Available online at www.sciencedirect.com

ScienceDirect

Journal of Applied Logic 5 (2007) 3–19

JOURNAL OF APPLIED LOGIC

www.elsevier.com/locate/jal

Question answering and database querying: Bridging the gap with generalized quantification

Antonio Badia [1]

Computer Engineering and Computer Science department, Speed School of Engineering, University of Louisville, Louisville KY 40292, USA

Available online 18 January 2006

Abstract

Even though Questions Answering and Database Querying have very different goals and frameworks, collaboration between the two fields could be mutually beneficial. However, the different assumptions in each field makes such collaboration difficult. In this paper, we introduce a query language with generalized quantifiers (QLGQ) and show how it could be used to help bridge the gap between the two fields.

Keywords: Generalized quantifier; Question answering; Query language; Formal linguistics

1. Introduction

Even though both Question Answering (QA) and Database Querying (DQ) aim at giving users access to large volumes of information, they have very different goals and frameworks. QA assumes questions posed in natural language over collections of documents (also in natural language); DQ assumes queries (questions posed in a formalized *query language*) over large collections of structured, well-organized data. However, there are also underlying commonalities to both technologies, derived from the fact that they both provide means for users to extract relevant information from an information repository. Thus, QA and DQ could benefit from sharing experiences, techniques and tools, especially as there are signs that both fields are converging: questions are asked over larger and larger sets of documents, hence QA systems need to care about efficiency (a strong database area), and list questions (similar to queries) are more and more common; databases are expanding their scope to deal with documents, and as queries get more complex and involve more complex data repositories, more flexible query languages are needed. However, there are considerable obstacles to integrating the technologies, due to the different paradigms in which each one is developed.

In this paper, we argue that an in-depth integration of both technologies is possible and desirable. We then introduce QLGQ, a query language with generalized quantifiers that can help bridge the gap. The paper is organized as follows: in the next section, we argue for the advantages of combining QA and DQ technologies, and show our vision of an integrated, encompassing information system. In Section 3, we introduce the concept of Generalized Quantifier (GQ), and in Section 4 we present QLGQ. Then, in Section 5 we show how the concept of GQ introduced in Section 3 can be extended to natural language interrogatives, thus providing a basis to extend QLGQ to deal with questions. In

E-mail address: abadia@louisville.edu (A. Badia).

[1] This material is based upon work supported by the National Science Foundation under a CAREER Grant No. 0347555.

1570-8683/$ – see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.jal.2005.12.007

Section 6 we discuss the role of QLGQ in our proposed framework, overview some desirable features of the extended language and also point out some important challenges to its implementation. Finally, we close in Section 7 with some considerations on this line of work and further research.

2. Motivation

In this section we discuss the reasons to attempt an integration of Question Answering (QA) and Database Querying (DQ) technologies. We argue that there are benefits to be achieved for both fields from such an integration, as well as a possibility to create smarter, more flexible information systems.

2.1. Question answering and database querying

QA systems provide the ability to answer questions posed in natural language by extracting, from a repository of documents, fragments of documents that contain material relevant to the answer. Evaluations in the Text Retrieval Conference (TREC) QA track [41], as well as ARDA's Advanced Question Answering for Intelligence (AQUAINT) program [4], have led to considerable advances in the field. Answering factual questions is performed with better and better accuracy; multiple forms of definition questions are processed correctly, and list questions retrieve sequences of answers with good recall from large text collections. These results and accompanying research have been the topic of several workshops at ACL (2001 and 2003), COLING (2002), LREC (2002), EACL (2003) and the AAAI Spring Symposium series (see [11] for an early such symposium)). Historically, QA has focused on issues of natural language understanding and document summarizing and categorization. Its research is centered on the usual issues of natural language processing: dealing with ambiguity, interpretation issues, and semantic analysis and representation [21,22,41]. It is necessary not only to understand the original question in a natural language, but to also find relevant information in documents. Most QA system proceed by:

(1) analyzing the input question, trying to build a semantic representation of what the question is about and classifying the type of question and the type of answer (factoid, list, definition) that would be expected;
(2) identifying candidate fragments among a repository of documents, usually extracting semantic information from the fragments that matches or can be made to correspond to the semantic representation of the answer; and
(3) ranking the fragments to determine if some of them contain an exact answer or, otherwise, partial or approximate answers to the question.

In order to carry out the second point, a few systems have reasoning abilities, although these are usually limited (see [36] for an example of a system with such abilities). The bulk of the process is in building semantic structures (frames, or predicate-argument structures), and detecting the *topic* and *focus* of the question, in order to guide the search for the answer. However, this processing is computationally intensive. In order to apply it to large collections of documents, many systems rely on Information Retrieval (IR) techniques, at least to focus the efforts in a smaller set of documents. This means generating a set of keywords from the question, which is then supplied to an IR engine, capable of producing an answer extremely fast—normally using some indexing technique like inverted indices [7,10]. Usually, this is combined with natural language techniques to further filter the documents selected, pinpoint relevant fragments in them, and ensure the quality of the answers. Techniques from Information Extraction (IE) can be used for this purpose. In IE, the question is substituted by a set of templates that have to be filled in with information, and therefore there is no need to analyze the question; all emphasis is in extracting information from the text [24,33]. However, there is much commonality between QA and IE.

In contrast, *database querying* has always been characterized by the need to access highly structured data—there is no question about the data's intended meaning, or structure—in large volumes, and with high efficiency. The query is expressed in a *formal language*; for the most successful database model, the relational model, this means using *Structured Query Language* (*SQL*), which can be seen as a syntactic variant of standard First Order Logic (FOL), with some additions and modifications to remedy a lack of expressive power. The interpretation of the query is given by the semantics of the formal language, and poses no problem; all the emphasis is in an extremely efficient procedure for obtaining the answer.

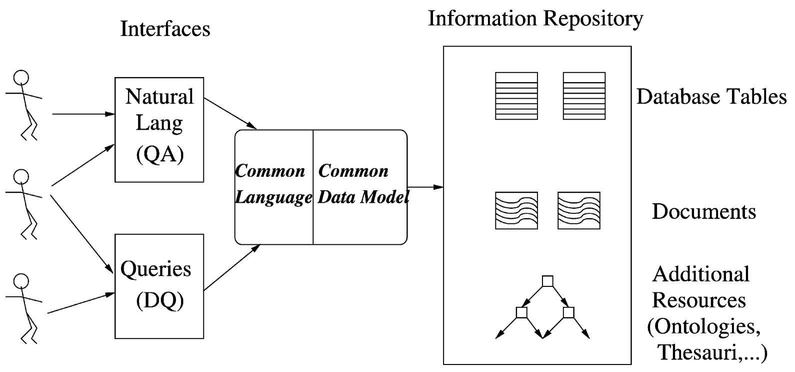

Fig. 1. An all-encompassing Information System.

With such different frameworks, goals and constraints, it is not surprising that QA and DQ have grown completely separated, without almost any common ground. The situation, however, may be changing soon. Databases have lately widened their scope with the incorporation of *semistructured* and *unstructured* data. Semistructured data is assumed not to have a strict type, but to posses irregular, partial organization [35]. Because, in addition, the data may evolve rapidly, the schema for such data is usually large, dynamic, and is not strictly respected. Data models have been proposed that try to adapt to those characteristics. One of the most widely known and used is XML [9]; HTML can also be considered a prime example of semistructured data. The importance and growth of the Web have made this a very important paradigm. As for unstructured data, this usually correspond to *documents*, understood here in a wide sense to encompass anything from books to emails, and anything in between: letters, technical reports,. . . .[2] In order to incorporate data in XML and in documents, database systems need to support query languages more flexible than SQL. While data in XML still allows for a formal treatment (see the proposal for XQuery, a query language for XML [45]), data in documents requires a radically different paradigm. Databases too have depended on IR technology to incorporate documents into their scope. Typically, a table is created that holds document *metadata* (data like author, size, type, provenance, etc.) and, as one of the attributes, the document itself. An inverted index is then built on this attribute; the latest SQL standard incorporates a predicate (`CONTAINS`) that allows elaborate keyword querying alongside traditional data querying. This makes it possible to look for documents using a combination of data and metadata predicates (i.e., "look for documents written by Melton after 2001 that contain the words 'SQL' and 'standard' in the same sentence"). However, no in-depth analysis of document content takes place; IR is solely based on keyword occurrence and disregards syntactic and semantic information.

2.2. *An integrated information system*

We envision a system containing information in many different formats, so database tables live side-by-side with documents and perhaps other types. In this system, information is accessed using a single language—the user may be allowed to enter natural language questions and/or formalized queries, but underlying all interfaces there is a centralized access pathway (see Fig. 1). When answering a user request, all datum (facts) that are relevant contribute to the answer, since all datum (facts) are visible as if they were all part of a seamless repository.[3] It must be pointed out that some existing systems use structured, semistructured and unstructured sources (see, for instance, the description of the QA system from the university of Amsterdam in [2]); however, the level of integration achieved is by no means complete.

