Natural Language Interpretations for Heterogeneous Database Access

Hodong Lee and **Jong C. Park**
Computer Science Division and AITrc
Korea Advanced Institute of Science and Technology
373-1 Gusung-dong, Yusong-gu, Daejon 305-701, South KOREA
{hdlee,park}@nlp.kaist.ac.kr

Abstract

In order to query into diverse types of databases and to integrate the resulting information, dispersed throughout the network in a specific domain, we must address complex problems due primarily to heterogeneity of the involved databases. In this paper, we propose to model access to heterogeneous databases, by interpreting natural language queries into queries in formal languages such as SQL, OQL, and CPL by accounting for various language-specific constructions including join relations, path expressions, and object bindings with domain resources and a common lexicon, in a combinatory categorial grammar framework. [1]

1 Introduction

Natural language database interfaces (NLDBs) are designed to free the user of the burden of familiarizing himself/herself with expressions of a formal query language in consulting the database of interest. They also free the user of the need to know the specifics of the database, as the choice of formal query languages is often tightly coupled with the way in which the databases are organized (Androutsopoulos et al., 1995; Lee and Park, 2001b).

While information access to heterogeneous data sources is an interesting issue in NLP fields such as open-domain question and answering, information retrieval and information extraction, there are complex problems due primarily to heterogeneity (Wong, 2000). That is, the databases are usually dispersed throughout the network in a specific domain as relational, object-oriented, object relational, temporal, or other flat file databases.

In order to provide access to heterogeneous databases, we propose to interpret natural language queries into queries in formal languages such as SQL, OQL, and CPL with a common natural language lexicon where SQL works for relational and object relational databases, OQL for object-oriented databases (Cattell, 1996), and CPL for structured-data file systems (Paton et al., 1999; Wong, 2000). An example query and its target interpretations are shown below:

(1) (a) 회색 전화기를 만드는 회사명은?
(hoy-sayk cen-hwa-ki-lul man-du-nun hoy-sa-myeng-un?)[2]
What are the names of companies which produce gray-colored telephone sets?

(b)
```
SELECT company.name
FROM   product, category, company
WHERE  product.catid=category.catid
       and product.cid=company.cid
       and product.color='gray'
       and category.catname='phone'
```

(c)
```
SELECT com.name
FROM   p in product, cat in p.catid,
       com in p.cid
WHERE  p.color='gray' and
       cat.catname='phone'
```

(d)
```
{ z.#name | \x <- product,
\y <- x.#category, \z <- x.#company,
x.#color="gray", y.#catname="phone"}
```

For the natural language query 1a, examples 1b, 1c, and 1d correspond to SQL, OQL, and CPL queries, respectively. In this example, query 1a is interpreted as 'a set of names of companies which produce the telephone sets in gray color' and represented in the first order notation with implicit quantifiers as shown below.

(2) $nameof(X, Y) \wedge company(X) \wedge productof(X, Z) \wedge phone(Z) \wedge colorof(Z, gray)$

While this meaning seems to be specific enough, it is nevertheless difficult to represent such meanings in database languages without any loss of semantics due to their differences in expressive power and inherent semantic limitation of a particular query language. In this paper, we propose to address this problem by utilizing the common lexical resource in interpreting natural language queries into queries in formal languages that show differences in the syntax, semantics and language paradigm. Due to these differences, the correct interpretation often requires

[1]This work was supported by the Korea Science and Engineering Foundation (KOSEF) through AITrc.

[2]The Yale transcription of a Korean query is shown in parentheses.

a different modeling of the syntactic and semantic mapping for the parsed information into the corresponding database languages. For example, relations such as '$company(X) \wedge productof(X, Z) \wedge phone(Z)$' in 2 is modeled as the join relations for SQL's select syntax (in 1b), path expressions for OQL (in 1c) and object bindings for CPL's set notation (in 1d). We will discuss more complex interpretations later.

In the present proposal, we make use of a combinatory categorial grammar (CCG) framework for the interpretation of natural language query expressions such as subject ellipsis, noun phrases, numerical expressions, coordination, and subordination into the corresponding SQL queries (Lee and Park, 2001b). This framework is portable and extensible to other database languages and logical forms using the features of CCG that is well known to provide a means of deriving all the levels of information for natural language, i.e., syntax, semantics and discourse, at the same time (Steedman, 2000; Lee and Park, 2001a). These features are enough to provide a simple, straitforward and modular translation with transparent CCG derivation (Lee and Park, 2001a).

