A Logical Language for Data and Knowledge Bases

Shamim Naqvi
Shalom Tsur

A Logical Language for Data and Knowledge Bases

Shamim Naqvi
Shalom Tsur

COMPUTER SCIENCE PRESS
New York

Library of Congress Cataloging-in-Publication Data

Naqvi, Shamim.
A logical language for data and knowledge bases.

Bibliography: p.
Includes index.
1. LDL (Computer program language) 2. Data base management 3 Artificial intelligence. I. Tsur, Shalom. II. Title.
QA76.73.L15N36 1989 005.75'6 89-682
ISBN 0-7167-8200-6

Printed in the United States of America

Computer Science Press
1803 Research Boulevard
Rockville, MD 20850

An imprint of W. H. Freeman and Company
41 Madison Avenue, New York, NY 10010
20 Beaumont Street, Oxford OX1 2NQ, England

1 2 3 4 5 6 7 8 9 0 RRD 7 6 5 4 3 2 1 0 8 9

PRINCIPLES OF COMPUTER SCIENCE SERIES

Series Editors

Alfred V. Aho, *Bell Telephone Laboratories, Murray Hill, New Jersey*
Jeffrey D. Ullman, *Stanford University, Stanford, California*

Theo Pavlidis
Algorithms for Graphics and Image Processing

C. K. Wong
Algorithmic Studies in Mass Storage Systems

David Maier
Theory of Relational Databases

Jeffrey D. Ullman
Computational Aspects of VLSI

Narain Gehani
Advanced C: Food for the Educated Palate

Narain Gehani
C: An Advanced Introduction

Narain Gehani
C for Personal Computers: IBM PC, AT&T PC 6300, and Compatibles

Leonard R. Marino
Principles of Computer Design

Christos Papadimitriou
The Theory of Database Concurrency Control

Michael Andrews
Computer Organization

Steven Tanimoto
Elements of Artificial Intelligence Using LISP

Egon Börger, Editor
Trends in Theoretical Computer Science

Martti Mäntylä
An Introduction to Solid Modeling

Jeffrey D. Ullman
Principles of Database and Knowledge Base Systems, Volumes I and II

Jim Moore
UNIX: The Minimal Manual

Shamim Naqvi and Shalom Tsur
A Logical Language for Data and Knowledge Bases

OTHER BOOKS OF INTEREST

Arto Salomaa
Jewels of Formal Language Theory

Jeffrey D. Ullman
Principles of Database Systems

Kurt J. Schmucker
Fuzzy Sets, Natural Language Computations, and Risk Analysis

Stuart C. Shapiro
LISP: An Interactive Approach

Preface

§

This is a comprehensive account of the syntax and semantics of a new programming language called $\mathcal{LDL}$. In it are provided descriptions, both at the informal and formal level, of every $\mathcal{LDL}$ construct. The informal descriptions do not presuppose a high technical inclination but nevertheless are precise. For the avid and mathematically inclined, advanced sections are provided detailing model-theoretic and constructive semantics of the language. These advanced sections themselves and by the nature of the language also serve as introductions to current research topics in the theory of Data and Knowledge Bases. The authors have striven for simplicity and brevity, but not at the expense of accuracy.

§

We have used both the terms *data* and *knowledge* in our title; their meaning in our context warrants some discussion. The distinctions between these concepts stem from the different computing cultures out of which they grew. Let us first discuss the notions of data and a database. A database is a system designed to query, update, and share a large volume of data. The basic premise is that the volume of data is too large to be contained in its entirety in the memory of a computer and hence, special access methods are required to retrieve the data efficiently from its secondary memory. Another premise is that the data are a permanent resource, to be shared by different application programs over a long time span. This premise, which is also known as the *data independence* requirement, implies that the query language must be *declarative* in nature so that subsets of the stored data can be specified in different application programs without any knowledge of the actual layout or the mode of access to these data. The task of translating and optimizing the declarative specification into an efficient access procedure is the responsibility of the database system. The technology to

implement this task, particularly within the context of relational database systems, has been constantly improved over the past decade and has reached a high degree of sophistication.

The database philosophy can thus be summarized as "simple but efficient." Performance in data retrieval is at a premium, and the complexity of the stored data is sacrificed in order to facilitate this objective. Typically, the data format is a simple record, of a predetermined length and with a fixed number of fields. This format is consistent with the relational data model. Another consequence of this approach is that database systems offer but a limited functionality: the query language enables only the specification of a limited class of queries, and certain information implicit in the data cannot be specified at all. Furthermore, data can only be retrieved or updated. Other operations on data, e.g., arithmetic, cannot be performed within the database system. To compensate for this limited functionality, database systems are used in conjunction with general-purpose programming languages: the database subsystem retrieves the data, and further manipulation is performed by the host programming language.

