only relevant pieces of information to be extracted are the targets, sources, selection conditions, and semantic role conditions. The development of the SG approach is aimed at combining the ease of use of natural language systems with the power of the Conceptual Query Language.

Figure 3 shows the overall approach to query formulation and processing in the CQL/NL approach. First, specified queries in a controlled form of natural language are preprocessed to extract system-recognized lexical terms. These terms are then mapped into CQL. The resulting CQL query is in turn transformed into a logical query. Finally, the constructed logical query is specified against the underlying database management system. The outward form of CQL/NL is therefore seen to be CNL, which in turn uses the underlying formal basis of the semantic grammar. In the rest of this section, the controlled natural language and the semantic grammar are discussed.

3.1 Controlled Natural Language

Noun phrases (NPs) talk about some set of real-world objects (called referents). Therefore, NP interpretation in linguistics deals with determining the referents a given NP is investigating [Mellish 1985]. There is an implied NP to every query. This realization allows us to rephrase queries into CNL queries, which are simpler in lexical structure and, therefore, facilitate query formulation and the incremental evaluation and processing of the meaning of queries.

Listing 1. Sample CNL Queries

CNL Query: Find the names of students enrolled-in the classes taught-by Jones.
CNL Query: List the students that are enrolled-in Jones' class.
CNL Query: What is the highest course-registration grade given by Jones in the Fall semester?
CNL Query: How many students are taking classes from Professor Jones?
CNL Query: Get the student_name and student_id for the students that are enrolled in section A of IDS 410 that is taught by Professor Smith.
CNL Query: Which secretary works-for the teacher who teaches the section with sec-no. A and c-id Math 500?
CNL Query: Who are the teachers that can teach the course in which student Marshall is enrolled?

Towards this end, in Owei [2000] we therefore adopted the way suggested in Chang and Sciore [1992] for addressing the difficulty users experience with NL database query languages (DBQLs), namely, to combine concept-based DBQL paradigms with NL approaches to enhance the overall ease of use of query interfaces. We used information extraction approaches to extract only the relevant search terms in queries expressed in controlled NL. The extracted terms referred to semantic constructs on a conceptual DB schema.

CNL is flexible enough to accommodate a wide range of linguistic styles and preferences, but structured so simply as to render IE easy. Transforming free, unrestricted NL queries into CNL queries eliminates the need for both full semantic and syntactic parsing. Our approach to developing CNL queries is described as follows. From a corpus of free, unrestricted NL queries posed by users, we decided on a set of simple but semantically equivalent wh-forms (i.e., the forms who, what, where, when, how, etc.), list-forms, find-forms, and get-forms into which users' unrestricted questions were translated. The resulting forms are used as our CNL forms. A sample of CNL queries is shown in Listing 1.

3.1.1 *Controlled Natural Language Filter in CQL/NL.* Figure 4 shows the query-processing stages in CQL/NL. The user enters a CNL query; for example, the user specifies the query, "What course is the student named Marshall taking from associate-professor Jones?" as already explained. As seen from this figure, the NL filter in our IE approach accepts CNL queries as input, and filters them only for the search parameters of the CQL template. The filtration process consists of verbiage suppression. For this, we adopt an approach similar to that used in many IE systems where noncontent terms, that is, texts that are not relevant to the semantics of the application, are considered to be excess verbiage and eliminated without loss of meaning [Andersen et al. 1986; DARPA 1992; Jacobs and Rau 1990]. The suppressed terms consist mainly of verb terms such as "is," and function words and articles such as "the," "a," and so on. Verbiage suppression produces a leaner restatement of the query, which is thereby cast in a form that is close to the semantic categories of the application domain. For example, "What course is the student named Marshall taking from associate-professor Jones?" is suppressed to "What course student named Marshall taking associate-professor Jones?" Notice that the resulting statement may be difficult for humans to comprehend, and can therefore be hidden from the user. The structure of the suppressed CNL query statement bears a lexical resemblance to a semantic but primitive grammar. We use this primitive grammar as our semantic grammar.

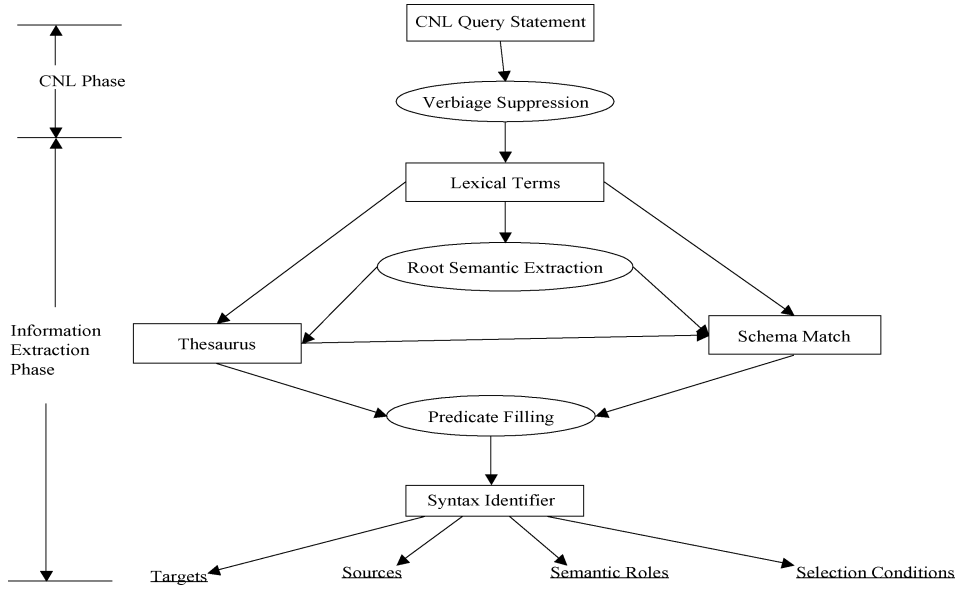

Fig. 4. CNL query processing.

Wh-Form → Target → Source → Source-attribute → Source Semantic → Source → Source-attribute
(Find, List and Get Forms) Value Role Value

Fig. 5. Semantic graph of Query 1.

3.1.2 *The Semantic Grammar.* Semantic grammars are generally simpler to process for information extraction and are domain-specific. Therefore they can be lexically tailored to the application. This means that each word position can have an unambiguous meaning with respect to the domain. The SG framework is developed by first drawing the semantic graph for the verbiage-suppressed versions of an initial set of CNL queries, and then defining the graph with a set of formal constructs. The approach is illustrated with our example Query 1: What course(s) is the student named Marshall taking from associate-professor Jones? As already explained, verbiage suppression to eliminate the underlined words in this CNL query results in query Q1*: "What course student named Marshall taking associate-professor Jones?"

We refer to queries transformed into this configuration as semantic grammatical (SG) queries, since they conform to the suppositions for our semantic grammars. The semantic graph for query Q1* is shown in Figure 5. In a similar fashion, a wide range of these semantic grammatical forms can be defined to accommodate a variety of user linguistic styles and preferences. Each SG form consists of a unique ordering of semantic predicates; it is therefore equivalent to a single semantic proposition specified by instantiating, or filling, the arguments of semantic predicates.

3.2 Specification of SG Forms

Formalizing the semantic grammar allows us to generate, define and process different SG forms. The semantic grammar is formally specified with a set of symbolic constructs.

Brackets "[]" specify an optional terms, braces "{}" represent possible repetition of terms, and diamonds "⟨⟩" separate logical units. The symbols are used to define our SG constructs as follows.

Semantic_Role := Relationship#1 [IntermediateEntity Relationship#2]
EntitySelected := EntityName { EntityAttributeName Entity AttributeValue }
EntityValuesOnly := { EntityAttributeName EntityAttributeValue }
Entity := EntityName.

The constructs are explained as follows.

—Semantic_Role: is a path between two entities with either one or two linking relationships. In case of two relationships, at least one must be given in the query, preferably the first one.

—Entity: is just an entity name. It must be given in the query. In certain cases, the semantic role between two entities is not given explicitly, but implied and must be deduced. Implied semantic roles that are part of a query-path are automatically determined by the system. An implied role may be different from the one implied by the most direct relationship between two entities. For example, "I take a course from teacher Jones" implies that Jones is not just capable of teaching this course in general, but is actually teaching this course now. The implied path between "teacher" and "course" must, therefore, be "teacher *teaches* section *is-section-of* course," and not "teacher *can-teach* course," which is semantically incorrect, although it is a shorter path with only one relationship.

—EntitySelected: denotes an entity name and a list of specific attributes and their values that are used in selection expressions. The attribute names or even the entity name may be omitted in the query, but all values must be given explicitly.

—EntityValuesOnly: is a list of attribute names and their values but without the entity name. Attribute names may be omitted but values are given explicitly.

The SG forms are formally defined using these constructs, as illustrated next with the SG form of sample query 1.

SG-1 Query Form

Wh-Form <> {[TargetAttributeName]} TargetEntity <> Source1Selected <> SourceSemanticRole <> Source2Selected <> (ImpliedSemanticRole).

Semantically, SG-1 is interpreted as follows.

> Find Target (or its attributes) such that a particular Source1 has Source SemanticRole with Target and also a particular Source2 is related to Target (via ImpliedSemanticRole).

SG-1 Example: What are the Ids and names of courses that student Marshall is taking from Jones?

After verbiage suppression the SG query is:

> what <> ids names courses <> student marshall <> taking <> jones

After reconstruction of the full mapping with synonyms "ids = c-id," "names = c-name," "courses = course," "taking = enrolled-in," we obtain:

> what <> c-id c-name course <> student s-name marshall <> enrolled-in course-registration is-enrollment-for <> teacher t-name jones

The implied relationship relating "teacher" and "course" via the "teaches" semantic role is deduced by the system and incorporated into the SG query. The query is therefore finally interpreted as

> what <> c-id c-name course <> student s-name marshall <> enrolled-in course-registration is-enrollment-for <> teacher t-name jones <> (teaches section is-section-of)

Semantically, this means:

> Find c-id, c-name of each course such that student with s-name marshall enrolled-in course-registration that in turn is-enrollment-for this course and also teacher with t-name jones teaches section that in turn is-section-of this course.

The SG forms for the other sample queries can be formally defined similarly. For example, queries Q2 and Q3 are formally defined as follows.

SG-2 (Long Query) Form

> WhForm <> {[TargetAttributeName]} TargetEntity <> TargetSemanticRole <> Source1Entity <> SourceSemanticRole <> Source2Selected.