First, we discuss problems in handling queries to heterogeneous databases. Among these problems, we focus in particular on interpretations of natural language queries into queries in formal languages such as SQL, OQL, and CPL in a CCG framework. In this process, we explain our method for representing natural language query in each database language, using features such as join relations, path expressions, object bindings and other formal language constructions, with a linguistic concern for the correct translation.

The rest of the paper is organized as follows. Some recent natural language database interfaces (NLDBs) will be discussed in Section 2. We discuss problems for heterogeneous databases in Section 3. Section 4 describes interpretations to formal query languages, i.e., SQL, OQL, and CPL. Section 5 describes our implemented system and results of our experiment. In Section 6, we discuss implications to our work.

2 Related Work

In this section, we show a brief introduction to CCG and NLDBs for various database languages.

2.1 Combinatory Categorial Grammar

Combinatory Categorial Grammars (CCGs) are combinatory extensions to the categorial grammars (Steedman, 2000). CCGs are among the lexicalized grammars, such as linear indexed grammars and tree adjoining grammars, and are known to provide a wide linguistic coverage and a way of processing sentences incrementally.

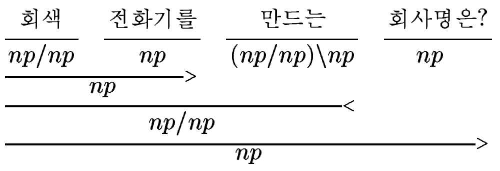
Rule	Rule Name (Symbol)
$X/Y \quad Y \rightarrow X$	Forward Application ($>$)
$Y \quad X\backslash Y \rightarrow X$	Backward Application ($<$)
$X \quad conj \quad X \rightarrow X$	Coordination ($< \phi^n >$)
$X/Y \quad Y/Z \rightarrow X/Z$	Forward Composition ($> B$)
$Y\backslash Z \quad X\backslash Y \rightarrow X\backslash Z$	Backward Composition ($< B$)
$X/Y \quad Y\backslash Z \rightarrow X\backslash Z$	Forward Crossed Comp. ($> B_x$)
$X \rightarrow T/(T\backslash X)$	Forward Type Raising ($> T$)
$X \rightarrow T\backslash(T/X)$	Backward Type Raising ($< T$)

Table 1: CCG Rules for Korean

System	DB	Grammar	Target Language
KID	OODB	DCG	OQL
TAMIC-P	RDB	CFG	SQL
NLTDB	TDB	HPSG	TSQL2
QWERTY	TDB	TLG	SQL/Temporal
NChiql	RDB	DG	SQL
N/S	RDB	CCG	SQL

Table 2: Approaches to NLDB

Table 1 shows the CCG reduction rules proposed for Korean (Park and Cho, 2000). Reduction rules for English are similarly defined (Steedman, 2000).

(3)


Example 3 shows the CCG derivation for the example query 1a with syntactic categories alone. Modifiers such as '만드는', that work in a way similar to relative clauses in English but form a noun phrase instead, are assigned the category $(np/np)\backslash np$, which receives a phrase of category np on its left (the third np; the directionality is indicated by the backslash \, that is, to the left) and then receives another np on its right (the second np; the directionality is indicated by the slash /, that is, to the right), to give rise to the phrase of category np.[3] Such computation is done by simple function applications.

2.2 Heterogeneous Database Interfaces

There have been much effort on developing such NLDBs since the 1960's (Androutsopoulos et al., 1995). Some of the recent proposals are summarized in Table 2. In this section, we review them briefly.

The KID (Chae and Lee, 1998) system transforms parse trees into OQL queries using syntactic patterns that are manually constructed from a sample query corpus. This system generates frame structures containing the object path for the computation of a path expression.

The NLTDB (Androutsopoulos et al., 1998) and QWERTY (Nelken and Francez, 2000) systems focus

[3] np is a shorthand for "noun phrase".

on queries with temporal expressions, with a specialized semantic representation language that can handle temporality. Example queries are shown below.

(4) Did any flight circle while runway 2 was open?
Which companies serviced BA737 in 1990?
During which years did Mary work in marketing?

The NChiql (Meng et al., 2001) system and the system proposed by (Lee and Park, 2001b) translate natural language queries into SQL statements based on extracted domain knowledge information such as database entities, attributes and relationships. (Meng et al., 2001) provides the query translating method with well-defined database semantics. And (Lee and Park, 2001b) provides a tool that can construct the lexicon automatically from database and domain information.