If our ultimate goal is to provide universal access to all available information through a common language, then it seems clear that several technologies will have to be combined, as each has strengths to offer. Likewise, each technology may benefit from contributions by others.

[2] Note that the label "unstructured" is obviously a misnomer, and corresponds to the fact that, unlike in the case of structured and semistructured data, where the structure is explicit (in record boundaries or tags), in natural language the structure is implicit, and needs to be unearthed by parsing. Also, the extreme flexibility of natural language makes it unlikely that a given document may fit in a pre-defined structure.

[3] Even if physically we are accessing separate (distributed) data sources, the goal is that all relevant connections among facts are drawn.

QA technology may benefit from some database support, for scalability and performance. Also, as answer sets are becoming standard part of QA tasks (the list task was offered in both the TREC 2001 and the TREC 2002 QA tracks) some database techniques may become relevant in QA. The list task requires systems to assemble an answer from information located in multiple documents. In TREC, a list question asks for different instances of a particular kind of information to be retrieved, so it can be considered as asking for multiple instances of the same factoid. Thus, an appropriate answer is a set of factoids (usually, each factoid is tagged to indicate its origin—document—but we disregard this for now) [41]. Note the similarities with a typical database answer, which is simply a set of facts.

Database technology may also benefit from QA support. Trying to come up with more flexible query systems has a long tradition in databases. The field of Cooperative Query Answering (CQA) is devoted to this. This field is very diverse; see the proceedings [3,12] for a current snapshot at the field. CQA may be composed of one or more of the following: query intent analysis; query rewriting; answer transformation; and answer explanation. The first part covers modeling user goals, beliefs, expectations and intentions. The second part covers transforming the query based on information about the contents of the database. Relaxation and generalization are two primary examples of these techniques; in relaxation, the original query is *weakened* so that it admits information that was not originally asked for but is closely related to the information requested. The third part covers intensional answers and dealing with presuppositions and assumptions. In intensional answers, a description for the answer is provided instead of (or besides) the *extension* of the answer (i.e. the list of database elements that constitutes the answer set). *Presuppositions* are defined as the set of propositions that the user needs to believe are true in order for the question to have several possible answers (or, equivalently, as the set of propositions such that, if they were false, would imply that the question has only one trivial answer—or none at all). When a presupposition is violated that fact should be pointed out.[4] Finally, answer explanation covers creating justifications for answers, that is, explanations of why certain items are (not) part of the answer set, including listing whatever facts and rules were used in constructing the answer set. Note how many of the tenets of CQA fit naturally into QA work. Another area where database work can benefit from QA is in natural language interfaces [14,32,37]. Even though this area is not currently as active as it used to be, there are still clear benefits in some cases to allowing the user to input a query in natural language, instead of a contrived, formal language [25]. Finally, the fact that databases are starting to incorporate documents means that tools for text analysis are of potential use to future database systems—as stated, current database systems rely on IR technology [7], which has severe limitations.

However, there are also significant issues to be solved before integration can be achieved. We trace the difficulties of using QA technology in databases to two main factors, which are inter-related: the different nature of the information search in QA and in databases, and the basis of the technology used in each field. As stated above, database query languages are heavily influenced by first order logic; any researcher in natural language processing is aware of the shortcomings of FOL for natural language analysis. For instance, logical form is quite different from grammatical structure; the sentences "all students are smart" and "some students are smart" have completely different translations, although they differ only in the determiner. As sentence structure is ignored, it is difficult to incorporate information about topic and focus in a logical analysis. Thus, any language that wants to support QA and DQ needs to be rooted in a paradigm that is both more powerful and more linguistically sensitive than FOL. We introduce a new query language, QLGQ (short for *Query Language with Generalized Quantifiers*) which may help, at least in a small measure, to bridge the gap, as it responds to these motivations.

2.3. Queries and questions

In this paper, we reserve the word "query" for database queries, that is, expressions in a query language which are meant to be ran against a database as they *formalize* a user's informational request; we reserve the word "question" for the user's request, as posed by the user in a natural language. The issue that we need to clarify, then, is what is the relationship between queries and questions. Note that queries are usually phrased as commands, but questions can be phrased as commands too: "Give me the name of the tallest mountain in Asia" instead of "What is the name of the tallest mountain in Asia?". Likewise, queries can be phrased as questions: "What are the mountains that are above 8,000 meters high?" can be used instead of "Give a list of all mountains that are above 8,000 meters high". What

[4] It is assumed that, since users are trying to learn some useful information, they expect that there are multiple possible answers for her or his query.

really seems to distinguish queries from questions is that usually the answer to a query is expected to be a set, i.e. a collection—as opposed to a single datum, as in the first example. However, we note that if we consider answers as sets (as we will in our technical approach) than single answers can be naturally incorporated. Another assumption in DQ is that queries will return data as opposed to, say, instructions or definitions. Thus, queries usually are not the answer to "How" or "Why" or "What is" questions, although they could in principle be if the actions, instructions, reasons or definitions are considered as data and stored in a database.[5] In this case, the questions can be paraphrased as "List the different ways in which..." or "List the reasons why..." or "Find the definition of...". Thus, this difference seems to do more with the way information is organized (and the type of things allowed in the repository) than a difference between question and query. Hence, despite some differences, questions and queries are obviously related; they are simply different parts of a spectrum. It seems feasible, then, to develop a single language in which both can be posed.

3. Generalized quantifiers

In the past, GQs have been a subject of interest mostly for logicians. Seminal work was carried out by Mostowski [31]; the original motivation was to be able to talk about properties that are not first order logic-definable. Lindstrom [28] refined the concept and studied logics with GQs in a general setting.[6] The publication of [6] made their importance evident in linguistics, and was followed by a wealth of research in the area [13,16,19,26,27,38,39,42–44]. We argue that the insights of the theory illuminates some aspects of query languages and of question answering, and therefore provides a tool to bridge the gap between the two fields.

There are two views of quantifiers, a relational and a functional view; they are formally equivalent. The relational view is more prevalent in query languages, while the functional view is more frequently used in formal linguistics. In this paper we will use the relational view, but all results and properties discussed hold in the functional view.

A *type* $t = [t_1, \dots, t_n]$ is a finite sequence of natural numbers. Given a set M (the domain), a *Generalized Quantifier* of type $t = [t_1, \dots, t_n]$ on M is an n-ary relation between subsets of $M^{t_1}, \dots, M^{t_n}$ (i.e. a subset of $\mathrm{P}(M^{t_1}) \times \cdots \times \mathrm{P}(M^{t_n})$, where $\mathrm{P}(A)$ denotes the *powerset* of set A). The arity of a quantifier of type $t = [t_1, \dots, t_n]$ is $\max_{i=1}^{n} t_i$ and its argument number is n. A GQ is called *monadic* if it is of arity 1, *polyadic* if it is of arity > 1, and it is called *unary* if its argument number is 1, *binary* if its argument number is 2, etc. Quantifiers of type $[1, 1]$ (binary monadic) are called *standard*, as this type has been found to be the most common type of GQs for natural language analysis [42] and for use in query languages [5,8,23]. We use Q as a variable over GQs, X, Y, Z, as variables over sets and write $Q(X, Y)$ to indicate that sets X and Y belong to the extension of Q, i.e. that they are in the relation denoted by Q.

Below we give a few natural examples (we will write particular quantifiers in boldface). In each case, the GQ is standard, and it corresponds directly to conventional English usage. $Q(X, Y)$ is read as "Q Xs are Ys", as in "all Xs are Ys", for the quantifier **all**.

all	$\{X, Y \subseteq M \mid X \subseteq Y\}$	**at most n**	$\{X, Y \subseteq M \mid \lvert X \cap Y\rvert \leqslant n\}$
some	$\{X, Y \subseteq M \mid X \cap Y \neq \emptyset\}$	**more**	$\{X, Y \subseteq M \mid \lvert X\rvert > \lvert Y\rvert\}$
no	$\{X, Y \subseteq M \mid X \cap Y = \emptyset\}$	**most**	$\{X, Y \subseteq M \mid \lvert X \cap Y\rvert > \lvert X - Y\rvert\}$
all but n	$\{X, Y \subseteq M \mid \lvert X - Y\rvert = n\}$	**half**	$\{X, Y \subseteq M \mid \lvert X\rvert = 2 \times \lvert Y\rvert\}$
at least n	$\{X, Y \subseteq M \mid \lvert X \cap Y\rvert \geqslant n\}$	**as many as**	$\{X, Y \subseteq M \mid \lvert X\rvert = \lvert Y\rvert\}$

Observe that the last two GQs are not first order logic-definable. The concept of GQ is an extremely powerful one; GQs can be added to a logic to extend greatly its expressibility, although there are limits to what can be expressed with GQs [15].[7]

Not every relation between subsets of the domain is considered a GQ. Intuitively, we would like a GQ to behave as a *logical* operator, in the sense that it should not distinguish between elements in the domain, and nothing should

[5] Although they usually are not. As we argue later, this is more an issue of *representation* than one of querying, though.

