

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

UMINF-05.09
March 2005

A Phrasal Approach to Natural Language Interfaces
over Databases

by

Michael Minock

ABSTRACT

This report introduces the STEP system for natural language access to relational databases. In contrast to most work in the area, STEP adopts a phrasal approach; an administrator couples phrasal patterns to elementary expressions of tuple relational calculus. This ‘phrasal lexicon’ is used bi-directionally, enabling the generation of natural language from tuple relational calculus and the inverse parsing of natural language to tuple calculus. 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. An on-line demonstration of STEP is accessible at <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 new interaction possibilities on devices ill-suited to windows, icons, menus and pointers. To get a grip on this problem, we focus on the case in which the information of user interest is within a relational database and interaction is purely textual; user requests are single sentences of natural language and answers are multiple sentences of natural language. Clarification dialogs, when necessary, are limited to multiple choice, yes/no responses from the user.

Historically projects with such assumptions have aroused great interest within the relational database and computational linguistics communities [3, 7]. The system RENDEZVOUS [6] was perhaps the first such system, but it is doubtful whether it was ever completed given the limited facilities of the time. In any case, and to the point here, Codd is very explicit about requiring a system to engage the user in clarification dialogs; *single shot* systems where requests are parsed to a formal language and then immediately applied to a back-end database were deemed likely to be misinterpreted and misprized by users. At the very least a system should be able to paraphrase the user's query back to them during answer presentation or ambiguity resolution. In short, *any practical natural language interface over databases must have query paraphrasing capabilities.*

Now it is not proper to say that no systems after RENDEZVOUS focussed on the paraphrasing problem, some did [10] [9]. But as systems transitioned from semantic grammars toward portable systems leveraging domain independent grammars [8], the focus on generating query paraphrases tended to be discarded. Though the idea of doing a full integration of these domain independent facilities is appealing, it is the opinion of this author that this is not yet feasible. In response to this, STEP¹, the system described in this report, adopts a phrasal approach to the configuration and maintenance of linguistic knowledge. Specifically the administrator authors a *phrasal lexicon* by coupling phrasal patterns to elementary expressions of a class of tuple calculus. This 'phrasal lexicon' is used bi-directionally, enabling the generation of natural language from tuple relational calculus and the inverse parsing of natural language to tuple calculus. 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 remainder of this report is organized as follows. In section 2 we shall informally² describe the *representations* underlying STEP. This includes³ the database schema, a decidable class of tuple query expressions, the phrasal lexicon and a set of sentence templates. Section 3 discusses how these representations are *processed* to support natural language generation and understanding. Section 4 describes our implementation and experiences with an on-line demonstration of STEP. Section 5 discusses this work in relation to prior work and section

¹Schema Tuple Expression Processor

²The formal descriptions of these representations are given in [1, 12, 14].

³We omit a discussion of the hash tables that contain attribute values, synonyms and parser speed up indices.

6 concludes.

2 Representations

2.1 Relational Schemas

STEP obtains schema information from the underlying database. This includes primary and foreign key constraints for each table as well as attribute type information. The following table definitions⁴ are a small part of the Mondial database [11].

Country(name, code, capital_{City}, population)

In_Continent(country_{Country}, continent)

Borders(country1_{Country}, country2_{Country})

City(name, country_{Country}, population)

2.2 Query Expressions

STEP restricts its core representation to a class of *tuple calculus*. Tuple calculus is in fact just a ‘syntactic sugar’ over first order logic that allows variables in queries to range over entire tuples, not over the values in the i -th place of a given predicate. In any case tuple calculus has a direct translation to classical first order logic (see [17]) and SQL for that matter.

Informally the given class of tuple calculus [12] are queries over a single free tuple variable where conditions are either simple or signed vectors of existentially quantified variables over conjunctions of simple or join conditions (See [12, 14]). By example the following two queries are within the restricted class: 1.) $\{x \mid \text{Country}(x) \wedge x.\text{population} < 10,000,000 \wedge \neg(\exists y_1) (\text{In_Continent}(y_1) \wedge y_1.\text{country} = x.\text{code} \wedge y_1.\text{continent} = \text{‘Asia’})\}$; 2.) $\{x \mid \text{Country}(x) \wedge (\exists y_1) (\exists y_2) (\exists y_3) (\text{Borders}(y_1) \wedge \text{Country}(y_2) \wedge \text{In_Continent}(y_3) \wedge y_1.\text{country1} = x.\text{code} \wedge y_1.\text{country2} = y_2.\text{code} \wedge y_2.\text{code} = y_3.\text{country} \wedge y_3.\text{continent} = \text{‘Asia’})\}$.

