A Dutch to SQL database interface using
Generalized Quantifier Theory

Dirk Speelman[1] Geert Adriaens[1,2]

[1] University of Leuven Center for Computational Linguistics,
Blijd-Inkomststraat 21, 3000 Leuven, Belgium
[2] Siemens-Nixdorf Software Centre Liège,
Rue des Forêts 2, 4020 Liège, Belgium

Abstract

This paper presents the treatment of quantification as it was implemented in a prototype of a natural language relational database interface for Dutch. It is shown how the theoretical 'generalized quantifier' apparatus introduced in formal semantics by Barwise and Cooper can be tuned to implementational feasibility. Compared to the traditional treatment of quantification, the alternative presented here offers greater expressive power, greater similarity to natural language and, as a consequence, the possibility of a more straightforward translation from natural language to formal representation.

1 INTRODUCTION

In the prototype at hand, as in many database interfaces, the natural language input is translated to a conventional formal query language, viz. SQL, the most widely used and supported of these languages. The resulting SQL queries can then be passed to an already existing SQL interpreter.

The translation procedure from Dutch to SQL is split up in two consecutive major steps, using a logic-based intermediate semantic representation called General Semantic Representation (GSR). The functionality of the whole database interface, including the SQL interpreter, was seen as a straightforward implementation of the formal semantic Montague-style (Montague, 1973) mechanism of indirect interpretation of natural language (see Fig. 1).

Figure 1: Major processing steps in the DB interface

'Grafting' formal semantic processing steps upon an NL, database interface architecture has been proposed and (successfully) worked out before in a somewhat comparable project carried out at the university of Essex (see De Roeck, Fox, Lowden, Turner & Walls, 1991). The main concern in that project was to clearly separate domain (= database) dependent semantic information from domain independent semantic information. In the project presented here a similar but more general objective was to maximize the separation of the NLP data and functionality of the system from its purely database oriented data and functionality, GSR being the interface structure.

The main topic of this paper, treated in section 3, is the application of 'generalized quantifier theory' in GSR. Having become classical in mathematical and some theoretical linguistic studies on quantification (see resp. Mostowski, 1957 and Barwise & Cooper, 1981), the theory is now beginning to be appreciated in AI (and NLP) for its richness and flexibility. Probably the best illustration of this upcoming interest is the incorporation of 'generalized quantifiers' in the popular Conceptual Graph knowledge representation formalism (see e.g. Sowa, 1991). A somewhat differently
oriented AI-application also using 'generalized quantifiers' can be found in (Kaan, Kas & Puhland, 1990). These applications concentrate on the expressive and inferential power of 'generalized quantifier theory' respectively. The program presented here additionally illustrates how the use of (a variant of) the theory reduces the complexity of implementing the translation from natural to formal and artificial language.

2 GSR: GENERAL OUTLINE

The question what GSR should look like was to a large extent tackled in a very pragmatic way. As far as the linguistic module of the program is concerned, the following criteria were formulated: GSR had to be a formal representation

(i) with sufficient expressive power so that every possibly useful query can be formulated in it in a not too complex fashion,
(ii) that is relatively easy to render computationally, starting off from natural language.

A general observation is that, considering the kind of NL sentences one can expect as input to the system, GSR inevitably had to differ from logical formalisms such as the ones used in formal semantics (focussing on propositions). In view of the general decision to work with intermediate semantic expressions the denotation of which is the answer to the NL questions, the basic types of complete expressions listed in Fig. 3 were found useful. In this figure \( \varphi \) stands for an arbitrary proposition in some logical language \( L \). The extension of \( L \) created by introducing these new types will be called \( L' \).