[6] Generalized quantifiers are sometimes called *Lindstrom quantifiers*.

[7] In the functional view, a generalized quantifier Q of type $[1, 1]$ is simply a function from sets of individuals to functions from sets of individuals to truth value, i.e. of type $[\mathrm{P}(M) \to [\mathrm{P}(M) \to 2]]$, where $\mathrm{P}(M)$ is the powerset of M, and $[X \to Y]$ is the set of all functions from X to Y. Clearly, our previous definition of type $[1, 1]$ quantifiers is equivalent to the present one. In particular, $\mathbf{all}(X)(Y) = 1$ iff $X \subseteq Y$, and so on. The appeal of the functional view is that it extends easily to other types; in particular, questions can be defined as functions from sets of objects to truth values (i.e. functions in $[\mathrm{P}(A) \to 2]$ for a certain A) [19].

be dependent on the domain M chosen. Logicality and other requirements are usually expressed by *axioms*, which in this context are constraints put on the definition of the quantifier. The following one has been deemed fundamental by most authors:

Definition 3.1 *(ISOM).* A quantifier Q follows ISOM if, whenever f is a bijection from M to M', then $Q_M(X, Y)$ iff $Q_{M'}(f[X], f[Y])$.[8]

This constraint appears in the classical papers by Mostowski and Lindstrom [28,31]. Both make it part of the definition of *generalized quantifier*, since it implies, as explained above, that GQs are *logical*.

There have been several other axioms considered. Some of them imply that the behavior of a quantifier is independent of the context, as is the case for the usual logic constants. This seems, therefore, a desirable property. There is an axiom that makes context independence a characteristic:

Definition 3.2 *(EXT).* Quantifier Q follows EXT if for all M, M', all X, Y such that $X, Y \subseteq M \subseteq M'$, $Q_M(X, Y)$ iff $Q_{M'}(X, Y)$.[9]

Why are standard quantifiers the preferred ones in linguistics? Intuitively, such quantifiers can be seen as giving an interpretation to *determiners* in sentence; the two arguments then correspond to a noun phrase (the "subject" of the sentence) and to a verb phrase (the "object" of the sentence). For instance, "all students are registered", gets formalized by the sentence

$$\mathbf{all}\{x \mid \mathit{Student}(x)\}\,\{x \mid \mathit{Registered}(x)\} \tag{1}$$

where the expression $\{x \mid \varphi(x)\}$ denotes the set of elements a that have the property φ (i.e. such that $\varphi(a)$ is true), and we have assumed suitable predicates in the formal model for the translation, purely for convenience—we will return to this topic later.

With this intended usage in mind, other axioms have been proposed, mainly to capture *semantic universals* (i.e. rules that seem to be followed by all natural language determiners). As an example, the following is considered by most authors:

Definition 3.3 *(CONS).* A standard quantifier Q is conservative if for all $X, Y \subseteq M$, $Q_M(X, Y)$ iff $Q_M(X, X \cap Y)$.

This gives the first argument a more important role than the second one; the asymmetry is justified by the difference between the noun phrase and the verb phrase in a sentence: note that the above example is trivially equivalent to "all students are students and registered". In a sense, the subject helps determine the universe of quantification: in natural language, unlike in logic, when we quantify we almost always do it with respect to a certain intended domain or context: thus, if someone says "All are vegetarian" this is likely to be in a context where some set (like the set of all students) has already been introduced—in other words, it is highly unlikely that what is meant is that everything in the universe is vegetarian [44]. However, this is exactly what $\forall x\ \mathit{Vegetarian}(x)$ means. Thus, standard quantifiers naturally provide a context in which to interpret the scope of quantification; we will come back to this observation later.[10] Interestingly, a standard quantifier can also be restricted to a particular context:

Definition 3.4. If Q is a standard quantifier, and $X, Y, Z \subseteq M$, then the *restriction* of Q to Z, in symbols Q^X_M, is defined by $Q^X_M(X \cap Z, Y)$ whenever $Q_M(X, Y)$.

[8] ISOM stands for *isomorphism*, since when M and M' are monadic structures f is an isomorphism [44].

[9] EXT stands for *extensionality*.

[10] Note that this is also possible in first order logic, where formulas like the one above are written as

$$\forall x\ (\mathit{Student}(x) \rightarrow \mathit{Vegetarian}(x))$$

but now we are relying on a (syntactical) convention to make the difference.

If a quantifier is extensional and conservative, its restriction is too. However, a quantifier obtained by restriction does not respect ISOM.

4. A Query Language with Generalized Quantifiers

In this section, we introduce QLGQ (Query Language with Generalized Quantifiers). We will show, together with a formal description of the language, a simple example. We will use a language made up of variables, constants, predicate and quantifier symbols (which roughly correspond to natural language determiners like *every*, *two*, *at least one*, *some*, *most* …) to write sentences about this world. In the following, we assume that a set V of variables, a set C of constants, a set R of relation names and a set Q of quantifier names are given. Expressions of the language and their corresponding set of free variables (denoted $Fvar(\varphi)$, for expression φ) are defined as follows.

Definition 4.1. The syntax of *QLGQ*(Q) is given as follows:

(1) Basic terms
  (a) If $x \in \mathrm{V}$ then x is a basic term. $Fvar(x) = \{x\}$.
  (b) If $c \in \mathrm{C}$ then c is a basic term. $Fvar(c) = \emptyset$.
(2) Set terms
  (a) If φ is a formula, and $\{x_1, \ldots, x_m\} \subseteq Fvar(\varphi)$, then $\{x_1, \ldots, x_m \mid \varphi\}$ is a set term.

$$Fvar(\{x_1, \ldots, x_m \mid \varphi\}) = Fvar(\varphi) - \{x_1, \ldots, x_m\}.$$

  We say that the arity of the set term is m.
(3) Formulas
  (a) If t_1, t_2 are basic terms, and θ is a comparison operator ($=$, $\neq$, $<$, $\leqslant$, …) then $t_1\ \theta\ t_2$ is a basic formula.

$$Fvar(t_1\ \theta\ t_2) = Fvar(t_1) \cup Fvar(t_2).$$

  (b) If $R \in \mathrm{R}$, with arity m, and $t_1, \ldots, t_m$ are basic terms, then $R(t_1, \ldots, t_m)$ is a basic formula.

$$Fvar\big(R(t_1, \ldots, t_m)\big) = Fvar(t_1) \cup \cdots \cup Fvar(t_m).$$

  (c) If $Q \in \mathrm{Q}$ is a quantifier name[11] and S_1, S_2 are set terms of arity m, then $Q(S_1, S_2)$ is a formula.

$$Fvar(Q(S_1, S_2)) = Fvar(S_1) \cup Fvar(S_2).$$

  (d) If φ_1 and φ_2 are formulas then $\varphi_1 \wedge \varphi_2$, and $\varphi_1 \vee \varphi_2$ are formulas.

$$Fvar(\varphi_1 \wedge \varphi_2) = Fvar(\varphi_1 \vee \varphi_2) = Fvar(\varphi_1) \cup Fvar(\varphi_2).$$

A *QLGQ*(Q) *sentence* τ is a formula with no free variables (i.e. $Fvar(\tau) = \emptyset$). A *QLGQ*(Q) *query* S is a set term without free variables (i.e. $Fvar(S) = \emptyset$). Formulas of type 3a and 3b are called *atomic*.

To give the semantics of the language, we introduce the idea of interpretations. An *interpretation* M is a tuple $(D, M_r, M_{\mathrm{C}}, M_{\mathrm{Q}})$ where D is a non-empty set (the domain), M_r is a function that associates, with each relation name $R \in \mathrm{R}$ a relation of the appropriate arity in D, M_{C} is a function that associates, with each constant $\mathbf{c} \in \mathrm{C}$ an element of D, and M_{Q} is a function which associates, with each quantifier name $Q \in \mathrm{Q}$ a binary relation over D. Thus, the semantics of the language associates with each quantifier a (binary) relationship defined over $\mathrm{P}(D)$, the powerset of D.

We are now ready to define the semantics function $[\![S]\!]_M$ for (set) terms S and the *satisfaction relation* $M \models \varphi$ for formulas φ of $QLGQ(\gamma)$.

Definition 4.2. Let M be an interpretation.

[11] For simplicity, we assume the type to be $[m, m]$, for $m \geqslant 1$. As stated above, these are the most common type in queries; however, incorporating other types presents no problem.