STEP extends this core class to allow for simple projection and aggregation⁵. For example the names of all countries may be obtained through $\{x.\text{name} \mid \text{Country}(x)\}$ and the average population of all countries through the expression $\{\text{AVG}(x.\text{population}) \mid \text{Country}(x)\}$. Hereafter when we refer to a ‘query’, we are referring to the restricted class [12], possibly extended with simple projection or aggregation. Queries of this sort go beyond the expressive power of conjunctive queries and are decidable for satisfiability⁶ and query containment [12, 4].

It should be noted that ‘logical form’ in STEP are such queries with additional *pragmatic features* attached. For example if the feature *count* is associated with a query, then it

⁴Primary keys are underlined and the foreign keys subscripted by their home relation.

⁵The form of answer aggregation here is weaker than complex GROUP BY ... HAVING constructs of SQL. Still, in every day language, such complex constructs are hard to express in single sentences of natural language.

⁶It may be decided if there exists a legal database for which the query returns answers.

is understood that the number of tuples are to be presented as the answer. Also it should be noted that the relations used within these expressions may be introduced ‘dummy’ relations that broaden the schema to cover additional domain and pragmatic concepts not (yet) covered by the database.

2.3 The Phrasal Lexicon

Administrators author the phrasal lexicon by specifying a set of *entries* to cover the schema. An entry is of the form $\langle q : p^+ \rangle$ where q is a query expression and p^+ is one or more phrasal *patterns* that express the meaning of q . Patterns specify either *head*, *modifier*, *complement* or *answer* material. In the phrase ‘Asian countries with less than 10,000,000 people’, ‘Asian’ is a modifier, ‘countries’ is a head and ‘with less than 10,000,000 people’ is a complement. Answer material indicates the manner in which answers to queries should be presented back to users. We shall illustrate different kinds of entries through examples.

$$\begin{aligned} \langle \{x|Country(x)\} : & \\ & Head[\text{‘countries’}](pl) \\ & Head[\text{‘country’}](sing) \\ & Head[\text{‘nations’}](pl) \\ & Head[\text{‘nation’}](sing) \end{aligned} \tag{e1}$$

In the entry e1 the elemental expression $\{x|Country(x)\}$ is coupled with four head patterns where features *sing* and *pl* indicate singular and plural forms.

$$\begin{aligned} \langle \{x|Country(x) \wedge x.population < c_1\} : & \\ & Complement[\text{‘with population less than } c_1 \text{’}] \\ & Complement[\text{‘with less than } c_1 \text{ people’}] \\ & Complement[\text{‘with fewer than } c_1 \text{ people’}] \end{aligned} \tag{e2}$$

Entry e2 includes a *template parameter* (c_1) that associates a value in the query with text in a pattern. In English the same complement serves for singular or plural here, so there is no need to condition on such features.

$$\begin{aligned} \langle \{x|Country(x) \wedge & \\ & (\exists y)(In_Continent(y_1) \wedge y_1.country = x.code \wedge y_1.continent = \text{‘Asia’})\} : & \\ & Modifier[\text{‘Asian’}] & \\ & Complement[\text{‘of Asia’}] \end{aligned} \tag{e3}$$

Entry e3 has an elemental expression with joins as well as an example of a modifier pattern.

$$\begin{aligned} \langle \{x|Country(x) \wedge & \\ & (\exists y_1)(\exists y_2)(Borders(y_1) \wedge Country(y_2) & \\ & \wedge y_1.country1 = x.country \wedge y_1.country2 = y_2.code \wedge *_{y_2}) : & \\ & Complement[\text{‘that border } ref(\{y_2|Country(y_2) \wedge *_{y_2}\}) \text{’}](pl) & \\ & Complement[\text{‘that borders } ref(\{y_2|Country(y_2) \wedge *_{y_2}\}) \text{’}](sing) \end{aligned} \tag{e4}$$

