Antonio Badia

Quantifiers in Action

Generalized Quantification in Query, Logical and Natural Languages

Springer

Quantifiers in Action

Generalized Quantification in Query, Logical and Natural Languages

ADVANCES IN DATABASE SYSTEMS
Volume 37

Series Editors

Ahmed K. Elmagarmid
Purdue University
West Lafayette, IN 47907

Amit P. Sheth
Wright State University
Dayton, Ohio 45435

For other titles published in this series, go to
www.springer.com/series/5573

Quantifiers in Action

Generalized Quantification in Query, Logical and Natural Languages

by

Antonio Badia
Computer Engineering and Computer Science Department
Speed School of Engineering
University of Louisville
Louisville, KY, USA

Springer

Author:
Antonio Badia
Computer Engineering and Computer Science Department
JB. Speed School of Engineering
University of Louisville
Louisville, KY 40292 USA
abadia@louisville.edu

Library of Congress Control Number: 2009921815

ISBN: 978-0-387-09563-9 e-ISBN: 978-0-387-09564-6

Printed on acid-free paper

9 8 7 6 5 4 3 2 1

springer.com

To Mindy, Emma and Gabi: I'm finally done!

Preface

The goal of this book is to help conquer what this author sees as dangerous divides in the study of databases in general, and of query languages in particular: the divide between theory and practice, and the divide between purely formal approaches and those with a wider concept of communication.

This is, of course, a very tall order, and there is no claim here that the mission has been accomplished. We live in an era of specialists (people who know a lot about a little), and the explosion of available material in any given field makes it quite difficult even to keep current with one's area, never mind exploring a different one. Presenting all the material this book touches upon in depth is nearly impossible. Rather, the book introduces the most basic results of different fields. It is then likely that an expert in any given area (logic, databases, or formal linguistics) will find that the material discussed here is very elementary. This is part of the price to pay when trying to reach the widest audience. The book is intended just to present the beginning of an exploration, and its aim is to encourage others to pursue research further. Necessarily, the coverage is limited and subjective, but hopefully it will become an *entry point* for the interested reader, who will be encouraged to delve further into the subject using some of the bibliography provided.

It may seem strange to pick a a technical, specialized topic like Generalized Quantification for this kind of endeavor. However, Generalized Quantifiers are a perfect example of a practical theory. An idea that started in the seminal papers by Mostowski ([76]) and Lindstrom ([70]) as a theoretical investigation into a new concept found unsuspected applications first in linguistics and then in query languages. Generalized Quantifiers motivate insights into optimization of set-based queries, Question Answering, and Cooperative Query Answering. This book tries to gather all those insights together. As such, the book tries to walk a line between theory and practice. There is out there a large collection of work on GQs; however, they all tend towards the purely theoretical side ([4]) or towards a particular application, usually linguistics ([94]). The book strives the cover the middle ground.

Depending on their point of view, readers may feel that we have gone too far to the theory side or to the application side. Others still may feel that we have not gone far enough. There are no doubt many other possible applications of the concept, and much more is known right now that we can hope to include in our first chapters. However, by bringing together all this material we hope to make people aware of what others are doing. If we happen to encourage collaborations and help establish links where before there were none, then we consider that we have done our job.

Before getting into the subject, it is time to give credit where credit is due. This book represents a summary of research carried out by the author over the past years. In particular, section 4.3 of chapter 4 is based on [10]; sections 6.2, 6.3 and 6.4 of chapter 6 are based on [13]; chapter 7 is based on [9] and chapter 8 is an update of work reported in [11]. Along the way, there were many people who helped carry out the research, and this is the opportunity to thank them, as they made this book possible. This list should certainly include the members of my doctoral dissertation committee (Dirk van Gucht, Ed Robertson, David Leake and Larry Moss), as it all got started there. The collaboration with Stijn Vansummeren was a pleasure; I learned much from it. My graduate students contributed considerably to this effort; special mention should go to Bin Cao, Brandon Debes and Dazhuo Li for their work in different parts of the project. Bin helped develop the interpreter introduced in chapter 5; Dazhuo worked an approach to linear prefixes, shown in chapter 6; and Brandon implemented a system that transformed our project from a paper-and-pencil approach to executable software[1]. This material is based upon work supported by the National Science Foundation under Grant No. IIS-0347555, and therefore special mention goes to the National Science Foundation, which generously supported the research described here under a CAREER Award, and to my two program managers, Le Gruenwald and Maria Zemankova, who exhibited a tremendous amount of patience with me -I am very indebted to both of them.