- Basic terms
  (1) For $a \in D$, then $[\![x]\!]_{M,a} = a$.
  (2) $[\![c]\!]_M = M_C(c)$.
- Set terms
  (1) For $a_{m+1}, \ldots, a_n \in D$,

$$[\![\{x_1, \ldots, x_m \mid \varphi(x_1, \ldots, x_m, x_{m+1}, \ldots, x_n)\}]\!]_{M,a_{m+1},\ldots,a_n}$$
$$= \{(a_1, \ldots, a_m) \in D^m \mid M, a_1, \ldots, a_m, a_{m+1}, \ldots a_n \models \varphi\}$$

- Formulas
  (1) In order to give proper semantics to comparisons, it is necessary to proceed with a case analysis. Since a basic term may be a constant or a variable, we have the following cases:[12]
    (a) For $a_1, a_2 \in D$, $M, a_1, a_2 \models x_1\theta x_2$ if $[\![x_1]\!]_{M,a_1}\theta[\![x_2]\!]_{M,a_2}$.
    (b) For $a \in D$, $M, a \models x\theta\, \mathbf{c}$ if $[\![x]\!]_{M,a}\theta\ [\![\mathbf{c}]\!]_M$.
    (c) $M \models \mathbf{c}_1\theta\ \mathbf{c}_2$ if $[\![\mathbf{c}_1]\!]_M\theta\ [\![\mathbf{c}_2]\!]_M$.
  (2) Let $n = |Fvar(R(t_1, \ldots, t_m))|$, with m the cardinality of R. Then $M, a_1, \ldots, a_n \models R(t_1, \ldots, t_m)$ if $(b_1, \ldots, b_m) \in M_R(R)$, where $b_i = a_j$ $(1 \leqslant i \leqslant m, 1 \leqslant j \leqslant n)$ if t_i is the variable x_j, or $b_i = [\![t_i]\!]_M$, $1 \leqslant i \leqslant m$, if t_i is a constant.
  (3) Let $Fvar(S_1) \cup Fvar(S_2) = \{x_1, \ldots, x_n\}$, and let $\vec{a} = (a_1, \ldots, a_n) \in D^n$. Define $\vec{a}_{S_i}$ as $(a_i)_{x_i \in Fvar(S_i)}$, for $i = 1, 2$. Then $M, \vec{a} \models Q(S_1, S_2)$ if $([\![S_1]\!]_{M,\vec{a}_{S_1}}, [\![S_2]\!]_{M,\vec{a}_{S_2}}) \in M_Q(Q)$.
  (4) Let $Fvar(\varphi_1) \cup Fvar(\varphi_2) = \{x_1, \ldots, x_n\}$, and let $\vec{a} = (a_1, \ldots, a_n) \in D^n$. Define, as before, $\vec{a}_{\varphi_1} = (a_i)_{x_i \in Fvar(\varphi_1)}$, and similarly for $\vec{a}_{\varphi_2}$. Then $M, \vec{a} \models \varphi_1 \wedge \varphi_2$ if $M, \vec{a}_{\varphi_1} \models \varphi_1$ and $M, \vec{a}_{\varphi_2} \models \varphi_2$. Similarly, $M, \vec{a} \models \varphi_1 \vee \varphi_2$ if $M, \vec{a}_{\varphi_1} \models \varphi_1$ or $M, \vec{a}_{\varphi_2} \models \varphi_2$.

In the context of QLGQ, a query is simply a set term. Since set terms can be nested, complex queries can be expressed by combining different GQs. As an example, we provide a series of queries in FOL, QLGQ and SQL. We assume a database with relations: Student(sid) which holds when the interpretation of sid denotes an individual that is a student; Professor(pid) which holds when the interpretation of pid denotes an individual that is a professor; Takes(sid,cid) that holds between student sid and course cid when sid is enrolled in cid, of giving course cid; and Teaches(pid,sid) which holds between two individuals if the first is a teacher of the second.

(1) Find the professors teaching some students.
  - $\{y \mid \exists\, x[\text{Student}(x) \wedge \text{Teaches}(y, x)]\}$
  - $\{y \mid \textbf{some}\{x \mid \text{Student}(x)\}, \{x \mid \text{Teaches}(y, x)\}\}$
  - `SELECT pid FROM Teaches, Student WHERE Teaches.sid = Student.sid`

(2) Find the professors teaching all students.
  - $\{y \mid \forall\, x[\text{Student}(x) \rightarrow \text{Teaches}(y, x)]\}$
  - $\{y \mid \textbf{all}\{x \mid \text{Student}(x)\}, \{x \mid \text{Teaches}(y, x)\}\}$
  -
```
SELECT pid FROM Teaches T WHERE NOT EXISTS
(SELECT sid FROM Student WHERE sid NOT IN
(SELECT sid FROM Teaches WHERE pid = T.pid))
```

(3) Find the professors teaching all but one of the students.
  - $\{y \mid \exists\, z(\text{Student}(z) \wedge \forall\, x[\text{Student}(x) \rightarrow (\text{Teaches}(y, x) \vee (\neg\text{Teaches}(y, x) \wedge x = z))]\}$
  - $\{y \mid \textbf{all but one}\{x \mid \text{Student}(x)\}, \{x \mid \text{Teaches}(y, x)\}\}$
  -
```
SELECT pid FROM Teaches T WHERE 1 =
(SELECT COUNT(distinct sid) FROM Student WHERE sid NOT IN
(SELECT sid FROM Teaches WHERE pid = T.pid))
```

(4) Find the professors teaching at least two students.
  - $\{z \mid \exists\, x \exists\, y[x \neq y\ \wedge \text{Student}(x) \wedge \text{Student}(y) \wedge \text{Teaches}(z, x) \wedge \text{Teaches}(z, y)]$
  - $\{y \mid \textbf{at least two}\{x \mid \text{Student}(x)\}, \{x \mid \text{Teaches}(y, x)\}\}$

[12] We disregard symmetric situations.

- `SELECT t1.pid FROM Teaches T1 T2 WHERE T1.pid = T2.pid AND T1.sid≠T2.sid`,
  or
  `SELECT pid FROM Teaches GROUP BY pid HAVING count(distinct sid) ⩾2`

(5) Find the professors teaching half of the students.
- Not expressible in FOL.
- $\{y \mid \textbf{half}\{x \mid \mathrm{Student}(x)\}, \{x \mid \mathrm{Teaches}(y, x)\}\}$

-
  ```
  SELECT pid FROM Teaches GROUP BY pid
  HAVING count(distinct sid) * 2 = (SELECT count(distinct sid) FROM STUDENT)
  ```

(6) Find the pairs of teachers teaching the same number of students.
- Not expressible in FOL.
- $\{x, y \mid \textbf{as many as}(\{z \mid \mathrm{Teaches}(x, z)\}\}, \{z \mid \mathrm{Teaches}(y, z)\}\})$
- Only expressible in several steps in SQL:
  ```
  WITH (SELECT pid, count(distinct sid) FROM TEACHES GROUP BY pid)AS
  NUM-STUDENTS (teacher, num) SELECT teacher, T.teacher FROM NUM-STUDENTS T
  WHERE num = T.num
  ```

Note that quantifiers now do not bind variables (as in FOL), but operate on sets. Note also that in examples (1) to (5) the question does not change except for the determiner. This is paralleled in QLGQ, while in FOL and in SQL we need to use diverse formulas to achieve a faithful translation. Note also that the last two formulas are not expressible in FOL, but can written in QLGQ (and in SQL, thanks to the fact that SQL adds, to logically-based operators, the ability to *count* which is lacking in FOL). Thus, QLGQ solves two problems commonly associated with FOL-based approaches to querying (which are the predominant approaches):

- the disassociation between FOL syntax and natural language syntax. After all, queries start as questions/requests in a natural language, which must be translated into the formal query language. It is well known that FOL does not follow natural language rules, and SQL inherits the problem from FOL. On the other hand, QLGQ is strongly inspired by the language QL of [6]; one of the major insights of this work is that it is possible to have formal languages that more closely resemble natural language structure. There are several major benefits for this, some of which will be examined later.
- QLGQ can be very expressive. In fact, QLGQ is not really a language but a *family* of languages; the parameter is the set of quantifiers used. Since the concept of GQ is extremely powerful, one can get different languages with different levels of expressivity by adjusting this parameter.

5. QLGQ in QA

One of the appeals of the concept of GQ is that it has been widely used in linguistic analysis; as a result, it can be used to analyze questions. In particular, the notion of interrogative quantifier has been proposed and analyzed [13,16,19,42]. Here, we build on such analysis to show how QLGQ can be used to formalize questions.