Entry e4 calls for sub-descriptions within its patterns, denoted by the function *ref* being

applied over an open sub-formula of the query expression. The symbol $*_{y_2}$ stands for a conjunction of conditions over the tuple variable y_2 relative to a query. For example relative to the second example query of section 2.2, $*_{y_2}$ would be $(\exists y_3)(In_Continent(y_3) \wedge y_2.code = y_3.country \wedge y_3.continent = \text{“Asia”})$. STEP extracts and merges such open sub-formulas during generation and understanding.

$$\langle \{x.population | Country(x)\} : \tag{e5}$$

$$\text{Head}[\text{‘populations’}](pl)$$

$$\text{Head}[\text{‘population’}](sing)$$

$$\text{Head}[\text{‘number of people’}](sing)\rangle$$

Entry e5 gives a name to the projection of the attribute *population* from the relation *Country*. Such entries enable users to refer directly to attributes as if they were concepts.

$$\langle \{x.population | Country(x) \wedge *_{x}\} : \tag{e6}$$

$$\text{Complement}[\text{‘of ref}(\{x | Country(x) \wedge *_{x}\})\text{’}]\rangle$$

Entry e6 names the relationship of the attribute *population* to that of the table for *Country*. Such bridging entries enable complements to attach to simple projection queries.

$$\langle \{x | City(x) \wedge x.name = c_1 \wedge \tag{e7}$$

$$(\exists y_1)(Country(y_1) \wedge x.country = y_1.code \wedge y_1.name = c_2)\} :$$

$$\text{Answer}[\text{‘}c_1, c_2\text{’}](value)$$

$$\text{Answer}[\text{‘}c_1 \text{ is located in the country } c_2\text{’}](where)\rangle$$

Finally entry e7 specifies how answers are to be presented. The first pattern states that to identify a city tuple in an answer it is sufficient to place the city names followed by the country name, as in ‘Umeå, Sweden’. The second pattern shows the response to a *where* typed query over cities.

The decidability of containment between value query expressions enables the compilation of a set of entries into a subsumption hierarchy. Such a hierarchy makes the semantic relationships within the phrasal lexicon explicit and thus helps validate the configuration process. Additionally this process can be viewed as a reverse engineering of the conceptual model from the underlying representational schema. Figure 1 shows the subsumption hierarchy for the portion of the mondial schema provided in section 2.1. The entries specified above are highlighted.

2.4 Sentence Templates

Though the administrator is not expected to work with the underlying sentence templates, below we show a portion of the templates for English.

$$\langle q(value) :$$

$$\text{Template}[\text{‘ref}(q)\text{’}]$$

$$\text{Template}[\text{‘(list|give me|...) \cdot ref}(q).\text{’}]$$

$$\text{Template}[\text{‘(which|what) \cdot ref}(f) \cdot (\text{is|are|do|does|has|have}) \cdot gapped(q, f)?\text{’}] \dots \rangle$$

We show here three basic template patterns associated with retrieving the literal answers to

