University of Umeå
Department of Computing Science
SE-901 87 Umeå, Sweden

UMINF-04.15
ISSN-0348-0542
June 2004

Natural Language Access to Relational Databases through STEP

by

Michael Minock
Department of Computing Science
The University of Umeå, Sweden 90187
Phone: +46 90 786 6398 FAX: +46 90 786 6126
Email: *mjm@cs.umu.se*

ABSTRACT

This paper introduces the STEP system for natural language access to relational databases. In STEP the administrator couples phrasal patterns to elementary expressions within a decidable fragment of tuple relational calculus. This phrasal lexicon serves as a bi-directional grammar, enabling the generation of natural language from tuple relational calculus and the inverse parsing of natural language to tuple calculus expressions. This ability to both understand and generate natural language enables STEP to engage the user in clarification dialogs when the parse of their query is of questionable quality. The STEP system is implemented and is currently being evaluated over a geography database. Additional evaluations are planned and will also be available on STEP's website (`http://www.cs.umu.se/˜mjm/step`).

1 Introduction

The benefits of reliable natural language interfaces would indeed be numerous. Not only would they enable PC users to more easily access specific, detailed information, but they would create other interaction possibilities on devices ill-suited for WIMP [1]. Furthermore, when one considers integrating speech recognition (i.e. speech to text) and speech generation (i.e. text to speech), the breadth of applicability is even wider. Still, at the core, we need systems that are able to answer plain textual natural language questions with plain textual natural language answers. To get a grip on this problem, we focus on the case in which the information of user interest is within a relational database. Specifically we make the following assumptions:

1. The domain of discourse is bounded.
   a. The domain has a conceptual model (i.e. an ER diagram)
   b. The domain has a representation model (i.e. a set of relational tables)
   c. The state of the tables of 1.b is truthful.
2. Interaction is purely textual.
   a. User requests are single sentences of natural language.
   b. Answers are multiple sentences of natural language.
   c. Clarification dialogs occur when the system has difficulty understanding a user's request. Within clarification dialogs, users are limited to multiple choice, yes/no responses.
3. Administrators are clever, though constrained.
   a. Administrators understand primary and foreign keys, SQL, and tuple calculus.
   b. Administrators have detailed knowledge of the database's conceptual model, schema and state.
   c. Administrators do not have detailed linguistic understanding.

Historically [6] projects with such assumptions have aroused great interest within the relational database and computational linguistics communities [3]. Recently however, interest in the area has waned, due to the area's difficulty, the prevalence of WIMP interfaces and the popularity of the semi-structured web. Naturally we believe that a fresh look at this area is now warranted. And the system STEP [2] represents just such an attempt.

STEP exploits recent advances in classes of decidable logic [2][7], the schema tuple query assumption [10] and efficient first-order theorem provers(e.g. `spass.mpi-sb.mpg.de`). Specifically the class of logic STEP considers is both decidable for containment and equivalence, and, when

[1]Windows, Icons, Menus and Pointers.
[2]**S**chema **T**uple **Q**uery **P**rocessor

expressed in query form, is syntactically closed over set operations. These properties [16] are fundamental to our representation and processing of linguistic and relational knowledge. We also borrow notions of minimality and lexically based linguistic knowledge to simplify the configuration and maintenance of the linguistic knowledge within STEP.

1.1 Organization of this Paper

In section 2 we shall describe the *representations* underlying STEP. This includes the database schema, the type of logical queries supported in STEP, database answers, the phrasal lexicon and a simple thesaurus. Section 3 discusses how these representations are *processed* to support natural language generation and understanding. Section 4 describes the implementation of STEP, shows its interface and gives some performance results. Section 5 discusses this work in relation to prior work. Section 6 discusses on-going and future efforts to evaluate STEP. Section 7 gives conclusions.

2 Representations

In this section we informally discuss the representations upon which STEP rests. The formal descriptions of these representations are given in previous work [5][1][8][10] [12].