(i) propositions (format: \( \varphi \)), to be used when people ask yes-or-no questions
(ii) set expressions (format: \( \{ x \mid \varphi \} \)), to be used when people ask non-numerical identity questions
(iii) mathematical expressions (format: \( \#(x \mid \varphi) \)), to be used when people ask for numerical information

Figure 3: GSR: types of expressions

3 FROM DUTCH TO GSR

3.1 \( \exists \) and \( \forall \): problems

The traditional way of coping with quantification in NL database interfaces is by using \( \exists \) and \( \forall \), the classical first order predicate logic (PL) instruments (see e.g. Warren & Pereira, 1982). This approach, however, does not meet the criteria set out above. To illustrate this, we basically rely on two observations Barwise & Cooper introduced, are left out. Fur-

| Example | Translation |
|---------|-------------|
| 1 | Zijn alle werknemers gehuwd? Are all employees married? | \( \forall x ( \text{employee}(x) \rightarrow \text{married}(x)) \) |
| 2 | Zijn beide werknemers gehuwd? Are both employees married? | \( \exists x \exists y (\text{employee}(x) \land \text{employee}(y) \land \text{married}(x) \land \text{married}(y)) \) |
| 3 | Zijn precies drie werknemers gehuwd? Are exactly three employees married? | \( \exists x \exists y \exists z (\text{employee}(x) \land \text{employee}(y) \land \text{employee}(z) \land \text{married}(x) \land \text{married}(y) \land \text{married}(z)) \) |
| 4 | Zijn meer dan de helft van de werknemers gehuwd? Are more than half of the employees married? | \( \forall x (\text{employee}(x) \rightarrow (x < \frac{1}{2}) \lor (x > \frac{1}{2})) \) |

Figure 4: Translation of quantification from Dutch to PL

A second, more serious reason for the inadequacy of \( \exists \) and \( \forall \) is that some forms of NL quantification can only be expressed in a very complex way (e.g. Fig. 4, examples 2 and 3) or simply cannot be expressed at all (e.g. Fig. 4, example 4). Here criterion (i) is not satisfied.

A third problem, mentioned in Kaan, Kas & Puhland (1990), is that in practice, e.g. in implementations, one is tempted to make rough translations, and to neglect nuances or strong conversational implications in natural language, when one is limited to \( \exists \) and \( \forall \). So, for instance, in Warren & Pereira (1982) 'a', 'some' and 'the' all are simply interpreted as \( \exists \).

3.2 L(GQ)': a solution

There are many ways to try and get around the shortcomings of the traditional approach. To score better on criterion (i), i.e. to increase expressive power, one could consider the introduction of numbers in the logical formalism. Only, one can imagine that, if made in an ad hoc way, this extension could result in a hybrid formalism (with respect to quantification) showing an even greater syntactical mismatch with NL (decreasing the score on criterion ii).

A solution for these problems was first explored by Montague (1973), and later thoroughly worked out by Barwise & Cooper (1981) in a formalism called L(GQ). In contrast to traditional PL, which only has \( \exists \) and \( \forall \), the language of generalized quantifiers L(GQ) specifies no limitation of the number of primitives to express quantification. All kinds of determiners can be used. The translation of the examples of Fig. 4 to L(GQ)' is given in Fig. 5. Some special notational conventions Barwise & Cooper introduced, are left out. Fur-
thermore a relational perspective (see Zwarts, 1983) is used.

\[
\begin{array}{|c|c|}
\hline
1 & \text{Zijn alle werknemers gehuwd?} \\
\sim & \text{all}(x \mid \text{employee}(x)), \text{married}(x) \\
\hline
2 & \text{Zijn beide werknemers gehuwd?} \\
\sim & \text{both}(x \mid \text{employee}(x)), \text{married}(x) \\
\hline
3 & \text{Zijn precies drie werknemers gehuwd?} \\
\text{exactly-3} & \text{is}(x \mid \text{employee}(x)), \text{married}(x)) \\
\hline
4 & \text{Zijn meer dan de helft van de werknemers gehuwd?} \\
\sim & \text{more-than-a-half}(x \mid \text{employee}(x)), \text{married}(x)) \\
\hline
\end{array}
\]

Figure 5: Translation of quantification from Dutch to L(GQ')

The denotation of L(GQ') determiners is defined at a meta-level. Some examples are given in (1) to (4). In these examples \(I\) stands for an interpretation function mapping an expression on its denotation.