The TAMIC-P (Klein et al., 1998) system interprets noun phrase queries according to possibly different perspectives in the social insurance databases. Example queries are shown below.

(5) Ersatzzeiten wegen Kindererziehung
Exemption times because of child raising

The system generates SQL statements for access to heterogeneous databases with domain knowledge which encodes a unified view of the domain data and the hierarchy of concepts which can be deduced from the databases and their relations (Matiasek et al., 1999).

While the system can analyze noun phrases with various adverbial phrases, it is not reported to be able to analyze more complex noun phrase queries such as those with subordinate or coordinate constructions, that are often observed in the queries and handled by our system. In addition, our system provides a more broad-scale access to heterogeneous databases by generating SQL, OQL, and CPL queries, among others.

3 Access to Heterogeneous Databases

There are several considerations that must be resolved in handling queries to databases (or data sources) which are "high in volume, highly heterogeneous and complex, constantly evolving, and geographically dispersed" (Wong, 2000).

Diverse types of databases: There are different types of high-level query languages, such as SQL, OQL, SQL/Temporal, and CPL. There are also databases (or data repositories) which only provide low-level primitives for database access.

Different database information: For the generation of formal queries into a specific database, the NLDB system must have information about database specifics, such as location, database name and type, supporting query language, table (or class) and column (or attribute) names, their types, and their structure. Thus the system requires methods for integrating and managing the different information, related to the multiple databases, in a manner similar to that in the class hierarchy and Wordnet (Matiasek et al., 1999).

Data in various languages: Since databases may be geographically dispersed, the data can also be written in diverse languages, which necessitates multilingual query translation (Thompson and Mooney, 1999; Matiasek et al., 1999; Lee and Park, 2001a). In interpreting multilingual queries to databases with domain information, the lexical selection must be dealt with word sense disambiguation (Palmer et al., 1999). (Lee and Park, 2001a) proposes a method for interpreting multilingual queries by augmenting the translation dictionary with database terminologies.

Large size of lexicon: In interpreting natural language queries to multiple databases, the system must handle large domain-specific lexical items according to the database objects. For example, in bioinformatics domain, there are more than a hundred million journal abstracts including so many protein names and author names. This problem gives rise to the need for approximation methods to deal with the large volume of domain-specific entities. While the system can reduce the size of the lexicon with this approximation, the lexicon could be much larger if we need to deal with multiple databases as well.

4 Interpretations to Database Query Languages

In this section, we show interpretations and their processing into query languages such as SQL, OQL, and CPL with the same lexical entries. These languages can be directly translated from the parsed result of CCG derivation which contains all the information required to generate queries in each formal language. Our interpretation method is also extensible to generate SQL/Temporal and its logical form queries with temporality with slightly modified lexical entries (Lee and Park, 2001a).

4.1 Representation of Lexical Entries

In CCGs, all the levels of information for natural language, such as syntax, semantics, and discourse, are integrated into the categorial lexicon as lexical entries. The following shows example lexical entries of a CCG for Korean.

```
(6) lex( 전화기를, np:[_,category,category.catname='phone']).
    lex( 회사명이, np:[company.name,company,_]).
    lex( 얼마인가, s:[A,B,C]\np:[A,B,C]).
```

The lexical entry consists of a lexical item and its CCG category. The CCG category is a pair of

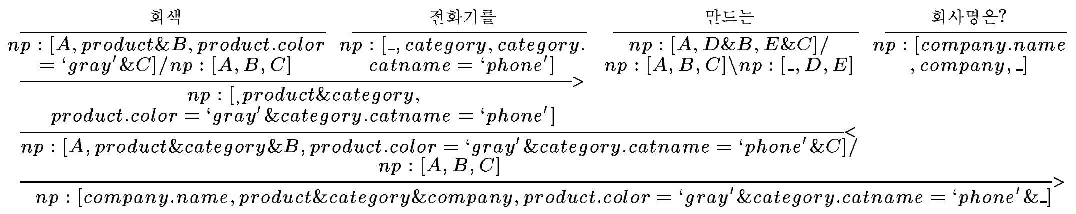

회색
np : [A, product&B, product.color = 'gray'&C]/np : [A, B, C]
전화기를
np : [_, category, category.catname = 'phone']
만드는
np : [A, D&B, E&C]/np : [A, B, C]\np : [_, D, E]
회사명은?
np : [company.name, company, _]
>
np : [,product&category, product.color = 'gray'&category.catname = 'phone']
<
np : [A, product&category&B, product.color = 'gray'&category.catname = 'phone'&C]/np : [A, B, C]
>
np : [company.name, product&category&company, product.color = 'gray'&category.catname = 'phone'&_]