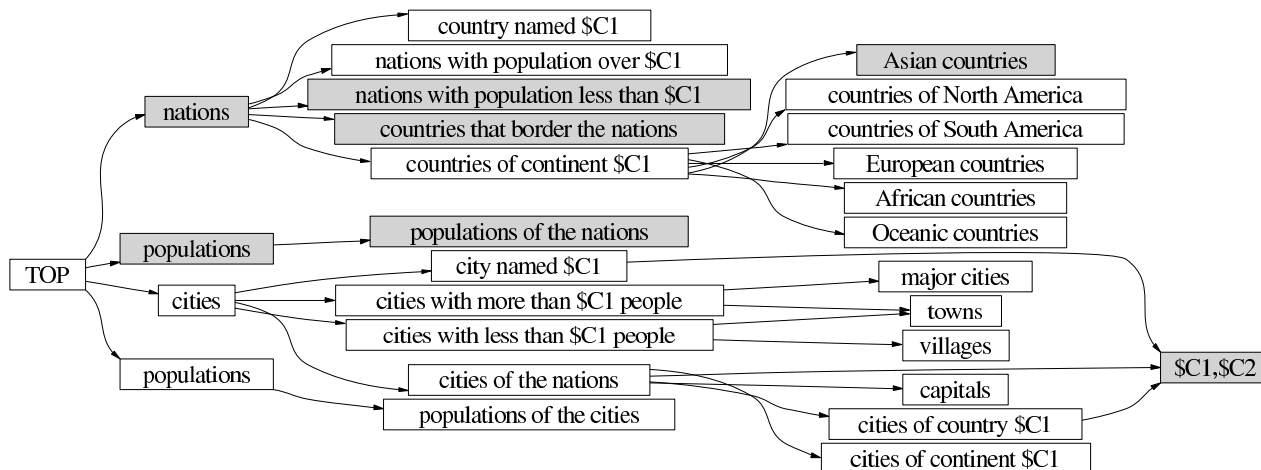


Figure 1: Subsumption hierarchy for a phrasal lexicon covering the schema of 2.1.

a query. The functions $ref(q)$ and $gapped(q, f)$ call for query descriptions where $gapped(q, f)$ is the query description of q with the previously interpreted query f filling a gap. The first template enables users to simply type a noun phrase request (e.g. “Asian countries”). The second template enables command type sentences (e.g. “Give me the Asian countries”). The third template enables users to pose a type of WH-question (e.g. “Which Asian Countries have fewer than 10,000,000 people?”).

- $\langle q(truth) \rangle :$
 $Template[‘(is|are) \cdot there \cdot ref(q)?’] \dots$
- $\langle q(count) \rangle :$
 $Template[‘how many \cdot ref(q) \cdot (is|are) \cdot there?’] \dots$
- $\langle q(when) \rangle :$
 $Template[‘when \cdot (is|are) \cdot ref(q).’] \dots$

These three templates account for special processing or interpretation of query answers associated with the pragmatic features $truth$, $count$ and $when$ respectively. The first template is used to answer yes/no queries (e.g. “Is there a country with more than 1,000,000,000 people?”), the second template returns the number of answers (e.g. “How many countries are in Asia?”) and the third template is used to answer where type queries (e.g. “Where is Riga?”).

3 Processing

3.1 Generation

The generation model of STEP is described in [13, 14] and thus we refer interested readers to these publications for details. In summary however, generation works by semantically sorting a query 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. This process is recursive, leading to descriptions of derived query forms within complement patterns. For example the description of the second query of section 2.2 leads to the sub-description: $ref(\{y_2 | Country(y_2) \wedge (\exists y_3)(In_Continent(y_3) \wedge y_2.code = y_3.country \wedge y_3.continent = "Asia")\})$ in the first complement pattern of the entry e4 of section 2.3. In total the resulting description of the query is ‘countries that border Asian countries’.

3.2 Understanding

We formulate sentence understanding as state-space search where the initial state is the input sentence and the goal state is a query expression that could have generated the input sentence. Along with book keeping structures, 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 parsed. Though this search process starts with the sentence templates of section 2.4, the interesting part has to do with recognizing the query expressions. Thus we isolate the treatment here to parsing $ref(q)$ and $gapped(q, f)$ expressions.

In the following σ stands for a word, ω for a sequence of words and ε the empty string. The state of the parse is represented by: $\langle \omega, q, \Gamma, h, f \rangle$ where ω is the sequence of strings remaining, q is a query expression (referred to as the current focus), Γ is a stack of query expressions with open sub-formulas, h is a head type marker and f , if present, is the query that will fill a gap. In more detail the stack Γ is represented by an expression $q_n \rightarrow q_{n-1} \rightarrow \dots \rightarrow q_1$, where each expression in q_i has a distinguished sub-formula \star_v where q_{i+1} is to fuse; the variable v in the specification of the sub-formula specifies the name that the free variable of q_{i+1} must adopt when q_{i+1} is fused with q_i . The head type marker h is either $h\rhd$ indicating that the head is expected, or h indicating that the head has already been obtained. The initial state of a query description parse is represented as $\langle \omega, \emptyset, \emptyset, \emptyset, \emptyset \rangle$ in the case of $ref(q)$ and $\langle \omega, \emptyset, \emptyset, \emptyset, f \rangle$ in the case of $gapped(q, f)$. The goal state is represented as $\langle \varepsilon, q, \emptyset, \emptyset, \emptyset \rangle$. In the inference rules below the symbol $_$ indicates a don’t care symbol that matches any material in the corresponding position.