The essential intuition is that interrogatives can be also seen as GQs by considering them as relations between sets: the set provided by the NP and VP in the question, and the answer. Different types of interrogative quantifiers can be considered; quantifiers of type [1, 1] are called *unary interrogative quantifiers* in [19], and quantifiers of type [1, 1, 1] are called *unary interrogative determiners*. Some examples of interrogative quantifiers are

who	$\{X, Y \subseteq M \mid X \cap PERSON = Y\}$
what	$\{X, Y \subseteq M \mid X \cap (M - PERSON) = Y\}$
$\textbf{which}^{\mathbf{Z}}_{\mathbf{n}}$	$\{X, Y \subseteq M \mid X \cap Z = Y \wedge \lvert Y\rvert = n\}$
$\textbf{which ones}^{\mathbf{Z}}$	$\{X, Y \subseteq M \mid X \cap Z = Y \wedge \lvert Y\rvert \geqslant 2\}$

Note that the last two are only defined on a restricted form, that is, it is considered that in proper usage they always make reference to some context. For context dependency, we note that context may be explicit (either in the same sentence, or, in a dialog, in previous sentences) or implicit. In database queries (where there are no dialogs), the context is always explicit (i.e. most queries are phrased with standard quantifiers, as we have seen). In QA, the context may be implicit, and it usually will be in systems that handle dialogs. Note also that **who** is restricted to the PERSON domain, while **what** is restricted to the complement of such domain; this kind of constraint is useful when trying to determine adequate answers.[13]

We illustrate the approach with an example. The question "Who takes CHEM 101?" gets formalized as

$$\mathbf{who}(\{x \mid \mathit{Take}(x, \mathit{CHEM}101)\}, Y) = (\{x \mid \mathit{Take}(x, \mathit{CHEM}101)\} = Y)$$

Thus, an answer is a set Y that is (extensionally) equal to $\{x \mid \mathit{Take}(x, \mathit{CHEM}101)\}$. Formulas in QLGQ can be easily extended to incorporate this type of formula. The most important change is the need to accommodate *set variables* in the language. This is due to the fact that answers are sets (when the answer is a factoid rather than a list, we have a singleton). However, unlike the queries we have previously shown, we cannot determine beforehand how this set will be constructed; in the end, it will be a list of constants. Therefore we cannot express it as $\{x \mid \varphi(x)\}$ where φ is a given formula in the language. Thus, we are simply indicating what makes a complete and correct answer (more on this later). As another example, the question "Which 3 take CHEM101?" is analyzed as follows: first, this question only makes sense in a context, i.e. we pick 3 out of some set determined contextually (this is the role of Z). Assume we are talking about students, i.e. $Z = \{x \mid \mathit{Student}(x)\}$. Then the formalization of the question is

$$\begin{aligned}&\mathbf{Which}_{\mathbf{3}}^{\{\mathbf{x}\mid\mathbf{Student(x)}\}}(\{y \mid \mathit{Takes}(y, \mathit{CHEM}101)\}, Y \wedge |Y| = 3)\\ &\quad = ((\{x \mid \mathit{Student}(x)\} \cap \{y \mid \mathit{Takes}(y, \mathit{CHEM}101)\}) = Y \wedge |Y| = 3)\end{aligned}$$

Thus, the answer to this question is a set with three elements that obeys the restriction imposed by the formula. Finally, we note that sometimes this formalization leaves out useful information: for instance, the question "What classes does John take?" is formalized as

$$\begin{aligned}&\mathbf{What}(\{x \mid \mathit{Takes}(\mathit{John}, x)\} \cap (M - \mathit{PERSON}), Y)\\ &\quad = ((\{x \mid \mathit{Takes}(\mathit{John}, x)\} \cap (M - \mathit{PERSON}) = Y)\end{aligned}$$

However, we know that the restriction to things ($M - \mathit{PERSON}$) is too wide; if there is a $\mathit{Class}(x)$ predicate, we can restrict ourselves to it.

Some examples of interrogative determiners are

what	$\{Z, X, Y \subseteq M \mid Z \cap X = Y \wedge \lvert X\rvert = 1\}$
which$^{\mathbf{W}}$	$\{Z, X, Y \subseteq M \mid (Z \cap W) \cap X = Y \wedge \lvert X\rvert = 1\}$
how many	$\{Z, X, \{n\} \subseteq M \mid \lvert X \cap Y\rvert = n\}$
whose	$\{Z, X, Y \subseteq M \mid Z \cap X = Y \wedge \forall y \in Y \exists p \in \mathit{PERSON}\ \mathit{Owns}(p, y)\}$

Note that **what** and **which** may have singular and plural readings; the correct reading can be inferred from the complete question: "What students attend the class?" is plural, and "What student won the contest?" is singular. The definition given above corresponds to the singular reading of both quantifiers; in the plural reading, the condition $|X| = 1$ is substituted by $|X| \geqslant 2$. Note also that **which** is considered a context dependent quantifier.

As an example, the question "What is the class taught by Paul?" gets formalized as

$$\mathbf{What}(\{x \mid \mathit{Class}(x)\}, \{x \mid \mathit{Teaches}(x, \mathit{Paul})\}, Y) = ((\{x \mid \mathit{Class}(x)\} \cap \{x \mid \mathit{Teaches}(x, \mathit{Paul})\}) = Y)$$

Note that determiners, unlike quantifiers, take an explicit restriction as an argument. However, this extra argument may not be the context, or the whole context, for the question. "Which students take CHEM101?" is interpreted as follows: now we are explicitly mentioning a context (the set of students), but in order to use **Which** there has to be a more restrictive context that applies (otherwise we would say "What students take CHEM101?"). Assume we were talking about Linguistics students (so $W = \{x \mid \mathit{Major}(x, \mathit{Linguistics})\}$). Then we obtain

[13] Even though any QA system incorporates this knowledge about **who**—and **what**—questions, and many times more sophisticated types [21], often this knowledge is implicit in the reasoning or the code of the system. The salient feature of this approach is that it incorporates these (and other) types of constraints in a declarative and organized manner.

$$\mathbf{Which}(\{x \mid \mathit{Student}(x)\}, \{y \mid \mathit{Takes}(y, \mathit{CHEM}101)\}, Y)$$
$$= ((\{x \mid \mathit{Student}(x)\} \cap \{x \mid \mathit{Major}(x, \mathit{Linguistics})\}) \cap \{y \mid \mathit{Takes}(y, \mathit{CHEM}101)\} = Y)$$

The approach is extensible to *modifiers*, under the intuition that modifier interrogatives like **when**, **where**, **how** can be treated as quantifiers. To achieve that, a *sorted* approach is required, i.e. one in which the domain has been divided into sets or *sorts*. We have already introduced the idea of sorts when we used PERSON to denote a specific subset of the domain; extending this idea yields other domains like PLACE and TIME. If one admits higher-order sorts, it is possible to represent concepts like MANNER, CAUSE and REASON (see [19] for the details).[14]

The modifier interrogative quantifiers are defined as follows. Let S be the union of all sets of sorts, and assume we have a number s of sorts. For any relation R of arity n, we extend it to a relation R' of arity $n + s$, where an element of each sort is incorporated: thus, $\langle r_1, \dots, r_n, a_1, \dots, a_s\rangle \in R'$ iff $\langle r_1, \dots, r_n\rangle \in R$ and $a_i \in S_i$, the ith sort. This will allow us to incorporate into the behavior of the modifier the implicit sort argument.

Definition 5.1. For $p \in \mathit{PLACE}$, $t \in \mathit{TIME}$, $m \in \mathit{MANNER}$, $c \in \mathit{CAUSE}$, $r \in \mathit{REASON}$,

$$\begin{array}{ll}
\mathbf{where} & \{R', R, Y \subseteq M \mid \{p \mid \langle r_1, \dots, r_n, a_1, \dots, a_n\rangle \in R' \wedge \langle r_1, \dots, r_n\rangle \in R\} = X\} \\
\mathbf{when} & \{R', R, Y \subseteq M \mid \{t \mid \langle r_1, \dots, r_n, a_1, \dots, a_n\rangle \in R' \wedge \langle r_1, \dots, r_n\rangle \in R\} = X\} \\
\mathbf{how} & \{R', R, Y \subseteq M \mid \{m \mid \langle r_1, \dots, r_n, a_1, \dots, a_n\rangle \in R' \wedge \langle r_1, \dots, r_n\rangle \in R\} = X\} \\
\mathbf{why_c} & \{R', R, Y \subseteq M \mid \{c \mid \langle r_1, \dots, r_n, a_1, \dots, a_n\rangle \in R' \wedge \langle r_1, \dots, r_n\rangle \in R\} = X\} \\
\mathbf{why_r} & \{R', R, Y \subseteq M \mid \{r \mid \langle r_1, \dots, r_n, a_1, \dots, a_n\rangle \in R' \wedge \langle r_1, \dots, r_n\rangle \in R\} = X\}
\end{array}$$

The p is one of the a_i in the definition of **where**, and likewise for the other definitions.

Note that we distinguish two senses of **why**, one asking for cause and one for reason. Again, how to actually set those apart in a natural language question may require some analysis.

As an example, the question "When did John take CHEM101?" can only be answered if the information about who takes what is augmented with temporal information. Thus, from $\mathit{Takes}(x, y)$ we obtain $\mathit{Takes}'(x, y, p, t, m, c, r)$, which is meant to represent the fact that x takes class y in location p at time t in the manner m for cause c and reason r. Then, the analysis of the sentence is

$$\mathbf{when}(\{p \mid \mathit{Takes}'(\mathit{John}, \mathit{CHEM}101, p, t, m, c, r) \wedge \mathit{Takes}(\mathit{John}, \mathit{CHEM}101)\} = X)$$

denotes the answer as the set X of time points when John takes CHEM101. The question "Why did John take CHEM101?" is analyzed as

$$\mathbf{why_c}(\{c \mid \mathit{Takes}'(\mathit{John}, \mathit{CHEM}101, p, t, m, c, r) \wedge \mathit{Takes}(\mathit{John}, \mathit{CHEM}101)\} = X)$$

when it refers to the cause of John taking this class, and as

$$\mathbf{why_r}(\{r \mid \mathit{Takes}'(\mathit{John}, \mathit{CHEM}101, p, t, m, c, r) \wedge \mathit{Takes}(\mathit{John}, \mathit{CHEM}101)\} = X)$$

when it refers to the reason that he is taking CHEM101.