\[
I(\text{all}(v, x)) = \begin{cases} 
\text{True} & \text{if } I(v \setminus x) = \emptyset \\
\text{False} & \text{otherwise}
\end{cases} \quad (1)
\]

\[
I(\text{the-some}(v, x)) = \begin{cases} 
\text{Undefined} & \text{if } \#I(v) \neq n \\
\text{otherwise} & \text{otherwise}
\end{cases} \quad (2)
\]

\[
I(\text{exactly-n}(v, x)) = \begin{cases} 
\text{True} & \text{if } \#(v \setminus x) = n \\
\text{False} & \text{otherwise}
\end{cases} \quad (3)
\]

\[
I(\text{more-than-a-half}(v, x)) = \begin{cases} 
\text{True} & \text{if } \#I(v) > \frac{n}{2} \\
\text{False} & \text{otherwise}
\end{cases} \quad (4)
\]

In Fig. 5 the structural similarity of the NL expressions is reflected in that of the L(GQ') expressions. Furthermore, all NL examples can be expressed almost equally easily in L(GQ'). By consequence, the formalism does not force people to be satisfied with rough translations. In short, the problems of traditional logical quantification are overcome.

### 3.3 L(GQ'): complications

Unfortunately, there are two reasons for not considering L(GQ') an ideal solution. The first problem actually is not typical of L(GQ), but of the fact that Barwise & Cooper take over the Montagovian way of coping with possible ambiguity due to phenomena of quantifier scope. In these cases one reading is generated in a straightforward way by Barwise & Cooper. To allow for alternative readings, they introduce extra machinery (called the 'quantification rule').

The latter mechanism, however convenient from a theoretical point of view, is rather implementation-unfriendly. It operates on complete structural descriptions (=non-trivial trees), and generates complete structural descriptions. Allowing for such a rule drastically changes the profile of the parser that is needed.

The second problem is that it is undesirable for GSR, being an interface language with a non-NLP module, to contain the set of (NL inspired) determiners that L(GQ') contains. It would probably be better if GSR had fewer primitives, preferably of a type not completely customary in traditional DBMSs.

### 3.4 GSR: an L(GQ') derivative

As a solution for these problems L(GQ') gets two new neighbours in the translation process, as shown in Fig. 6.

\[
\text{NL} \rightarrow \text{SRI} \rightarrow \text{L(GQ')} \rightarrow \text{GSR}
\]

Figure 6: Major processing steps in the NLP subsystem

In order to avoid the application of the 'quantification rule', the choice has been to first generate an expression that is neutral with respect to the scope of its quantifiers (SRI1), and then solve the scope problem in a second step, hereby generating an L(GQ') expression. The trick of first generating a scope-neutral expression is not new. For instance, it is used in the LOQUI system (see Gailly, Ribbens & Binot, 1990). The originality lies rather in the effort to respect well-formedness in the scope-neutral expressions.

Informally speaking, SRI is a predicate-logical formalism in which the arguments of the predicates are internally structured as the NL arguments of verbs. The most important consequence is that determiners are located within the predicate-arguments. To give an example, 'Werken alle werknemers aan twee projecten?' (Do all employees work on two projects?) would be represented as (5). For identity and cardinality questions the formats in Fig. 3 are made superfluous by the pseudo-determiners WH and CARD. For instance, the question 'Welke werknemers werken aan twee projecten?' (Which employees work on two projects?) is translated to (6).

\[
\text{work}(\text{all}(x \mid \text{employee}(x)), \exists(x \mid \text{project}(x))) \quad (5)
\]

\[
\text{work}(\text{WH}(x \mid \text{employee}(x)), \exists(x \mid \text{project}(x))) \quad (6)
\]

The translation of NL to SRI is a straightforward compositional process, comparable to the Barwise & Cooper processing of readings for which no 'quantification rule' is needed. The algorithm for going from SRI to L(GQ') is given in Fig. 7.

If an SRI expression contains a pseudo-determiner WH or CARD, the schema in Fig. 7 is adapted as follows. In the first step the arguments with real determiners are replaced by variables \(v_1\) up to \(v_n\), and the argument WH(S) or CARD(S) is replaced by a special variable \(v_0\). Further, the result \(\varphi\) of the normal second step is turned into a set expression or a numerical expression \(\{v_0 \mid S_0 \wedge \varphi\}\) or \(\#(\{v_0 \mid S_0 \wedge \varphi\})\). The third step, which is \(\varphi\)-internal, remains unchanged.