2.1 Relational Schemas

Given a database name and server address, STEP obtains the schema definition from the underlying database over which it is applied. This includes primary and foreign key constraints for each table as well as type information for each attribute. Currently the type system in STEP is rather weak; attributes are either `STRING` or `NUMERIC`. The following set of table definitions are a part of the Mondial database [9].

Example 1 *(Part of the MONDIAL Database Schema)*
Continent(name, area)
Country(name, code, **capital**, area, population)
Encompasses(**country**, **continent**, percentage)
City (name, **country**,population,longitude,latitude)
Ethnic_group (**country**, name, percentage)
Language (**country**, name, percentage)
Religion (**country**, name, percentage)
Borders (**country1**, **country2**, length)

where the primary keys are underlined and the foreign keys are shown in **boldface**.

2.2 Schema Tuple Expressions

STEP restricts its core representation of queries and answers to *schema tuple expressions* [10]. Informally schema tuple expressions are tuple relational calculus expressions over a single free tuple variable, limited to only existential quantification while admitting negation. Such expressions go well beyond the expressiveness of conjunctive queries [4] and, under certain restrictions, are a a variant of the guarded fragment of first order logic [12]. The key offerings of schema tuple expressions is that they are: 1.) decidable for satisfiability; 2.) decidable for containment; 3.) closed over syntactic difference.

2.2.1 Schema Tuple Queries

To represent *"the non-Asian countries with more than 100000000 people"*, we use the following schema tuple query expression:

$$\begin{array}{l}\{x|Country(x)\wedge \\ \quad \neg(\exists y_1)(Encompasses(y_1)\wedge y_1.country = x.code \wedge y_1.continent = \text{“Asia”})\wedge \\ \quad x.population > \text{“100000000”}\}.\end{array}$$

Which corresponds to the SQL:

```
SELECT *
FROM  COUNTRY as x
WHERE NOT exists(
  SELECT *
  FROM   ENCOMPASSES as y1
  WHERE  y1.country = x.code and y1.continent = 'Asia') AND
    x.population > '100000000';
```

We have adopted a compact `ASCII` based representation of schema tuple expressions that we shall use throughout this paper. The query above is specified as:

```
{x | Country(x),
     !{Encompasses(y1), y1.country=c.code, y1.continent='Asia'},
     x.population > 100000000}
```

It should be noted that, although the core queries are of the schema tuple type, STEP allows for simple projection, aggregate functions and counts. For example the names of all countries may be obtained through `{x.name | Country(x)}`, the average area of all countries through `{AVG(x.area) | Country(x)}` and the total number of countries through `|{x | Country(x)}|`. Finally we may also specify queries that simply check the truth of an expression, for example `{Country(x), x.population>1000000000}`. We term standard queries as *tuple* queries, projection and simple [3] aggregation queries as *value* queries, queries computing counts as *count* queries and queries seeking to determine the truth of an expression as *truth* queries.

[3]The form of answer aggregation within schema tuple query expressions is weaker than the complex GROUP BY ... HAVING constructs of SQL. Still, in every day language, these constructs are hard to express in single sentences of natural language.

2.2.2 Answers

Answers are the tuple sets that are returned from queries. Normally answers are simply sets of tuples for which the query is true. However in STEP answers are nested structures that supply the joined tuples that establish the answer tuple's membership in the query. For example the query:

```
{x | Country(x),
  {City(y), x.capital=y.code, y.population > 10000000}}
```

Returns the nested[4] tuples:

```
{{Country(x), x.name='South Korea', x.code='ROK',
   x.capital='Seoul', x.area=98480, x.population=45482291,
   {City(y), y.name='Seoul', y.country='ROK',
      y.population=10229262, y.logitude=127.0, y.latitude=37.6}}
  ...}
```

Syntactically the notation for an answer is a schema tuple expression. This enables the same generation mechanism to be employed to describe queries and answers alike.

