An Efficient Denotational Semantics for Natural Language Database Queries

Richard A. Frost and Randy J. Fortier

University of Windsor, Canada

Abstract. Early work on natural language database query processing focused on theories of compositional semantics. Recent work concentrates on the translation of NL queries to SQL where semantics is primarily used in an ad hoc manner to guide syntactic translation. Here, we argue that there remains a need for an efficiently-implementable denotational semantics for NL DB queries, and show how this can be achieved by integrating a relatively little-known semantics for transitive verbs with a new efficiently-implementable semantics for negation.

1 Introduction

1.1 Why Do We Need a Denotational Semantics?

Researchers have studied NL DB query processing for over forty years. Androutsopoulos et al (1995) survey early work. Since then, there has been the annual NLDB conference. A review of this and other literature, shows that a) although early work focused on formal compositional semantics, none of the proposed semantics has gained widespread use, and b) much recent work has focused on translating NL queries to SQL, rather than attempting to directly interpret the queries with respect to a formal denotational semantics. Although remarkable achievements have been made in this endeavor, some shortcomings remain:

1) Hsu and Parker (1995) argue that it is not easy to express queries containing generalized quantifiers (GQ) in SQL. In particular it is difficult to create efficient SQL expressions for such queries. An example GQ query is "*Find all patient-disease pairs such that patient* `p` *has [some | all | no | not all] symptoms associated with disease* `d`". They propose an extended SQL, and a translator from their extended SQL to an optimized expression in conventional SQL. As such, they do not claim that SQL is unable to accommodate GQ queries, but that efficient formulations cannot be easily expressed.

2) Rao et al (1996) extend the work of Hsu and Parker and show that even the optimal translation of "*no*" and "*not all*" types of GQ queries to SQL results in "abysmal performance results". They do not criticize Hsu and Parker, but identify the reason for poor performance as resulting from the hidden complementation in such queries. They continue by showing why SQL and conventional database data structures are ill-equipped to deal with such queries.

3) As a corollary of 2) above, we claim that there is no efficient compositional and orthogonal method for translating quantified queries to SQL. A method

Z. Kedad et al.(Eds.): NLDB 2007, LNCS 4592, pp. 12–24, 2007.

is "compositional" if the meaning of a composite query is computed from the meanings of its parts using a small set of rules. A method is "orthogonal" if components have meanings that are independent of context, and a change to a component of a query only affects the meaning of the whole query in a well-defined manner. To illustrate the problem, suppose that we have an `OrbitsRelation` containing the tuples (Phobos, Mars), (Mars, Sol) etc., but not (Sol, anything). Consider the queries q1 = *"List all things that orbit a planet"* and q2 = "*List all things that orbit no planet"*. SQL translations are:

```
q1 = SELECT SubjectName FROM OrbitsRelation
     WHERE ObjectName IN (SELECT Name FROM Planets)
q2 = SELECT SubjectName FROM OrbitsRelation
     WHERE ObjectName NOT IN (SELECT Name FROM Planets)
```

Now consider the queries q3 = "*Does Sol orbit a planet*" and q4 = "*Does Mars orbit no planet"*. A compositional approach would include the relations returned by q1 and q2 respectively, and the resulting answers would be False for q3 and True for q4, which are correct. However, a compositional translation of the query q5 = "*Does Sol orbit no planet*", which should make use of the relation returned by q2, would return the incorrect answer due to the fact that Sol does not appear as subject in the orbits relation. To obtain the correct answer to queries of the form "*Does X REL no Y*", we need an SQL expression which is quite different in structure from the SQL expression for queries "*Does X REL a Y*", and which, according to Rao et al, would be highly inefficient. Because these two types of query have the same syntactic structure, a compositional and orthogonal method should translate both to SQL expressions with similar structure. Such a method would result in inefficient expressions for both types of query. In a sense, there is an impedance mismatch between NL and SQL.