3.2.1 Initial matching

$$\frac{\langle \sigma \cdot \omega, \emptyset, \emptyset, \emptyset, - \rangle \quad q_h : Modifier[\sigma \cdot \omega']}{\langle \omega'', q_h, \emptyset, h\rhd, - \rangle} \text{ where } \omega = \omega' \cdot \omega'' \quad (1)$$

$$\frac{\langle \sigma \cdot \omega, \emptyset, \emptyset, \emptyset, - \rangle \quad q_h : Head[\sigma \cdot \omega']}{\langle \omega'', q_h, \emptyset, h, - \rangle} \text{ where } \omega = \omega' \cdot \omega'' \quad (2)$$

These rules prime the parsing process by matching⁷ an initial modifier or head pattern

⁷Matching occurs either *strongly* or *weakly*. A match is strong when a parameter of a pattern matches values of the parameter’s attributes within the database. A match is weak, when a parameter does not match any value within the database nor lexicon. A strong match does not incur cost, while a weak match incurs additional cost.

to the first word(s) of the sequence.

3.2.2 Standard matching

$$\frac{\langle \sigma \cdot \omega, q_h, \Gamma, h \triangleright, - \rangle \quad q'_h : \text{Modifier}[\sigma \cdot \omega']}{\langle \omega'', q_h \cap q'_h, \Gamma, h \triangleright, - \rangle} \text{ where } \omega = \omega' \cdot \omega'' \text{ and } q_h \cap q'_h \neq \emptyset \quad (3)$$

$$\frac{\langle \sigma \cdot \omega, q_h, \Gamma, h \triangleright, - \rangle \quad q'_h : \text{Head}[\sigma \cdot \omega']}{\langle \omega'', q_h \cap q'_h, \Gamma, h, - \rangle} \text{ where } \omega = \omega' \cdot \omega'' \text{ and } q_h \cap q'_h \neq \emptyset \quad (4)$$

$$\frac{\langle \sigma \cdot \omega, q_h, \Gamma, h, - \rangle \quad q'_h : \text{Complement}[\sigma \cdot \omega']}{\langle \omega'', q_h \cap q'_h, \Gamma, h, - \rangle} \text{ where } \omega = \omega' \cdot \omega'', \omega' \text{ is plain and } q_h \cap q'_h \neq \emptyset \quad (5)$$

These rules attach modifiers, heads and complements to an unchanging focus.

3.2.3 Stack management

$$\frac{\langle \sigma \cdot \omega, q, \Gamma, h, - \rangle \quad q'_h : \text{Complement}[\sigma \cdot \omega' \cdot \text{ref}(q''_h)]}{\langle \omega'', q''_h, q_h \cap q'_h \rightarrow \Gamma, h' \triangleright, - \rangle} \text{ where } \omega = \omega' \cdot \omega'' \text{ and } q_h \cap q'_h \neq \emptyset \quad (6)$$

$$\frac{\langle \omega, q_h, q'_h \rightarrow \Gamma, h, - \rangle}{\langle \omega, \text{fuse}(q'_h, q_h), \Gamma, h', - \rangle} \quad (7)$$

$$\frac{\langle \varepsilon, q, \emptyset, h, \emptyset \rangle}{\langle \varepsilon, q, \emptyset, \emptyset, \emptyset \rangle} \quad (8)$$

These rules manage the stack of focuses. Rule 6 enables focus to shift to sub-descriptions. Rule 7 pops the stack to return to a prior focus. Rule 8 pops the last type head marker to arrive at a goal state.

3.2.4 Gap management

$$\frac{\langle \omega, \emptyset, \emptyset, \emptyset, f_h \rangle}{\langle \omega, \mathbf{f}_h, \emptyset, h \triangleright, \emptyset \rangle} \quad (9)$$

$$\frac{\langle \sigma \cdot \omega, q, \Gamma, h, f_{h'} \rangle \quad q'_h : \text{Complement}[\sigma \cdot \omega' \cdot \text{ref}(q''_h)]}{\langle \omega'', \mathbf{f}_{h'}, q_h \cap q'_h \rightarrow \Gamma, h' \triangleright, \emptyset \rangle} \text{ where } \omega = \omega' \cdot \omega''. \quad (10)$$

These rules control for some movement phenomenon. Rule 9 lets the filler fall immediately into place. This would cover sentences such as “which Asian countries have — *more than 10,000,000 people?*” where the gap ‘Asian’ countries inserts at ‘—’. Rule 10 covers longer distance cases such as “which Country is *the city named Paris in* —?”. In this later case the resulting query must be rearrange to make the filler concept the free variable.