2.3 The Phrasal Lexicon

The phrasal lexicon is the primary knowledge representation that administrators must author to tie STEP to their domain database. In short, the administrator authors a set of *entries* to cover the domain schema. An entry associates a single elementary schema tuple expression with a set of *patterns*, each pattern providing an alternative way to express the elementary expression. Because of the decidability of containment over schema tuple expressions, the entries are compiled into a *subsumption hierarchy* which organizes the phrasal lexicon for processing and inspection. We shall now delve into the finer details of these representations.

2.3.1 The Entries

We shall illustrate the form of entries in the phrasal lexicon mainly through examples. Here we start with a very simple entry:

```
<{x | Country(x)}:
   H[pl    :'countries'],
   H[sing  :'country'],
   H[pl    :'nations'],
   H[sing  :'nation']>
```

[4]There is a choice in how to represent answers in nested form. Either all the bindings that make a tuple satisfy the query are supplied, or only one such binding is supplied. For efficiency concerns we have chosen the later policy.

In this entry the elemental expression is `{x | Country(x)}` coupled with four patterns. These pattern are all of type *head*, signified with `H[...]`. The other pattern types, which we shall see below, are *modifier* (`M[...]`) and *complement* (`C[...]`) patterns. The *features*[5] `sing` and `pl` control the applicability of these patterns.

As a slightly more complicated example we have the entry:

```
<{x | Country(x),x.population < $c1}:
   C[   :'with population less than $c1']
   C[   :'with less than $c1 people']
   C[   :'with fewer than $c1 people']>
```

Here we see a more complex elemental pattern that includes a *template parameter*, signified `$c1`. Note that, in English at least, the same complement serves in case of singular or plural so there is no need to condition on those features. Also note that the pattern types are complements, meaning that their text will follow the that of the head (e.g. "Countries *with fewer than 1000000 people*".).

We encounter an even more complex elemental pattern in the next entry:

```
<{x | Country(x),
        (Encompasses(y1),y1.country=x.code,y1.continent= $c1}}:
  C[    :'in the continent $c1']>
```

Here we see joins being introduced in the elemental tuple expression.

Some entries call for sub-descriptions within their patterns. This is so in the following entry:

```
<{x | Country
        {Borders(y1),Country(y2),
          y1.country1=x.country,y1.country2=y2.code,*}}:
   C[   :'that border D({y2|Country(y2),*},(indef pl)']>
```

The special function `D` yields a description of a query under a given set of features. The symbol `*` stands for any arbitrary tuple expression fragment appearing at the given position.

The following entry handles projection over the attribute `population`.

```
<{x.population | Country(x),*}:
   H[sing    :'population of D({x|Country(x),*},{indef sing})'],
   H[pl      :'populations of D({x|Country(x),*},{indef pl})']>
```

2.3.2 The Subsumption Hierarchy

Figure 1 shows the hierarchy that is automatically compiled from entries in the phrasal lexicon. This hierarchy is instrumental in both generation and understanding processes covered in the section 3. The compilation process often expose errors that administrators might have made during populating or refining the phrasal lexicon.

[5] The possible features for the current English version of STEP are: `sing` for singular, `pl` for plural, `def` for definite, `indef` for indefinite, and `u` to signal that a pattern may only be used for understanding, not generation. This set is expected to grow in the coming months, but hopefully not by much.