4) Existing NL to SQL translators continue to be error-prone and unable to answer queries that one would reasonably expect. Bhootra (2004) conducted an experiment with Microsoft's English Query (EQ) and Access English Language Front End (ELF) (www.elfsoft.com). A set of questions was created which could reasonably be asked of the Northwind sample database that was shipped with MS SQL. The questions were used to create interfaces to the database using tools provided with EQ and ELF. The interfaces returned 42% (EQ) and 19% (ELF) incorrect answers for the set of questions that were used to generate them. The lack of accuracy of NL to SQL translators has also been identified by Popescu et al (2003) who introduce a new approach, together with a system called PRECISE, for constructing reliable NL DB query interfaces. The method involves matching words in the query to attributes in the database and a subsequent syntactic translation of the NL query to an SQL expression. Although their approach addresses the problem of accuracy, it does not address the problem of generalized quantifiers, nor does it appear to be possible to extend the approach to cover modal or intensional databases (see next).

5) SQL cannot express queries with modal or intensional constructs, such as *"has it always been the case that . . ."* and "*Does Paul believe that ...*". These

limitations are being addressed, by others, who have proposed to extend SQL or to create new SQL-like query languages.

We conclude that there remains a need for an efficiently-implementable denotational semantics for NL DB queries: a) to accommodate queries containing generalized quantification, b) to extend the scope of query processors beyond the first-order expressibility of SQL, and 3) to bridge the gap between Linguists who want to explain natural language, and Computer Scientists who want to build efficient NL interfaces.

In the following, we present an efficient semantics for a subset of NL DB queries. The semantics is a denotational semantics in that all denotations are functions (some of which are constant valued functions) and that the denotations of compound expressions are computed from the denotations of their components using function application and function composition only, according to their syntactic structure.

1.2 An Overview of the Proposed Approach

Work on the semantics of natural language queries falls into three categories:

1) Work, usually carried out by Computer Scientists, who develop new ad hoc but efficient semantics that are not based on established linguistic theories. Examples are described in the survey by Androutsopoulos et al (1995). More recent work includes Owei (2003), Popescu et al (2003), Duesterhoeft and Thalheim (2004), Little et al (2004), Tseng and Fan (2005), and Boonjing and Hsu (2006).

2) Work based on well-defined formal theories, usually carried out by Linguists who are not particularly concerned with efficiency. An example is Nelken and Francez (2002) which contains reference to many other similar papers.

3) Work which involves the modification and/or extension of established linguistic theories for use in database query processing. Most of this is based on Montague-like Semantics (Montague 1974). Examples are: Main and Benson (1983), Clifford and Warren (1983), Clifford (1990), Lapalme and Lavier (1993), Frost and Boulos (2002), Lee and Park (2002), and Cimiano et al (2007).

Our approach falls into category 3). We add a little-known explicit semantics for transitive verbs to Montague Semantics. The effect is similar to that proposed by Main and Benson (1983), and later by Clifford (1990). We convert the extended semantics to a more efficiently-implementable form based on sets and relations rather than characteristic functions. Finally, we add a relatively-new efficient semantics for negation proposed by Frost and Boulos (2002).

Our approach differs from most of the approaches in category 3 (except for Frost and Boulos) - we define denotations directly in terms of database relations, whereas all other researchers have used Montague's intensional logic (IL) as an intermediate representation. None have been able to extend their approach to handle arbitrary negation, and none have been able to directly generate efficiently-implementable denotations of queries without subsequent post-processing of those denotations. Montague stated that IL was dispensable, and

we believe that it should dispensed with early in the process in order to obtain an efficient and compositional denotational semantics. We discuss, in section 4, what we have achieved, and what more needs to be done.

2 Montague Semantics (MS) and Its Shortcomings

The following is a brief introduction to some of Montague's ideas. In particular, we will say little here about modal or intensional aspects of natural language as these topics require substantial background discussion which can be found in, for example, (Montague 1974), Partee (1975), and Dowty, Wall and Peters (1981).

Montague claimed that natural language can be described in terms of a formal syntax and an associated compositional semantics. The relationship between syntax and semantics is similar to that in the denotational-semantics approach to the formal specification of programming languages, with the exception that expressions of a natural language have first to be "disambiguated" before interpretation. Such disambiguation involves mapping natural-language expressions to one or more unambiguous expressions in a syntactic algebra. These expressions are then mapped to semantic expressions through a homomorphism.