3.2.5 Fudging

$$\frac{\langle \sigma \cdot \omega, -, -, X, - \rangle}{\langle \sigma' \cdot \omega, -, -, X, - \rangle} \text{ with cost } \Delta_{before}(\sigma, \sigma') \text{ where } X \text{ is either } \emptyset \text{ or } h \triangleright \quad (11)$$

$$\frac{\langle \sigma \cdot \omega, -, -, h, - \rangle}{\langle \sigma' \cdot \omega, -, -, h, - \rangle} \text{ with cost } \Delta_{after}(\sigma, \sigma') \quad (12)$$

These rules alter, with additional cost, the input sequence into a form that is more likely to be parsed. The cost of such operations is based on the type of words considered and whether the fudging occurs before or after the head of the focus has been matched. The functions $\Delta_{before}(\sigma, \sigma')$ and $\Delta_{after}(\sigma, \sigma')$ supply this cost. Note that this approach generalizes to adding and deleting words as well. For example the cost of 10 units to drop the preposition ‘in’ after the head is specified by making $\Delta_{after}(\text{‘in’}, \varepsilon) = 5$.

3.2.6 Search

Uniform cost search is employed to find the least cost parse of the query description; parser states are expanded in order of cost. To keep search states properly ordered, each inference rule applies with the cost of 1 unit which may be added to in the case of weak matches or fudging operations. When new states are generated they are added to a hash table that is subsequently checked to avoid generating repeat states, essentially giving an Early type parsing algorithm.

Because each application of an operator costs 1 unit, we naturally prefer the smallest ‘parse tree’ and thus capture the well known preference of minimal attachment [2]. To capture late closure [2], we refine our cost model to make any number of consecutive applications rule 7 or 8 cost 1 unit. Thus the correct parse of ‘Countries that border countries of Asia’ costs 5 units and is shown below.

Initial State:

$\langle \text{'Countries that border countries of Asia'}, \emptyset, \emptyset, \emptyset, \emptyset \rangle$

Step 1: Apply rule 2 with first pattern of entry e1.

$\langle \text{'that border countries of Asia'}, \{x_1 | \text{Country}(x_1)\}, \emptyset, \text{country}, \emptyset \rangle$

Step 2: Apply rule 6 with first pattern of entry e4

$\langle \text{'countries of Asia'}, \{x_2 | \text{Country}(x_2)\},$
 $\{x_1 | \text{Country}(x_1) \wedge (\exists y_{2,1})(\exists y_{2,2})(\text{Borders}(y_{2,1}) \wedge \text{Country}(y_{2,2}) \wedge$
 $y_{2,1}.\text{country1} = x_1.\text{country} \wedge y_{2,1}.\text{country2} = y_{2,2}.\text{code} \wedge *_{y_{2,2}})\} \rightarrow \emptyset,$
 $\text{country} \triangleright, \emptyset \rangle$

Step 3: Apply rule 4 with first pattern of e1

$\langle \text{'of Asia'}, (\text{from above}), (\text{from above}), \text{country}, \emptyset \rangle$

Step 4: Apply rule 5 with second pattern of e3

$\langle \varepsilon, \{x_2 | \text{Country}(x_2) \wedge$
 $(\exists y_4)(\text{In_Continent}(y_4) \wedge y_4.\text{country} = x_2.\text{code} \wedge y_4.\text{continent} = \text{'Asia'})\},$
 $(\text{from above}), \text{country}, \emptyset \rangle$

Step 5: Apply rule 7

$\langle \varepsilon, \{x_1 | \text{Country}(x_1) \wedge (\exists y_{2,1})(\exists y_{2,2})(\exists y_4)(\text{Borders}(y_{2,1}) \wedge$
 $\text{Country}(y_{2,2}) \wedge \text{In_Continent}(y_4) \wedge y_{2,1}.\text{country1} = x_1.\text{code} \wedge$
 $y_{2,1}.\text{country2} = y_{2,2}.\text{code} \wedge y_4.\text{country} = y_{2,2}.\text{code} \wedge y_4.\text{continent} = \text{'Asia'})\},$
 $\emptyset, \text{country}, \emptyset \rangle$

Step 6: Apply rule 8 to yield a goal state.

