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,
Blijde-Inkomststraat 21, 3000 Leuven, Belgium
[2] Siemens-Nixdorf Software Centre Liège,
Rue des Fories 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[1]. 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)[2]. 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 propagated and (succesfully) 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.

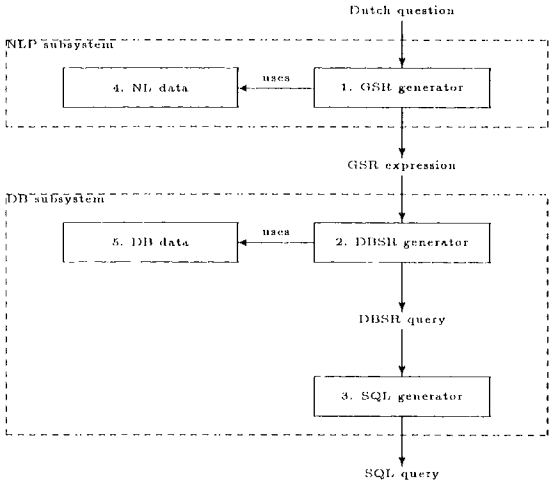

Figure 2: General architecture of the program

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

[1] In this paper the actual implementation is not in focus (see Speelman, 1992).

[2] Within a framework of machine translation, we can say that GSR is a kind of logic-based interlingua.

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 pragmatical 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 reach 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 φ 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: φ), 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 (1981) made to show a fundamental difference in the natures of NL and PL. Their observations will be 'transposed' to the computational application at hand.

The first observation is illustrated in figure 4, which contains some Dutch questions and their most natural PL' counterparts. Whereas the Dutch sentences have the same syntactic structure, their PL' counterparts have different formats. These and many other examples suggests that there is no trivially compositional way of translating NL expressions to their nearest PL' equivalents. The problem is that the quantificational information, which in NL has a fixed location, is spread over the PL' expression in a seemingly arbitrary way. It may be concluded that criterium (ii) for a good GSR is violated.

1	Zijn *alle* werknemers gehuwd ? 'Are all employees married?' $\rightsquigarrow$ $\forall x(employee(x) \rightarrow married(x))$
2	Zijn *beide* werknemers gehuwd? 'Are both employees married?' $\rightsquigarrow$ $\exists x_1(\exists x_2((x_1 \neq x_2) \wedge employee(x_1) \wedge employee(x_2) \wedge \forall y(employee(y) \rightarrow ((y = x_1) \vee (y = x_2))) \wedge married(x_1) \wedge married(x_2)))$
3	Zijn *precies drie* werknemers gehuwd? 'Are exactly three employees married?' $\rightsquigarrow$ $\exists x_1(\exists x_2(\exists x_3($ $(x_1 \neq x_2) \wedge (x_1 \neq x_3) \wedge (x_2 \neq x_3) \wedge$ $employee(x_1) \wedge employee(x_2) \wedge employee(x_3)$ $\wedge \forall y((married(y) \wedge employee(y)) \rightarrow$ $((y = x_1) \vee (y = x_2) \vee (y = x_3)) \wedge$ $married(x_1) \wedge married(x_2) \wedge married(x_3))))$
4	Zijn *meer dan de helft van de* werknemers gehuwd? 'Are more than half of the employees married?' $\rightsquigarrow$ —

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 criterium (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 implicatures 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 criterium (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 criterium 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.

1	Zijn *alle* werknemers gehuwd ?
$\rightsquigarrow$	$all(\{x \mid employee(x)\}, \{x \mid married(x)\})$
2	Zijn *beide* werknemers gehuwd?
$\rightsquigarrow$	$the\_2(\{x \mid employee(x)\}, \{x \mid married(x)\})$
3	Zijn *precies drie* werknemers gehuwd?
$\rightsquigarrow$	$exactly\_3$ $(\{x \mid employee(x)\}, \{x \mid married(x)\})$
4	Zijn *meer dan de helft van de* werknemers gehuwd?
$\rightsquigarrow$	$more\_than\_a\_2\_th$ $(\{x \mid employee(x)\}, \{x \mid married(x)\})$

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.