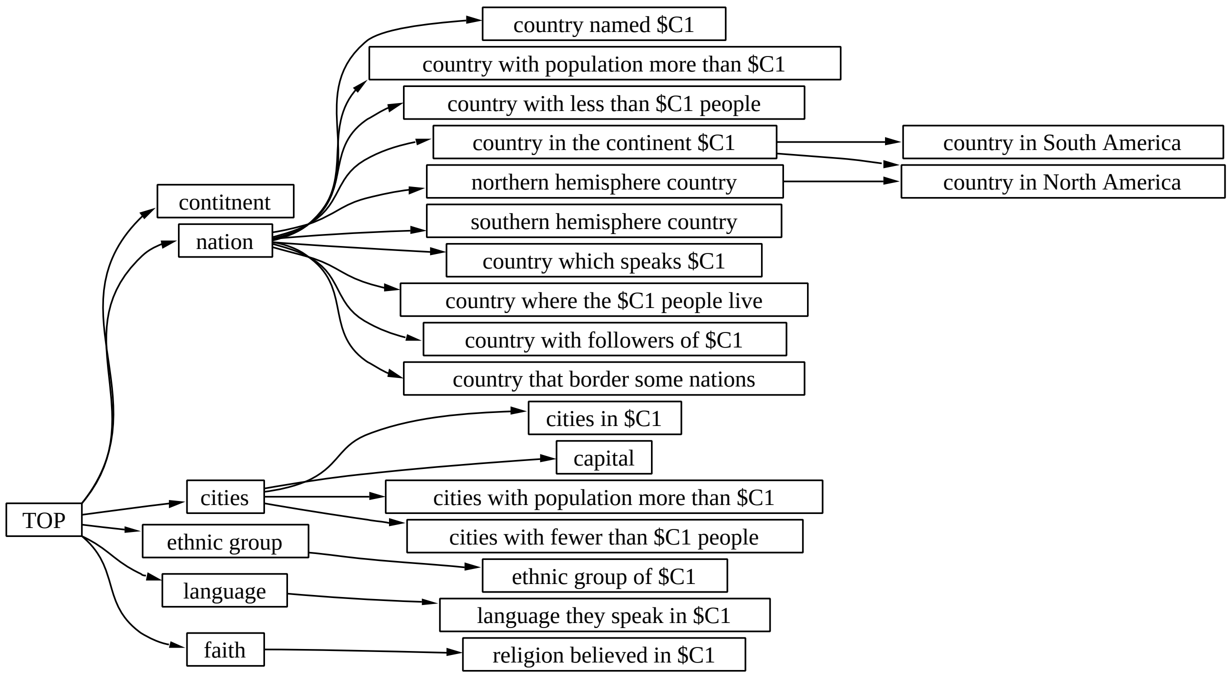

TOP
contitnent
nation
cities
ethnic group
language
faith
country named $C1
country with population more than $C1
country with less than $C1 people
country in the continent $C1
northern hemisphere country
southern hemisphere country
country which speaks $C1
country where the $C1 people live
country with followers of $C1
country that border some nations
country in South America
country in North America
cities in $C1
capital
cities with population more than $C1
cities with fewer than $C1 people
ethnic group of $C1
language they speak in $C1
religion believed in $C1

Figure 1: Subsumption Hierarchy.

2.3.3 Word Hashes

The first words of modifier, head and complement patterns are hash indexed to their corresponding patterns during the construction of the subsumption hierarchy. These hash-indices are essential to making the parsing process efficient.

2.4 Simple Word Lists

2.4.1 Attribute Value Arrays

At start-up, STEP accesses the database to build hash tables containing the value for each string typed, non-foreign key attribute. These attribute value hashes help guide parsing and also give a very quick way to identify query misconceptions (e.g. “a country named Africa.”).

2.4.2 Thesaurus

The thesaurus within STEP is very simple. It lists synonyms and translations for underlying database values. In all cases there is a distinguished value that is the one used in the actual database.

2.5 The Language Model

Though the administrator is not expected to work with the underlying language model, the specification of basic sentence forms is template based and is relatively easy to specify. Below we show a portion of the templates for English.

```
<TUPLE-QUERY: 'FETCH SPEC(*) D(QUERY,*)'.>
<TRUTH-QUERY:  'BE(*) there SPEC(*) D(QUERY,*)?'>
<COUNT-QUERY:  'How many  D(QUERY,*+pl) BE(*+pl) there?'>
<VALUE-QUERY:  'WH-WORD BE(*) SPEC(*) D(QUERY,*)?'>

<TUPLE-QUERY ANSWER: 'SPEC(*) D(QUERY,*) BE(*) D(ANSWER,*).''>
<TRUTH-QUERY TRUE: 'Yes, there BE(*) D(QUERY,*).'>
<TRUTH-QUERY FALSE: 'No, there BE(*) not D(QUERY,*).'>
<COUNT-QUERY NUM: 'There BE(*) NUMBER(NUM,*) D(QUERY,*).'>
<VALUE-QUERY ANSWER: 'D(ANSWER,*) BE(*) SPEC(*) D(QUERY,*).'>
```

