A Phrasal Generator for Describing Relational Database Queries

Michael J. Minock
mjm@cs.umu.se
The University of Umeå
Umeå, Sweden

Abstract

This paper proposes a technique to generate single sentence natural language descriptions for a wide class of relational database queries. Such a capability meets an important need in the area of cooperative information systems.

The approach to describing queries is phrasal and is restricted to tuple relational queries using positive or negatively signed sequences of existential quantifiers over conjunctions of conditions. Query containment and equivalence are decidable for this class and this property is exploited in the maintenance and use of the phrasal lexicon.

1 Introduction

Often relational database schemas are delivered as collections of oddly named tables and attributes. Users and administrators are expected to query, integrate and otherwise maintain such systems. Natural language generation has been seen as an important part of improving the understandability of relational database schemas(McKeown, 1985).

The focus here, however, is not to explain or describe database schemas, but rather to describe database queries. While at first this may seem to be of limited value, we shall see that many techniques in cooperative information systems(Gaasterland et al., 1992) require 'query' descriptions as an integral part of their communication process. The 'query' being described is not usually the user's own query, but rather some derived expression that may be written in the form of a query. A set of cooperative techniques that require such description services shall be reviewed in this paper. We view the cooperative information system that bundles these techniques, as essentially providing the services of a strategic text planner. The cooperative information system decides communication content with communication acts consisting of template sentences with embedded requests for 'query' descriptions.

Given that we seek to describe queries, we must contend with the fact that there are infinitely many syntactically correct queries over a given schema. It is critical that the generation system provide adequate *coverage* over some *well defined* portion of this space. In this paper the space of coverage mirrors a recently defined class of *schema tuple queries*(Minock, 2002). Because of the natural closure properties of this language, and its decidability for equivalence and containment, it is reasonable to assume that many cooperative techniques may generate output 'queries' within this form.

The author of the query description system is assumed to be a database administrator. Thus we must adopt a generation technique that does not require a deep understanding of linguistics. Moreover the administrator must be given a structured method of authoring so that they may declare a schema *covered*. That is the system should faithfully, if not always elegantly, describe all queries of the form that are posed over the database schema. Given these requirements we adopt a phrasal approach that couples parameterized queries with patterns. The parameterized queries are within the class of the identified form and the patterns are simply modifier, head, complement triples.

1.1 Organization of this Paper

Section 2 shall review cooperative information systems and shall illustrate how such systems play the role of strategic natural language generators. Section 3 shall give a brief introduction to the language in which queries must be expressed. Section 4 describes the phrasal lexicon and section 5 describes the generation process. Section 6 discusses this work in the context of prior work and gives future directions.

2 Cooperative Information Systems

Cooperative information systems(Gaasterland et al., 1992) seek to extend conventional database query-answer dialogues with the principles of cooperative conversation(Grice, 1975). Thus the response to the user's query is richer than simply presenting the answers that meet the conditions of their query. Often such responses are conceptual and may in fact be specified with derived 'query' expressions. We now cover specific cooperative techniques that have such 'conceptual' outputs.

2.1 Cooperative Techniques

A user may have a *query misconceptions*, meaning that the user is unaware that their query presupposes an illegal state of the database. For example assume that a state law in Ohio is that all mayors must be over 25 years old. If a user is unaware of this restriction and issues a request "list all female mayors in Ohio that are younger than 23", they should be informed that "it is impossible for people under 25 to be mayors of cities in Ohio." Such a conceptual response is in the form of a 'query'. Gal (Gal and Minker, 1988) uses integrity constraints to explain query misconceptions back to the user.

Related to query misconceptions, a user should be made aware of a *false presupposition* they have about the database state. A false presupposition is an assumption that is implicit in a user's query, though false. The system CO-OP(Kaplan, 1982) used a limited theory of cooperation to correct false presuppositions. For example assume that a user requests: "Give the cities with population over 2 million in the state of Alaska or North Dakota that have a female mayor" A traditional system would say, "none." A system that could detect false presuppositions would respond, "there are no *cities in Alaska or North Dakota with a population over 2 million*." This is a description of the minimal failing sub-query of the users original query. Efficient algorithms exist to find minimal failing sub-queries (MFS) as well maximal succeeding sub-queries (MSS)(Godfrey, 1994).