In MS, each disambiguated syntactic expression of English denotes a function in a function space constructed over a set of entities, the Boolean values `True` and `False`, and a set of states, each of which is a pair consisting of a "possible world" and a point in time. The functions are defined using the notation of lambda calculus. Each syntactic category is associated with a single semantic type. Each syntax rule is associated with a semantic rule which shows how the meanings of composite expressions are computed from the meanings of their constituents. The primary rule for syntactic composition is juxtaposition. The primary rule for semantic composition is function application.

Ignoring intensional aspects, common nouns such as "*planet*" and intransitive verbs such as "*spins*" denote predicates over the set of entities, i.e. characteristic functions of type `entity → bool`, where `x → y` denotes the type of functions whose input is a value of type `x` and whose output is of type `y`.

One of Montague's insights is that proper nouns do not denote entities directly. Rather, they denote functions defined in terms of entities. For example, the proper noun "*Phobos*" denotes the function `λp p Phobos` where `Phobos` represents the entity Phobos. (For readers not familiar with the lambda calculus, the expression `λx e` denotes a function which, when applied to an argument `y`, returns as result the expression `e` with all instances of `x` in it replaced by `y`.)

According to the rules proposed by Montague, the phrase "*Phobos spins*" is interpreted as follows, where `a => b` indicates that `b` is the result of evaluating `a`. Note that in this paper the denotation of a word is indicated by non-italic monospaced font. For example, `spins` is shorthand for the denotation of the word "spins", which according to Montague is a unary predicate:

```
(λp p Phobos) spins => spins Phobos
```

Quantifiers such as "*every*", and "*a*" denote higher-order functions of type `(entity→bool)→((entity→bool)→bool)`, e.g. the quantifier "*every*" denotes the

function λpλq ∀x (p x) → (q x), where → is overloaded to denote logical implication here. Accordingly, the phrase "*every planet spins*" is interpreted as:

```
(λpλq ∀x p(x) → q(x)) planet spins
     => (λq ∀x planet(x) → q(x)) spins
     => ∀x planet(x) → spins(x)
```

Constructs of the same syntactic category denote functions of the same semantic type, e.g. the phrases "*Mars*" and "*every planet*" are both deemed to denote functions of type `(entity → bool) → bool`.

The resulting approach is highly orthogonal: many words that appear in differing syntactic contexts denote a single polymorphic function thereby avoiding the need to assign different meanings in these different contexts. For example, the word "*and*", which can be used to conjoin nouns, verbs, term-phrases, etc., denotes the polymorphic function `λgλfλx (g x) & (f x)`. For example, the phrase "*Phobos and Deimos spin*" is interpreted as shown below (where identifiers representing entities begin with a capital letter):

```
=> ((λgλfλx (g x) & (f x))(λp p Phobos) (λp p Deimos)) spin
=> (λx ((λp p Phobos) x) & ((λp p Deimos) x) ) spin
=> ((λp p Phobos) spin) & ((λp p Deimos) spin)
=> (spin Phobos) & (spin Deimos)
```

Montague Semantics has a number of shortcomings when used as a basis for the interpretation of NL DB queries: a) it is not fully compositional as it does not provide a direct denotation for transitive verbs. Instead, MS uses a convoluted syntactic process involving "relational notation" and a "delta *" operator (see page 216 in Dowty et al 1981, for details), b) MS cannot accommodate queries such as "*Does Phobos orbit and Luna orbit Mars.*" due to the fact that the phrases "*Phobos orbit*" and "*Luna orbit*" cannot be given straightforward denotations using function application, as the input types of the denotations of "*Phobos*" and "*Luna*" are incompatible with the type of the denotation of "*orbit*", c) direct implementation of MS is computationally intractable. Phrases such as "*every planet spins*" require all entities in the universe of discourse to be examined (see the interpretation of this phrase given earlier), and d) Montague gave no details of how negation should be accommodated w.r.t. the closed world assumption. We address these issues in the next section.

3 The Proposed Approach

3.1 A Little-Known Semantics for Transitive Verbs

It is possible to give a direct denotation for transitive verbs thereby avoiding a convoluted manipulation of an intermediate representation. We begin by noting that although Montague defined the denotation of proper nouns as, for example, `λp p Phobos`, he viewed such denotations as being of type `(entity → bool)`

→ `bool`. This creates a difficulty when, attempting to define a denotation for transitive verbs, e.g. the denotation of "*discover*" would have to be of type:

`((entity → bool) → bool) → (entity → bool)` so that the denotation of "*discovered Phobos*" would be of the correct type for input to the denotation of "*Hall*". This does not appear to be possible.