In these templates, all-caps symbols stand for various word (phrase) sets. For example `FETCH` denotes ‘List’, ‘Give me’, ‘Print’, etc. `WH-WORD` denotes ‘Which’,‘Where’,‘Who’, etc. The `*` symbol stands for feature sets which must agree across the entire sentence. `*+feature` asserts that `feature` is within the feature set. Naturally the word sets may be indexed through features as in `SPEC(*)`. Based on what the feature set is, this could yield ‘a’,‘the’,‘some’,‘any’, etc. Places where query and answer descriptions are substituted are specified with `D(QUERY,*)` and `D(ANSWER,*)`.

3 Processing

Now that we have described the representations underlying STEP, we shall describe the generation process that maps query expressions to natural language sentences, and, inversely, the understanding process that maps natural language sentences to query expressions.

3.1 Generation

The generation model of STEP is described in [11][13] and thus, for reasons of space, we refer interested readers to these publications for details. In summary however, generation works by semantically sorting a schema tuple expression into the subsumption hierarchy, fetching the patterns of its immediate parents, and combining such patterns so that features agree and the resulting phrase consists of a set of modifiers, followed by a single head, followed by a set of complements. It should be noted that this process is complicated by the fact that generation may be recursive.

3.2 Understanding

Our approach to understanding natural language is to reformulate it as a classical state-space search problem. The initial state of this search is the input sentence and the goal state is a query expression that could have generated the input sentence. Intermediate states have a query that accounts for a prefix of the input sentence as well as the remainder of the sentence yet to be parse. States also have an accrued *cost* which is the sum cost of how much ‘fudging’ was required to match input words with phrasal patterns. Finally states include some book keeping structures including a reference stack of introduced tuple variables and a status of either `head-seeking`, `complement-seeking`, or `sentence-token-seeking`. The operators that determine successor states are divided into three types: *pattern matching*, *stack* and *'fudging'*.

3.2.1 The Matching Operators

Depending on status, matching operators match a modifier, head, complement or sentence pattern to the prefix of the remainder of the sentence. The system uses a left corner type strategy, because each pattern, as stated in section 2.3, is indexed by a first word token. When matching operators apply, they consume tokens in the remainder of the sentence and incorporate the elementary tuple expression of the matched pattern into the candidate query expression. Based on the type of match, the status may be altered (e.g. from `head-seeking` to `complement-seeking`) and new variables may be pushed onto the reference stack.

3.2.2 The Stack Operators

The stack operators pop reference variables from the stack. For example in the parse of "Cities in countries in Europe *above the 50th parallel*", the variable for the country in Europe must be removed from the reference stack so that the complement may attach to the variable for city. Another similar type of operator is the one that pops a query description, shifting status from complement seeking to `sentence-token-seeking`.

3.2.3 The 'Fudging' Operators

The 'fudging operators' help soften up the sentence so that matching operators might apply. They do this through adding a word to the remainder of the sentence or by dropping the first word on the remainder of the sentence. The costs associated with such actions are dependent on the type of words involved. For example adding the preposition 'in' is relatively cheap, while adding 'not' is quite expensive. And dropping a preposition is cheaper than dropping an unrecognized word which, in turn, is much cheaper than dropping a word that matches an attribute value domain.

Given this search based formulation, we employ uniform cost search [15] to find the least cost parse of the query. Solutions exceeding a certain cost must be paraphrased back to the user for confirmation. Solutions exceeding an even greater cost are deemed complete failures. Semantically distinct solutions within a fixed cost of the best solution are presented as rival parses.