Query relaxation is useful when a query has no matching tuples. During query relaxation conditions may be loosened or alternate entity types may be queried. For example when asking for flights from Dulles airport to La Guardia with a Sunday morning departure at 10 am, a relaxed query might return a 9:32 am flight from Dulles arriving at JFK. An entity type relaxation might offer a train or a bus trip rather than airplane flight. It is important to describe the relaxed query to the user before flooding them with extensional answers.

Intensional query answering (Imielinski, 1988) (Shum and Muntz, 1987) provides a summary answer rather than the entire tuple extension satisfying the query. If you are asking for all employees who make over 100,000$, instead of listing every single manager and Joe Star engineer, it is better to report "all the managers and the engineer named 'Joe Star'." This intensional response is more informative in the case that the user does not interpret an enumeration of all the managers names to mean 'all managers'. Once again the ability to describe queries is important.

The CARMIN system (Godfrey et al., 1994) includes an integrated explanation and answer presentation system. System explanations are based on the proof path used by a PROLOG meta-interpreter. Aspects of what to include and how to coordinate these explanations are also addressed(Gaasterland and Minkler, 1991). Natural language descriptions of the user query and relaxation process are generated for the cooperative information system CoBase(Chu et al., 1996),(Minock and Chu, 1996).

2.2 Cooperative Information Systems Serving as 'Strategic' Planners

We propose a modular approach in which the cooperative information system decides 'what' to say and a query description sub-system decides 'how' to say it.

Figure 1 shows an architecture for this approach. The cooperative information system consists of sub-systems that perform misconception detection, false presupposition detection, query relaxation and intensional answer generation. The user interface consists of sub-systems that perform query formulation, query description and answer presentation. Both the interface and the cooperative information system have access to the domain database.

A user is assumed to compose their query through some type of query formulator. This may be as primitive as a text box in which to type a logical query expression, to as advanced as a full natural language understanding system. Whatever the formulator type, it may be helpful to provide the user a natural language description of the query, q, that they have formulated.

After the user has verified their query, it is passed from the user interface to the misconception detection sub-system. If the query contains a misconception, then the offending portion of the query that caused the misconception, m, is reported as being 'impossible' and execution terminates. If the query contains no misconception, then subsequent flow depends on whether the user's query returns answers. If it does not return answers, the query is passed to the false presupposition detection sub-system. This sub-system identifies a minimal failing sub-query of the query. This minimal failing sub-query, s, is described to the user as not returning answers in the current database state. The minimal failing sub-query is then generalized to an answer returning query by the query relaxation sub-system. The fact that this relaxed query, a, generates answers is communicated to the user. Now that either the original query, or the derived relaxed query is answer generating, we then check whether a suitable in-

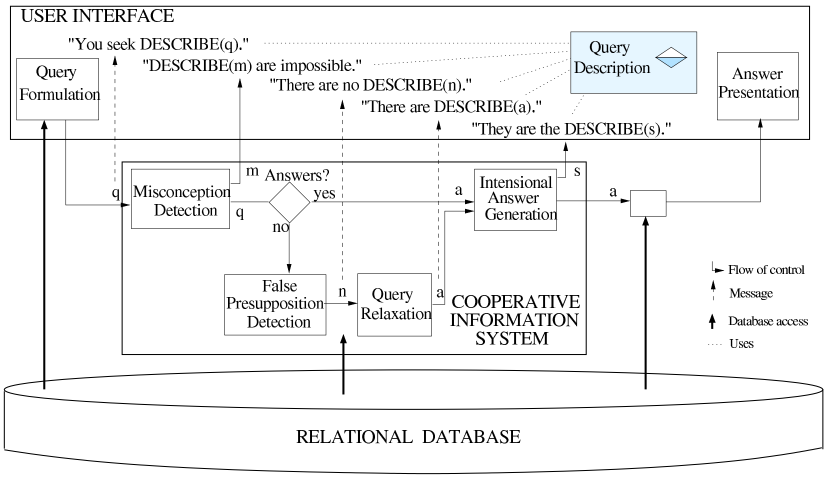

Figure 1: A cooperative information system as a strategic planner.

tensional summary may be returned as a substitute for fetching all the extensional answers to the query. The intensional answer generator either decides that no suitable summary exists, or it terminates the with a description of the summary 'query' s. If no suitable summary exists, then the extensional answers are retrieved from the database and are passed to the answer presentation sub-system.