The semantic meaning of SG-2-long form is:

> Find Target (or its attributes) that has TargetSemanticRole with Source1 that in turn has SourceSemanticRole with a particular Source2.

SG-2 (Short Query) Form

> WhForm <> {[TargetAttributeName]} TargetEntity <> TargetSemanticRole <> Source2Selected.

This query form is interpreted as meaning

> Find Target (or its attributes) that has TargetSemanticRole with a particular Source2.

SG-3 (Long Query) Form

WhForm <> {[TargetAttributeName]} TargetEntity <> TargetSemanticRole <> Source1ValuesOnly <> (ImpliedSemanticRole) <> Source2Entity.

The semantic meaning of SG-3-Long is taken as

Find Target (or its attributes) that has TargetSemanticRole with Source2 so that a particular Source1 has ImpliedSemanticRole with a Source2.

SG-3-Short Query Form

Wh-Form <> {[TargetAttributeName]} TargetEntity <> TargetSemantic-Role <> Source1ValuesOnly <> Source1Entity

The semantic meaning of this short form is:

Find Target (or its attributes) that has TargetSemanticRole with Source1.

Designers can follow a similar approach to define other forms as desired. Furthermore, already defined and existing SG forms can be logically combined to compose additional and more complex forms. This aspect of the study is currently under development.

4. IMPERFECT INFORMATION IN CONCEPT-BASED QUERIES

Imperfect information in DB systems can be classified into five types: uncertainty, incompleteness, imprecision, vagueness, and inconsistency. Parsons [1996] performs a trawl of the literature to provide a detailed discussion of the different types. A definition of each type is given in Table I. Also identified are some studies that address the different categories. As seen from Table I, imperfect information has been studied extensively in relational systems, to a lesser degree in object-oriented DB systems, and even less so in concept-based query systems. For example, Wald and Sorenson [1990] and Wu and Ichikawa [1992] deal with the issue in concept-based queries. In these two studies, however, the treatment is limited to ambiguous concept-based queries (in Wald and Sorenson [1990]) and incomplete concept-based queries (in Wu and Ichikawa [1992]). In this article, we address all the types of interpretations, and in this section, we examine what imperfect information means in the context of concept-based query languages.

4.1 Types of Imperfect Concept-Based Queries

As already discussed, in concept-based query languages, query specification is done against DB conceptual schemas. Direct use is made of semantic abstractions, such as entities, attributes, relationships, or roles in query formulations. Imperfection in concept-based queries can therefore be attributed to one or more of these semantic abstractions. In the remainder of this section, the different types of imperfection in concept-based queries are briefly described. They are considered in detail in subsequent sections, where techniques of how they are resolved in CQL/NL, our CNL system are also discussed.

Table I. General Classification of Imperfect Information in DB Systems

Type of Imperfect Information	Definition	Studies	Type of DB System
Uncertainty	A data value is not known for certain.	[Motro 1993, 1990], [Lee 1992a], [Prade and Testamale 1984, 1987b], [Barbara et al. 1990], [Wong 1982], [Cavallo and Pittarelli 1987], [Pittarelli 1994], [Lee 1992b] [Kiessling et al. 1994], [Kornatzky and Shimony 1994]	Relational OODB
Incompleteness	A pertinent piece of information is missing.	[Codd 1979], [Lien 1979], [Zaniolo 1982], [Williams and Kong 1988], [Demolombe and Farinas del Cerro 1988], [Zicari 1993], [Codd 1975], [Grant 1974, 1977, 1979, 1980], [Grant and Minker 1986], [Vassilliou 1979, 1981], [Goldstein 1981], [Biskup 1983], [Lipski 1979], [Winslett 1988] [Zicari 1990], [George et al. 1991]	Relational OODB
Imprecision	A situation in which the value of data does not meet the necessary degree of precision.	[Baldwin and Zhou 1984], [Dubois and Prade 1991], [Zivelli 1986], [Buckles and Petry 1987, 1982], [Prade and Testamale 1987a], [Shenoi and Melton 1989], [Buckles and Petry 1983], [Morrissey 1992], [Lee 1992a], [Leung et al. 1989] [Anwar et al. 1992]	Relational OODB
Vagueness	The value of data is given as a vague predicate, e.g., "age = old."	[Bosc and Prade 1993], [Prade and Testamale 1984], [Motro 1988]	Relational
Inconsistency	A variable is assigned two or more conflicting values.	[Wald and Sorenson 1990], [Markowitz and Shoshani 1989], [Czejdo and Embley 1987]	Relational

Semantically Mismatched Queries (Query Modification Problem). This involves slightly changing one correct query to another correct query. In terms of the taxonomy in Table I, queries in this group are either imprecise or inconsistent.

Example: "Find the students who are taking algebra" can be modified to "Find the students taking algebra courses taught by Professor Jones."

Missing Information. Queries having this type of imperfection are partially complete. An incomplete query is syntactically valid, but semantically incomplete. The formulation of missing information queries is such that the underlying target DBMS is unable (due to some unspecified information) to process it to return data. In terms of the classification discussed above, concept-based queries with missing information suffer from incomplete information on one or more semantic abstractions.

Example: How many students are taught by teacher?

Inexplicit Queries. A concept-based query can be imperfect because it is not a syntactically complete sentence or it contains noun phrases that corefer to other noun phrases. The former results in elliptical statements. The latter gives rise to anaphoric expressions. Both have the same effect of being semantically vague. Therefore, in terms of the general classification in Table I, inexplicit conceptual queries exhibit vagueness.

Example: What courses is the student named Marshall taking from Professor Jones, and Ford from Professor Smith?

Example: Find each student that is taking algebra. Also display the grade he or she got in it.

Ambiguous Queries. The wording of concept-based queries could be such that they can have multiple different meanings. Such queries are considered to be ambiguous. We can identify two classes of ambiguous queries. In one class are queries, the formulation of which is such that there is semantic conflict or mismatch between the set of query graphs generated.

Example: Find the teachers that each secretary works for; also list their titles.

Ambiguity in this case results from and is indicated by the presence of multiple query paths with differing meaning for the given query formulation. Queries of this kind suffer from inconsistency, in the sense of Table I.

In the second class are queries whose formulations result in unambiguous query paths, but the use of some nonschema term or expression renders their meaning ambiguous.

Example: Find the teachers that every secretary works for.

The obvious difficulty here is that of deciding the interpretation with the desired meaning. The best fit for queries of this kind, in terms of the classification in Table I, is vagueness.

5. SEMANTICALLY MISMATCHED QUERIES (QUERY MODIFICATION PROBLEM)

From the perspective of a query processor, a query represents a precise formulation of the search intended by the user. In this respect, there are two important factors that affect the precision of a formulated query: the terms, words, codes, or values used to describe what the user wants to retrieve; and the logic relating different sets of values [Meadow 1992]. The clear meaning of a query can, therefore, be corrupted by either of these two factors. Query rectification becomes necessary when either a semantic or a syntactic gap exists between the formulated query and the intention of the user. The purpose of rectification is to bring about semantic and syntactic convergence between the formulated and the intended queries. Rectification is, therefore, needed to resolve an imperfection in a query. We view query rectification as consisting of two aspects: modification and refinement. Query modification is taken up in

this section. In subsequent sections, the other types of imperfection and their rectification methods are addressed.

5.1 Query Modification

Query modification is done to remedy a fully formulated, syntactically valid query that is erroneous in that it lacks either semantic or syntactic convergence. A CQL/NL query can be modified by the following.

1. Selecting a new query. In this case, another query that conforms with the user's intended query is specified and processed as already discussed for a new query.
2. Tinkering with a query. This involves slightly altering the current query to change the semantic proposition expressed by the query. The two cases below illustrate what we mean by "tinkering" with a query.

Case 1: Query 2, "Find the students who are taking algebra" can be modified to Query 2-1: "Who are the students enrolled in mathematics?" Notice that the new query maintains the same SG form as the old one.

Case 2: Query 2 can also be changed to Query 2-2: "Find the students taking algebra courses taught by Professor Jones." In this case, the resulting query has an SG form that is different from, but related to, that of the changed query. By "related" we mean that both SG forms share some common segments on the SG graph.

To modify a query in CNL form, the user simply clicks the "modify" button on the input form. A new query formulation workspace is then presented to the user to input the new query. Figure 6 illustrates the modification of Query 2.

Clearly, in query modification, the user is changing one correct query to another correct query. For this reason, the system does not provide any enhanced support for this task beyond what is provided for the formulation of a new query.

6. MISSING INFORMATION

Query refinement in the context of missing information deals with queries that are incomplete. We, therefore, use query refinement in this sense to refer to the process of improving the semantic gap in partially complete queries. Based on our SG framework, refinement in this sense can be viewed as belonging to one of the following: unrecognized term, dangling SG pattern, or fragmented SG pattern.

Unrecognized Term. This type of refinement involves resolving a lexical term that is not recognized by the system and, therefore, cannot be automatically hooked to a database content term. Handling this type of refinement problem is discussed in detail in Owei [2000] and, therefore, not dealt with here.