The essential part in Fig. 7 is the procedure that determines the possible scope-configurations. In the program only one, the most probable scope-configuration is generated. The algorithm states that the earlier some quantifier occurs in the NL expression, the larger its scope should be in the L(GQ') expression. In the
gram, this procedure proved to be amazingly accurate seen in Fig. 2, a GSH, expression is first translated to a

employees married?).

question is 'Zijn alle werknemers gehuwd?' (Are all

SIL to L(GQ)' to (SP, is given in (7) up to (9). 'The

semantic building blocks of traditional computer sci-

tics (see (1)-(4)). In the resulting I,(GQ)' derivative,

was originally the recta-level definition of their seman-

the interpreters used it, the database subsystem with

arate module, any extension or alteration of cat, be

probability, is generated. Since the procedure is a sep-

What remains to be overcome, is the fact that intro-

a NLP-type recognition tank. This problem is solved by giving L(GQ)' a righthand neigbonr (see Specimen, 1992, 85 98). The fllture goal, how-

formalism called DBIL This was clone for reasons of

4 FROM GSR TO SQL

As the NLP subsystem, the database subsystem is fully

implemented. However, we shall restrict ourselves to a

very brief sketch of its functionality here. As can be

seen in Fig. 2, a GSR expression is first translated to a

formalism called DBSR. This was done for reasons of

modularity, primarily for facilitating the extension of the

system to different target languages.

DBSR, which stands for DataBase specific Semantic

Representation, is a declarative relational database

query language that is both close to GSR and easily

translatable to any of the commercialized RDBMS

query languages. Apart from the treatment of quantifi-

cation the formalism is very similar to relational calcu-

lus. The major effort in the step from GSR to DBSR

lies in adapting GSR-terminology to concrete names

of tables and columns of a database. This is done using

a DB-lexicon, which can be seen as an augmented

ER-model of a database.

The last step, from DBSR to SQL, is extremely

straightforward. Sets and cardinality expressions are

translated to (sub)queries. Relations between sets or

cardinality expressions are translated to conditions for

(sub)queries.

For completeness, an example of the database sub-

system output is given. For the last example of the

foregoing section a DBSR expression and an SQL query

are given in (10) and (11) respectively. YES contains

only 'Yes'.

\[
\{ x_1 \mid \text{employee}(x_1) \} \cup \{ x_1 \mid x_1 \text{ married} = \text{"T"} \} = \emptyset
\]

\[
\text{SELECT } *
\text{ FROM YES}
\text{ WHERE NOT EXISTS}
( \text{SELECT } x_1*
\text{ FROM employee } x_1
\text{ WHERE NOT }(x_1 \text{ married} = \text{"T"}))
\]

\[
\{ x_1 \mid \text{employee}(x_1) \} \cup \{ x_1 \mid \text{ married}(x_1) \} = \emptyset
\]

Figure 7: Schema for translation from SR1 to L(GQ)'

NL fragment that was tested extensively with the pro-

gram, this procedure proved to be amazingly accurate
(see Speelman, 1992, 85–98). The future goal, however,
is that instead of one most probable reading a list of all possible readings, tagged with a degree of
probability, is generated. Since the procedure is a sepa-
rate module, any extension or alteration of can be
made without affecting the rest of the program.

What remains to be overcome, is the fact that intro-

Introducing a large set of determiners in GSR would burden the interpreters used in the database subsystem with

an extra, NLP-type recognition task. This problem is solved by giving L(GQ)' a righthand neigbonr (see Fig. 6 in which the determiners are replaced by what

was originally the meta-level definition of their semantics (see (1)-(4)). In the resulting L(GQ)' derivative, called GSR, the number of primitives (set, set inter-

section, set difference, set cardinality, ...) is drastically reduced. Furthermore, the new primitives are much

closer to, and even at the heart of, the procedural and semantic building blocks of traditional computer science in general, and of relational DBMSs in particular.