The solution follows from the fact that the type of denotations such as `λp p Phobos` is more polymorphic than Montague stated. It is of type `(entity → *) → *` where `*` can be any type. This allows us to define the denotations of transitive verbs directly as follows : `discover = λz z(λxλy disc_pred(y,x))`

This is similar to, but not the same as, that proposed by Main and Benson (1983) and Clifford (1990). The following uses this denotation. The polymorphic type of `λq q Phobos` allows the lambda conversion at step 3:

```
      Hall          ( discovered              Phobos )
(λp p Hall) ((λz z(λxλy disc_pred(y,x))) (λq q Phobos))
=> (λp p Hall) ((λq q Phobos)(λxλy disc_pred(y,x)))
=> (λp p Hall) ((λxλy disc_pred(y,x)) Phobos)
=> (λp p Hall) (λy disc_pred(y,Phobos))
=> (λy disc_pred(y,Phobos)) Hall
=> disc_pred(Hall,Phobos)
```

Barbara Partee (personal communication) has pointed out that the above approach is not standard in linguistics, but that Dr. Kratzer, at the University of Massachusetts Amherst, has done something similar (Kratzer 2003), and Hendricks (1993) has proposed type-lifting to achieve a similar result in a more powerful semantic theory. Discussion of polymorphic types for transitive verbs and termphrases occurred over twenty years ago (e.g. Partee and Rooth 1983). Also, it is most likely that Montague was aware of the polymorphic type of termphrases, but chose to fix the type for linguistic reasons as it can be argued that the simpler type is more linguistically plausible. In addition, similar formalizations have been proposed in the context of logic programming, e.g. Blackburn and Bos (2005) who attribute it to Robin Cooper at Goteborg University.

The second problem of not being able to provide a compositional semantics for phrases such as "*Phobos orbits*" can be easily solved by extending MS to allow denotations to be created through function composition as well as function application. For example, the denotation of "*Phobos orbits*" is `phobos . orbit`. Thus, queries such as "*Does Phobos orbit and Deimos orbit Mars?*" are now interpreted as shown below using the denotation of "*and*" given earlier:

```
((phobos.orbits) and (luna.orbits)) mars
=>((phobos.orbits)mars) & ((luna.orbits)mars)
=>(phobos(orbits mars)) & (luna(orbits mars))
```

This approach accommodates a wide range of queries such as "*Did Hall discover and is Mars orbited by Phobos*", etc.

3.2 An Efficient Set-Theoretic Version of Montague Semantics

The computational intractability of MS results from denotations such as ∀x `planet(x)` → `spins(x)` which require that characteristic functions of sets (the denotations of common nouns and intransitive verbs) be applied to all entities in the universe of discourse. Other researchers, referred to earlier, who have based their semantics on Montague's approach, have not addressed this problem. Our solution is to use the sets themselves as denotations, rather than their characteristic functions, and convert all other denotations appropriately, e.g.

```
spins  = {Mars, Jupiter, Phobos, ..      moon = {Phobos, Deimos, ..
planet = {Mars, Jupiter, Mercury, ..     antibiotic = [Penicillin, ..
person = {Hall, Kuiper, Kowal, Fleming ..
```

The denotations of proper nouns are defined in terms of set membership, e.g.:

```
phobos = λp Phobos ∈ p
```

Quantifiers are defined in terms of set operators, e.g.:

```
every = λsλt s ⊆ t          a = λsλt s ∩ t  = {}
```

Conversion of the denotation of "*discover*" to a set-theoretic version gives:

```
discover = λq {x |(x,image_x) ∈ collect disc_rel & q image_x }
     where disc_rel = {(Hall,Phobos), (Hall,Deimos),
                       (Kuiper,Nereid),(Fleming, Penicillin), etc.}
```