$\langle \varepsilon, (\text{from above}), \emptyset, \emptyset, \emptyset \rangle$

3.3 Interaction and Answer Presentation

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. Because queries here may be tested for equivalence, ‘semantic distinctness’ is determined algorithmically.

Once a query is deemed appropriate, based on its pragmatic features (e.g. *where*) and semantics, an answer pattern is obtained from the phrasal lexicon. This often causes an extension of the query to fetch additional relevant information. Finally the query is translated to SQL and applied over the database. The answer pattern determines how the resulting tuples are presented to the user.

4 Implementation and Experiences

STEP is currently about 10,000 lines of LISP code and is run as a server that is called through CGI from a browser. STEP issues satisfiability queries to the SPASS theorem prover and relational queries to a PostgreSQL database. STEP supports concurrent users through simple server side queuing and HTML embedded session identifiers.

At start up time STEP compiles the phrasal lexicon into a subsumption hierarchy and builds relevant hash tables. To compile the 85 entries of the Geography domain, SPASS

must decide 1513 satisfiability tests. Remarkably the whole compilation takes under 35 seconds⁸ on a DELL Dimension 8300. The on-line performance of STEP is also reasonable. Provided that there are not too many simultaneous requests, system response time is in the neighborhood of 2 to 5 seconds for queries over the Geography domain. These initial performance measures indicate that STEP’s generation and understanding techniques are able to run, though perhaps just barely, on today’s high powered servers.

A full demonstration of STEP over a geography domain [11], complete with a link to its configuration, has been continuously available for anonymous querying since June 2004. Over this period approximately 140 different visitors (myself not included) have issued in the neighborhood of 1000 questions to STEP. The purpose of this initial testing was not to carefully measure the accuracy of STEP, but was rather to build up a relatively large sample of real queries and in doing so help determine where system development efforts should be placed; naturally STEP has undergone extensive development and some refinement over this period. Still, based on a cursory analysis of more recent system logs, STEP has done a reasonably accurate job of satisfying user requests. In the future more systematic studies will be undertaken over different domains to quantify this.

5 Related Work

A comprehensive review of the pre-mid 90’s work done in natural language interfaces over databases appears in [3, 7]. In relation to this work, STEP shares an almost identical vision to Codd’s RENDEZVOUS system [6]. Surprisingly there has been fairly little work on the relational query paraphrasing that Codd called for. The most advanced of the prior systems seems to be REMIT [10]. Similar to STEP, REMIT maps relational calculus expressions to natural language. REMIT does this in two phases: firstly the relational calculus expression is mapped to a predicate/argument structure; secondly the predicate/argument structure is mapped to English. In the first phase a *focus* is determined which is the ‘root’ variable of the given query. The focus of a query in REMIT corresponds roughly to the free variable x in queries here. REMIT, however, does not place restrictions on the class of tuple calculus and thus the determination of a focus is not as straight forward as with STEP. In any case once it determines a focus, REMIT follows the syntactic structure of the query and aggregates linguistic knowledge attached to attributes of the current focus. REMIT recursively applies the same strategy to satellite variables reached through joins. It is not clear what the configuration requirements of REMIT are, though they appear to be considerable. In contrast, the configuration of STEP is well spelled out. Finally the semantic basis of STEP’s generator gives a clear path toward integrating specialized domain dependent language.

Although none of the following systems generate clarification dialogs or query paraphrases, they deserve mention. The system described in [16] is similar to STEP in that both address mapping browser initiated user questions to SQL. STEP however uses a phrasal approach in contrast to more traditional grammar formalism used in [16]. STEP is somewhat

⁸This cost should grow logarithmically with the size of the phrasal lexicon.

similar to Microsoft’s English Query[5] in that an administrator supplies essentially phrasal attachments to the schema. However English Query does not seem to be mapping into an internal representation other than SQL and is simply presenting answers as tables resulting from SQL evaluation.

Another recent system worth noting is the PRECISE system [15]. 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 tables, 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 answer presentation problem. To their credit, however, PRECISE publicly offers an on-line demonstration.