6.1 Dangling Query (Missing Tail Problem)

Queries in this class have SG patterns that have their trailing ends missing, as shown in Figure 7. The problem here is that, due to the missing tail, the system

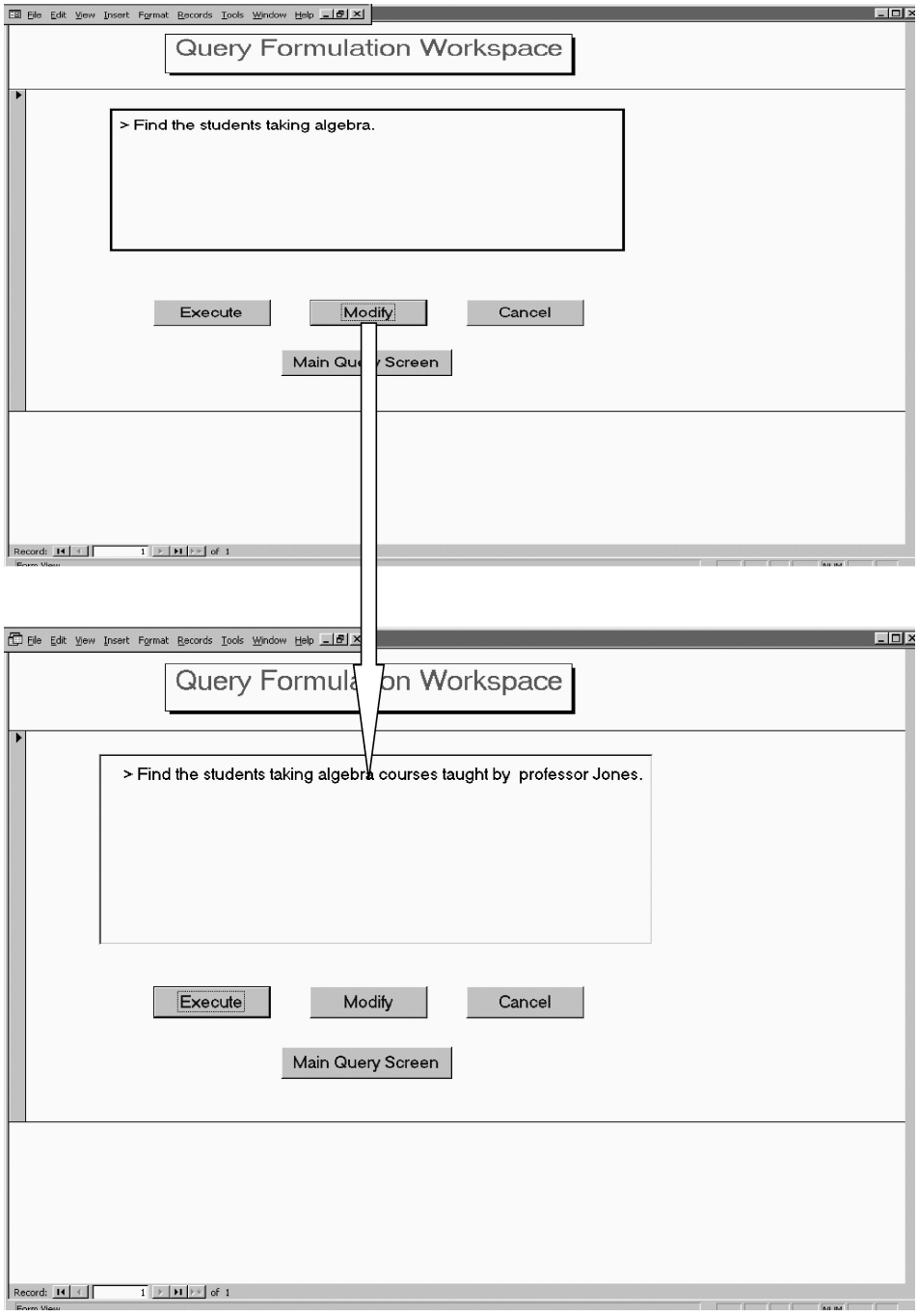

Fig. 6. Use of "Modify" button to change Query 2.

Fig. 7. Dangling SG pattern of an ill-stated query.

Wh-form -→ target aggregation term → Target -→ ……..……

Fig. 8. Dangling SG pattern of Query 3.

is unable to process the query. To refine the query, the problem is detected first, by matching it with any of the SG forms. Thereafter, the system returns an error message to the user. In Owei [2000] missing elements are hooked manually by the user. In this section, an interactive approach based on Skolemization of the missing elements is used. The approach uses a Skolem variable to compute the missing elements. The process is algorithmically described as follows.

Step 1: Assign Skolem variable $k(a_1, a_2, \ldots)$ to the missing segment of the SG query, where k is a schema entity and $\{a_i\}$ the set of attributes of k.

Step 2: Starting with the SG query, as the initial SG query, construct a simple SG path to each semantic neighbor from the tail entity by assigning the entities that are semantically adjacent to the tail entity to the Skolem variable one at a time.

Step 3: Return the set of computed paths plus the initial SG query to the user and ask the user to select the desired path(s).

Step 4: With the selected path as the new SG query, repeat Steps 2 and 3.

Step 5: Repeat Step 4 until the desired query path is attained.

Step 6: Stop.

As an example, suppose that the user specifies Query 3 against the schema. The SG pattern of this query is that of a truncated SG form 4, where the truncation occurs at the end of the pattern. The pattern is shown in Figure 8.

Query 3: What is the number of students?

If P_{sg} represents the known part of the SG query and r_{sk} the semantic role connecting P_{sg} to the assigned Skolem variable $k(a_1, a_2, \ldots)$, then Step 1 gives the parametric SG paths $P_{sg}r_{sk}k(a_1, a_2, \ldots)$. For this example, P_{sg} is "What number Students." Therefore, the Skolemized SG query is, "What number Students $r_{sk}k(a_1, a_2, \ldots)$."

In Step 2, the parametric variable $k(a_1, a_2, \ldots)$ is assigned the members of the set {Course, Teacher}. The corresponding set of semantic roles for r_{sk} is {enrolled-in (for Course), advised-by (for teacher)}. The computed SG paths are:

"What number Students enrolled-in Course having c-id (?), c-name(?), c-advisor(?)"

"What number students advised-by Teacher having t-id(?), t-name(?), t-age(?), adviser-code(?)

Step 3 therefore returns the following paths to the user.

1. "What number Students"
2. "What number Students enrolled-in Course having c-id (?), c-name(?), c-advisor(?)"
3. "What number students advised-by Teacher having t-id(?), t-name(?), t-age(?), adviser-code(?)

Assuming that the user is interested in finding the number of students that are registered for the course whose c-id is ids 410, then Path 2, "What number Students enrolled-in Course having c-id (?), c-name(?), c-advisor(?)," is picked by the user.

With Path 2 now as the new initial SG query, Step 2 computes the new paths:

"What number Students enrolled-in Course consisting-of Section having c-id(?), sec-no(?), t-name(?)"

"What number Students enrolled-in Course can-be-taught-by Teacher having t-id(?), t-name(?), t-age(?), advisor-code(?)."

The new set of paths now returned to the user is:

4. "What number Students enrolled-in Course having c-id (?), c-name(?), c-advisor(?)"
5. "What number Students enrolled-in Course consisting-of Section having c-id(?), sec-no(?), t-name(?)"
6. "What number Students enrolled-in Course can-be-taught-by Teacher having t-id(?), t-name(?), t-age(?), advisor-code(?)."

The user now picks Path 4. Since this is the same as the new initial SG query, this tells the system that this is the SG query of the query desired by the user. This terminating query is referred to as the "terminal query," or more appropriately the "terminal SG query."

Once the terminal SG query is attained, the user is prompted to instantiate the attributes of the Skolem entity, in this case c-id (?), c-name(?), c-advisor(?), whichever applies. For this example, the instantiation c-id = "ids 410" is made. Listing 2 shows the user–system interaction to resolve this query.

Suppose instead that the user is interested in finding the number of students that are registered for section A of ids 410. In this case, Path 5 in Step 3 is chosen: "What number Students enrolled-in Course consisting-of Section having c-id(?), sec-no(?), t-name(?)." The attribute-value instantiations c-id = ids 410 and sec-no = A are made to give the instantiated terminal SG query, "What number Students enrolled-in Course consisting-of Section having c-id 410, sec-no A."

6.2 Fragmented Query (Disconnected Segment Problem)

The SG patterns of queries in this group consist of disconnected segments of the form shown in Figure 9. The problem here is that the system cannot assign values to the missing segment. The task in refining this query is to compute the intermediate schema elements needed to connect the two segments. Here too, to correct this problem, the problem is detected by first matching the statement with the SG forms. The system then returns an error message to the user. Here

Listing 2. CQL/NL Output for Dangling Query 3

```
Ill-Stated query - missing tail:
Input your query: What is the number of students?

SUPPRESSED QUERY: What number students

Error: unknown query.

Please choose from the following the option that is closest to the intended query:
(1)  What number Students
(2)  What number Students enrolled-in Course having c-id(?), c-name(?), c-advisor(?)
(3)  What number students advised-by Teacher having t-id(?), t-name(?), t-age(?), advisor-code(?)

Please input an option number: 2

Please choose again from the following the option that is closest to the intended query:
(1)  What number Students enrolled-in Course having c-id(?), c-name(?), c-advisor(?)
(2)  What number Students enrolled-in Course consisting-of Section having c-id(?), sec-no(?), t-name(?)
(3)  What number Students enrolled-in Course can-be-taught-by Teacher having t-id(?), t-name(?), t-age(?),
     advisor-code(?)

Please input an option number: 1

Please enter value(s) for: c-id(?), c-name(?), c-advisor(?), whichever applies:
c-id = ids 410
c-name =
c-advisor =

MATCHED SG QUERY:  What number Students enrolled-in Course having c-id ids 410

PSEUDO-ENGLISH: What number of Students enrolled-in the Course having c-id ids 410.
```

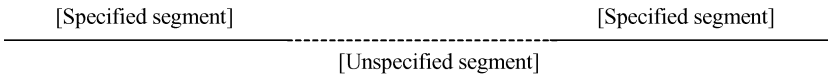

Fig. 9. Fragmented SG pattern of an ill-stated query.

too, a Skolem variable is used to compute the missing elements. Algorithmically, the process is as follows.

Step 1: Pick one segment of the disconnected SG query.

Step 2: Assign Skolem variable $k(a_1, a_2, \ldots)$ to the missing elements of the selected SG query.

Step 3: If no semantic role on the SG query can be used to compute the missing semantic neighbor of the tail entity, then

- Step 3.1: Starting with this Skolemized SG query, as the initial SG query, construct a simple SG path to each semantic neighbor from the tail entity by assigning the entities that are semantically adjacent to it to the Skolem variable one at a time.
- Step 3.2: Return the set of computed paths plus the initial SG query to the user.
- Step 3.3: Repeat Steps 1 to 3.2 with the second segment of the disconnected SG query.
- Step 3.4: Ask the user to select the desired path(s) from each computed segment.
- Step 3.5: Compute the intermediate schema elements connecting the two segments, as shown in Figure 10. (Where the user picks two or more subpaths from one or more of the segments in

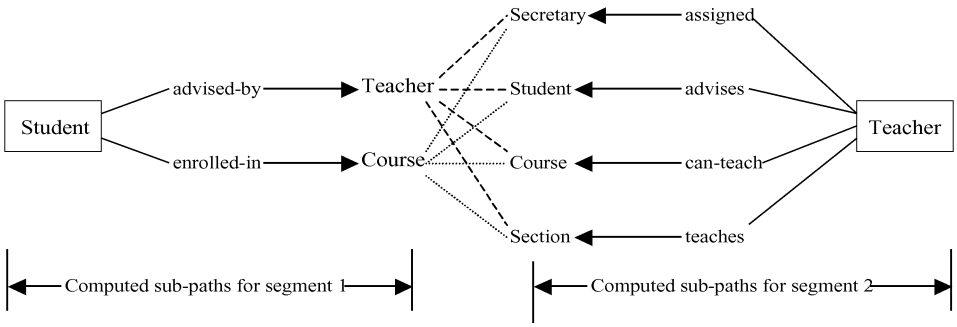

Fig. 10. Computed paths with no initial semantic role on SG query.