Of course the proposed architecture leaves many issues unspecified. For example how does one pick a single minimal failing sub-query? How does one decide to relax the user's query? When is a summary appropriate in place of a full extensional answer. Still the important point to note here is that the cooperative information system makes such decisions based on semantic and pragmatic issues. These considerations determine the *quantity*,*quality* and *relation* of the content to be expressed. The tactical decision of the *manner* in which to express such content is left up to the sub-system that describes 'queries'.

3 A Class of Describable Queries

This paper now turns to the relatively pure problem of generating natural language descriptions for a broad class of relational database queries. Let us start by considering the following relational schema:

Person(<u>name</u>, gender, age, *city*)

City(<u>cityName</u>, population, *mayor*, state, country)

Knows(<u>*knower*, *known*</u>, opinion)

The semantics here are those of standard relational databases. The <u>underlined</u> attributes are the primary keys of the relations and the *italicized* attributes are foreign keys[1]. A wide variety of queries from simple to somewhat complex may be expressed over this schema. The goal here will be to describe such queries. The following queries are of interest:

1.) "Men living in 'Paris' or 'Nice'."

2.) "People living in cities with populations of over 100,000 people."

3.) "People not living in cities with populations of over 100,000 people."

4.) "People who know people living in 'Nice'

5.) "People who do not know people living in 'Nice'

6.) "People who know and like themselves"

7.) "People who know all people living in 'Nice'

The first five queries above may be described using our current approach. Query six has a reflexive reference that we are not yet able to handle. Query seven may not be expressed within the language we limit ourselves to - the language $\mathcal{L}$.

Definition 1 *(The language $\mathcal{L}$)*
$\ell \in \mathcal{L}$ if ℓ is in the form:

$$R(x) \bigwedge_{i=1}^{k} s_i \cdot (\exists \vec{y}_i)\Psi_i$$

where x is the only free variable of ℓ, $R(x)$ is the range condition for x, s_i is a positive or negative ($\neg$) sign, $\vec{y}_i$ is a finite sequence of existentially quantified variables and Ψ_i is a conjunction of range conditions, simple conditions, set conditions and join conditions.

[1]Those who are familiar with databases will note that this schema makes the rather simplistic assumption that all city names in the world are unique.

The example queries from above are shown here. See (Minock, 2002) for a more complete discussion of $\mathcal{L}$. Each query $\{x|\ell\}$ returns a set of tuples. For the first six queries, $\ell \in \mathcal{L}$.

1.) $\{x|Person(x) \wedge x.gender = \text{`male'} \wedge x.age \geq 18 \wedge$
  $x.city \in \{\text{`Paris'}, \text{`Nice'}\}\}$
2.) $\{x|Person(x) \wedge (\exists y)$
  $(City(y) \wedge y.population > 100000 \wedge$
  $y.cityName = x.city)\}$
3.) $\{x|Person(x) \wedge \neg(\exists y)$
  $(City(y) \wedge y.population > 100000 \wedge$
  $y.cityName = x.city)\}$
4.) $\{x|Person(x) \wedge (\exists y)(\exists z)($
  $(Know(y) \wedge Person(z) \wedge$
  $\mathbf{z}.city = \text{`Nice'} \wedge$
  $x.name = y.knower \wedge y.known = z.name)\}$
5.) $\{x|Person(x) \wedge \neg(\exists y)(\exists z)($
  $(Know(y) \wedge Person(z) \wedge$
  $\mathbf{z}.city = \text{`Nice'} \wedge$
  $x.name = y.knower \wedge y.known = z.name)\}$
6.) $\{x|Person(x) \wedge (\exists y)$
  $(Know(y) \wedge y.opinion = \text{`like'}$
  $x.name = y.knower \wedge \mathbf{y.known = x.name})\}$

Query seven may not be expressed using $\mathcal{L}$

7.) $\{x|Person(x) \wedge (\forall z)(\exists y)$
  $(Person(z) \wedge z.city = \text{`Nice'} \Rightarrow$
  $Know(y) \wedge x.name = y.knower \wedge$
  $y.known = z.name)\}$

Naturally all of these queries may be expressed using standard SQL.