4 The STEP System

Figure 2 shows the architecture of STEP. STEP is currently about 10,000 lines of LISP code and is run as a server that is called through CGI from a web browser. STEP issues satisfiability queries to the SPASS theorem prover and relational queries to a back-end PostgreSQL database. At start

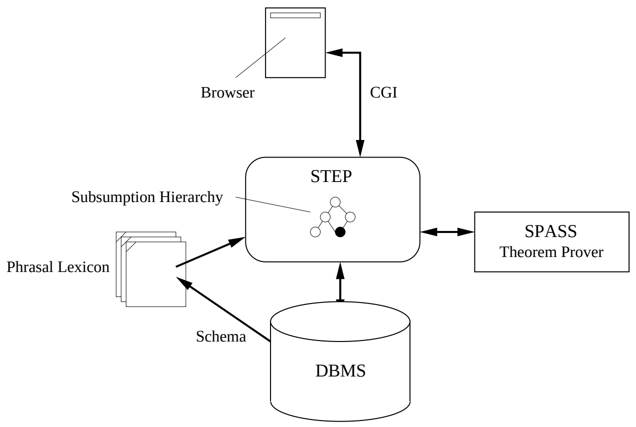
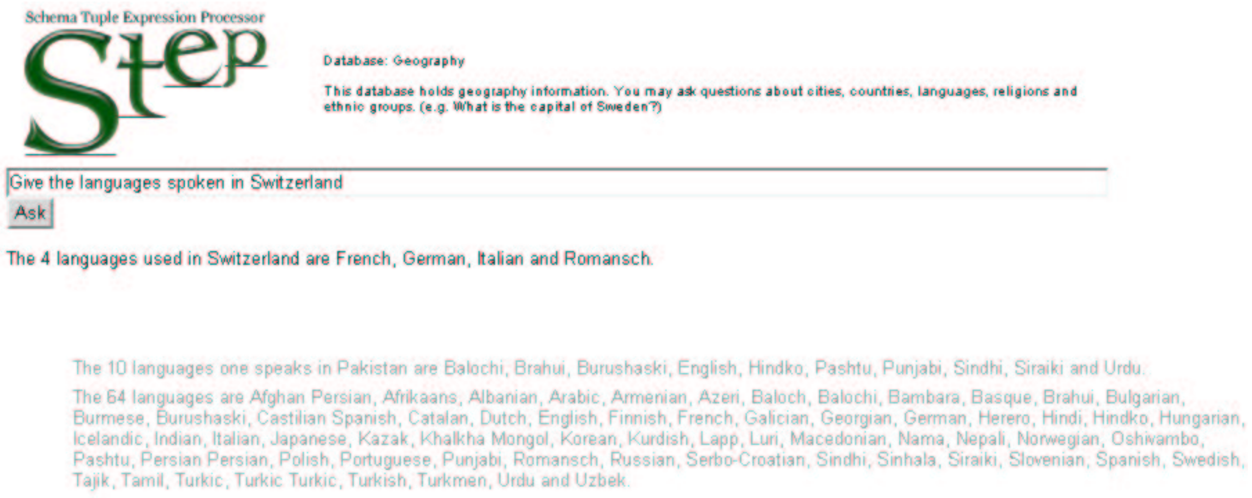


Figure 2: System Architecture.

up time STEP compiles the phrasal lexicon into a subsumption hierarchy, builds the token hash-indices into the patterns and obtains the attribute value arrays from the database. The system is then ready to accept browser-submitted requests.



Database: Geography

This database holds geography information. You may ask questions about cities, countries, languages, religions and ethnic groups. (e.g. What is the capital of Sweden?)

Give the languages spoken in Switzerland

The 4 languages used in Switzerland are French, German, Italian and Romansch.

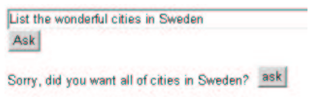
The 10 languages one speaks in Pakistan are Balochi, Brahui, Burushaski, English, Hindko, Pashtu, Punjabi, Sindhi, Siraiki and Urdu.