The approach can be extended to modifier interrogative determiners like **in which**, **for what (reason)**, **at what (time)**, **in which (manner)**; the interested reader is referred to [19].

6. The role of QLGQ

Recall that our aim is to provide a unifying framework in which research from QA and DQ can be incorporated. By extending QLGQ syntax and semantics to work with these formalizations of natural language quantifiers, determiners and interrogatives, we have a common language that can be applied to both databases and collections of facts extracted from documents, thus providing a framework in which to integrate questions and queries. Moreover, the framework

[14] Obviously, such domains may have internal structure; we do not enter into these details in this paper, but we shall have something more to say later.

is declarative, high-level and extensible. For instance, the approach can be extended to multiple questions ("Which country was visited by which person?", or "Who bought what?") or to questions that utilize (declarative) quantifiers ("Who bought every item in the store?" or "Who visited at least two countries?"). Thus, QLGQ (with the extension proposed here) is very well suited to be the *common language* in which to represent both questions and queries that our framework calls for (see Fig. 1). Its similarity to the natural language (surface) structure means that the formalization effort is simplified, while the fact that it is a formal language with well defined semantics means that reasoning and search procedures can be carefully constructed and analyzed for correctness. Thus, it is a reasonable effort to formalize questions in QLGQ, while it is also a reasonable effort to represent (SQL) queries in QLGQ.

However, it could be argued that the framework presented is too general to be of real use in QA. Analyzing the task as a relationship between questions and answers misses much of the detailed work needed to produce answers. For instance, we have mapped questions to flat structures made up of predicate expressions; for quantifiers, the question was mapped to a single set expression/formula, while for determiners we divided the question into two parts—approximately corresponding to an NP/VP analysis of the English expression. However, real QA systems would strive to distinguish a *pattern* and a *focus* in a question, besides a set of keywords (possibly connected in a predicate-argument structure) [36]. Thus, a ternary relationship between a question pattern, a question focus and an answer may be a more realistic framework. But the analysis may not stop there; a 5-ary relationship between a question pattern, a question focus, a question type, a question vocabulary (the keywords used) and an answer may be proposed. Different systems may propose and use different components. Many resources (like Named Entity Recognizers, thesauri or ontologies, semantic parsers) and many steps may be involved in generating such components. Our analysis deliberately stopped at the topmost (most coarse) level in order to abstract from these details. This is in line with out goal of providing a general framework, and is consistent with a need to further elaborate the framework. We next show some characteristics of the approach that show its promise, and discuss some of its shortcomings, in order to provide a better perspective.

6.1. Some applications

An ability of the approach is to help determine the type of expected answer, an information that QA systems put to good use. The approach correctly predicts that **who** questions asks for persons, **what** questions for objects (non person entities), and **how many** for cardinal determiners of the type **exactly_n**. Clearly, this is only the beginning of the typology needed by a QA system, which requires quite a bit more precision [21], but recall that we do not analyze the internal structure of the query, so this is just an initial assessment that can guide further processing. Furthermore, the notion of answer can now be formally analyzed. It is assumed that an answer is always a set, and that a question has only one *correct and complete* answer. Thus, if the question "Which students have taken all classes offered by Peter?" refers to Paul and Mary, this means that the only correct and complete answer is the set {Paul, Mary}. This discards the set {Paul} as a good answer, as it is not complete (i.e. it is a *partial answer*), or the set {John} as it is not correct. This correct and total answer is the one that a QA system strives to find, but if it is not found, partial answers may be offered. Note that, for a given answer S, the set of all partial answers of S forms a lattice (the empty answer is excluded), and its elements can be given a partial order. Note also that this order can be used to *rank* partial answers, and to order questions.[15]. For question f, let as denote by $Ans(f)$ the complete and correct answer to f. As in [19], we say that question f subsumes question g (in symbols, $f \leqslant g$) iff $Ans(g) \subseteq Ans(f)$. This relation is a partial order; if question f subsumes question g, then $Ans(g)$ is a partial answer for f. We can exploit this fact together with the following:

Definition 6.1. Let Q be a standard quantifier or interrogative quantifier. Then $Q(X)_1 = \{A \mid Q(A, X)\}$ and $Q(X)_2 = \{B \mid Q(X, B)\}$. Let D be an interrogative determiner. Then $D(X, Y)_2 = \{A \mid D(X, Y, A)\}$.

[15] Note that there is also in QA a notion of exact answer, but it is somewhat different. Since in QA systems an important issue is that the paragraph returned may contain extraneous information, i.e. *more* than the answer, we have a possibility not considered here—such paragraphs would be considered completely incorrect in the GQ framework. Curiously, TREC 2002 judged such answers as inexact and did not allow them to contribute to the score [41]

Definition 6.2. An interrogative quantifier Q is decreasing iff for all $X, Y \in M$, if $X \subseteq Y$ then $Q(X)_2 \leqslant Q(Y)_2$.[16] Similarly, an interrogative determiner D is decreasing iff for all $X, Y, Z \in M$, if $X \subseteq Y$ then $D(X, Z)_2 \leqslant D(Y, Z)_2$.

The interesting fact, proved in [19], is that argument interrogative quantifies are decreasing, as are argument interrogative determiners. Therefore, there is an interplay between different questions and their answers that can be exploited by a system: starting with query g, we can obtain query f such that $f \leqslant g$ by relaxing g. Answering f will yield an answer that contains the answer of the original query. This ability can be used in several ways; for instance, queries may be relaxed to retrieve related, relevant information, in order to give the most informative answer. In QLGQ there are two ways to relax a query: the first one is to relax the GQ used, the second one is to relax the set terms. We give a brief example of each technique (this example is taken from [5]). For relaxing the GQ, assume the query *"Are all students enrolled?"*, expressed as follows in QLGQ:

$$\{() \mid \mathbf{all}(\{x \mid \text{Student}(x)\},\ \{x \mid \text{Enrolled}(x)\})\}$$

The quantifier **all** can be computed as follows: $\mathbf{all}(X, Y)$ is true if $|X \cap Y| = |X|$. Suppose that in the course of computing the answer, we find out that actually $|X \cap Y| = |X|/2$. Then the answer is *no*, but we also know from our computations that half of them are (since $|X \cap Y| = |X|/2$ corresponds to the quantifier *half of Xs are Ys*), so we can answer *No, but half of them are*.

The second technique, relaxing the set terms, has been exploited before in the research literature. The novelty here is that we can determine when it is a good idea to do so, based solely on the structure of the query. Since the answer depends not only on the set terms involved but on the quantifiers used, we have to determine what the effect of relaxing the set term is going to be on the query. This can be achieved by using a property of GQs that we define next:

Definition 6.3. A standard GQ Q is *upward monotonic* in the first argument if whenever $Q(X, Y)$ and $X' \subseteq X$, it is the case that $Q(X', Y)$. A standard GQ Q is *downward monotonic* in the first argument if whenever $Q(X, Y)$ and $X \subseteq X'$, it is the case that $Q(X', Y)$. The same definition applies to the second argument.

A quantifier may be upward monotonic in one argument and downward monotonic on the other. We denote upward monotonicity by using a $\uparrow$ and downward monotonicity by using a $\downarrow$ and putting them in front (behind) of the symbol *MON* to denote the behavior of the first (second, respectively) argument. For instance, **all** is $\downarrow MON \uparrow$, **some** is $\uparrow MON \uparrow$, and **no** is $\downarrow MON \downarrow$.

A simple example will show the utility of the concept: the query *"List the professors who teach all students with a GPA of 3.0 who are friends of Peter"*, can be written in QLGQ as follows:

$$\{pr \mid \mathbf{all}\ (\{st \mid \text{Teaches}(pr, st) \wedge GPA(st, 3.0)\},\ \{st \mid Friend(st, \mathbf{Peter})\})\}$$

If this query returns an empty answer and we want to relax it, and since **all** is $\downarrow MON \uparrow$, we have two choices: to tighten the set term that is the first argument to the quantifier, or to relax the set term that is the second argument to the quantifier. Thus, both tightening the first set (which amounts to restricting the set of students under consideration) or relaxing the second (which amounts to considering more people besides Peter's friends) should help get an answer.