Finally, I cannot honestly say that Mindy, Emma or Gabi's contributions have been technical. In fact, I cannot say that repeatedly asking "Tickle me, papa!" is a contribution at all. But I know that without them, I would have never spent the time and energy I did on this, or any other project -it would just not seem worthwhile.

Louisville, KY

Antonio Badia
December 2008

[1] The result of his efforts is accessible at `http://qlgq.spd.louisville.edu/index.php`.

Contents

1

Introduction

This monograph is written with two purposes: to help bridge the gap between theory and practice in databases, and to help bridge the gap between research in query language and research in other areas (outside databases, and some outside Computer Science) that are clearly related to querying. The motivation behind the book, then, is a belief that the gaps exist, and that this is a bad situation.

Databases are a strange area. On the one hand, they are clearly an *applied* field. Databases (and related services) are a multi-billion, world-wide software industry. Some advances (for instance, in query optimization) make it to market in relatively short time. Changes in the industry (for instance, the move to *data warehouses*) provoke comparable changes in the field, creating whole subareas of research. On the other hand, databases are a *theoretical* field. Since the definition of the relational model by Codd, its main concepts have been tied to logic ideas, and logician's methods have been used to study the model abstractly. This study has blossomed in entirely new subareas, like *descriptive complexity* ([56]) and *finite model theory* ([27, 69]). Thus, databases is, at the same time, a clearly applied and a clearly theoretical field of study.

This would be a good thing if theory and practice walked hand in hand. However, this does not always happen. To quote a familiar dictum, *"In theory, theory and practice are the same. In practice, they are not."*[1]. While occasionally theory occupies itself with questions of practical interest, and practice does generate questions of theoretical interest, the mainstream body of theory and practice remain separated. This is not due to the will of the researchers -many theoreticians find delight in seeing theoretical developments put to good use in practice; and they are also eager to apply their skills to problems motivated by new areas of practice. It is also the case, in my opinion, that many practitioners consider the establishment of a solid, formal foundation for a practical problem a positive development. In fact, most database

[1] This saying, as well as the next one, have been attributed to a variety of people, so they will go uncredited here.

A.Badia, *Quantifiers in Action: Generalized Quantification in Query. Logical and Natural Languages*, Advances in Database Systems 37, DOI: 10.1007/978-0-387-09564-6_1,

people (certainly this author) adhere to another familiar dictum, *"There is nothing as practical as a good theory"*. But theory and practice have different goals, demand different approaches and produce different results. Most of the time, the disconnect is present. It is for this reason that it is a pleasure to work on a subject that gives an opportunity to bridge this gap.

As for our second purpose, it is not difficult to realize, after a bit of thought, that querying is, first and foremost, a *linguistic* activity: queries are posed in a language. However, there are two levels at which this activity takes place. First, a cognitive agent has to come up with a question. Such agents are, usually, human beings, and the questions they come up with are expressed in natural language. Only when the necessity to pose the question to a computer arises there is a corresponding need to translate the question into a formal language. Let us agree to reserve the term *question* to refer to expressions in a natural language, and *query* to refer to expressions in a formal language. The database researcher (and the theoretical computer scientist) deals with queries *only*. The linguist (and the philosopher) deals with questions *only*. But surely a link is missing. In real life, queries start as questions. Making sure a query correctly reflects the intended question is important. Learning which kinds of questions (as opposed to queries) are most useful (or common, or important, or whatever other measure one wants to use) is also important. This importance has been reflected in the fast development, in the last few years, of the field of Question Answering, in which users pose questions to the computer directly in natural language, and hence it is the software that must come up with an appropriate translation from question to query ([99]). Information Retrieval, also prevalent nowadays as a research field due to the explosion of the World Wide Web, straddles a middle line between queries and questions -at least, that is how this author regards keyword search ([14, 18, 73]). It is perhaps time, then, to take the relationship between questions and queries seriously.

Once our purpose has been explained, the next question is: why use Generalized Quantification as the tool to develop a research plan for this purpose?