In the above, the `collect` function is defined such that it returns a new binary-relation containing one binary tuple `(x, image_x)` for each member of the projection of the left-hand-column of `disc_rel`, where `image_x` is the image of `x` under the relation `disc_rel`, e.g. `collect disc_rel => {(Hall, {Phobos,Deimos}),etc.` An example application of `discover` is:

```
  discover phobos =>
λq {x |(x,image_x) ∈ collect disc_rel & q image_x }(λp Phobos ∈ p)
     => {x |(x,image_x) ∈ collect disc_rel & (λp Phobos ∈ p) image_x }
     => {x |(x,image_x) ∈ collect disc_rel & (Phobos ∈ image_x)}
     => {Hall}
```

One disadvantage of converting MS to a set-theoretic form is some loss of orthogonality. For example, the word "*and*" now needs more than one denotation:

```
term_and    = λpλqλs (p s) & (q s)          noun_and = λsλt s ∩ t
transvb_and = λpλqλr (p r) ∩ (q r)
```

Even with this slight loss of orthogonality, the mini-semantics is highly compositional: a) denotations are created using function application and function composition according to the syntactic structure of the query. For example, the query "*Did Kuiper discover and Hall discover a moon?*" is evaluated as:

```
(term_and (kuiper . discover) (hall . discover))(a moon)
```

and, b) words and phrases of the same syntactic category (e.g. "*a moon*" and "*Phobos*") denote semantic values of the same type as required by Montague.

It should be noted that syntactic ambiguity is accommodated by having the parser generate more than one disambiguated form. Semantic ambiguity is resolved, in a linguistically simple, yet efficient way, by assuming a right–to–left distribution. For example, according to our semantics, the query *"Did Kuiper discover and Hall discover a moon"* would be rewritten to:

```
(Kuiper (discover (a moon))) & (hall (discover (a moon)))
```

To obtain the answer to the other reading of this query, it would have to be restated as "*Was a moon discovered by Kuiper and Hall*'. (The denotations of passive forms of verbs are defined using the inverse of the associated relations).

3.3 Accommodating Negation

Linguists have studied negation extensively (e.g. Iwanska 1992). However, no efficiently-implementable compositional semantics for accommodating arbitrary negation in NL DB queries exists. The problem can be illustrated by considering the following queries with respect to the `disc_rel` relation given earlier: "*Did Fleming discover no moon?*" and "*Did Lewis discover no moon?*". Most compositional semantic theories will return the correct answer for the first query but the wrong answer to the second query (with respect to the closed-world assumption which is appropriate for many database applications). This is because `disc_rel` does not contain `Lewis` in its left-hand column owing to the fact that Lewis did not discover anything according to this database. One solution to this problem is to extend the discover relation to include `(x, ''nothing'')` for all entities `x` in the domain of discourse which do not already occur in the left-hand column. This is clearly impractical for all but very small databases, and is useless for databases with infinite domains. A more practical solution, proposed by Frost and Boulos (2002), is based on the notion that a set can be represented in two ways: explicitly by enumerating its members, or implicitly by enumerating the members of its complement. When a set is computed as the denotation of a phrase that involves negation, it is represented as a complement. The set operators are redefined to take complements as operands.

We now show how our approach to transitive verbs can be integrated with the above approach to negation. We begin by introducing two "constructors" `SET` and `COMP` to distinguish between sets defined in the usual way, and those which are defined by enumerating the elements of their complement, e.g.

```
  SET  {Phobos,Deimos,etc.} denotes the moons
COMP {Phobos,Deimos,etc.} denotes the non moons
```

Operations on sets and complements are defined as follows:

```
c_member e  (SET  s) = e ∈ s
c_member e  (COMP s) = not (e ∈ s)
c_union     (SET  s) (SET  t) = SET  (s ∪ t)
c_union     (SET  s) (COMP t) = COMP (t -- s)
c_union     (COMP s) (SET  t) = COMP (s -- t)
c_union     (COMP s) (COMP t) = COMP (s ∩ t)
c_intersect (SET  s) (SET  t) = SET  (s ∩ t)
```

```
c_intersect (SET  s) (COMP t) = SET  (s -- t)
c_intersect (COMP s) (SET  t) = SET  (t -- s)
c_intersect (COMP s) (COMP t) = COMP (s ∪ t)
c_subset    (SET  s) (SET  t) = s ⊆ t
c_subset    (SET  s) (COMP t) = (t -- s) = t
c_subset    (COMP s) (SET  t) = (all_entities -- s) ⊆ t
c_subset    (COMP s) (COMP t) = t ⊆ s
```

where `--` is set difference, `all_entities` denotes the set of all entities in the universe of discourse, and the definition of the set-cardinality operator # is extended as follows:

```
#(SET s) = #s      and #(COMP s)= #all_entities -- #(SET s)
```