This idea can be used in questions too, and may be useful for QA. Assume that a question asks "What are the mountains higher than 10,000 meters in Nepal?". No answer can be found to this query. Analyzing its formalization

$$\begin{aligned}&\mathbf{What}(\{x \mid Mountain(x) \wedge Height(x, y) \wedge y > 10{,}000\},\\ &\qquad \{x \mid Mountain(x) \wedge Located(x, Nepal)\}, Y)\end{aligned}$$

we can see that the set $\{x \mid Mountain(x) \wedge Height(x, y) \wedge y > 10{,}000\}$ is empty, i.e. there are no mountains on Earth (and therefore, in Nepal) which are higher than 10,000 meters. This could be used to relax the query by changing the constraint $y > 10{,}000$ to some realistic value (perhaps guided by already extracted information). Note that sometimes, no set term is empty, and yet there is no answer. If one asks "What are the names of mountains in Spain that are higher

[16] Note that for an interrogative quantifier, its second argument (denoted by $Q(X)_2$) is unique and is the answer.

than 8,000 meters?" the answer is empty. One can analyze this as:

$$\textbf{What}(\{x \mid \mathit{Mountain}(x) \wedge \mathit{Located}(\mathit{Spain}, x)\},$$
$$\{x \mid \mathit{Mountain}(x) \wedge \mathit{Height}(x, y) \wedge y > 8{,}000\}, Y)$$

In this case, neither set is empty but the intersection is. This means that there are mountains in Spain, and there are mountains that high, but none of them is in Spain. Thus, we can choose to relax either set (or both) until an answer is obtained.

As another illustration of the capabilities of the framework, we show some other properties of GQs that can be used in efficient QA (also taken from [5]). As pointed out in previous examples, GQs take as arguments set terms that represent NPs in natural language sentences. The existence assumptions of natural language usually imply that the denotation of these NPs (and therefore, of the set terms) is not empty. When this is not the case, the extension of a GQ becomes trivial. Assume, for instance, that the quantifier **all** is given the empty set as its first argument. Then $\textbf{all}(\emptyset, Y)$ is true for any set Y (since the semantics of **all** require that $\emptyset \subseteq Y$, which holds trivially). It is expected that GQs will divide the domain of discourse into two non-empty regions: that of sets that belong to the GQs extension, and that of sets that do not. This is called *acting as a sieve* in [6], where it is expected that GQs will act as sieves: "It is often assumed in normal conversation that NPs denote sieves." [6, p. 179], and also: "With some NPs it is harder to contradict the assumption that the denotation is a sieve." [6, p. 181] (Barwise and Cooper call this kind of quantifier *strong* and the rest *weak*). The importance of this concept lies in the fact that when a GQ does not behave as a sieve, this is a good indication that some presupposition may have been violated. The preceding intuition can be formalized as follows:

Definition 6.4. Let Q be a standard quantifier or interrogative quantifier. Q behaves as a sieve in its ith argument ($i = 1, 2$) whenever for all $X \subseteq D$, $Q(X)_i \neq \emptyset$ and $Q(X)_i \neq \mathrm{P}(D)$. Q behaves as a sieve when it behaves as a sieve in any argument.

Given a quantifier Q, it is possible in many cases to find necessary and sufficient conditions for the quantifier to behave (or not) as a sieve. For instance, **all** does not behave as a sieve if and only if its first argument is empty or its second argument is D; **some** does not behave as a sieve if and only if its first argument is empty or its second argument is empty; and **at least 2** does not behave as a sieve when its first argument or its second argument have cardinality $\leqslant 1$. We call these the *sieve conditions* of the quantifier (many of these conditions are pointed out in [6]). The relevance of this concept is that it corresponds to the idea of *degenerate queries* [17], and can be used to formalize the idea of a *violated presupposition*. An example will make this connection clear: to express the query *"find the students who are taking all the courses taught by Peter"*, we use the formula

$$\{\mathit{Sname} \mid \textbf{all}\ (\{\mathit{crs} \mid \mathrm{Lectures}(\textbf{Peter}, \mathit{crs})\},\ \{\mathit{course} \mid \mathrm{Takes}(\mathit{Sname}, \mathit{course})\})\}$$

If the first set term, which denotes the set of courses taught by Peter, is empty, the query returns all the students that are taking at least one course. To see why, remember that $\textbf{all}(X\ Y)$ is true iff $(X \subseteq Y)$. Thus, if $X = \emptyset$, the formula is true for any Y. But note that X being empty means that Peter is not teaching any course, in which case the query makes little sense (most English speakers will agree that, in formulating the query, we are assuming that *there are* courses being taught by Peter).

6.2. Other perspectives

One of the best assets of the generalized quantifier approach is the large body of knowledge already accumulated in linguistics and logic research and that can be applied to the present task. It is well known that using generalized quantification in linguistic analysis has a long and rich tradition, from issues of expressive power [26,27,38,40,43], to dealing with anaphora [18] and plural and mass terms [29]; what is interesting to point out is that there is also discussion of the semantics and pragmatics of questions in this line of research; see [13,19,20,39] as a sample. Generalized quantifiers have already been applied to computational tasks, besides query languages [34,37]. Finally, we point out that application of logic methods to QA is a nascent area of interest [22,30].

6.3. Challenges

While the approach proposed holds considerable promise, there are several substantial issues that need to be dealt with. Here we overview some of them.

Recall that our aim is to provide a unifying framework for information access, where both QA and DQ are supported. For this, it is not enough to provide a single language; it is also necessary to decide how information in tables and documents can be stored and accessed via a common system. Note that we have assumed all along that the right predicates are available to form the correct set terms for each phrase in the query or question. However, a large part of the work in a QA system is precisely to decide the right structures to represent the information in a query or a document (and therefore, the *vocabulary* available); thus, our assumption hides quite a considerable amount of work. But considerable research in QA is devoted already to the task, and can be reused in this framework.

A especially difficult problem to integrate QA and DQ is the different nature of information in databases and documents. In databases, the vocabulary comes given by the *schema* of the database (i.e., in relational databases, the name of the relations in the database and the names of the attributes on those relations). This schema is fixed beforehand, and is determined in part by issues of database design [1]. In documents, on the other hand, the vocabulary is built dynamically based on what text analysis finds in the documents that constitute the collection. Thus, this vocabulary is not fixed in advance and comes determined by what the system is able to find—and, most of the time, natural language sentences yield structures that are considerably different from a database schema. Not only do sentences provide facts that do not adjust to a predefined schema; natural language freely mixes raw facts, metadata about the facts (including links among them), and information that is usually not captured in databases, like explanations, justifications, etc. Proof of this fact is that database query languages rarely—if ever—are used to ask *Why... ?* or *How... ?* questions. This is an extremely important and difficult issue, which is outside of the scope of this paper.

At the language level, there are also important problems that are to be resolved. For instance, even though factoids and lists can be dealt with in the proposed framework, definitions are not addressed. Even within the narrow issue of factoids, there are important differences between the way information is access in QA and in DQ. One important difference is that in a query, the type of answer is fixed by the query expression and trivial to determine; therefore there is no need for further analysis of the query expression. However, an interesting possibility that our approach opens up is the analysis of the query to determine if a *topic* and a *focus* can be associated with it, perhaps with the help of additional structures, like a thesaurus. This would allow us to reformulate a query over a database in order to obtain additional information from a set of documents.

One problematic case is that of superlatives. Assume the question: "Which is the highest mountain in the world?". In a database, we would expect to find a table with information about mountains; for simplicity, assume a table named `Mountains` with attributes `name` and `height`. To obtain an answer to the question, we first need to calculate the largest height, then look for it; this is broken down into two steps in SQL as follows:

```
SELECT name FROM Mountains
WHERE height = (SELECT max(height) FROM Mountain)
```

There is an embedded subquery (after the equal sign) that computes the maximum height using an *aggregate function*;[17] once this is done, the actual value returned is used to compute the overall query. This approach is not feasible in a QA system, for two reasons: first, it is much more efficient to look for a passage that directly answers the question (with a statement like "Mt. Everest is the highest mountain in the world"), instead of finding all mountain height information and doing a comparison. But second, even if we decided to carry out this approach (perhaps because a direct answer was not found), the database uses the *closed world assumption*, which means that it assumes that it has information about all the mountains in the world (at least, all the mountains that matter: if a mountain is not in the database, it does not exist) [1]. This assumption makes the comparison sensible. However, texts are more open-ended; not such assumption can reasonably be made. Thus, QA systems will attempt to find a direct answer to the question by locating a sentence or paragraph in a document that states the needed information. Even though QLGQ as defined does not support aggregate functions, they are not difficult to incorporate into the language. The problem is that one

[17] Aggregate functions in SQL are functions that compute a value out of a set of values; currently, *minimum*, *maximum*, *count*, *sum* and *average* are recognized by the standard—although many others are to become also part of it.

should be able to regard such functions not only as a computation (as in SQL) but also as defining a complex term that may have an exact match in some frame or similar structure extracted from the document repository. Note that being able to consider such functions both ways opens up the possibility of obtaining an answer in two ways: if a QA system was unable to find a straight answer, it could try to gather information about mountains and their heights; compute the highest height among those found and use this information to return an answer—with the caveat that the answer was found indirectly and based on the available information. Note that if a document states "Mount Everest is the highest mountain in the world" we can be certain that we have a correct and complete answer (as far as the document is not mistaken), while if the answer is computed we do not have that certainty—but a computed answer, good to the best of our knowledge, is better than nothing.