Figure 1: A CCG Derivation of Example 1a

the syntactic and semantic information that are interwoven in the following way. Elementary CCG (syntactic) categories include *np* and *s* (and *vp* for abbreviatory purposes), and CCG categories are recursively defined as either X/Y or $X\backslash Y$, where X and Y are also CCG categories, including elementary categories. Each elementary CCG (syntactic) category X is augmented with an appropriate semantic information Z so that the resulting form $X : Z$ is a CCG category (Steedman, 1996).

In our proposal, the semantic information is encoded as a suitable fragment of SQL, OQL, and CPL, with slots modeling query results, tables/collections against which query runs (collections indicate complex objects such as sets, bags and lists of structured data), and query conditions[4], bracketed by '[' and ']'. For example, in SQL statement 1b, in the first entry in 6, '전화기를' is assigned the syntactic category 'np' and the semantic information which encodes the fact that the database attribute 'category.catname' has the value 'phone' in the table for 'category'. 'category' is described in the FROM clause of SQL and 'category=phone' in the WHERE clause. In the second entry in 6, 'company.name' is encoded in the SELECT clause. In the last entry in 6, the intransitive verb '얼마인가' (el-ma-in-ka) is taken to have ambiguous meanings, including 'what is the price of', 'what is the amount of', 'how much is', and the like. During the parsing process, this kind of lexical ambiguity can be resolved by accounting for the semantics such as the SELECT clause information and condition attributes in the WHERE clause. We encode synonyms in order to refer to the representative entry (Lee and Park, 2001b). For example, TV has synonyms in Korean such as '텔레비젼', '텔레비', '티비', '티브이', 'television', and so on. The lexical entry for 'TV' is described as in example 7a, and its synonym '텔레비젼' as in example 7b. This '텔레비젼' refers to the representative entry 'TV' via the tag 'alias' and the syntactic categories of 'TV' are assigned to '텔레비젼'.

(7) (a) lex(TV, np:[_,category,category.catname='TV']).

(b) lex(텔레비젼, alias:'TV').

[4]We call each slot as SELECT, FROM and WHERE slot for convenience. And all such information is commonly required in SQL, OQL, and CPL.

4.2 Interpretation into SQL Expressions

Example 1b is translated with the sample database schema, which is a relational data model, consisting of product, company, and category table. In addition to product information such as price, name, color, product ID and size, product table has category ID (i.e. catid) and company ID (i.e. cid) which uniquely identify category and company of corresponding products. Company table and category table have, respectively, maker and class information of the product. Figure 1 shows a CCG derivation of query 1a.

In the translation to SQL statement from the resulting semantics in Figure 1, implicit join relations such as 'product.catid = category.catid' and 'product.cid = company.cid' have to be extracted by checking FROM slot and added to the resulting SQL statement. These implicit join relations are extracted from relational database schema and encoded into domain knowledge as shown below.

(8) join(product, category, 'product.catid=category.catid').
join(product, company, 'product.cid=company.cid').

In this manner, the system can parse more complicated query sentences as in 9 with coordination and subordination into the result semantics of CCG derivation in 10.

(9) 29인치 텔레비젼과 바이오싱싱냉장고를 각각 한 대씩 사면 가격은 얼마나 되나요?
(29-in-chi theyl-ley-pi-cyen-kwa pa-i-o-sing-sing-nayng-cang-ko-lul kak-kak han-day-ssik sa-myen ka-kyek-un el-ma-na doy-na-yo?)
If (I) buy a 29-inch television set and Bio-Singsing refrigerator, what is the price for each of them?

(10) [product.price,product,cond[*,product&category,product.size=29&category.catname='TV']] and [product.price,product,cond[*,product&category,product.name='Bio-Singsing'&category.catname='refrigerator']]

Sentences with coordination may usually be syntactically and semantically ambiguous, and thus have multiple candidate readings (Park and Cho, 2000). (Lee and Park, 2001b) provides the interpretations, including the account for the ambiguity, for the queries with coordination. In addition, we can simply optimize the resulting statement. For example, in interpreting queries with coordination, if FROM and WHERE slots of translated semantics have the same attributes and conditions, the result can be a single statement whose SELECT clause contains the unified attributes from both SELECT slots of the resulting semantics.