The concepts of knowledge and knowledge system have grown from the artificial intelligence culture. The problem of *knowledge representation* is a long-standing field of inquiry and efforts in this area have concentrated on the design of (in-memory) data structures to encode "real-world" situations. The best-known result in this respect is the use of frames as a knowledge representation device. Another AI effort concentrates on the use of logics and inference methods, often in conjunction with knowledge representation structures, to infer new knowledge from existing knowledge. So far, the commercial results of this research manifest themselves in the form of expert systems. The expert system paradigm assumes the existence of a human expert, whose knowledge in some specific domain, e.g., a medical specialty, can be articulated and mapped into a knowledge representation structure. The structure can be queried and can, in conjunction with a logic, be used to infer new knowledge. This is known as the "inference engine" of the expert system. The logical conclusions of the deductive process are dispensed as "advice." The term *knowledge base* is thus used in AI to denote the knowledge representation structure over which the logical inferences are performed.

Let us now briefly contrast these two cultures. The emphasis in AI is on generality. Complex structures are used to represent classification schemes, types, and other features pertinent to a real world situation. This contrasts with the simple record structures used in the database world for factual knowledge representation. At the same time, the total volume of knowledge

used in an expert system is small and can be accommodated entirely in main memory. Hence, the use of an in-memory data structure to represent knowledge. The volume concern in databases that dictates the use of simple structures, is nonexistent in the AI context, as is the second database concern of data independence. Expert systems in their present commercial form are autonomous programs that do not share data. Hence, there is no need for a declarative form of data and knowledge access. Indeed, the mode of access in knowledge bases is by navigation through the structure, and the results may be dependent on the particular route taken. By contrast, a database user does not know the details of the data representation and the results of his or her query are independent of these details.

$\mathcal{LDL}$ as a system covers the middle ground between databases and knowledge bases. On the one hand, it is committed to the database requirements of volume and data independence; the $\mathcal{LDL}$ language is purely declarative and the $\mathcal{LDL}$ system comes with a highly sophisticated optimizer, designed to improve the performance of the complex queries that are generated by the system. On the other hand, it provides some of the structural richness and inference capabilities found in knowledge bases. In particular, users can represent their knowledge within complex objects, e.g., lists, as well as set types. The inference capability of the language enables the user to define knowledge by *intension*, i.e., by the specification of rules that derive relations from other relations. The terminology for systems of this type has not crystallized yet, and they are variously referred to as *logic databases, deductive databases,* or, to compound the confusion with expert systems, *knowledge bases*. From now on we will use these terms synonymously.

From the database perspective logic databases extend the generality of the traditional systems in the sense that they embody some of the functionality that otherwise had to be provided by the general-purpose application language system. Another important difference between the traditional host- and embedded-language organization and logic databases is the elimination of the *impedance mismatch* in the former case. The impedance mismatch is the result of the combination of a procedural application language with a declarative database query language. Severe inefficiencies may result in the execution because of the inability to optimize a query over these incompatible subsystems. In $\mathcal{LDL}$ the application language and the query sublanguage are the same. Hence, no impedance mismatch exists, and queries can be efficiently executed.

From the AI perspective the $\mathcal{LDL}$ system can serve as the substrate upon which some of the richer features, such as classification and type inheritance schemes, can be built. This would endow expert systems with a database

capability they lack at present. We note that the present trend of using expert system shells in conjunction with conventional database systems suffers from the same impedance mismatch as the conventional database application language systems. As in the conventional case, it is impossible to optimize queries over the navigation-oriented, procedural knowledge structure of the shell and the declarative database subsystem.

The evolution of the thinking that eventually led to $\mathcal{LDL}$ had its origins in the areas of relational databases and logic programming. All of these technologies have strengths, yet on its own, each is insufficient to meet the objectives that we set out to achieve. A point of departure could be to combine a logic programming language, e.g., PROLOG, with a relational database system. Unfortunately, a combination of this type suffers from two major shortcomings. The first shortcoming is that the computational mechanisms of PROLOG, *viz.*, SLD-resolution and backtracking, attempt to compute a single proof at a time. This results again, in an impedance mismatch, since the (default) mode of computing *all* of the answers to a query can now only be achieved by repeated proofs, one proof at a time, and the query optimization mechanisms of the underlying database, which are set oriented, cannot be utilized. The second problem in this approach is that during the execution of a proof the computation may not terminate; hence, the completeness of the result cannot be guaranteed. The approach adopted in $\mathcal{LDL}$ is to retain Horn-clause logic and extend it but, in contrast to PROLOG, to use a different interpretation of the logic that is more attuned to applications with large volumes of data. The interpretation used is *model based*. The answer to a query is computed in a bottom-up application of the rule set to the stored and the partially derived data until no further results can be produced, i.e., a *fixpoint* has been reached. Since this method could generate a great amount of irrelevant and redundant data, if implemented naively, the $\mathcal{LDL}$ system uses special compilation techniques developed for this purpose that not only ensure an efficient execution but also guarantee the completeness of the query-answering process. Fixpoint semantics, together with the compilation methods, are the key ingredients of this new technology that enables the realization of integrated logic database systems and removes the shortcomings in some of the other systems mentioned.