The 64 languages are Afghan Persian, Afrikaans, Albanian, Arabic, Armenian, Azeri, Baloch, Balochi, Bambara, Basque, Brahui, Bulgarian, Burmese, Burushaski, Castilian Spanish, Catalan, Dutch, English, Finnish, French, Galician, Georgian, German, Herero, Hindi, Hindko, Hungarian, Icelandic, Indian, Italian, Japanese, Kazak, Khalkha Mongol, Korean, Kurdish, Lapp, Luri, Macedonian, Nama, Nepali, Norwegian, Oshivambo, Pashtu, Persian Persian, Polish, Portuguese, Punjabi, Romansch, Russian, Serbo-Croatian, Sindhi, Sinhala, Siraiki, Slovenian, Spanish, Swedish, Tajik, Tamil, Turkic, Turkic Turkic, Turkish, Turkmen, Urdu and Uzbek.

Figure 3: Basic Querying.

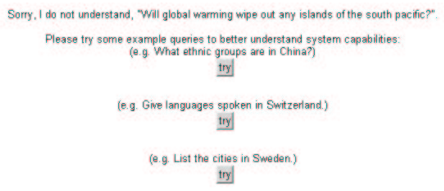
Figure 3 shows the state of the web-based interface for a user session over STEP. Note that the current database is the `Geography` database, which has a simple paragraph description. Users enter their queries on the input field and obtain answers in the area immediately below. Also shown are the results from prior queries. These are retained so that the user may cut and paste results to other documents and so that the user is reminded of the value constants understood by the system.

Figure 4a shows the result of a noisy parse. In this case the system has guessed, more or less, what the user is asking for. In figure 4b we see the result of a query that the system is unable to even guess a parse for. In this case the system presents the user with several example queries to remind the user of what types of queries may be answered.


List the wonderful cities in Sweden

Sorry, did you want all of cities in Sweden?

(a) Confirmation


Sorry, I do not understand, "Will global warming wipe out any islands of the south pacific?".

Please try some example queries to better understand system capabilities:
(e.g. What ethnic groups are in China?)

(e.g. Give languages spoken in Switzerland.)

(e.g. List the cities in Sweden.)

(b) Failure

Figure 4: Clarification Dialogs.

Overall, the STEP system has reasonable performance. From the user's perspective the overall response time is the important measure, and this depends on many factors in the networked database case. Typically however, with STEP running on a SPARC 10 over a relatively efficient network, response times are in the neighborhood of 2 to 5 seconds.

From the point of view of STEP, the overriding cost is the cost of satisfiability tests issued to SPASS. With careful optimization the average satisfiability test for common queries is now around 0.14 seconds on a SPARC 10. To give a general sense of the practical performance of the system, note that it takes around 48 seconds of computer time on a SPARC 10 to build the phrasal lexicon of figure 1 - a task that involves 269 satisfiability calls to SPASS.

5 Related Work

A comprehensive review of the pre-mid 90's work done in natural language interfaces over databases appears in [3]. In relation to the work covered there, STEP shares an almost identical vision as Codd's RENDEZVOUS system [6]. In fact some of Codd's seven steps appear almost verbatim in section 1 of this paper.

In contrast to prior work however, STEP is unique in that it restricts the form of the queries to a highly expressive class of schema tuple queries [12]. This restricted class has the desirable property of being decidable for satisfiability and subsumption while being closed over basic syntactic set operations of difference, complementation and union [10]. These properties have been exploited in STEP's approach to generate natural language generation. Of note STEP's generation component is the only 'semantic' relational query paraphraser [13], where by 'semantic', we mean that semantically equivalent though syntactically different queries will necessarily map to equivalent sets of natural language descriptions.

Though recently the research community has largely ignored the problem of natural language interfaces to database, Microsoft has fielded the English Query product. STEP is somewhat similar to English Query in that an administrator supplies phrasal attachments to the schema, however, unlike STEP, English Query does not address the paraphrasing problem at all and seems to be using SQL as its core formal query language. Furthermore answers are presented as tables - the result of evaluating SQL. While this may be adequate for many tasks, it is not clear how such a system should be modified to meet assumption 2 of section 1.