In semantics 10, the coordinate relation 'and' is modeled as SQL function 'UNION' with intuitive meaning. While the nested relation 'cond' represents additional conditions from the meaning for subordinate item '면', containing meanings like 'if' in English, of word '사면' in 9. This relation is implemented as flattened up nested conditions except when there are correlated conditions between self referenced tables. Due to this relation, 10 can be translated into SQL statement 11 without any nesting.

```
(11) (SELECT product.price FROM product, category
     WHERE product.catid=category.catid and
           product.size=29 and category.catname='TV')
     UNION
     (SELECT product.price FROM product, category
     WHERE product.catid=category.catid and
           product.name='Bio-Singsing'
           and category.catname='refrigerator')
```

Nested relation 'among' from word markers, such as '중' (cwung) , '중에' (cwung-ey) and '가운데' (ka-wun-dey), which has meanings like 'among' in English, can be mapped to the set of results in nested query, similar to SQL function 'IN'. But 'IN' operator is not easily utilized due to the restrictions of object types. 'among' would be mapped to 'EXISTS' instead of 'IN', which is, however, not the correct mapping. In this case, since the expressive power of SQL is not enough, i.e. with no support for recursive structures in SQL92, it is difficult to translate such natural language queries into SQL statements correctly.

4.3 Interpretation into OQL Expressions

OQL is an SQL-like but its superset language for object-oriented database. This language provides object-oriented features, i.e., path expression, complex object, polymorphism, operation invocation, and so on.

OQL example 1c is translated with the schema mentioned in the previous sample database for SQL except that product class (which is known as table in SQL) has the object reference for category (catid) and company (cid) attribute. And each category and company class do not contain category ID and company ID attribute. In this schema, OQL statement 1c is translated from the resulting semantics in Figure 1 by finding path expression for the object and substituting the path to a renamed label. Since the path expression encodes an implicit join relation in SQL, OQL statement makes use of the path expression instead of join relation.

We locate this path from the domain knowledge which can be extracted from reference relations of the schema described in the object definition language. The extracted domain knowledge can be described as shown below.

```
(12) entry( product).
     path( category, 'product.catid').
     path( company, 'product.cid').
```

This resource encodes the query graph, which stands for object reference structure for the database and can be automatically generated by traversing the graph from entry class. In the resource 12, $entry(A)$ predicate represents the entry point and the class A has the path expression as the same name of class. $path(A, B)$ represents the fact that class A has the path expression B. We explain the translation steps as follows:

1. Sorting FROM slot of resulting semantics in the order of classes described in 12. This order is written in a list.
2. With the resource 12, we replace each class in the sorted FROM slot to path expression and define alias for the path expression with binding operator 'in' as shown in FROM clause of example 1c. Path expression of the class described in path predicate in 12 is replaced with pre-defined alias generated during this step.
3. We replace classes in SELECT and WHERE slots to aliases. So we obtain the resulting OQL statement 1c.

This path expression is used for representing complex object. For example, if we assume that category class is embedded in product class as a set without object reference (i.e. catid), we can obtain the category information with the same path expression shown above.

With the path expression, the example 10 can be translated to the corresponding statement in a form similar to OQL statement 1c with the 'UNION' function. And since binding operator 'in' is used with complex objects in OQL, if relation 'cond' in semantics 10 is replaced into 'among', the result can be correctly translated into OQL statement 13.

```
(13) (SELECT p.price FROM p in product WHERE a in
     (SELECT pp FROM pp in product, cat in product.catid
     WHERE pp.size=29 and cat.catname='TV') and a=p)
     UNION
     (SELECT p1.price FROM p1 in product WHERE a1 in
     (SELECT * FROM pp1 in product, cat1 in product.catid
     WHERE p1.name='Bio-Singsing'
           and cat1.catname='refrigerator') and a1=pp1)
```

4.4 Interpretation into CPL Expressions

CPL is a high-level query language providing access to relational databases and databases composed of structured data files. This is possible for the CPL's ability to represent complex objects. This feature makes it possible for the system to query more general databases. Representation of complex objects in CPL is similar to path expression of OQL. But CPL uses notations '<-', '<--', and '<---', respectively, for binding set, bag and list objects. Thus we need to know the types of objects to translate into CPL queries correctly.