Wh-form -→ target aggregation term → Target -→ …….. -→ Source-attribute value

Fig. 11. Fragmented SG pattern of Query 4.

Step 3.4, Step 3.5 is repeated for each subpath. The resulting separate full paths are then logically combined, either conjunctively or disjunctively, into a complex path.)

Step 4: If a semantic role on the SG query can be used to compute the missing semantic neighbor of the tail entity, then

Step 4.1: Add the edge from the known semantic role to the neighboring entity to the SG query.

Step 4.2: Taking the new SG query as the original SG query, repeat Steps 1 to 4.

Step 5: Execute the query.

Step 6: Stop.

Suppose, for example, that Query 4 is specified against the schema. This query has the fragmented SG pattern shown in Figure 11. The pattern is that of a disconnected SG form 4.

Query 4: How many of the students from Professor Jones?

The SG query for this example is "How many students Professor Jones." A disconnection exists between students and professor, and there is no semantic role either from student or from professor that can be used to compute the missing semantic neighbor of student or professor. The applicable algorithmic step here is Step 3. The result of Step 3.1 to Step 3.3 is shown in Figure 10. In Step 3.4, assume that the user picks the Student — enrolled-in — Course edge from segment 1 and the Teacher – teaches — Section edge in segment 2. Then Step 3.5 gives the path shown in Figure 12. Listing 3 shows the output for Query 4.

Fig. 12. Computed final path with no initial semantic role.

Listing 3. Output for Fragmented Query 4

```
Ill-Stated query - missing segment without connecting semantic role:
Input your query: How many of the students from Professor Jones?
SUPPRESSED QUERY: How many students professor jones
Error: unknown query.

Please choose from the following the option that is closest to the intended query:
(1)  Student advised-by Teacher AND Teacher is-assigned Secretary
(2)  Student advised-by Teacher AND Teacher advises Student
(3)  Student advised-by Teacher AND Teacher can-teach Course
(4)  Student advised-by Teacher AND Teacher teaches Section
(5)  Student enrolled-in Course AND Teacher is-assigned Secretary
(6)  Student enrolled-in Course AND Teacher advises Student
(7)  Student enrolled-in Course AND Teacher can-teach Course
(8)  Student enrolled-in Course AND Teacher teaches Section

Please input an option number: 8

MATCHED SG QUERY: How many student enrolled-in course consists-of section is-taught-by teacher t-name jones

PSEUDO-ENGLISH: How many student enrolled-in course that in turn consists-of section that in turn is-taught-by teacher
having t-name jones?
```

Now, suppose instead that Query 5-1 is specified.

Query 5-1: How many students are enrolled in section with c-id ids 410 and sec-no A?

The SG query here is, "How many students enrolled section c-id ids 410, sec-no A." The disconnection in this case occurs at the semantic role "enrolled." Since "enrolled" is a synonym for "enrolled-in" in the application domain, algorithmic Step 4 is applicable. Step 4.1 adds the edge "enrolled-in—Course" to the SG query, as shown in Figure 13. Step 4.2 then computes the rest of Figure 13. If the user selects the Student–enrolled-in–Course–Course– is-section-of – Section(c-id, sec-no) path, then the query finds all the students that are enrolled in the course which has section with c-id ids 410 and section number A. The number of students that qualify is returned. Listing 4 shows an output for this interpretation of the query.

If instead the user selects the Student – enrolled-in – Course —- Teacher – is-taught-by — Section(c-id, sec-no) path, the query requires finding the students that are enrolled in all the courses that can be taught by the teacher who teaches the section having c-id ids 410 and sec-no A. The number of qualifying students is returned. If this interpretation of the query is taken as Query 5-2, then Listing 5 shows an output for the query.

6.3 Queries with Disconnected Segment and Missing Tail

Queries with both a missing segment and a missing tail are first processed to bridge the missing segment. The missing tail information is then resolved thereafter. As an example, suppose that the user specifies Query 6: How many students are enrolled in section?

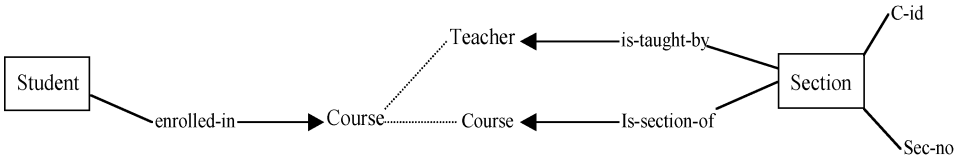

Fig. 13. Computed paths with an initial semantic role on SG query.

Listing 4. Output for Fragmented Query 5-1

```
Ill-Stated query - missing segment with connecting semantic role:
Input your query: How many students are enrolled in section with c-id ids 410, sec-no A?
SUPPRESSED QUERY: How many students enrolled section c-id 410, sec-no A
Error: unknown query.

Please choose from the following the option that is closest to the intended query:
(1) Student enrolled-in Course AND Section is-section-of Course
(2) Student enrolled-in Course AND Section is-taught-by Teacher

Please input an option number: 1

MATCHED SG QUERY: How many students enrolled-in course consists-of section c-id 410, sec-no A

PSEUDO-ENGLISH: How many students enrolled-in course that in turn consists-of section having c-id 410, sec-no A?
```

Listing 5. Output for Fragmented Query 5-2

```
Ill-Stated query - missing segment with connecting semantic role:
Input your query: How many students are enrolled in section with c-id ids 410, sec-no A?
SUPPRESSED QUERY: How many students enrolled section c-id 410, sec-no A
Error: unknown query.

Please choose from the following the option that is closest to the intended query:
(1) Student enrolled-in Course AND Section is-section-of Course
(2) Student enrolled-in Course AND Section is-taught-by Teacher

Please input an option number: 2

Please choose again from the following the option that is closest to the intended query:
(1) Student enrolled-in Course AND Course can-be-taught-by Teacher AND Section is-taught-by Teacher
(2) Student enrolled-in Course AND Section is-taught-by Teacher

Please input an option number: 1

MATCHED SG QUERY: How many students enrolled-in course can-be-taught-by teacher teaches section c-id 410, sec-no A

PSEUDO-ENGLISH: How many students enrolled-in course that in turn can-be-taught-by teacher that in turn teaches
section having c-id 410, sec-no A
```

The paths computed in this case are the same as Figure 13 without the attributes c-id and sec-no of Section. The missing segment is filled as already described, so that the two interpretations corresponding to the selections in Query 5-1 and Query 5-2 are:

Query 6-1: Student–enrolled-in–Course–Course–consists-of–Section
Query 6-2: Student–enrolled-in–Course—-Teacher–teaches–Section

In trying to process these paths further, as required by Step 5 of the algorithm for queries with fragmented segments, the missing tails are encountered. At this stage, the problem is solely that of a missing tail and is resolved as such. What is missing are instantiated attribute values for the source entity Section. The problem is therefore resolved as already described by Skolemization of the

Listing 6. Output for Query 6-1

```
Ill-Stated query - missing segment and missing tail:
Input your query: How many students are enrolled in section?
SUPPRESSED QUERY: How many students enrolled section
Error: unknown query.

Please choose from the following the option that is closest to the intended query:
(1)  Student enrolled-in Course AND Section is-section-of Course
(2)  Student enrolled-in Course AND Section is-taught-by Teacher

Please input an option number: 1

Intended query: Students enrolled-in Course consisting-of Section c-id(?), sec-no(?), t-name(?)

Please enter value(s) for: c-id(?), sec-no(?), t-name(?), whichever applies:
c-id = ids 410
sec-no = A
t-name =

MATCHED SG QUERY: How many students enrolled-in course consists-of section c-id 410, sec-no A

PSEUDO-ENGLISH: How many students enrolled-in course that in turn consists-of section having c-id 410, sec-no A?
```

missing tail (see example for Query 3). The paths computed and returned to the user by the system are:

1. Student–enrolled-in–Course–Course–consists-of–Section having c-id(?), sec-no(?), t-name(?)"
2. Student–enrolled-in–Course–Course–consists-of – Section–is-taught-by–Teacher for Query 6-1, and
3. Student–enrolled-in–Course —- Teacher–teaches–Section having c-id(?), sec-no(?), t-name(?)" for Query 6-2.

If the values ids 410 and A are assigned, respectively, to c-id and sec-no in 1 and 3, the resolved queries become:

Query 6-1: Student–enrolled-in–Course–Course–consists-of–Section having c-id ids 410, sec-no A

Query 6-2: Student–enrolled-in–Course —- Teacher–teaches–Section having c-id 410, sec-no A

Listings 6 and 7 show the outputs for these queries.

7. INEXPLICIT QUERIES

Inexplicitness and ambiguity usually characterize human expressions. The expressive power of a natural language system like CQL/NL is therefore enhanced by an ability to handle such expressions. CQL/NL uses semantic knowledge, contextual information, hooking of unrecognized and unhookable terms, and query rectification to resolve inexplicit and ambiguous expressions. We view inexplicit expressions as elliptical or anaphoric and handle them accordingly. Inexplicit queries are discussed in the current section, and ambiguous queries in the next.