An example of the complete procedure, going from

SR1 to L(GQ)' to GSR, is given in (7) up to (9). The question is 'Zijn alle werknemers gehuwd?' (Are all

employees married?).

\[
m\text{arried}(\text{all} | \text{employee}(x))
\]

\[
\text{all}(\{ x_1 \mid \text{employee}(x_1) \} \cup \{ x_1 \mid \text{ married}(x_1) \})
\]

\[
\{ x_1 \mid \text{employee}(x_1) \} \cup \{ x_1 \mid \text{ married}(x_1) \} = \emptyset
\]

5 IMPLEMENTATION

The system is written in Common Lisp (according to the de facto standard Steele, 90) and generates standard SQL queries (ISO). It has proved to be a perfectly

portable product. Originally written on a Macintosh SE/30, it has afterwards been tested on several Symbols, Macintosh and PC platforms.

The major modules of the linguistic component are a 'letter tree' tool for efficient communication with the

lexicon, a transition network based morphological analysis tool, and an augmented chart parser for syntactic and semantic analysis.

6 CONCLUSION

In some subfields of formal semantics the traditional

logical apparatus for quantification, i.e. the use of

$\exists$ and $\forall$, is being abandoned in favor of 'generalized quantifiers', because the latter are both closer to nat-

ural language and richer in expressive power. In this

text it has been shown how this theory can be put to

use in a natural language database interface, an-

other field in which $\exists$ and $\forall$ had become traditional.

Some modifications had to be made in order to ren-

der the theoretical 'generalized quantifier' approach
more implementation-friendly. The major modifications were the introduction of a separate module to replace the 'quantification rule', and the shift from meta-level to logical representation of some settheoretical primitives.

References

[1] Harwise, J. & Cooper, R. (1981). 'Generalized Quantifiers and Natural Language'. Linguistics and Philosophy 4, 159-219.

[2] Codd, E.F. (1970). 'A Data Sublanguage Founded on the Relational Calculus'. ACM SIGFIDET Workshop on Data Description, Access and Control, November 1971.

[3] De Roeck, A.N., Fox, C.J., Lowden, R. G.T., Turner, R. & Walls, B.R. (1991). 'A Natural Language System Based on Formal Semantics'. Proceedings of the International Conference on Current Issues in Computational Linguistics. 268-281.

[4] Gailly, P.J., Ribbens, D. & Binot, J. L. (1990). 'La quantification en Traitement du Language Naturel'.

[5] ISO TC97/SC21/WG3 and ANSI X3H2 (1987). ISO 9075 Database Language SQL.

[6] Kaan, E., Kas, M. & Puhlau, R. (1990). 'Een procedure voor coderen met kwantoren'. TABU Bulletin voor Taalwetenschap 20 (4), 205-221.

[7] Montague, R. (1973). 'The Proper Treatment of Quantification in Ordinary English'. Hintikka J., Moravcsik J. & Suppes P. (eds.) Approaches to Natural Language. Dordrecht : Reidel. 221-242.

[8] Mostowski, A. (1957). 'On a Generalization of Quantifiers'. Fund. Math. 44, 12-36.

[9] Sowa, J.F. (1991). 'Towards the Expressive Power of Natural Language'. J.F. Sowa (ed.), Principles of Semantic Networks. San Mateo, California : Morgan Kaufmann. 157-189.

[10] Spelman, D. (1992). Een prototype voor een database-interface in Lisp. Vertaling van Nederlandse vragen naar SQL-queries, Computational Linguistics Thesis, University of Leuven (in Dutch).

[11] Steele, G.L. (1992). Common Lisp : The Language. Second Edition. Digital Press : Bedford MA.

[12] Warren, D.H. & Pereira, F.C.N. (1982). 'An Efficient Easily Adaptable System for Interpreting Natural Language Queries'. American Journal of Computational Linguistics 8, 110-119.

[13] Zwart, F. (1983). 'Determiners: a relational perspective'. Ter Meulen (ed.) Studies in Modeltheoretic Semantics. Foris : Dordrecht. 37-62.