Theorem 1 *($\mathcal{L}$ is decidable for $\subseteq, =$ and disjointness) if $q_1 \in \mathcal{L}$ and $q_2 \in \mathcal{L}$ then there exists a sound and complete inference mechanisms to decide if the three predicates:*
1.) $\{x_1|q_1\} \subseteq \{x_2|q_2\}$
2.) $\{x_1|q_1\} = \{x_2|q_2\}$
3.) $\{x_1|q_1\} \cap \{x_2|q_2\} = \emptyset$.
are necessarily true over the set of all database instances.

See (Minock, 2002) for the proof of this theorem. These properties will be used to maintain and select entries from the phrasal lexicon.

4 The Phrasal Lexicon

The approach here generates a single highly aggregated sentence of natural language that describes a query built over a formula in $\mathcal{L}$. The knowledge used to achieve this is a *phrasal lexicon*. The phrasal lexicon, denoted *PL*, consists of a set of *n entries* where each entry is a *parameterized query/pattern* pair. The *i*-th entry is $\langle\{x|\ell_i\} : p_i\rangle$ where $\{x|\ell_i\}$ is a parameterized query and p_i is a single pattern.

A parameterized query is simply a query defined using a formula in $\mathcal{L}$ in which constants may be parameters. Thus the query $\{x|Person(x) \wedge x.gender = c_1\}$ is a parameterized query where the constant c_1 could be 'male' or 'female'. The constant c_1 is said to be a parameter. We may also have set valued parameters as in: $\{x|Person(x) \wedge x.city \in C_1\}$.

A pattern is simply the three *phrases*: "[modifier] head [complement]". Phrases consist of plain text, possibly including parameters[2]. The modifier, head and complement distinction is best illustrated with example. The simple description,"Young people living in London" has "young" as a modifier, "people" as the head, and "living in London" as the complement. Thus it would be represented: "[Young] people [living in London]". The [phrase] syntax within a pattern signifies that groups of phrases may collect in such positions during aggregation. By contrast there can only be one head. Thus we may generate the aggregated pattern "[Young, employed] people [of the female gender, living in 'London']".

We now shall now cover the different types of entries within the phrasal lexicon. Special attention will be paid to insure that the entries completely cover the database schema over which queries may be posed.

4.1 Simple Entries

Let us start with the simplest type of entry. Here we specify the pattern associated with the condition free database relation *Person*:

$\langle\{x|Person(x)\}$:
  "[ ] people [ ]"$\rangle$

Now we see the entry for the relation *Person*, the attribute *gender*, and the operator $=$.

$\langle\{x|Person(x) \wedge x.gender = c_1\}$:
  "[ ] people [of the c_1 gender]"$\rangle$

The "[ ]" specifies that there is an an empty modifier for this pattern. The head is "people". Finally the complement phrase "[of c_1 gender]" has the parameter c_1.

There may be more than one entry with equivalent parameterized queries. For example:

$\langle\{x|Person(x) \wedge x.gender = c_1\}$:
  "[c_1] people [ ]"$\rangle$

Naturally We may also have constants specified in the queries as well.

$\langle\{x|Person(x) \wedge x.gender =' male'\}$:
  "[ ] **males** [ ]"$\rangle$

[2]As we shall see later, the complement phrase may also contain a recursive call to describe a sub-query.

In this final case we see that the head itself has been changed from the default for the relation. This may only occur once during a generation. The head is said to be *open* if it contains the same value as the head for the condition free entry over the relation. Otherwise it is said to be *frozen* and may not be combined with other patterns that alter the head.

Given the above entries we may generate the description of the query:

$\{x|Person(x) \wedge x.gender = \text{‘male’}\}$

As “people of the male gender”, “male people”, or “males”. The simple heuristics we employ prefer the last form over the first two.

We must fully populate the lexicon to cover all of the attribute/basic operator combinations for each relation in the schema. If there are 6 basic operators $(>, \geq, =, \neq, <, \leq)$ and two set operators $(\in, \notin)$, and each attribute/operator combination makes sense, then the minimal number of simple entries required by the schema in section 3 is: $(8 \times 4 + 1) + (8 \times 5 + 1) + (8 \times 3 + 1)$. The ‘+1’ terms signify the entry for the case where no conditions are applied.