Generalized Quantification started as a narrow concept applied to a particular problem and then, like many good ideas, took off with a life of its own. The great logician Mostowski considered the limitations of expressive power of first order logic (henceforth, FOL) and focused on one that seemed somewhat troublesome to him: the inability of FOL to distinguish between finite and infinite sets ([76]). The concept of infinity being so important for the foundations of mathematics, he considered that FOL had to be able to capture the notion (there were contrary opinions: for other people, the notions of finite and infinite sets were not purely logical notions and hence did not belong in FOL). So he considered how to add to FOL the ability of expressing infinity. The challenge was doing so in a manner that was *minimal*, in the sense that minimal changes were introduced to both the syntax and the semantics of the language. He introduced a new symbol, $\mathbf{Q}_{\aleph}$, and used in formulas as follows:

$$Q_{\aleph}\ x\ \phi(x)$$

where ϕ was an arbitrary FOL formula with x the only free variable. The interpretation of this formula is, as usual, given in terms of whether an arbitrary model $\mathcal{M}$ would satisfy it[2]. Given the set of values a from the universe of $\mathcal{M}$ such that $\mathcal{M}$ satisfies $\phi(a)$ (that is, when a is substituted by x, we get a sentence true in $\mathcal{M}$), Mostowski stipulated that if such a set is infinite numerable, then the sentence above is true in $\mathcal{M}$.

The subtle but important idea behind that notion is that, by introducing a new logical operator that captures exactly the notion we intended to capture ("infiniteness"), we are adding to the language *only* the necessary machinery to deal with the new concept. By studying the resulting language, one could learn what the concept exactly entails. For instance, if some property can also be expressed in the extended language that could not be expressed in plain FOL, that such property must be somehow related to infinity. A trivial example is that of finiteness, which can be expressed in the language simply by using negation on the formula above.

A few years later, as part of its meta-logic studies on exactly what constitutes a logic (and on what makes FOL such an important language among many possible) Lindstrom revisited the issue of Generalized Quantification and realized that it could be put in a more general setting ([70]). Just like he was doing for other logic concepts, Lindstrom formally defined the idea of Generalized Quantification, of which Mostowski's quantifier was only one example[3].

The idea then became part of the logician's tool and was studied by researchers; however, it didn't make any impact outside the logical community. But this all changed with time. Nowadays, the concept of Generalized Quantification is heavily used in at least two communities outside its original setting: theoretical Computer Science and Formal Linguistics.

Theoretical Computer Science's main theme is the study of computational complexity, which classifies problems according to the resources (time and space) that any algorithm needs to compute a solution to the problem. Since there can be many algorithms for a given problem, careful study of the problem is needed to establish such results -which should hold for all possible algorithms that compute solutions to the problem. In [30], Fagin introduced the idea of using some logic language to describe the problem, and studying the properties of such logic in order to determine properties of the problem. This was a significant new viewpoint, since logics are *declarative* languages: they simply allows us to specify *what* the problem is, not *how* a solution looks like (which is what algorithms do). Thus, the study linked issues of computational complexity with issues of expressive power in logic, and a large number of results followed, starting a subfield that is usually called *declara-*

[2] For readers not acquainted with logic, chapter 2 introduces all needed basic notions.

[3] Lindstrom's definition is given in chapter 3.

tive complexity. The explosion of results was facilitated, in part, by the body of knowledge that logicians had developed over time.

The idea had a direct impact on query languages. It is well known that relational query languages are simply versions (more or less disguised) of FOL. The limitations in expressive power of FOL have became an important issue and, since Generalized Quantifiers offer a way to overcome some of those limitations, logics with Generalized Quantifiers have been studied in depth in this setting. As its name indicates, Generalized Quantifiers are an extension of the idea of quantifier in FOL. There, the notion of quantifier is *fixed*: first-order quantifiers are a closed class, capturing only the most basic concept. Generalized Quantifiers consider the class open: you can add new quantifiers simply by defining them, subject to some very basic rules that make sure any quantifier behaves in a logical way (what this means will be defined in Chapter 3). It turns out that this is a natural way to capture properties that are not expressible in FOL; this makes Generalized Quantifiers a great tool to investigate expressive power, and to add it in a controlled manner to query languages.

Formal linguists attempt to give a description of the meaning of natural language statements in a formalized language. FOL is one of the popular choices for a target language. The inadequacies of such a choice have been known for a long time. In a work that touched most later research, Richard Montague argued that natural language could be successfully modeled with tools that were better suited for the task ([26]). However, he used heavy equipment: a logic derived from lambda calculus that contained many second-order constructs. Surely there had to be something between FOL and the sophisticated tools of Montague that was adequate for the analysis of at least some simple fragments of natural language. In 1981, in a seminal paper, Barwise and Cooper introduced the idea of using FOL extended with Generalized Quantifiers for the formal analysis of linguistic expressions ([16]). Using a suitably simplified notion of Generalized Quantification, Barwise and Cooper established a basic framework that has lasted until today and has originated its own significant body of research (see, for instance, [94]).