In only one line, in the definition of `c_subset`, is it necessary to refer to the set of all entities. This is where we need to determine if a set represented by an enumeration of the members of its complement is a subset of a set that is represented by an explicit enumeration of it members (this computation occurs in the evaluation of the denotation of queries such as "*Does every thing that is orbited by no moon spin?*"). Fortunately, this part of the definition can be replaced by the following which refers only to the size of the set of all entities and not to the entities themselves:

```
c_subset (COMP s)(SET t) = ( #(s ∪ t) = #all_entities)
```

Redefinition of the denotations of most words is straightforward:

```
moon       = SET {Deimos, Phobos, etc.}    planet = SET {Jupiter,Mars}
spins      = SET {Jupiter,Mars,Phobos,etc} thing  = COMP {}
antibiotic = SET {Penicillin, etc.}
deimos     = λs c_member Deimos s           mars = λs c_member Mars s
a          = λsλt #(c_intersect s t) > 0    every = λsλt c_subset s t
no         = λsλt #(c_intersect s t) = 0    non  = λs COMP s
not        = s COMP s etc.
```

Evaluation of the denotation of the phrase "*non moon that spins*" would result in the following (assuming that the denotation of *that* is set to `noun_and`):

```
c_intersect (COMP{Phobos,Deimos, etc.})(SET {Jupiter,Mars,Phobos,etc})
     =>SET {Mars, Jupiter, etc.}
```

The problem with negation in queries such as *"Did Lewis discover no moon?"* is now solved by redefining the denotation of each transitive verb so that the function begins by applying the predicate given as argument to `SET{}` (representing the empty image of all entities that do not appear on the left-hand side of the associated relation), otherwise the result is returned in the form of a complement. If the predicate fails, the result returned is the same as that returned by the original denotation of the verb.

According to this approach, the new denotation of " `discover`" is:

```
discover = λq COMP({a |(a,b) ∈ disc_rel} -- result),if q(SET{}) = True
       SET result, otherwise
       where result = {x |(x,image_x) ∈ collect disc_rel & q image_x }
```

Now, the interpretation of "*discovered no moon*" returns the correct answer w.r.t. the `disc_rel` and *"Did Lewis discover no moon?"* is interpreted as:

```
=> (λp c_member Lewis p) (COMP ({a|(a,b) ∈ disc_rel} -- result))
       where
       result = {x|(x,image_x) ∈ collect disc_rel & (no moon) image_x}
=> (λp c_member Lewis p) (COMP ({Hall, Kuiper, Kowal, Fleming, etc}
                                -- {Fleming,etc})
=> (λp c_member Lewis p) (COMP {Hall, Kuiper, Kowal, etc.})
=> c_member Lewis (COMP {Hall, Kuiper, Kowal, etc.})
=> True
```

3.4 A Note on Compositionality, Efficiency, and Orthogonality

The semantics is highly compositional in that each word (after syntactic disambiguation) has a single denotation (meaning), and the meaning of composite queries is computed by function application and function composition only, in an order that is determined by the syntactic structure of the query. In addition, syntactic subcomponets of a query have meanings that can be computed independently of the whole query. Compositionality can be proven formally by induction on the length of expressions that can be derived from the context-free grammar (CFG) of the query language. The base case states that basic terms (words) of the query language have denotations of the required semantic type for their syntactic category (this follows directly from the semantic definitions that we have presented). The inductive step shows that the denotations of compound expressions that are created though use of any CFG rule are of the correct semantic type for the syntactic category, under the assumption that this is true for their components (this can be shown by considering each rule separately, and showing that for each alternative the computed denotation has the correct type. This also follows directly from our semantic definitions). In addition, the semantics overcomes the problem of "hidden complementation" discussed in section 1.1 item 2. The quantifier "*no*" can be treated in the same way semantically as "*a*" and "*every*" without incurring the inefficiency (and in some cases intractability) that would occur if complements of unary relations were enumerated explicitly.