Listing 7. Output for Query 6-1

```
Ill-Stated query – missing segment and missing tail:
Input your query: How many students are enrolled in section
SUPPRESSED QUERY: How many students enrolled section
Error: unknown query.

Please choose from the following the option that is closest to the intended query:
(1) Student enrolled-in Course AND Section is-section-of Course
(2) Student enrolled-in Course AND Section is-taught-by Teacher

Please input an option number: 2

Please choose again from the following the option that is closest to the intended query:
(1) Student enrolled-in Course AND Course can-be-taught-by Teacher AND Teacher teaches Section
(2) Student enrolled-in Course AND Section is-taught-by Teacher

Please input an option number: 1

Intended query: Student enrolled-in Course AND Course can-be-taught-by Teacher AND Teacher teaches Section c-id(?),
sec-no(?), t-name(?)

Please enter value(s) for: c-id(?), sec-no(?), t-name(?), whichever applies:
c-id = ids 410
sec-no = A
t-name =

MATCHED SG QUERY: How many students enrolled-in course can-be-taught-by teacher teaches section c-id 410, sec-no A

PSEUDO-ENGLISH: How many students enrolled-in course that in turn can-be-taught-by teacher that in turn teaches
section having c-id 410, sec-no A
```

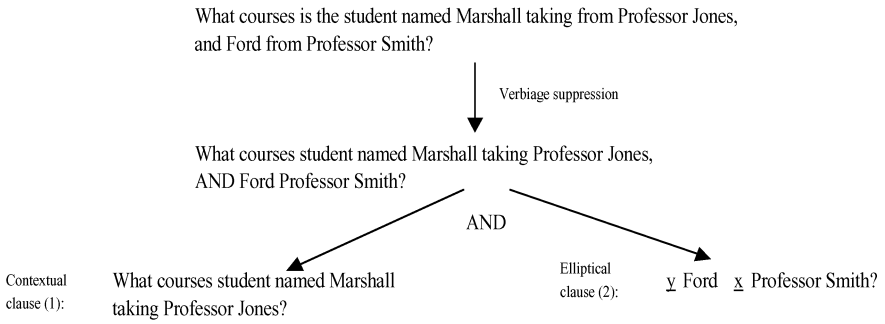

Fig. 14. Decomposition of elliptical query into contextual and elliptical clauses.

7.1 Elliptical Queries

An elliptical expression is a clause that is not a syntactically complete sentence. The query, "What courses is the student named Marshall taking from Professor Jones, and Ford from Professor Smith?" is elliptical. The verbiage suppressed CNL counterpart is, "What courses student named Marshall taking Professor Jones, and Ford Professor Smith?" Figure 14 shows that this expression can be decomposed into the contextual clause, "What courses student named Marshall taking Professor Jones?" and the elliptical clause, ". . . Ford . . . Professor Smith?" This suggests that elliptical queries can be processed in two main steps: the contextual clause, which is a complete sentence, and the elliptical clause, which is not a syntactically complete sentence but whose structure depends on the structure of the contextual clause. Therefore, the contextual clause is processed further without difficulty as any other query. The elliptical clause can be made

syntactically complete by assigning Skolem variables x and y to the missing terms as follows: "y Ford x from Professor Smith?"

In Owei [2000], elliptical clauses are resolved manually by requiring the user to assign values to the Skolem variables x and y. In this article, the process is automated, with the user only serving an assistive function in an interactive approach involving the user and the system. The approach here adopts an assumption that is common to most analyses of ellipses: that the completed phrase structure of the elliptical clause is the same as that of the contextual clause. The problem then reduces to finding the structural correspondence between the two clauses. This requires that the break between the two clauses first be identified. Thereafter, the Skolem variables can be instantiated. A logical operator provides the connection, and hence the breakpoint, between the two clauses.

At the SG level, the contextual clause is either an SG query with a complete, that is, recognizable, SG form, or one with missing information that can be resolved, as discussed. For the example query, for the structure of the elliptical clause to correspond to that of the contextual clause at the SG query level, the following instantiation of the Skolem variables is required: $x =$ "taking " and $y =$ "what course student named." This results in the SG query, "What courses student named Marshall taking Professor Jones AND what courses student named Ford taking Professor Smith?"

The process of resolving elliptical queries can therefore be described algorithmically as follows.

Step 1: Compute the path of the SG query.

Step 2: Tag the SG query segment corresponding to the portion of the path that is a complete target to source path. Mark this query segment as the contextual clause.

Step 3: Repeat Step 1 and Step 2 until only a subpath is left which is not a complete target to source path. Tag the SG query corresponding to this subpath as the elliptical clause.

Step 4: Return to the user the SG queries for the tagged elliptical clause and the tagged contextual clause that is antecedent to the elliptical clause. Request user to validate the two clauses and to mark the corresponding objects in the two clauses.

Step 4.1: If the user is unable to validate the clauses, then ask the user to manually correct them.

Step 5: Using the information on the corresponding objects, assign Skolem variables to the missing terms in the elliptical clause.

Step 6: Set the Skolem variables in the elliptical clause to their counterparts in the contextual clause.

Step 7: Process the resulting SG query.

Step 8: Stop.

Listing 8 shows the result for the example query (Query 7): What courses is the student named Marshall taking from Professor Jones and Ford from Professor Smith?

Listing 8. Output for Query 7

```
Ill-Stated query – Incomplete statement:
Input your query: What courses is the student named Marshall taking from Professor Jones and Ford from Professor Smith?
SUPPRESSED QUERY: What courses student named Marshall taking Professor Jones and Ford Professor Smith
Error: unknown query.

Intended query:
(1)  courses student named marshall taking professor jones AND courses student named ford taking professor smith
(2)  Re-enter intended query

Please input an option number: 1

MATCHED SG QUERY:  What courses student named marshall taking professor jones and courses student named ford
taking professor Smith

PSEUDO-ENGLISH: What courses student named marshall taking professor jones and what courses student named ford
taking professor Smith
```

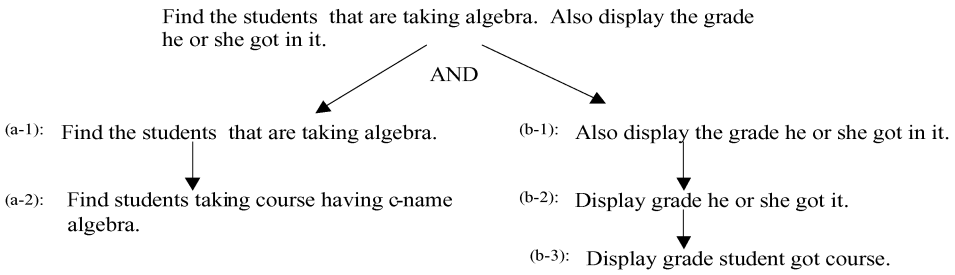

Fig. 15. Decomposition of an anaphoric query.

7.2 Anaphoric Queries

Queries in which noun phrases (NPs) corefer to other noun phrases are referred to as anaphoric. For example, the query, "Find each student that is taking algebra. Also display the grade he or she got in it," is an anaphoric query. In conceptual queries, NPs refer to entities, which are therefore considered to be first-class objects here.

Figure 15 shows the decomposition of the query into "Find the students that are taking algebra" (Statement (a-1)), and "Also display the grade he or she got in it" (Statement (b-1)). Statement (a-1) is processed as a query with a missing tail, as already discussed. This statement becomes, "Find students taking course having c-name algebra." In Statement (b-2), "Display" is hooked through its lexical synonym "Find." Based on the application database context, "grade" is also hooked. The phrase "he or she got in it" is unrecognized and therefore needs to be resolved.

The disjunctive terms "he" and "she" are replaced with the Skolem variables $k1$ and $k2$. Working backwards progressively, the coreferent of $k1$ and $k2$ can be determined. Since grade is a relationship attribute and $k1$ and $k2$ are in the local proximity of grade, the anaphora is first viewed as an intrasentential anaphora. An attempt is therefore first made to bind the anaphora within the local context of the sentence. First, $k1$ and $k2$ cannot refer to grade for two reasons: grade is not a first-class object in this sentence, and taking grade as the referent

will result in the SG tautology, "Display *grade* **grade** got it." There is no other possible coreferent within the sentence. Therefore the antecedent sentence is now considered, thereby making the anaphora a discourse, or intersentential, anaphora. The sentence, "Find students taking course having c-name algebra" now comes into view. The only first-class objects in this sentence are "Course" and "Student." The system therefore returns these two objects to the user to bind. Assume that "Student" is picked. The updated consequent sentence is now, "Display grade Student got it."

The anaphora "it" remains to be bound. The same process used to bind "he or she" to student is repeated. The Skolem variable $k3$ now replaces "it." The relationship attribute *grade* together with the entity Student now defines a local intrasentential domain within which the binding of the anaphora can be initially attempted. This domain defined by the semantic relationship "Student enrolled-in Course" contains "Student" and "Course" as the only first-class objects, and hence the only candidates for binding. The user is therefore presented with these two objects to choose from again. Suppose that "Course" is picked this time. The sentence now becomes, "Display grade Student got Course," which determines the semantically correct path.

Note that if the user had picked "Student" instead, the statement would then read, "Display grade Student got Student." This defines a recursive path that is semantically inconsistent with the local intrasentential domain determined by *grade* and *Student*. If the attempt to resolve the anaphora within the local context of the sentence in which it occurred had failed, then, as before, the antecedent sentence will have been considered. Listing 9 shows the result for this query (Query 8).

8. AMBIGUOUS QUERIES

By ambiguous queries we mean queries that are syntactically correct, but semantically ambiguous in that their senses can be mapped to different meanings. Unlike elliptical and anaphoric queries, ambiguous queries can be remedied either through the process of refinement or through the process of resolving inexplicit queries, depending on the nature of the ambiguity.

8.1 Handling Ambiguity as an Anaphora

Supposed that the user poses Query 9: Find the teachers that each secretary works for; also list their titles. This query is ambiguous because it can be validly mapped to the two meanings shown in Figure 16. The source of the ambiguity is seen to lie in the anaphora "their" of Statement (c-2), which can reference "secretaries," as in Statement (c-2-1) or "teachers," as in Statement (c-2-2). Two different meanings are thereby imparted by "their" to the query. The query is handled in the same way as an anaphoric query to remove the ambiguity. In this case, "their" is Skolemized to transform the subsequent query into the queries (1) list secretaries titles and (2) list teachers titles. The resulting subsequent queries are returned to the user to pick the intended query. Listing 10 shows the interaction for this example.