Another recent system has been the PRECISE system [14]. PRECISE addresses the problem of reliability through the notion of *semantically tractable queries*. In essence, the idea is that most user questions are simple enough that a unique match exists between the user's question and names for database table, attributes and values. The strategy behind PRECISE is to accept only such semantically tractable queries, while requiring the user to restate other queries. Additionally the system has the added benefit of obtaining much of its configuration automatically from the database over a marked up schema. PRECISE is impressive, however it is unclear how it will be able to handle complex, truly ambiguous queries and whether it will ultimately address the paraphrasing or answers presentation problem.

6 Evaluation

Initial user experiences with STEP have been promising. Users appear able to actually use the system. Unfortunately users don't distinguish between the quality of the database and the quality of the interface, thus we find ourselves spending time cleaning up the geography database. Still progress is steady and we anticipate wider and wider use experiments through our web interface. Through careful reading of systems logs and direct user feedback, we shall refine the design of the interface, improve coverage of the phrasal lexicon, and generally improve the performance and stability of the system.

Of course a second type of evaluation that we require is to assess how easy STEP is to author over new databases. Because of the difficulty of 'selling' untested technology into organizations, we intend to test the administration properties of STEP in an advanced database course next school year. Students will be asked to author a phrasal lexicon to cover a database schema in their third system assignment. Optionally, the students will be invited to integrate STEP into their term projects. Through this experience, we will better document and test the system, preparing for a possible future release of the STEP system.

7 Conclusions

This paper has introduced the STEP system. STEP leverages recent advances in the classes of decidable logic, lexically bound linguistic knowledge and the schema tuple query assumption to take a fresh pass at the classical problem of natural language access over databases. The bi-directionality of STEP's phrasal lexicon enables query paraphrases to be presented to users when their questions parse with difficulty. STEP is actively being evaluated for usability over a geography database, accessible at STEP's website (`http://www.cs.umu.se/~mjm/step`).

8 Bibliography

References

[1] S. Abiteboul, R. Viannu, and V. Hull. *Foundations of Database Systems 3rd edition*. Addison Wesley, 1995.

[2] H. Andreka, J. van Benthem, and I. Nemeti. Modal languages and bounded fragments of predicate logic. *Journal of Philosophical Logic*, 27:217–274, 1998.

[3] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. Natural language interfaces to databases–an introduction. *Natural Language Engineering*, 1(1):29–81, 1995.

[4] A. Chandra and P. Merlin. Optimal implementation of conjunctive queries in relational databases. In *Proc. 9 of the ACM Sym. on the Theory of Computing*, pages 77–90., 1977.

[5] E. Codd. Relational completeness of data base sublanguages. In R. Rustin, editor, *Database Systems*, pages 33–64. Prentice-Hall, 1972.

[6] E. Codd. Seven steps to rendezvous with the casual user. In *IFIP Working Conference Data Base Management*, pages 179–200, 1974.

[7] E. Grädel. On the restraining power of guards. *Symbolic Logic*, 64:1719–1742, 1999.

[8] M. Levene and G. Loizou. How to prevent interaction of functional and inclusion dependencies. *Information Processing Letters*, 71:115–125, 1999.

[9] W. May. Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität at Freiburg, Institute for Informatik, 1999.

[10] M. Minock. Knowledge representation using schema tuple queries. In *KRDB '03*. IEEE Computer Society Press, 2003.

[11] M. Minock. A phrasal generator for describing relational database queries. In *Proc. of the 9th EACL workshop on natural language generation*, Budapest, Hungary, April 2003.

[12] M. Minock. Managing fine-grained representations of completeness. Technical Report 04.05, The Univeristy of Umeå, Umeå, Sweden, February 2004.

[13] M. Minock. Modular generation of relational query paraphrases (to appear). *Journal of Language and Computation special issue on Formal Aspects of NLG*, 2004.

[14] A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Intelligent User Interfaces*, 2003.

[15] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, 2003.

[16] S. Shieber. The problem of logical-form equivalence. *Computational Linguistics*, 19(1):179–190, 1994.