As a consequence of the compositionality and the efficient treatment of negation, our semantics is highly orthogonal: a) the meaning of words and phrases within a query is, in most cases (with the exception of "*and*", independent of their context, b) if the meaning of a word is changed, the effect is propagated through the evaluation process in a well-defined way, and c) words and phrases of the same syntactic category, such as "*a*" and "*no*" can be interchanged without affecting the efficiency of the semantic evaluation process.

3.5 An Implementation and Evaluation

We have implemented the semantics directly in Miranda, a higher-order functional programming language. Example results are:

```
every (thing $that (orbits (no moon))) (orbits (no planet))    => False
a (non moon) (orbits sol)                                      => True
every moon (orbits (no moon))                                  => True
sol (orbits (a (non moon)))                                    => False
not (every moon) (is_orbited_by phobos)                        => True
a (moon $that (was_discovered_by hall))(does(not(orbit earth)))=> True
moon $that (was_discovered_by hall) => SET  [Phobos, Deimos]
orbits (no planet)              => COMP [Phobos, Deimos, Nereid, etc.]
non moon                        => COMP [Phobos, Deimos, Nereid, etc.]
discovered(a(moon $that (does(not(orbit (mars $term_or uranus)))))
                                => SET [Kowal, etc.]
```

To determine the viability of the approach, we have integrated the semantics with a functional parser using techniques described in Frost (2006), and have deployed the resulting application in a Public-Domain SpeechWeb (Frost 2005) which provides a speech interface. Details of how to access the speech interface from a PC through the Opera web browser are described in (Frost and Fortier 2007). A video demonstration is available at:

`http://www.cs.uwindsor.ca/~speechweb/movie.mov`

4 Concluding Comments

We have identified a need for an efficiently-implementable denotational semantics for NL DB queries. We have integrated an explicit little-known denotation for transitive verbs with Montague Semantics. Then, unlike others, we have transformed the extended semantics to a computationally-tractable form by replacing characteristic functions of sets by the sets themselves and modifying all denotations accordingly. We then added a relatively-new semantics for negation. The resulting approach has three advantages compared to others: a) all forms of generalized quantification are accommodated efficiently in a compositional and orthogonal way, b) the explicit denotation for transitive verbs improves compositionality, and c) by defining the denotations of words, phrases, and queries directly in terms of database relations, we avoid the impedance mismatch that occurs between NL and SQL as discussed in section 1.1.

The approach accommodates queries containing common and proper nouns, transitive and intransitive verbs, conjunction, disjunction, and arbitrarily-nested quantification and negation. However, this is insufficient for many applications. We are currently extending the semantics to accommodate prepositional phrases as in *"Did Galileo discover Europa with a telescope?"*, indirect objects as in *"Did Galileo give Europa its name?"*, and aggregates as in *"What is the average mass*

of the planets?." We will then begin the more-ambitious task of accommodating modal and intensional constructs. At that point we will compare our approach with that of others who are developing extended forms of SQL.