Finally, we point out that some of the definitions given may have to be amended to deal with the difficulties (faced by all QA system) of gathering and extracting information from natural language documents. For instance, the completion needed to answer modifier questions may not always be available, so we need to relax it: instead of having a complete extension for each predicate used, we may have to define a set of extensions determined by whatever information is available: thus, given relation $R(r_1, \ldots, r_n)$, $R_p(r_1, \ldots, r_n, p)$ is an extension when only location information is available; $R_t(r_1, \ldots, r_n, t)$ is an extension when only temporal information is available, and so on. This is motivated by the fact that information contained in natural language sentences (or paragraphs) may vary widely, and partial information is more the norm than the exception. Also, spatio-temporal information is many times implicit and hard to determine. Note, though, that such limited extensions are still good enough to answer some questions (but not others), and can be used in our framework.

7. Conclusions and further research

We have argued that the integration of QA and DQ technologies is possible and desirable. As a first attempt towards such an integration, we have introduced QLGQ, a query language with generalized quantifiers. Thanks to the generality and power of the concept of generalized quantifier, we have been able to expand the language to represents questions in addition to queries.

This is a novel and promising area of research. However, it is clear that we have just scratched the surface. In particular, and beyond the challenge of defining a common query language, there are complex issues of information representation, and of search and reasoning capabilities on such a representation. A complete and total solution will require considerable further research, and will call for collaboration among linguists, logicians, and computer scientists. In fact, one of the reasons our approach can be considered as potentially successful is that there is a large body of research (in linguistics and applied logic, mainly) that still has not been applied to QA or DQ. One of our goals is to call attention to the potential of this cross-disciplinary, mutually enriching approach, and encourage others to explore it.

References

[1] S. Abiteboul, R. Hull, V. Vianu, Foundations of Databases, Addison-Wesley, Reading, MA, 1995.

[2] D. Ahn, V. Jijkoun, J. Kamps, G. Mishne, K. Muller, M. de Rijke, S. Schloback, The University of Amsterdam at TREC 2004, in: TREC 2004 Conference Notebook, National Institute of Standards and Technology, 2004.

[3] T. Andreasen, A. Motro, H. Christiansen, H.L. Larsen (Eds.), Flexible Query Answering Systems, 5th International Conference, FQAS, Springer, 2002.

[4] Advanced research development agency (arda), http://www.ic-arda.gov/.

[5] A. Badia, Cooperative query answering with generalized quantifiers, Intelligent and Cooperative Information Systems (1998) 75–97.

[6] J. Barwise, R. Cooper, Generalized quantifiers and natural language, Linguistic and Philosophy 4 (1981) 159–219.

[7] R. Belew, Finding Out About, Cambridge University Press, 2000.

[8] A. Badia, M. Gyssens, D. Van Gucht, Query Languages with Generalized Quantifiers, Kluwer Academic Publisher, 1995, Chapter 11.

[9] T. Bray, J. Paoli, C.M., Sperberg-McQueen (Eds.), Extensible Markup Language (XML) 1.0, W3C Recommendation, second ed., http://www.w3.org/TR/REC-xml-20001006.

[10] R. Baeza-Yates, B. Ribeiro-Neto, Modern Information Retrieval, Addison-Wesley, Reading, MA, 1999.

[11] V. Chaudhri, R. Fikes (Eds.), Question Answering Systems: Papers from the AAAI Fall Symposium, North Falmouth, MA, AAAI Press, 1999.

[12] H. Christiansen, M. Hacid, T. Andreasen, H.L. Larsen (Eds.), Flexible Query Answering Systems, 6th International Conference, FQAS, Springer, 2004.

[13] G. Chierchia, Questions with quantifiers, Natural Language Semantics 1 (1993) 181–234.

[14] W. Chu, F. Meng, Database query formation from natural language using semantic modeling and statistical keyword meaning disambiguation, Technical report, UCLA Computer Science Department, 1999.
[15] A. Dawar, L. Hella, The expressive power of finitely many generalized quantifiers, in: Proceedings of the 9th IEEE Symposium on Logic in Computer Science, 1994.
[16] P. Gardenfors (Ed.), Generalized Quantifiers, Reidel Publishing Company, 1987.
[17] T. Gaasterland, P. Godfrey, J. Minker, An overview of cooperative answering, J. Intelligent Inform. Syst. 1 (1992) 123–157.
[18] J.M. Gawron, S. Peters, Anaphora and Quantification in Situation Semantics, Center for the Study of Language and Information, 1990.
[19] J.J. Gutierrez Reixach, Questions and generalized quantifiers, in: A. Szabolcsi (Ed.), Ways of Scope Taking, Kluwer Academic Publishers, Dordrecht, 1997.
[20] J. Groenendijk, M. Stokhof, Questions, in: J. van Benthem, A. ter Meulen (Eds.), Handbook of Logic and Language, Elsevier, Amsterdam, 1997.
[21] E. Hovy, L. Gerber, U. Hermjakob, M. Junk, C.-Y. Lin, Question answering in webclopedia, in: Proceedings of The Ninth Text REtrieval Conference (TREC 9), NIST Special Publication 500-249, 2000, http://trec.nist.gov/pubs/trec9/t9-proceedings.html.
[22] S.M. Harabagiu, D.I. Moldovan, M. Pasca, M. Surdeanu, R. Mihalcea, R. Girju, V. Rus, F. Lactusu, P. Morarescu, R.C. Bunescu, Answering complex, list and context questions with lcc's question-answering server, in: E.M. Voorhees, D.K. Harman (Eds.), The Tenth Text REtrieval Conference (TREC 2001), National Institute of Standards and Technology, 2001.
[23] P.Y. Hsu, D.S. Parker, Improving SQL with generalized quantifiers, in: Proceedings of the Tenth International Conference on Data Engineering, 1995.
[24] D. Israel, D. Appelt, Introduction to information extraction, in: Tutorial in the IJCAI-99 Conference, 1999.
[25] A. Isard, J. Oberlander, C. Matheson, I. Androutsopoulos, Speaking the users' languages, IEEE Intelligent Syst. 18 (1) (2003) 40–45.
[26] E. Keenan, L. Moss, Generalized quantifiers and the expressive power of natural language, in: [42].
[27] E. Keenan, D. Westerstahl, Generalized quantifiers in linguistics and logic, in: [42].
[28] P. Lindstrom, First order predicate logic with generalized quantifiers, Theoria 32 (1966) 186–195.
[29] G. Link, Algebraic Semantics in Language and Philosophy, CSLI Publications, 1998.
[30] D. Moldovan, S. Harabagiu, R. Girju, P. Morarescu, F. Lacatusu, A. Noischi, A. Badulescu, O. Bolohna, Lcc tools for question answering, in: E.M. Voorhees, D.K. Harman (Eds.), The Tenth Text REtrieval Conference (TREC 2001), National Institute of Standards and Technology, 2001.
[31] A. Mostowski, On a generalization of quantifiers, Fund. Math. 44 (1957) 12–36.
[32] N. Ott, Aspects of the automatic generation of sql statements in a natural language query interface, Inform. Syst. 17 (2) (1992) 147–159.
[33] M.T. Pazienza (Ed.), Information Extraction, Lecture Notes in Artificial Intelligence, vol. 1299, Springer-Verlag, 1997.
[34] J.J. Quantz, How to fit generalized quantifiers into terminological logics, in: 10th European Conference on Artificial Intelligence, 1992.
[35] P. Buneman, S. Abiteboul, D. Suciu, Data on the Web: From Relations to Semistructured Data and XML, Morgan Kaufman, 1999.
[36] S. Harabagiu, S. Narayanan, Question answering based on semantic structures, in: Proceedings of the International Conference on Computational Linguistics (COLING 2004), 2004.
[37] D. Speelman, A natural language interface that uses generalized quantifiers, in: Proceedings of the 5th Twente Workshop on Language Technology, 1996.
[38] E. Thijsse, Counting quantifiers, in: [42].
[39] J. van Benthem, Questions about quantifiers, J. Symbolic Logic 49 (1984) 443–466.
[40] J. van Eijk, Generalized quantifiers and traditional logic, in: [42].
[41] E.M. Voorhees, Overview of the TREC 2003 question answering track, in: The Twelfth Text Retrieval Conference, NIST Special Publication, SP 500-255, 2003.
[42] J. van Benthem, A. ter Meulen (Eds.), Generalized Quantifiers in Natural Language, Foris Publications, 1985.
[43] D. Westerstahl, Branching generalized quantifiers and natural language, in: P. Gardenfors (Ed.), Generalized Quantifiers, Reidel Publishing Company, 1987.
[44] D. Westerstahl, Quantifiers in formal and natural languages, in: D. Gabbay, F. Guenther (Eds.), Handbook of Philosophical Logic, vol. IV, Reidel Publishing Company, 1989, Chapter IV.
[45] Xquery 1.0: An xml query language w3c working draft 11 February 2005, http://www.w3.org/TR/2005/WD-xquery-20050211/.