4.1.1 Aggregating Simple Entries

Now we show how simple entries combine to describe queries with more than one condition. In addition to the previous entries, assume that we also have the following entry:

$\langle\{x|Person(x) \wedge x.city \in C_1\}$:
“[ ] people [living in C_1]”$\rangle$

Now suppose that the query for the “males living in Paris or Nice” needs to be described. The query is:

$\{x|Person(x) \wedge x.gender = \text{‘male’} \wedge$
$x.city \in \{\text{‘Paris’, ‘Nice’}\}\}$

This may be rewritten as a combination of filled in parameterized queries from the phrasal lexicon.

$\{x|Person(x) \wedge x.gender = \text{‘male’}\} \cap$
$\{x|Person(x) \wedge x.city \in \{\text{‘Paris’, ‘Nice’}\}\}$

Excluding the permutations of the ‘Paris’ and ‘Nice’, the possible ways to *combine* the 3 patterns that match the first query and the 1 pattern that matches the second are:

1.) [ ] males [living in ‘Paris’ or ‘Nice’]

2.) [male] people [living in ‘Paris’ or ‘Nice’]

3.) [ ] people [of the male gender, living in ‘Paris’ or ‘Nice’]

4.) [ ] people [living in ‘Paris’ or ‘Nice’, of the male gender]

Using a simple heuristic of minimizing sentence length, the first choice is preferred. Extending the heuristic to communicate the maximum amount of information in *k* symbols, we induce the ordering presented here.

4.2 Join Entries

Now we face the issue of representing the patterns associated with join conditions within queries. First we shall consider the simple case of joining over foreign keys. This accounts for the bulk of meaningful many-to-one and one-to-many relationships. Then we shall consider the more complicated case involving many-to-many relationships[3].

4.2.1 One-to-many and many-to-one Relationships

Each attribute that can be meaningfully joined with another must be considered. For the example of section 3 this would amount to a total of 6 attribute matches. Assuming that we are only interested in equality joins, then we must account for $2 \times 2 \times 6$ entries to cover this space. The reason for the first doubling is that one must take into account direction when one describes joins. The second doubling occurs because one must also consider the negative case in which the query specifies that answers do <u>not</u> participate in such relationships.

For the join involving $person : city \longrightarrow^{+} city : cityName$ we have the entry:

$\langle\{x|Person(x) \wedge (\exists y)(City(y) \wedge$
$x.city = y.cityName \wedge \Phi\}$:
“[ ] people [that live in $GEN(\{y|City(y) \wedge \Phi\})$”]$\rangle$

The negative case is almost identical. The entry for $person : city \longrightarrow^{-} city : cityName$ is:

$\langle\{x|Person(x) \wedge \neg(\exists y)(City(y) \wedge$
$x.city = y.cityName \wedge \Phi\}$:
“[ ] people [that don’t live in
$GEN(\{y|City(y) \wedge \Phi\})$”]$\rangle$

The key issue to note here is that the complement phrase has a recursive call. This will cause a completely new generation problem to be instantiated.

Assuming that we have simple entries covering *city*, we may now generate descriptions such as “[adult] [males] [that live in ([ ] cities [with populations over 100,000 people])”. The material enclosed within

[3]For those familiar with Entity-Relationship modeling, the goal is to generate descriptions of an entity’s participation in one-to-many and many-to-many relationships. We skip one-to-one relationships because of their simplicity. Note that we do not yet handle either reflexive (self-joining) relationships or general *n*-ary relationships, even though $\mathcal{L}$ admits such queries

parenthesis shows the solution to the recursive description problem.

4.2.2 Many-to-many relationships through joins

We now consider a case of many-to-many type relationships. These present a special, though not insurmountable difficulty. There is only one possible many-to-many relationship in the example of section 3. This is the relationship that many people may know many other people. Especially vexing is the fact that attributes may also be involved within the many-to-many relationship. For example the opinion one has about who one knows. Consider the following entry:

$\langle \{x | Person(x) \wedge$
$(\exists y)(\exists z)(Knows(y) \wedge Person(z) \wedge$
$opinion = c_1 \wedge x.personId = y.knower \wedge$
$y.known = z.personId \wedge \Phi\}:$
"[ ] people [who know and c_1 some
$GEN(\{z | Person(Z) \wedge \Phi\})$]"$\rangle$

The only way to precisely control this is to make multiple join entries for each combination of given attributes in the joining relation. Thankfully relations that bridge many-to-many relations are often of fewer attributes. In the case of the example in this paper the many-to-many relationship is taken care of with 4 entries.