Example 1d is translated with the schema modeled as product table containing category and company field as a set of records instead of category ID (catid) and company ID (cid) attributes in SQL schema. For the interpretation into 1d, the domain resource is described as in 14.

```
(14) cpl_entry( product).
     cpl_field( category, 'product.category', set).
     cpl_field( company, 'product.company', set).
```

This resource is extracted from the CPL data model, describing database name and all the complex objects such as set, bag and list in the database. *cpl_entry*(A) represents database name and entry point of the whole database regarded as a set of records. *cpl_field*(A, B, C) represents field name A with type C and path B from entry point. CPL syntax uses set notation such as {S|C}; Set S corresponds to SELECT part in SQL; Condition C corresponds to FROM and WHERE parts. From the resulting semantics in Figure 1, CPL statement can be translated by following similar steps for OQL statement except the set notation and using binding symbol, i.e. '<-', '<--', and '<---'.

Resulting semantics 10 can be translated into the corresponding CPL statement shown below.

```
(15) let \a ==
        { x.#price | \x<-product, \y<-x.#category,
              x.#size = 29, y.#catname = "TV"};
     let \a1 ==
        { x1.#price | \x1<-product, \y1<-y1.#category,
              x1.#name = "Bio-Singsing",
              y1.#catname = "refrigerator"};
     (union( a, a1));
```

In 15, we are assuming that `union(#1,#2)` is a pre-defined function like 'UNION'. In CPL interpretations, it is also possible to translate queries like 13 with a binding symbol.

5 Implementation

Figure 2 shows the implemented client/server system architecture. The system is divided into three parts: the query processing engine, the query generation module, and the client (web/application) interaction module.

The query processing engine is the core part for interpreting natural language with a lexicon and linguistic resources such as morphological analysis, unknown words resolution and word class inference for domain-related terminologies, for example, product code in e-commerce domain and protein name in bioinformatics domain. The query generation module directly translates the resulting semantics from query processing engine into expressions in target database languages such as SQL, OQL, and CPL. The system generates the database query with resources including syntactic information of the database language and domain-specific knowledge such as schema information and type/name of databases. The client interaction module communicates with a web-based or application-based clients and delivers the results of the user's query.

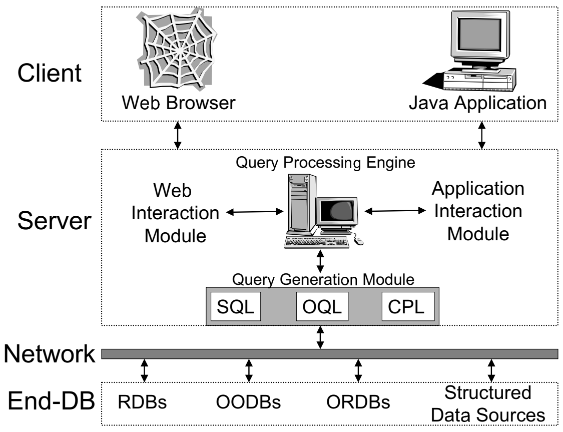


Figure 2: System Architecture

The server part of the system is implemented on UNIX in SICStus Prolog, Java and C. With this environment, we experiment the system with target database(s) composed of more than 30 thousand data entries from the shopping-mall websites. We have constructed the test corpus, with the help of 53 persons working on software-related issues, composed of 297 valid queries that contain compound nouns, some type of modifiers, various ellipses, numerical expressions, subordination, and coordination (Lee and Park, 2001b).

The system marked 708.03 msec as the average translation time for the average query with 6.6 space-delimited lexical items on the SUN Enterprise 250 with 1 GB main memory. The system was able to generate SQL expressions without any missing conditions and relations from 256 queries among 297 queries (i.e. precision 86.2%) with semi-automatically constructed 900 thousand lexical entries. Results from the remaining 41 queries are analyzed to contain errors that are mainly due to unregistered lexical entries in the domain, insufficient semantic information of the involved lexical entries and wrong categories described in the lexical entities. Lexical entries with insufficient semantics include special meanings in the domain such as the comparison by the *join* database operator and finegrained semantics for the compound nouns.