$$I(all(\varphi, \chi)) = \begin{array}{ll} True & (\text{if } (I(\varphi) \setminus I(\chi)) = \emptyset) \\ False & (\text{otherwise}) \end{array} \quad (1)$$

$$I(the\_n(\varphi, \chi)) = \begin{array}{ll} Undefined & (\text{if } \#(I(\varphi)) \neq n) \\ I(all(\varphi, \chi)) & (\text{otherwise}) \end{array} \quad (2)$$

$$I(exactly\_n(\varphi, \chi)) = \begin{array}{ll} True & (\text{if } \#(\varphi \cap \chi) = n) \\ False & (\text{otherwise}) \end{array} \quad (3)$$

$$I(more\_than\_a\_n\_th(\varphi, \chi)) = \begin{array}{ll} True & (\text{if } \#(I(\varphi) \cap I(\chi)) > \#(I(\varphi))/n) \\ False & (\text{otherwise}) \end{array} \quad (4)$$

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 uncustomary 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.

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 (SRI), 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 projekten?' (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 projekten?' (Which employees work on two projects?) is translated to (6).

$$work(all(\{x \mid employee(x)\}), 2(\{x \mid project(x)\})) \quad (5)$$

$$work(WH(\{x \mid employee(x)\}), 2(\{x \mid project(x)\})) \quad (6)$$

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_0) or CARD(S_0) is replaced by a special variable v_0. Further, the result φ 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 φ-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

$$P(D_1(S_1), D_2(S_2), \ldots, D_n(S_n))$$
$$\downarrow$$
Every argument $D_i(S_i)$ is replaced by a new, unique variable v_i $(i \in \{1, \ldots, n\})$
$$\downarrow$$
$$\ldots P(v_1, v_2, \ldots, v_n) \ldots$$
$$\downarrow$$
An independent procedure is run to determine the probable or possible scope of the determiners. The determiners are wrapped around the initial proposition according to this scope. Formally the scope-determining procedure generates for every probable or possible reading a permutation f of $\{1, \ldots, n\}$.
$$\downarrow$$
$$D_{f(1)}(\ldots D_{f(2)}(\ldots D_{f(n)}(\ldots P(v_1, v_2, \ldots, v_n) \ldots) \ldots) \ldots)$$
$$\downarrow$$
The remaining lacunes are filled up by adding, as shown, to every determiner D_i its original domain-set S_i, and the variable v_i that was introduced before to replace $D_i(S_i)$ $(i \in \{1, \ldots, n\})$
$$\downarrow$$
$$D_{f(1)}(S_{f(1)}, \{v_{f(1)} \mid D_{f(2)}(S_{f(2)}, \{v_{f(2)} \mid \ldots D_{f(n)}(S_{f(n)}, \{v_{f(n)} \mid P(v_1, v_2, \ldots, v_n)\}) \ldots\})\})$$

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

NL fragment that was tested extensively with the program, 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 separate 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 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 neigbour (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 intersection, 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?).

$$married(all(\{x \mid employee(x)\})) \tag{7}$$

$$all(\{x_1 \mid employee(x_1)\}, \{x_1 \mid married(x_1)\}) \tag{8}$$

$$\{x_1 \mid employee(x_1)\} \setminus \{x_1 \mid married(x_1)\} = \emptyset \tag{9}$$

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 quantification the formalism is very similar to relational calculus. The major effort in the step from GSR to DBSR lies in adapting GSR-terminology to concrete names of tabels 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 subsystem 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'.

$$\{x_1 \mid employee(x_1)\} \setminus \{x_1 \mid x_1.married = \text{'T'}\} = \emptyset \tag{10}$$

```
SELECT *
FROM YES
WHERE NOT EXISTS
( SELECT X1.*
  FROM EMPLOYEE X1
  WHERE NOT (X1.MARRIED = 'T'))
```
(11)

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 Symbolics, 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 natural 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, another field in which $\exists$ and $\forall$ had become traditional. Some modifications had to be made in order to render 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] Barwise, 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, B. 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. & Puhland, R. (1990). 'Een procedure voor redeneren 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] Speelman, 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] Zwarts, F. (1983). 'Determiners: a relational perspective'. Ter Meulen (ed.) *Studies in Modeltheoretic Semantics.* Foris : Dordrecht. 37-62.