4.3 Coverage and extension of the phrasal lexicon

If we may guarantee that there are a sufficient set of entries to fully cover the schema, we may declare the phrasal lexicon to be *covered* with respect to the schema. Thus based on the approach above, it takes 123 entries to cover the schema in section 3.

Naturally we may improve the phrasal lexicon by extending it to cover more specific entries. For example, the following entry will simplify some descriptions considerably:

$\langle \{x | Person(x) \wedge x.gender =$ 'male' $\wedge x.age \geq 18\}:$
"[ ] **men** [ ]"$\rangle$

5 The Generation Process

Beyond simple aggregation, we have not yet described how the entries within the phrasal lexicon are used to obtain natural language descriptions of a query. We begin with a definition of what constitutes a description and then we show a relatively efficient mechanism to actually obtain such descriptions. Finally we say a little about how specific descriptions are selected.

5.1 Descriptions Defined

A *description* of a query is the aggregate pattern generated through applying a *covering*, *non-redundant* subset of the phrasal lexicon over the query.

Before we define what a covering, non-redundant subset of the the phrasal lexicon is, we must resolve formal difficulties associated with parameters in the phrasal lexicon. Because there are a finite number of constants in the formula specifying the query (ℓ) and a finite number of parameters within the entries of the phrasal lexicon (*PL*), we may consider the expansion of the phrasal lexicon to be all combinations of constants from within ℓ substituted in place of parameters within the entries of *PL*. This generates the finite expanded lexicon PL^{ℓ} where all parameters are bound.

We now define the subsets of the expanded lexicon that *cover* the query without *redundancy*. A set of lexical entries $s \in 2^{PL^{\ell}}$ are said to *subsume* $\{x|\ell\}$ iff the set intersection of all the queries within the entries of s necessarily contain $\{x|\ell\}$ where $\top$ is substituted for Φ in every (parameterized) query. Another, more concise way of saying this, is that the extension of the entries s contains $\{x|\ell\}$. A set of lexical entries $s \in 2^{PL^{\ell}}$ is said to *cover* $\{x|\ell\}$ iff there is no other s' where $s' \supset s$ where s' subsumes $\{x|\ell\}$, and the extension of s' are properly contained within the extension of s. Finally a set $s \in 2^{PL^{\ell}}$ is *redundant* iff some $s' \subset s$ has the same extension as s.

5.2 Obtaining Descriptions

To quickly identify subsets of the phrasal lexicon that cover the user query without redundancy, we organize the phrasal lexicon into a *subsumption hierarchy*[4]. The phrasal lexicon is compiled into a subsumption hierarchy by sorting entries down into the hierarchy. Nodes correspond to parameterized queries, though entries with equivalent parameterized queries are collapsed into a single node with multiple attached patterns. When we sort the query into the subsumption hierarchy, the set of immediate parents identify the set of entries that cover the user query without redundancy.

Figure 2 illustrates the subsumption hierarchy for the entries we have considered thus far. The upper portion of the hierarchy consists of the nodes that correspond to the 123 entries we must provide if we wish to cover the given database schema. The lower portion of the hierarchy consists of nodes representing entries that are meant to make generation more precise. Figure 2 also shows two queries sorted into the hierarchy.

The only complication in the sort procedure is handling parameters and open formula of the lexical entries. The way to handle parameters, is to allow them to range over any domain value except a distinguished

[4]The notion of subsumption hierarchy here corresponds with that used in description logics. However it should be noted that the logic here differs considerably from the unary and binary predicates of typical description logics. See (Minock, 2002) for a full discussion of this.