From the foregoing it appears that a complete discussion of $\mathcal{LDL}$ would entail both the language semantics and the compilation aspects of the system. However, the methodology adopted for this book is *language centered*. It covers the area by successive descriptions of containing subsets of the language, each one having a more powerful semantics than its predecessor. This way, we proceed from a simple but pure Horn-clause subset of the language

toward the full language and its extensions, such as the use of negation, sets, and procedural extensions, while retaining its declarative character. In this process we make occasional references to some of the compilation techniques used but do not provide a complete coverage of them. General compilation techniques for logic databases have been published in technical conferences and journals. The subject is currently an active area of research and is likely to remain so for the foreseeable future. We expect to see a constant improvement of these methods as time progresses. While this is an indication of a healthy state of affairs from the perspective of the active database researcher in the field, it is still too early to summarize these techniques so as to present them in a coherent fashion to a wider audience. Hence, we have restricted our presentation of these techniques to those cases where understanding them is essential to understanding the accompanying material.

On the other hand, the theory underlying the language, the language specification itself, and its semantics have matured to the extent that a technology transfer to a wider audience is feasible and desirable. The technology can then be evaluated and appreciated by this community, concurrently with the ongoing research in compilation techniques. In our view this wider audience consists of database professionals who are interested in evaluating the technology for their own purposes and advanced computer science students who have a background in database theory and/or logic programming. We hope that the exposition of the ideas to this larger audience will result in their application to real problems and their critical evaluation. We have endeavored to accompany the book with a set of meaningful examples that expose the power of $\mathcal{LDL}$ and describe, by means of an extended example, a problem representative of the class of "data-dredging" problems. This class of inductive data analysis problems appears to be particularly suitable for treatment by the technology presented in this book.

The level of presentation of the material is split between an introductory level required for the writing of simple programs and an advanced level required for the deeper understanding of the issues involved. The introductory level can be read independently of the advanced level.

The software that implements the ideas presented in the book is the property of the shareholder companies of MCC. We hope that we will be able to make it available to academic and other nonprofit institutions in the near future. We expect that the experimentation with the system itself will further enhance the understanding of the issues involved and ultimately will lead to a wide adoption of this new technology.

§

We are grateful to Kevin Greene, Pat Lincoln, Jeff Naughton, Jeffrey D. Ullman, Moshe Vardi, and members of the Languages Group at MCC for reading earlier drafts of this manuscript and pointing out several errors, omissions, and inconsistencies. Fagin's early result on the inexpressibility of the transitive closure in first-order languages, and other related results on this topic, were brought to our attention by Moshe Vardi. We thank Danette Chimenti and Ruben Gamboa for providing application programs for Appendices C and D. These reviews, comments, and additions have substantially improved our presentation. Remaining errors and omissions are, of course, the responsibility of the authors.

Austin, Texas —*SAN*
January 16, 1989 —*ST*

Acknowledgment

The $\mathcal{LDL}$ system stems from a proposal by Shalom Tsur and Carlo Zaniolo. As it stands now, it is the accumulation of the work—research on semantics of database languages, inventing compilation and implementation strategies, and good old-fashioned programming—of many people. This book is a description of the result of their efforts. We know that it does not capture the quality and excellence of all their ideas. Our hope is that, in some small way, it can convey the sense of excitement and pleasure that we have felt in working with this wonderful group of people—members of the Languages Group at MCC, visitors, consultants, and friends. Although we give a list of names below, we remain convinced that some have been left out and hope that those not mentioned will, as in the past, understand and alert us to this failing .

Hassan Aït-Kaci
François Bancilhon
Catriel Beeri
Danette Chimenti
Ruben Gamboa
Tony O'Hare
Paris Kanellakis
Charlie Kellog
Ravi Krishnamurthy
Arshad Matin
Tom McLellan
Kayliang Ong
Raghu Ramakrishnan
Domenico Sacca
Oded Shmueli
Leona Slepetis
Peter Song
Millie Villarreal
Carolyn West
Denise White
Carlo Zaniolo

Contents

List of Programs