We expect that these three types of errors can be fully accounted for by expanding or modifying the categorial lexicon. The problem of unregistered lexical entries and wrong categories can also be dealt with by adding and modifying the corresponding entries to the lexicon, respectively. The problem of insufficient semantics can also be addressed by encoding the detailed semantics into the lexical entries, but it appears that in such a case, the semantics must be devised manually. Our system is expected to generate over 90% of the correct expressions when the lexicon is properly enhanced to contain no unregistered lexical entries. The other type of errors is the heuristic algorithm problem for product code identification during the stage of processing unknown words. This problem can be partially resolved by adding appropriate heuristic rules to the algorithm.

6 Discussion

In this work, we have described problems in interpreting the natural language queries to multiple heterogeneous databases and shown an implemented system in a combinatory categorial grammar framework (Lee and Park, 2001b). This system interprets natural language queries to queries in SQL, OQL, and CPL for a broad-scale access to the heterogeneous databases. The reason for choosing such database languages is that SQL is the most popular one and OQL is the ODMG standard language for object-oriented database. CPL is used as the core database query language for the Kleisli and TAMBIS systems which are known as successful access models to multiple heterogeneous databases consisting of mainly structured data sources in bioinformatics domain (Paton et al., 1999; Wong, 2000).

Since CPL is not familiar to the general public with a more mathematical style, these systems mainly provide form-based interfaces. While there are limitations in the form-based interfaces for highly complex data, natural language interfaces can be combined with CPL in a mutually beneficial way. Our system utilizes a query system like Kleisli, as a kind of middleware, so that it can be applied to many domains with heterogeneous data sources.

References

I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. 1995. Natural Language Interfaces to Databases - An Introduction. *Natural Language Engineering*, 1(1):29–81.

I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. 1998. Time, Tense and Aspect in Natural Language Database Interfaces. *Natural Language Engineering*, 4(3):229–276.

R. G. G. Cattell. 1996. *The Object Database Standard: ODMG-93.* Morgan Kaufmann.

J. Chae and S. Lee. 1998. Frame-based Decomposition Method for Korean Natural Language Query Processing. *Computer Processing of Oriental Languages*, 11(4):353–379.

A. Klein, J. Matiasek, and H. Trost. 1998. The treatment of noun phrase queries in a natural language database access system. In *COLING-ACL'98 workshop on the computational treatment of nominals*, pages 39–45.

H. Lee and J. C. Park. 2001a. Automatic Augmentation of Translation Dictionary with Database Terminologies in Multilingual Query Interpretation. In *ACL-EACL Workshop on Human Language Technologies and Knowledge Management*, pages 113–120.

H. Lee and J. C. Park. 2001b. Translating Natural Language Queries into Formal Language Queries with Combinatory Categorial Grammar. In *International Conference on Computer Processing of Oriental Languages*, pages 41–46.

J. Matiasek, A. Klein, and H. Trost. 1999. TAMIC-P: A System for NL Access to Social Insurance Databases. In *Applications of Natural Language to Information Systems*, pages 209–214.

X. Meng, S. Wang, and K. F. Wong. 2001. Overview of A Chinese Natural Language Interface to Databases: NChiql. *Computer Processing of Oriental Languages*, 14(3):213–232.

R. Nelken and N. Francez. 2000. Querying Temporal Databases Using Controlled Natural Language. In *COLING*, pages 1076–1080.

M. Palmer, D. Egedi, C. Han, F. Xia, and J. Rosenzweig. 1999. Constraining Lexical Selection Across Languages Using Tree Adjoining Grammars. In *TAG+3 Workshop Proceedings*, CSLI volume.

J. C. Park and H. J. Cho. 2000. Informed Parsing for Coordination with Combinatory Categorial Grammar. In *COLING*, pages 593–599.

N. W. Paton, R. Stevens, P. G. Baker, C. A. Goble, S. Bechhofer, and A. Brass. 1999. Query Processing in the TAMBIS Bioinformatics Source Integration System. In *11th Int. Conf. on Scientific and Statistical Databases (SSDBM)*, pages 138–147.

M. Steedman. 1996. *Surface Structure and Interpretation.* Number 30 in Linguistic Inquiry Monographs. MIT Press.

M. Steedman. 2000. *The Syntactic Process.* MIT Press.

C. A. Thompson and R. J. Mooney. 1999. Automatic Construction of Semantic Lexicons for Learning Natural Language Interfaces. In *AAAI/IAAI*, pages 487–493.

L. Wong. 2000. Kleisli, a Functional Query System. *Journal of Functional Programming*, 10(1):19–56.