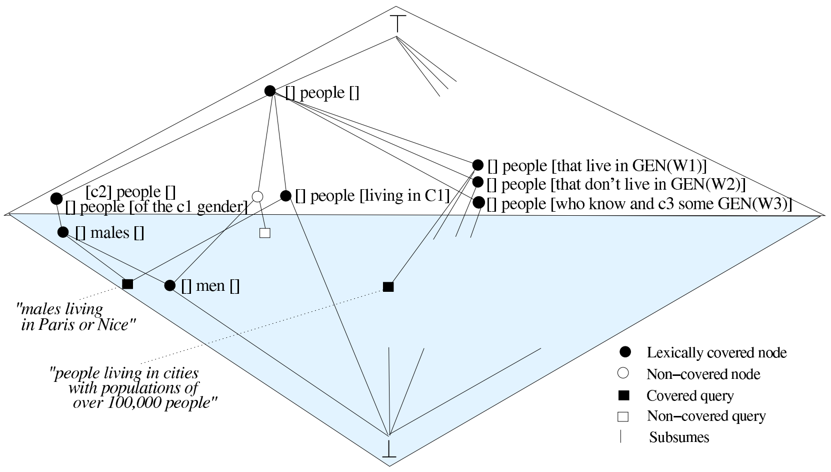
⊤
[] people []
[c2] people []
[] people [of the c1 gender]
[] people [living in C1]
[] people [that live in GEN(W1)]
[] people [that don't live in GEN(W2)]
[] people [who know and c3 some GEN(W3)]
[] males []
[] men []
"males living in Paris or Nice"
"people living in cities with populations of over 100,000 people"
⊥
● Lexically covered node
○ Non–covered node
■ Covered query
□ Non–covered query
| Subsumes

Figure 2: The phrasal lexicon organized as a subsumption hierarchy.

symbol *unkown*. This is recored by inserting the simple condition that the parameter does not equal *unknown*. Open formula Φ are simply assumed to be true and are replaced with the symbol $\top$.

Given that the parameterized queries and queries requiring description tend to be of limited size, we assume that deciding subsumption between two parameterized queries takes constant time. If the number of entries in the phrasal lexicon is n, then $O(n)$ nodes are in the subsumption hierarchy. It thus costs $O(n)$ subsumption operations to sort a query into the hierarchy.

5.3 Selecting Descriptions

Given a set of nodes that cover a query without redundancy, we may select combinations of associated patterns from this set of nodes to generate an actual description. The conditions guiding this process are that only one pattern is selected from each node and at most one pattern of the combination may over-ride the default head. A heuristic that we adopt is to prefer the shortest sentence. It is likely that a greedy technique is sufficient to enable the identification of such a minimal combination. Occasionally the whole generation process recurs when there is a Φ term within a complement. Once all the textual material is obtained from the recursive calls, a final process reorders the constituents so that the shorter phrases appear before longer phrases.

6 Discussion

It has been observed that fielded NLG systems tend to have pipelined architectures with the vast majority using some type of semantic network based representations as the common knowledge representation language(Reiter, 1994). Surface realization tends to be carried out using unification grammars(Kay, 1979)(Penman Project, 1989)(Elhadad, 1993). A common use for such grammars is enforcing number, gender, and case agreement.

The system here is also pipelined. The cooperative information system is granted the strategic decision about ‘what’ to express and the query description generator decides ‘how’ to express description requests from the cooperative information system. Thus the language being shared here is the language of query expressions, which are analogous to a fragment of first order logic. The description system interacts with the semantical system to obtain the most succinct descriptions of a ‘query’, but it does not pass any information back to the cooperative information system. This is in contrast to more general techniques that explicitly plan content through complex plan operators(Moore and Paris, 1989) or schemata(McKeown, 1985) using established rhetorical theories(Mann and Thompson, 1988). The specific, cooperative information strategy adopted here is less flexible, but the knowledge specification task is simplified considerably.

The choice of using a non-feature based phrasal grammar is based on the relative ease by which non-expert administrators might provide such phrasal knowledge. It is also anticipated that case and number errors will be of only minor annoyance and that clever administrators might be able to author phrases so that such errors are minimized. Currently it is assumed that a single, highly aggregated sentence, may describe a query. Certainly there is some limit to the number of query conditions that may be aggregated into a single sentence. Techniques to break up of the sentences must be entertained if we are to scale to more complex queries. Issues such a pronominalization and ellipsis are not yet addressed in this work, but will become more important as we consider how to span complex query descriptions over multiple sentences.