Listing 9. Output for Query 8

```
Ill-Stated query:
Input your query: Find each student that is taking algebra.  Also display the grade he or she got in it.
SUPPRESSED QUERY: Find student taking algebra.  Display grade he or she got it.
Error: unknown query.

Please choose from the following the option that is closest to the intended query:
(1)  Find student taking algebra. Display grade he or she got it.
(2)  Find student taking course algebra. Display grade he or she got it.

Please input an option number: 2

Unknown term "algebra".

Please choose from the following the option that describes the unknown term:
(1)   c-id = algebra
(2)   c-name = algebra
(3)   c-advisor = algebra
(4)   Other. I would like to assign the unknown term myself

Please input an option number: 2

Intended query:
(1)   student taking course c-name algebra. Display grade he or she got it.
(2)   Re-enter intended query

Please input an option number: 1
Unknown term "he or she".

Please choose from the following the option that describes the unknown term:
(1)  he or she = student
(2)  he or she = course
(3)  Other. I would like to assign the unknown term myself

Please input an option number: 1

Intended query:
(1)  student taking course c-name algebra. Display grade student got it.
(2)  Re-enter intended query

Unknown term "it".

Please choose from the following the option that describes the unknown term:
(1)  it = student
(2)  it = course
(3)  Other. I would like to assign the unknown term myself

Please input an option number: 1

Intended query:
(1)  student taking course c-name algebra. Display grade student got course.
(2)  re-enter intended query

Please input an option number: 1

MATCHED SG QUERY: Find student taking course c-name algebra. Display grade student got course

PSEUDO-ENGLISH: Find the student taking course having c-name algebra. Also display the grade student got in course
```

8.2 Handling Ambiguity as a Refinement Problem

Now assume that the query posed is Query 10: Find the teachers that every secretary works for. The ambiguous meanings of this query are portrayed in Figure 17. It is seen that the lack of clarity derives from the multiple references of the scoping term “every.” To resolve the ambiguity, “every” is replaced with the Skolem variable c, which is then bound in turn to “each” and “all,” that is, the set of possible unambiguous system-synonyms for “every.” The set of synonyms is returned to the user to select the intended meaning. After selection, the query is reconstructed and processed using the chosen term. Listing 11 shows the interaction.

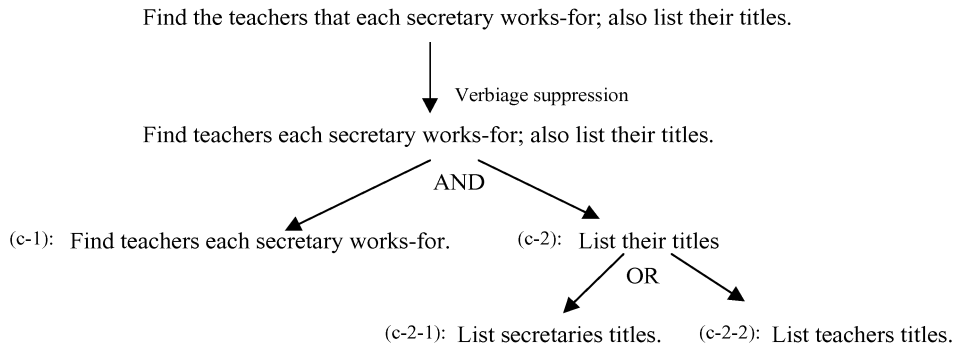

Fig. 16. Resolving ambiguity as an anaphora.

Listing 10. Output for Query 9

```
Ill-Stated query:
Input your query: Find the teachers that each secretary works for; also list their titles.
SUPPRESSED QUERY: Find teachers secretary works; list their titles.

Error: unknown query.

Please choose from the following the option that is closest to the intended query:
(1) find teachers secretary works. List their titles
(2) re-enter intended query

Please input an option number: 1

Unknown term "their".

Please choose from the following the option that describes the unknown term:
(1) their = teacher
(2) their = secretary
(3) Other. I would like to assign the unknown term myself

Please input an option number: 1

Intended query:
(1) find teachers secretary works. List teachers titles.
(2) re-enter intended query

Please input an option number: 1

MATCHED SG QUERY: Find teachers assigned secretary. List teachers titles

PSEUDO-ENGLISH: Find the teachers assigned secretary. List teachers titles
```

8.3 Special Case of Ambiguity: Dealing with Queries with Subtle Semantic Differences

As the discussion above indicates, query ambiguity arises from poorly worded statements. One other manifest effect of ambiguity is that, although two queries may sound almost the same, they may be semantically different: that is, the meaning of one may be different from the meaning of the other. Consider the following two classes of queries.

AQ 1.1: How many courses is student Marshall taking from (Professor) Jones?

AQ 1.2: How many courses is student Marshall taking with (student) Smith?

AQ 1.3: How many courses is student Marshall taking with (Professor) Smith?

AQ 2.1: What courses is student Marshall taking with (student) Smith?

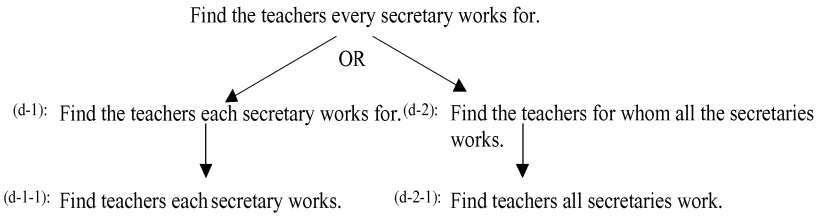

Fig. 17. Resolving ambiguity through query refinement.

Listing 11. Output for Query 10

```
Ill-Stated query:
Input your query: Find the teachers that every secretary works for.
SUPPRESSED QUERY: Find teachers every secretary works.
Ambiguous term "every".

Please choose from the following possible replacements the option that describes the ambiguous term:
 (1)  each
 (2)  all

Please input an option number: 2

Intended query:
 (1)  find teachers all secretaries works
 (2)  re-enter intended query

Please input an option number: 1

MATCHED SG QUERY: ) find teachers assigned all secretaries

PSEUDO-ENGLISH: Find the teachers assigned all secretaries
```

AQ 2.2: What courses is student Marshall taking with (Professor) Smith?

AQ 2.3: What courses is student Marshall taking from (Professor) Jones?

Clearly, these are valid queries, the meaning of which humans would readily understand. This, however, may not be the case with machines. In the CNL system, differentiating the queries in each group is difficult because of their identical verbiage suppressed equivalents:

AQ 1.1*: How many courses student Marshall taking Professor Jones?

AQ 1.2*: How many courses student Marshall taking student Smith?

AQ 1.3*: How many courses student Marshall taking Professor Smith?

AQ 2.1*: What courses student Marshall taking student Smith?

AQ 2.2*: What courses student Marshall taking Professor Smith?

AQ 2.3*: What courses student Marshall taking Professor Jones?

The interpretation of “from” in AQ1.1 and AQ2.3 poses no problems. However, on examining AQ1.2 and AQ2.1, it is seen that the source of the problem lies in the interpretation of “with.” In standard language parsing techniques, the meaning of AQ1.2 and AQ2.1 can be resolved only by resolving the meaning of “with.” Consider AQ1.2. Is the question dealing with courses in which both of them are enrolled? Is Smith another student enrolled in, that is, taking, the same courses as Marshall? If that is the intent, then the query can be stated less ambiguously as, “Find the number of courses in which both Marshall and Smith

Listing 12. Resolving Queries with Subtle Semantic Differences

```
Input your query: How many courses is student marshall taking with student smith?
SUPPRESSED QUERY: How many courses student marshall taking student smith
MATCHED SG QUERY: Find number courses has course-registration enrolls student s-name marshall and student s-name smith
PSEUDO-ENGLISH:  Find the number of courses that has course-registration that enrolls student with s-name marshall
and student with s-name smith.
The total number of such courses is 1.
```

```
Input your query: What courses is student marshall taking with student smith?
SUPPRESSED QUERY: What courses student marshall taking student smith
MATCHED SG QUERY: Find c-id courses has course-registration enrolls student s-name marshall and student s-name smith
PSEUDO-ENGLISH:  Find the c-id of courses that has course-registration that enrolls student with s-name marshall and
student with s-name smith.
Selected record from entity "course":
102 . . . biology . . . 1,2
```

are enrolled," or, "How many courses are Marshall and Smith taking together?" If, however, the question deals with courses taken by Marshall that are taught by Smith, then the query is the same as AQ 1.1, and this has already been solved. In this case, Smith is a student playing the role of a teacher teaching Marshall (as in graduate teaching assistant). Deciphering the intent of these queries depends on the contextual meaning of "from" and "with." This is a difficult task with sentence parsing approaches.

In the SG approach, this difficulty is addressed with the use of the semantic roles of entities: if student Smith is teaching the classes, then Smith is playing the role of a teacher, and the query is not well formulated. Instead of "student Smith," the appropriate phrase to use is "teacher Smith." As before, this reduces to AQ 1.1, which is already solved. Conversely, if, as in AQ 1.3 and AQ 2.2, Smith is a teacher enrolled in the same classes as Marshall, then Smith is playing the role of a student. The appropriate phrase in this case is "student Smith." AQ1.3 and AQ 2.2, respectively, then reduce to AQ 1.2 and AQ 2.1, and their corresponding suppressed forms to AQ 1.2* and AQ 2.1*. In the CNL system the underlying query paths constructed for these two queries lead from the student entity back to the student entity; that is, it has the form Student -→ Student. This is seen by the CNL system, and thus by CQL, as a recursion on the student entity. The result of processing these two queries in CNL is given in Listing 12.

9. DISCUSSION AND CONCLUSION

According to the proof-theoretic view of databases, a DB is seen as a certain logical theory, that is, as a set of axioms. As discussed in Reiter [1984], a DB is then the collection of all true statements that can be deduced from the axioms using those axioms in all possible combinations. It is therefore the set of theorems that can be proved from those axioms. A query on a DB can then be seen as a particular combination of axioms on the DB. It follows, then, that evaluating a query reduces to proving a theorem [Date 2000]. It also follows that a query that cannot be answered by a set of DB axioms is equivalent to a theorem that cannot

be proved. Stated differently, such a query represents a statement or a set of statements that cannot be deduced to be true from the existing set of DB axioms. The inability to deduce the truth about a statement is a phenomenon arising from imperfect information with respect to the universe of discourse (UoD).

The issue of imperfect information is extensively, but not fully, researched in the relational DB context. For example, in dealing with missing information, Date [2000] extends the types of nulls are include "does not apply," "does not exist," and "undefined," and then argues that theoretically, the types of nulls are infinite. It seems that an $(n+2)$-valued logic is needed to deal with n types of nulls. Where n is greater than 1, this logic approach to null resolution is known to entail too many difficulties [Date 2000]. Researchers therefore question the wisdom of allowing for more than two types of nulls. Another difficulty in dealing with multiple types of nulls is in the interpretation: what does a result with a combination of different types of nulls, or even the same type of null, mean? These questions are yet to be resolved.