6 Conclusions

The architectural of STEP is somewhat unusual. STEP does not use a general purpose grammar for syntactic analysis, but rather uses a phrasal lexicon authored specifically over the underlying database. Input sentences are parsed via a closed set of inference rules. These inference rules employ a Montague like strategy where logical form is incrementally built up as the input is scanned from left to right. These inference rules also cause special *fudging operations* which, at a cost, deliberately add, drop or alter words in the input sentence to help find a parse, albeit one of less confidence.

‘Logical form’ in STEP consists of expressions in a class tuple calculus with attached pragmatic features. The relations used within these expressions are over database as well as extended pragmatic and conceptual relations. Such an extended vocabulary of relations makes it possible that STEP might provide meta level responses for queries over meaningful relationships not (yet) covered in the underlying database. The actual class of tuple calculus expressions is a decidable for emptiness, containment and equivalence. Such capabilities are useful in providing cooperative responses [14].

The advantage of STEP’s phrasal approach to parsing is that it avoids many of the difficulties associated with ambiguity and idiosyncrasies in large scale grammars. Additionally, via fudging operations, STEP finds acceptable parses for many non-grammatical inputs – a common occurrence in practice. Finally a phrasal approach allows for easier specification of idiomatic and idiosyncratic domain language. A disadvantage to this approach is that a specific phrasal lexicon will need to be authored for each new database, though we envision tools to assist in this process; entries are relatively well structured and can be compiled into a subsumption hierarchy which makes their semantic relationships explicit. A question of course, is how well will STEP handle the syntactic complexities of real language. Of course it is still too early to answer this question fully, but the thesis here is that through integrating large electronic dictionaries into the parsing process and through relying on clarification

dialogs, STEP will ultimately work well enough to be practical.

References

- [1] S. Abiteboul, R. Viannu, and V. Hull. *Foundations of Database Systems*. Addison Wesley, 1995.
- [2] J. Allen. *Natural Language Understanding (2nd edition)*. Benjamin/Cummings, 1995.
- [3] I. Androutsopoulos and G.D. Ritchie. Database interfaces. In R. Dale, H. Moisl, and H. Somers, editors, *Handbook of Natural Language Processing*, pages 209–240. Marcel Dekker Inc., 2000.
- [4] P. Bernays and M. Schönfinkel. Zum Entscheidungsproblem der mathematischen Logik. *Mathematische Annalen*, 99:342–372, 1928.
- [5] A. Blum. Microsoft english query 7.5: Automatic extraction of semantics from relational databases and OLAP cubes. In *Proc. of VLDB*, pages 247–248, 1999.
- [6] E. Codd. Seven steps to rendezvous with the casual user. In *IFIP Working Conference Data Base Management*, pages 179–200, 1974.
- [7] A. Copestake and K. Sparck Jones. Natural language interfaces to databases. *The Natural Language Review*, 5(4):225–249, 1990.
- [8] B. Grosz, D. Appelt, P. Martin, and F. Pereira. Team: An experiment in the design of transportable natural-language interfaces. *AI*, 32(2):173–243, 1987.
- [9] J. Ljungberg. Paraphrasing SQL to natural language. In *Proc. of RIAO 91*, Barcelona, 1991.
- [10] B. G. T. Lowden and A. N. de Roeck. The REMIT system for paraphrasing relational query expressions into natural language. In *Proc. Int’l. Conf. on Very Large Data Bases*, Kyoto, Japan, August 1986.
- [11] W. May. Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999.
- [12] M. Minock. Knowledge representation using schema tuple queries. In *Proc. of KRDB*, pages 51–62, Hamburg, Germany, 2003. IEEE Computer Society Press.
- [13] M. Minock. A phrasal generator for describing relational database queries. In *Proc. of the 9th EACL workshop on natural language generation*, pages 63–70, Budapest, Hungary, April 2003.

- [14] M. Minock. Modular generation of relational query paraphrases. *Journal of Language and Computation special issue on Formal Aspects of NLG*, 2005. To appear.
- [15] A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *Intelligent User Interfaces*, 2003.
- [16] B. Thalheim and T. Kobienia. Generating DB queries for web NL requests using schema information and DB content. In *Proc. of NLDB*, pages 205–209, 2001.
- [17] J. Ullman. *Principles of Database and Knowledge-Base Systems*. Computer Science Press, Rockville, Maryland, 1989.