Though the idea of using a phrasal grammar is not new(Reiter, 1990), nor is using classification in text generation(Reiter and Mellish, 1992), the approach here is new in regard to exploiting the properties of the query formation language $\mathcal{L}$. As long as the reasoning task is able to present its results as expressions within this language (or, more liberally disjunctions of $\mathcal{L}$ expressions) then there may indeed be a high degree of modularity between the reasoning system and the generation component(Shieber, 1994).

7 Conclusions

This paper proposes a scalable and structured approach to generating natural language descriptions for a broad class of relational database queries. The simplicity of the phrasal approach enables clever database administrators to author the system without requiring specialized linguistic knowledge. The firm semantic basis of the approach lends a great deal of structure to the authoring process. Notably an administrator can declare their schema 'covered' once they have provided lexical entries for a bounded set of simple and join conditions.

8 Bibliography

References

W. Chu, H. Yang, Chiang K., M. Minock, G. Chow, and C. Larson. 1996. Cobase: A scalable and extensible cooperative information system. *Intelligent Information Systems*, 6(3):223–259.

M. Elhadad. 1993. FUF: the Universal Unifier. User Manual Version 5.2. Technical report, Computer Science, Ben Gurion University.

T. Gaasterland and J. Minkler. 1991. User needs and language generation issues in a cooperative answering system. In *ICLP Workshop on Adv. Logic Programming Tools and Formalisms for Language Proc.*

T. Gaasterland, P. Godfrey, and J. Minker. 1992. An overview of cooperative answering. *Intelligent Information Systems*, 1(2):127–157.

A. Gal and J. Minker. 1988. Informative and cooperative answers in databases using integrity constraints. In *Natural Language Understanding and Logic Programming*, pages 277–300.

P. Godfrey, J. Minker, and L. Novik. 1994. An architecture for a cooperative database system. In *Proceedings of the 1994 International Conference on Applications of Databases*.

P. Godfrey. 1994. Minimization in cooperative response to failing database queries. Technical report, University of Maryland Dept. of Computer Science, College Park, MD.

P. Grice. 1975. Logic and conversation. In P. Cole and J. Morgan, editors, *Syntax and Semantics*. Academic Press.

T. Imielinski. 1988. Intelligent query answering in rule based systems. In J. Minker, editor, *Foundations of Deductive Databases and Logic Programming*. Morgan Kaufman Publishers.

S. Kaplan. 1982. Cooperative responces from a portable natural language query system. *Artificial Intelligence*, 19:165–187.

M. Kay. 1979. Functional grammar. Fifth Annual Meeting of the Berkeley Linguistics Society.

W. Mann and S. Thompson. 1988. Rhetorical structure theory. *TEXT*, 8(3):243–281.

K. McKeown. 1985. *Using Discourse Strategies and Focus Constraints to Generate Natural Language Text*. Cambridge University Press.

M. Minock and W. Chu. 1996. Interactive explanation for cooperative information systems. In *Proceedings of the 9th International Symposium on Methodologies for Intelligent Information Systems*.

M. Minock. 2002. Beyond query containment: a decidable language supporting syntactic query difference. Technical Report 02.21, The Univeristy of Umea, Umea, Sweden, December.

J. Moore and C. Paris. 1989. Planning text for advisory dialogue. In *Proceedings of the 27th Meeting of the Association for Computational Linguistics*.

Penman Project. 1989. Penman documentation: the Primer, the User Guide, the Reference Manual, and the Nigel manual. Technical report, USC/Information Sciences Institute.

E. Reiter and C. Mellish. 1992. Using classification to generate text. In *Proceedings of the 30th Annual Meeting of the Association for Computational Linguistics.*, pages 265–272.

E. Reiter. 1990. Generating appropriate natural language object descriptions. phd thesis. *Harvard*.

E. Reiter. 1994. Has a consensus NL generation architecture appeared, and is it psychologically plausible? In *Proceedings of the 7th. International Workshop on Natural Language generation*, pages 163–170.

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

C. Shum and R. Muntz. 1987. Implicit representation for extensional answers. In L. Hershberg, editor, *Expert Database Systems*. Tysons Corner.