Even less research has been done in the area of imperfect information in conceptual, that is, semantic, data models. To address this issue, the question, "What is an incomplete DB?" needs to be examined in the context of DBs based on semantic data models. On the conceptual level, it is a set of possible (complete) DBs, one of which corresponds to the true state of the UoD. Stated formally, what is the answer to the query q_R applied to an imperfect semantic DB? If D is a semantic DB, then $\{q_r : r \in D\}$ is the set representation of the knowledge of the answer to q. It is the possible answer q_D, which may be expressed as $q_D = \{q_r : r \in D\}$. This gives a set of possible answers, each computed from and corresponding to one possible state of the real world. Therefore, the answer is another imperfect DB. The inherent ambiguity in q_D cannot be resolved without additional information. Indeed, it is this need for additional information that renders the DB imperfect.

In the query processing approach underlying CQL/NL [Owei 2000] and CQL [Owei and Navathe 2001b, a], every query is equivalent to a subgraph induced from the conceptual schema against which the query is specified. Imperfection in these systems can, therefore, be defined as follows. If due to some flaw in the query formulation, the system is unable to navigate the induced query-graph from source to target or, where this is possible, it is unable to extract semantically correct data, the query formulation is said to be imperfect.

It may be argued that it is meaningless to talk of imperfect information at the conceptual schema level, that it cannot be ascertained that there is imperfect information at this level. Therefore it does not have the same significance or meaning that it has at the logical and implementation levels. Consider the following. To assert that there is imperfect information at a level, it must be possible to verify the assertion against another level that is exogenous to the level with the purported imperfect information. At the logical level it is the relational schema. To say that there is imperfect information on a conceptual schema is to say that it is known that this information exists, and that this information is known (this is not true for the logical level), since the conceptual level represents the UoD. But if this is already known, then no imperfection exists. Therefore, at the exact moment when this is known, the "imperfect" information

is no longer imperfect. In a concept-based query language like CQL/NL, the concepts in the modeled real-world are known, or assumed to be known (since it uses this real-world concept captured in the conceptual data model).

An area in which the concept-based approach can benefit is in the incorporation of intelligent tools and techniques into query systems. In intelligent query answering the intent of a query is analyzed to provide generalized, neighborhood, or associated information relevant to the query. An approach adopted in the more recent studies is to exploit the rich semantic information of knowledge-rich DBs to determine the intent of queries. Query intent analysis can be performed on query statements that are not well formulated or are difficult to interpret, in order to clarify the intent of the user. Once the intent is determined, the query can be restated either automatically or cooperatively, with the help of the user, in a form that is easily interpreted.

Intelligent approaches can also be used to provide computer-aided query formulation systems (CAQFSs) to facilitate user formulation of concept-based queries. Support for computer-aided query formulation is the more common application of intelligent query-answering tools in natural language query systems. In those systems where it is provided, the approach is usually assistive, with the user interacting with the system to incrementally formulate the query. This usually takes the form of the user responding to prompts and cues from the system.

These advances in the area of intelligent tools and techniques can therefore be applied to facilitate the formulation of abbreviated concept-based queries. We have attempted to apply this to resolving queries with imperfect information in this article. To the best of our knowledge, no other concept-based query language provides this extended level of assistance for query formulation.

Finally, to test the suitability of conceptual approaches to query formulation, in Owei et al. [2002] a statistical experiment was conducted to probe end-users' reaction to using CQL, vis-à-vis SQL, as a database query language. The comparison focused on the effect of the two different database query language interfaces on user performance (as measured by query formulation time, query correctness, and users' perception) in a query-writing task with varying difficulty levels. Statistically significant differences between the two query languages were found.

The results indicate that end-users perform better with CQL and have a better perception of it than of SQL. There were significantly more accurate formulations with CQL than with SQL. Also, the groups with CQL took significantly less time than the groups with SQL. The CQL subjects perceived their query language to be easier to use than their SQL counterparts felt about SQL; they also felt more satisfied with CQL than the SQL subjects were with SQL. These differences were more pronounced when query-difficulty level was considered. The statistical significance of the differences increased with the complexity of the query. The scores indicate that users are more likely to perform better with CQL than with SQL, and that they are more likely to harbor a more favorable perception of it than of SQL. The abbreviated concept-based approach in CQL, therefore, clearly offers an alternative approach to end-users for querying databases.

In summary, this article examined the notion of imperfect information as it applies to concept-based query languages. Although imperfect information is extensively studied in the context of relational database systems and to a far less degree in object-oriented database systems, it is relatively almost entirely uninvestigated in concept-based query language systems. Since concept-based query languages are beginning to find their way into the commercial arena (e.g., Conquer-II [Bloesch and Halpin 1997] and QBD* [Angelaccio et al. 1990]), and since imperfect information is an inevitability in "real life" information systems, it is therefore imperative that research be conducted on handling imperfection in concept-based query languages. That is what we have done here. However, we consider the work reported in this article only as an initial effort. We limited this article to dealing with the more common types of imperfection. Uncertainty and fuzziness in concept-based queries were not considered. In the future, therefore, we plan to study these issues. We would also like to study how to deal with concept-based queries that contain two or more types of imperfection. Lastly, we would also like to study the possibility of using the rich semantic information contained in conceptual database schemas to repair imperfect answers retrieved by imperfect queries or imperfections such as gaps or nulls in the extensions of databases. Finally, in the future, we would like to extend the study to addressing imperfect information in abbreviated concept-based query languages, such as CQL.

REFERENCES

ANDERSEN, P. M., HAYES, P. J., HEUTTNER, A. K., SCHMANDT, L. M., AND NIRENBERG, I. B. 1986. Automatic extraction. In *Proceedings of the Conference of the Association for Artificial Intelligence* (Philadelphia), 1086–1093.

ANGELACCIO, M., CATARCI, T., AND SANTUCCI, G. 1990. QBD*: A graphical query language with recursion. *IEEE Trans. Softw. Eng. 16*, 10, 1150–1163.

ANWAR, T. M., BECK, H. W., AND NAVATHE, S. B. 1992. Knowledge mining by imprecise querying: A classification-based approach. In *Proceedings of the Eighth International Conference on Data Engineering*, F. Golshani, Ed. (Tempe, Ariz., Feb. 3-7), 622–630.

BALDWIN, J. F. AND ZHOU, S. Q. 1984. A fuzzy relational inference language. *Fuzzy Sets Syst. 14*, 155–174.

BARBARA, D., GARCIA-MOLINA, H., AND PORTER, D. 1990. A probabilistic relational data model. In *Proceedings of the 1990 EDBT Conference*, 60–74.

BATRA, D. 1993. A framework for studying human error behavior in conceptual database modeling. *Inf. Manage. 25*, 121–131.

BATRA, D. AND SEIN, M. K. 1994. Improving conceptual database design through feedback. *Int. J. Hum. Comput. Stud. 40* 653–676.

BATRA, D., HOFFER, J. A., AND BOSTROM, R. P. 1990. Comparing representations with relational and EER models. *CACM 33*, 2, 126–139.

BERZTISS, A. T. 1986. Data abstraction in the specification of information systems. In *Proceedings of the IFIP World Congress 86*, 83–90.

BERZTISS, A. T. 1993. The query language Vizla. *IEEE TKDE 5*, 5, 813–825.

BISKUP, J. 1983. A foundation of Codds's relational maybe-operations. *ACM TODS 8*, 4, 608–636.

BLOESCH, A. C. AND HALPIN, T. A. 1997. Conceptual queries using Conquer-II. In *Proceedings of the ER'97: 16th International Conference on Conceptual Modeling* (Los Angeles), 112–126.

BONNISSONE, P. P. AND TONG, R. M. 1985. Editorial: Reasoning with uncertainty in expert systems. *Int. J. Man Mach. Stud. 22*, 241–250.

BOSC, P. AND PRADE, H. 1993. An introduction to fuzzy sets and possibility theory based approaches to the treatment of uncertainty and imprecision in database management systems. In *Proceedings*

of the Second Workshop on Uncertainty Management in Information Systems: From Needs to Solutions (Catalina, Calif.).

BUCKLES, B. P. AND PETRY, F. E. 1982. A fuzzy representation of data for relational databases. *Fuzzy Sets Syst. 5* 213–226.

BUCKLES, B. P. AND PETRY, F. E. 1983. Extension of the fuzzy database with fuzzy arithmetic. In *Proceedings of the IFAC Symposium* (Marseilles), 421–426.

BUCKLES, B. P. AND PETRY, F. E. 1987. Generalized database and information systems. In *Analysis of Fuzzy Information* 2, J. C. Bezdek, Ed., CRC Boca Raton, Fla., 177–201.

CAVALLO, R., AND PITTARELLI, M. 1987. The theory of probabilistic databases. In *Proceedings of the Thirteenth VLDB* (Brighton), 71–81.

CHAN, H. C. 1989. A knowledge level user interface using the ER model. PhD dissertation, The University of British Columbia, Vancouver, BC, Canada.

CHAN, H. C., WEI, K. K., AND SIAU, K. L. 1993. User-database interface: The effect of abstraction levels on query performance. *MIS Quart. 17*, 4, 441–464.

CHANG, T. AND SCIORE, E. 1992. A universal relation data model with semantic abstractions. *IEEE TKDE 4*, 1, 23–33.

CODD, E. F. 1975. Understanding relations, Installment No. 7, *FDT Bull. ACM SIGMOD 7*, 7, 23–28.

CODD, E. F. 1979. Extending the database relational model to capture more meaning. *ACM TODS 4*, 4, 379–434.

CZEJDO, B. AND EMBLEY, D. 1987. An approach to computation specification for an E-R query language. In *Proceedings of the Sixth International Conference on the E-R Approach* (New York, Nov. 9–11), 307–322.

DARPA. 1992. MUC-4. In *Proceedings of the Third Message Understanding Conference* (McLean, Va.).

DATE, C. J. 2000. An *Introduction to Database Systems*, 17th ed., Addison-Wesley, Reading, Mass.

DEMOLOMBE, R. AND FARINAS DEL CERRO, L. 1988. An algebraic evaluation method for deduction in incomplete databases. *J. Logic Program. 5*, 183–205.

DUBOIS, D. AND PRADE, H. 1988. *Possibility Theory: An Approach to the Computerized Processing of Uncertainty*. Plenum, New York.