We have, then, a theoretical concept that can have practical application, and that has spanned several areas of research. The book's thesis is that Generalized Quantification is a good idea that can be profitable applied to practical matters of questioning and querying. Going back to the old saying that *there is nothing as practical as a good theory*, Generalized Quantification fits the bill perfectly. At the same time, applying the idea may not be easy. Using the concepts introduced here in a practical scenario involves being able to implement them efficiently (always a strong consideration in databases), and showing its usefulness for practical purposes. Thus, unlike past research that dealt with theoretical issues of complexity and expressive power, the aim here is to work with a suitably limited version of the concept but to show that such version can still be useful and can be implemented efficiently. This endeavor is helped by the inter-disciplinary approach mentioned above, as

the inspiration of what version of the concept to use in queries comes from research on linguistics. And this is the other main goal here: to show how concepts and techniques developed in one field help illuminate challenges in others. The book starts with a formal view of queries and gradually widens the scope to incorporate more and more issues about questions. To develop this program, Chapter 2 gives some basic background in logic focused on the traditional notion of quantification. The idea is to make the rest of the book accessible to readers without a background in logic, making the book as self-contained as possible. Readers with an adequate background may skip this chapter, which only introduces elementary concepts. Chapter 3 introduces the notion of Generalized Quantifier (in fact, two definitions are given) and some basic properties. We have chosen to introduce further properties later, right when they are needed, so that their motivation is clear. Chapter 4 introduces the Query Language with Generalized Quantifiers ($QLGQ$), which provides us with a general setting in which to study issues of quantification without being distracted by this or that particular syntax. The language is a simple variation of Datalog-like notations, and hopefully most readers will have no trouble becoming familiar with it. Chapter 5 introduces what we consider the most common case of Generalized Quantification for practical use, and concentrates on giving an efficient implementation for this case. Chapter 6 studies the use and implementation of (generalized) quantifier prefixes. We will see that, even in the simplest of settings, some questions come up that may not be evident on first thought. Then chapter 7 introduces, following a more linguistic motivation, the use of Generalized Quantification to deal with *pragmatic* issues in querying, and chapter 8 finishes the change to a linguistic setting by showing the use of Generalized Quantifiers in Question Answering. Finally, chapter 9 sketches how Generalized Quantifiers can be used in other settings, for instance distributed computing, so prominent nowadays with the raise of peer-to-peer systems and cloud computing. While the research in this chapter is just in its beginning stages, it will hopefully be enough to show the adaptability and promise of the approach. We close the book with a short discussion in chapter 10.

2

Basic Concepts

2.1 From Propositional to First Order Logic

In this chapter we introduce some basic logic concepts, focusing on those ideas related to quantification. This chapter is for readers with no background in logic, and it only includes some core concepts that facilitate further reading.

Propositional (also called *zero-order*) logic is the basic building block of First-Order Logic (recall that we abbreviate it as FOL). In fact, we have the same basic logic connectives:

- if ψ, φ are formulas, then $\psi \wedge \varphi$ is a formula (conjunction);
- if ψ, φ are formulas, then $\psi \vee \varphi$ is a formula (disjunction);
- if ψ is a formula, then $\neg\psi$ is a formula (negation);

It is customary to also allow formulas of the form $\psi \rightarrow \varphi$ and $\psi \leftrightarrow \varphi$, but they are considered just syntactic sugar, since $\psi \rightarrow \varphi$ is equivalent to $\neg\psi \vee \varphi$, and $\psi \leftrightarrow \varphi$ is equivalent to $(\psi \rightarrow \varphi) \wedge (\varphi \rightarrow \psi)$. Because of the recursive nature of the definition, arbitrarily complex formulas can be formed.

What truly distinguishes FOL from propositional logic is *quantification*. Propositional logic allows only *propositions*, statements that are either true or false: "yesterday it rained", "3 is greater than 5". In FOL, statements are made about individuals, by stating relationships that may hold among them. As an example, the statement "3 is greater than 5" would be expressed in FOL with a formula like $>(\mathbf{3}, \mathbf{5})$, where the symbol '$>$' denotes a binary relation. In propositional logic, such an statement would be considered atomic (without parts), and denoted by a single symbol, like P. In FOL, statements are no longer atomic. Since they are about individuals, though, the next thing we need is some way to tell *which* individuals we are talking about. FOL uses *terms* to denote individuals. Terms are of two basic types: *constants* or names that denote a certain individual unequivocally, or *variables*, which stand for an individual without denoting a particular one.

FOL languages then, will be made up of atomic formulas constituted by relations and terms. Such formulas can then be combined with the Boolean

A.Badia, *Quantifiers in Action: Generalized Quantification in Query. Logical and Natural Languages*, Advances in Database Systems 37, DOI: 10.1007/978-0-387-09564-6_2,