References

1. Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural Language Interfaces to Databases: An Introduction. J. of Lang. Engineering 1(1), 29–81 (1995)
2. Blackburn, P., Bos, J.: Representation and Inference for Natural Language. A First Course in Computational Semantics. CSLI Publications, Stanford, CA (2005)
3. Bhootra, R.: Natural Language Interfaces: Comparing English Language Front End and English Query. Master's Thesis, Virginia Commonwealth Univ (2004)
4. Boonjing, V., Hsu, C.: A New Feasible Natural Language Database Query Method. International Journal on Artificial Intelligence Tools 15(2), 323–330 (2006)
5. Cimiano, P., Haase, P., Heizemann, J.: Porting Natural Language Interfaces Between Domains — An Experimental User Study with the ORAKEL System. In: Proc. 12th Int. Conf. Intelligent User Interfaces IUI'07, pp. 180–189 (2007)
6. Clifford, J.: Formal Semantics and Pragmatics for Natural Language Querying. In: van Rijsbergen, C.J. (ed.) Cambridge Tracts in Theoretical Computer Science 8, Cambridge University Press, Cambridge (1990)
7. Clifford, J., Warren, D.S.: Formal Semantics for Time in Databases. ACM Trans. on Database Systems 8(2), 215–254 (1983)
8. Dowty, D.R., Wall, R.E., Peters, S.: Introduction to Montague Semantics. D. Reidel Publishing Company, Dordrecht, Boston, Lancaster, Tokyo (1981)
9. Duesterhoeft, A., Thalheim, B.: Linguistic based search facilities in snowflake-like database schemes. Data and Knowledge Engineering 48, 177–198 (2004)
10. Frost, R.A.: Realization of Natural-Language Interfaces using Lazy Functional Programming. ACM Comput. Surv. 38(4) Article No. 11 (2006)
11. Frost, R.A.: Call for a Public-Domain SpeechWeb. Commun. ACM 48(11), 45–49 (2005)
12. Frost, R.A., Boulos, P.: An efficient compositional semantics for natural-language database queries with arbitrarily-nested quantification and negation. In: Cohen, R., Spencer, B. (eds.) Advances in Artificial Intelligence. LNCS (LNAI), vol. 2338, pp. 252–267. Springer, Heidelberg (2002)
13. Frost, R.A., Ma, X., Shi, Y.: A Browser for a Public-Domain SpeechWeb. Accepted for WWW 2007 (2007)
14. Hendriks, H.: Studied flexibility: categories and types in syntax and semantics. Doctoral Thesis, Universiteit van Amsterdam (1993)
15. Hsu, P-Y., Parker, D.S.: Improving SQL with Generalized Quantifiers. In: Proc. 11th Int. Conf. on Data Engineering, pp. 298–305 (1995)
16. Iwanska, L.: A General Semantic Model of Negation in Natural Language: Representation and Inference. Doctoral Thesis, Computer Science, University of Illinois at Urbana-Champaign (1992)
17. Kratzer, A.: The event argument and the semantics of verbs, (2003), `http://semanticsarchive.net/GU1NWM4Z`
18. Lapalme, G., Lavier, F.: Using a functional language for parsing and semantic processing. Computational Intelligence 9, 111–131 (1993)
19. Lee, H., Park, J.C.: Interpretation of Natural Language Queries for Relational Databases with Combinatory categorial Grammar. Int. J. Comput. Proc. Oriental Lang 15(3), 281–303 (2002)

20. Little, J., de Ga, M., Ozyer, T., Alhajj, R.: Query Builder: A Natural Language Interface for Structured Databases. In: Aykanat, C., Dayar, T., Körpeoğlu, İ. (eds.) ISCIS 2004. LNCS, vol. 3280, pp. 470–479. Springer, Heidelberg (2004)
21. Main, M.G., Benson, D.B.: Denotational semantics for natural language question answering programs. American J. of Comput. Ling. 9(1), 11–21 (1983)
22. Montague, R.: In Formal Philosophy: Selected Papers of Richard Montague. Thomason, R.H. (ed.) Yale University Press, New Haven CT (1974)
23. Nelken, R., Francez, N.: Bilattices and the Semantics of Natural Language Questions. Linguistics and Philosophy 25(1), 37–64 (2002)
24. Owei, P.: Development of a Conceptual Query Language: Adopting the User-Centered Methodology. The. Computer Journal 46(6), 602–624 (2003)
25. Partee, B., Rooth, M.: Generalized conjunction and type ambiguity. In: Bauerle, R., Schwarze, C., von Stechow, A. (eds.) Meaning, Use and Interpretation of Language, pp. 361–383. Mouton de Gruyter, Berlin (1983)
26. Partee, B.H.: Montague Grammar and Transformational Grammar. Linguistic Inquiry 6(2), 203–300 (1975)
27. Popescu, A.M., Etzioni, O., Kautz, H.: Towards a Theory of Natural Language Interfaces to Databases, IUI'03 Miami, Florida, pp. 149–157 (2003)
28. Rao, S., Badia, A., Van Gucht, D.: Processing queries containing generalized quantifiers. Tech. Report TR 428. Computer Science, Indiana Univ. (1996)
29. Tseng, F.S.C., Fan, T.K.: Extending the Concepts of Object Role Modeling to Capture Natural Language Semantics for Database Access. In: Proc. IASTED Conference on Databases and Applications (2005)