DUBOIS, D. AND PRADE, H. 1991. Certainty and uncertainty of (vague) knowledge and generalized dependencies in fuzzy databases. In *Proceedings of the International Fuzzy Engineering Symposium on 91* (Yokohama), 239–249.

GEORGE, R., BUCKLES, B. P., AND PETRY, F. E. 1991. An object-oriented data model to represent uncertainty in coupled artificial intelligence-database systems. In *The Next Generation of Information Systems: From Data to Knowledge, Lecture Notes in Artificial Intelligence vol. 611*, Springer-Verlag, Berlin.

GOLDSTEIN, B. S. 1981. Constraints on null values in relational databases. In *Proceedings of the VLDB Conference*, 101–110.

GRAHNE, G. 1991. The problem of incomplete information in relational databases. In *Lecture Notes in Computer Science, vol. 554*, Springer-Verlag, Berlin.

GRANT, J. 1974. Incomplete models. *Notre Dame J. Formal Logic XV*, 4, 601–607.

GRANT, J. 1977. Null values in a relational database. *Inf. Proc. Lett. 6*, 5, 156–157.

GRANT, J. 1979. Partial values in a tabular database model. *Inf. Proc. Lett. 9*, 2, 97–99.

GRANT, J. 1980. Incomplete information in a relational database. *Funda. Inf. III*, 3, 363–378.

GRANT, J. AND MINKER, J. 1986. Answering queries in indefinite databases and the null problem. *Adv. Comput. Res. 3*, 247–267.

HALPIN, T. A. 1995. *Conceptual Schema and Relational Database Design*. 2nd ed., Prentice-Hall, Sydney, Australia.

HALPIN, T. A. 1996. Business rules and object-role modeling. *Database Prog. Des. 9*, 10, 66–72.

HALPIN, T. A. AND ORLOWSKA, M. E. 1992. Fact-oriented modelling for data analysis. *J. Inf. Syst. 2*, 2, 1–23.

HALPIN, T. A. AND PROPER, H. A. 1995a. Subtyping and polymorphism in object-role modeling. *Data Know. Eng. 15*, 251–281.

HALPIN, T. A. AND PROPER, H. A. 1995b. Database schema transformation and optimization. In

OOER'95: Object-Oriented and Entity-Relationship Modeling, Lecture Notes in Computer Science, vol. 1021, Springer-Verlag, Berlin.

IMIELINSKI, T. AND LIPSKI, W. 1981. On representing incomplete information in a relational database. In *Proceedings of the Seventh International Conference on VLDBs* (Cannes, Sept. 9–11), 388–397.

IMIELINSKI, T. AND LIPSKI, W. 1984. Incomplete information in relational databases. *JACM 31*, 4, 761–791.

JACOBS, P. S. AND RAU, L. F. 1990. SCISOR: Extracting information from on-line news. *CACM 33*, 11, 88–97.

JARVENPAA, S. L. AND MACHESKY, J. J. 1989. Data analysis and learning: An experimental study of data modeling tools, *Int. J. Man Mach. Stud. 31*, 367–391.

KIESSLING, W., LUKASIEWICZ, T., KOSTLERG., AND GUNTZER, U. 1994. The TOP database model—Taxonomy, object-orientation and probability. In *Proceedings of the Workshop on, Uncertainty in Deductive Systems and Databases* (Ithaca, N.Y.).

KORNATZKY, Y. AND SHIMONY, S. E. 1994. A probabilistic object-oriented data model. *Data Knowl. Eng. 12*, 143–166.

LEE, S. K. 1992a. Imprecise and uncertain information in databases: An evidential approach. In *Proceedings of the Eighth Intenational Conference on Data Engineering*, (Tempe, Az. Feb. 3–7) F. Golshani, Ed. 614–621.

LEE, S. K. 1992. An extended relational database model for uncertain and imprecise information. In *Proceedings of the Eighteenth Conference on VLDB* (Vancouver, B. C., Canada), 211–220.

LEUNG, K. S., WONG, M. H., AND LAM, W. 1989. A fuzzy expert database system. *Data Knowl. Eng. 4*, 287–304.

LEWIS, D. D. AND JONES, K. S. 1996. Natural language processing for information retrieval. *CACM 39*, 1, 92–101.

LIEN, Y. E. 1979. Multivalued dependencies with null values in relational databases. In *Proceedings of the Fifth Intenational Conference on VLDB* (Rio de Janeiro), 61–66.

LIPSKI, W. 1979. On semantic issues connected with incomplete information databases. *ACM TODS 4*, 3, 262–296.

MARKOWITZ AND SHOSHANI. 1989. Abbreviated query interpretation in EER oriented databases. In *Proceedings of the Eighth Intenational Conference on E-R Approach* Toronto, Oct. 18–20, 325–344.

MEADOW, C. T. 1992. *Text Information Retrieval Systems.* Academic Harcourt Brace Jovanovich, New York.

MELLISH, C. S. 1985. *Computer Interpretation of Natural Language Descriptions*. Wiley, New York.

MORRISSEY, J. M. 1992. Representing and manipulating uncertain data. *Int. J. Man Mach. Stud. 36*, 183–189.

MOTRO, A. 1988. VAGUE: A user interface to relational databases that permit vague queries. *ACM TOIS 6*, 3, 187–214.

MOTRO, A. 1990. Accommodating imprecision in database systems: Issues and solutions. *SIGMOD REC. 19*, 4, 69–74.

MOTRO, A. 1993. Sources of uncertainty in information systems. In *Proceedings of the Second Workshop on Uncertainty Management and Information Systems: From Needs to Solutions* (Catalina, Calif.).

NIJSSEN, G. AND HALPIN, T. 1989. *Conceptual Schema and Relational Database Design*. Prentice-Hall, Sydney, Australia.

OWEI, V. 1994. Framework for a conceptual query language for capturing relationship semantics in databases. PhD dissertation, The Georgia Inst. of Technology, Atlanta, Ga.

OWEI, V. Oct. 2000. Natural language querying of databases: An information extraction approach in the conceptual query language. *Int. J. Hum. Compu. Stud. 53*, 4, 439–492.

OWEI, V. AND NAVATHE, S. B. 2001a. A formal basis for an abbreviated concept-based query language. *Data Knowl. Eng. 36*, 2, 109–151.

OWEI, V. AND NAVATHE, S. B. 2001b. Enriching the conceptual basis for query formulation through relationship semantics in databases. *Inf. Syst. 26*, 6, 445–475.

OWEI, V., NAVATHE, S. B., AND RHEE, H. 1997. Natural language query filtration in the conceptual

query language. In *Proceedings of the 30th Hawaii International Conference on System Sciences*, (Maui, Hawaii January).

OWEI, V., RHEE, H. AND NAVATHE, S. B. 2002. An abbreviated concept-based query language and its exploratory evaluation. *J. Syst. Softw.* (to appear).

PARSONS, S. 1996. Current approaches to handling imperfect information in data and knowledge bases. *IEEE TKDE 8*, 3, 353–372.

PITTARELLI, M. 1994. An algebra for probabilistic databases. *IEEE TKDE 6*, 293–303.

PRADE, H. AND TESTEMALE, C. 1984. Generalizing database relational algebra for the treatment of incomplete or uncertain information and vague queries. *Inf. Sci. 34*, 115–143.

PRADE, H. AND TESTEMALE, C. 1987a. Fuzzy relational databases: Representational issues and reduction using similarity measures. *J. Am. Soc. Inf. Sci. 38*, 118–126.

PRADE, H. AND TESTEMALE, C. 1987b. Representation of soft constraints and fuzzy attribute values by means of possibility distributions in databases. In *Analysis of Fuzzy Information, 2*, J. C. Bezdek, Ed., CRC, Boca Raton, Fla., 213–229.

REITER, R. 1984. Towards a logical reconstruction of relational database theory. In *On Conceptual Modelling*, M. Brodie, J. Mylopoulos, and J. W. Schmidt, Eds., Springer-Verlag, New York.

SHENOI, S. AND MELTON, A. 1989. Proximity relations in the fuzzy relational database model. *Fuzzy Sets Syst. 31*, 285–296.

SIAU, K. L., CHAN, H. C., AND WEI, K. K. 1995. The effects of conceptual and logical interfaces on visual query performance of end users. In *Proceedings of the ICIS* (Amsterdam, The Netherlands, Dec. 10–13), 225–235.

VASSILLIOU, Y. 1979. Null values in data base management: A denotational semantics approach. In *Proceedings of the ACM-SIGMOD Intenational Conference on Management of Data*, 162–169.

VASSILLIOU, Y. 1981. Functional dependencies and incomplete information. In *Proceedings of the Conference on VLDB*, 260–269.

WALD, J. A. AND SORENSON, P. G. 1990. Explaining ambiguity in a formal query language. *ACM TODS 15*, 2, 125–161.

WILLIAMS, M. H. AND KONG, Q. 1988. Incomplete information in a deductive database. *Data Knowl. Eng. 3*, 197–220.

WINSLETT, M. 1988. A model-based approach to updating databases with incomplete information. *ACM TODS 13*, 2, 167–196.

WONG, E. 1982. A statistical approach to incomplete information in database systems. *ACM TODS 7*, 470–488.

WU, X. AND ICHIKAWA, T. 1992. KDA: A knowledge-based database assistant with a query guiding facility. *IEEE TKDE 4*, 5.

ZADEH, L. A. 1965. Fuzzy sets. *Inf. Control 8*, 338–353.

ZADEH, L. A. 1983a. Commonsense knowledge representation based on fuzzy logic. *Computer*, 61–65.

ZADEH, L. A. 1983b. The role of fuzzy logic in the management of uncertainty in expert systems. *Fuzzy Sets Syst. 11*, 199–227.

ZANIOLO, C. 1982. Database relations with null values. In *Proceedings of the ACM Symposium on Principles of Database Systems* (Los Angeles), 27–33.

ZICARI, R. 1990. Incomplete information in object-oriented databases. *SIGMOD REC. 19*, 3, 5–16.

ZICARI, R. 1993. Databases and incomplete information. In *Proceedings of the Second Workshop on Uncertainty Management in Information Systems: From Needs to Solutions* (Catalina, Calif.).

ZVELLI, A. 1986. A fuzzy relational calculus. In *Proceedings of the First Intenational Conference on Expert Database Systems* (Charleston, S.C.), 311–326.

Received November 2000; revised May 2001; accepted March 2002