Computer-Assisted Query Formulation

Alvin Cheung
University of Washington
akcheung@cs.washington.edu

Armando Solar-Lezama
MIT CSAIL
asolar@csail.mit.edu
Foundations and Trends® in Programming Languages
Volume 3, Issue 1, 2016

Editorial Board

Editor-in-Chief

Mooly Sagiv
Tel Aviv University
Israel

Editors

Martín Abadi
Microsoft Research & UC Santa Cruz
Anindya Banerjee
IMDEA
Patrick Cousot
ENS Paris & NYU
Oege De Moor
University of Oxford
Matthias Felleisen
Northeastern University
John Field
Google
Cormac Flanagan
UC Santa Cruz
Philippa Gardner
Imperial College
Andrew Gordon
Microsoft Research & University of Edinburgh
Dan Grossman
University of Washington

Robert Harper
CMU
Tim Harris
Oracle
Fritz Henglein
University of Copenhagen
Rupak Majumdar
MPI-SWS & UCLA
Kenneth McMillan
Microsoft Research
J. Eliot B. Moss
UMass, Amherst
Andrew C. Myers
Cornell University
Hanne Riis Nielson
TU Denmark
Peter O'Hearn
UCL
Benjamin C. Pierce
UPenn
Andrew Pitts
University of Cambridge

Ganesan Ramalingam
Microsoft Research
Mooly Sagiv
Tel Aviv University
Davide Sangiorgi
University of Bologna
David Schmidt
Kansas State University
Peter Sewell
University of Cambridge
Scott Stoller
Stony Brook University
Peter Stuckey
University of Melbourne
Jan Vitek
Purdue University
Philip Wadler
University of Edinburgh
David Walker
Princeton University
Stephanie Weirich
UPenn

Full text available at: http://dx.doi.org/10.1561/2500000018
Editorial Scope

Topics

Foundations and Trends® in Programming Languages publishes survey and tutorial articles in the following topics:

- Abstract interpretation
- Compilation and interpretation techniques
- Domain specific languages
- Formal semantics, including lambda calculi, process calculi, and process algebra
- Language paradigms
- Mechanical proof checking
- Memory management
- Partial evaluation
- Program logic
- Programming language implementation
- Programming language security
- Programming languages for concurrency
- Programming languages for parallelism
- Program synthesis
- Program transformations and optimizations
- Program verification
- Runtime techniques for programming languages
- Software model checking
- Static and dynamic program analysis
- Type theory and type systems

Information for Librarians

Foundations and Trends® in Programming Languages, 2016, Volume 3, 4 issues. ISSN paper version 2325-1107. ISSN online version 2325-1131. Also available as a combined paper and online subscription.
Computer-Assisted Query Formulation

Alvin Cheung  
University of Washington  
akcheung@cs.washington.edu

Armando Solar-Lezama  
MIT CSAIL  
asolar@csail.mit.edu
## Contents

1 Introduction 2

2 Query Processing 5
   2.1 Relational DBMS and Query Languages ............... 5
   2.2 DBMS as a library .................................. 6
   2.3 The ORM approach .................................. 7
   2.4 Query Execution .................................. 8

3 Program Synthesis 13
   3.1 The Problem ....................................... 13
   3.2 Deductive Synthesis ............................... 14
   3.3 Inductive Synthesis ............................... 15

4 Using Verified Lifting to Rewrite Code into SQL 19
   4.1 Interacting with the DBMS ......................... 20
   4.2 QBS Overview ................................... 22
   4.3 Theory of Finite Ordered Relations .............. 29
   4.4 Synthesis of Invariants and Postconditions ...... 39
   4.5 Formal Validation and Source Transformation ..... 44
   4.6 Preprocessing of Input Programs ................. 46
   4.7 Experiments ...................................... 49
   4.8 Summary .......................................... 59
5 Assisting Users Specify Database Queries
  5.1 Intended Users .............................................. 61
  5.2 Usage Model .................................................. 64
  5.3 Search Algorithms .......................................... 68
  5.4 Query Refinement ........................................... 79

6 Conclusion and Future Work
  6.1 Beyond Input-Output Examples .............................. 83
  6.2 Extending System Capabilities .............................. 84
  6.3 Refinement Techniques ..................................... 84
  6.4 Combining Different Inference Algorithms ................ 85

References ......................................................... 87
Abstract

Database management systems (DBMS) typically provide an application programming interface for users to issue queries using query languages such as SQL. Many such languages were originally designed for business data processing applications. While these applications are still relevant, two other classes of applications have become important users of data management systems: (a) web applications that issue queries programmatically to the DBMS, and (b) data analytics involving complex queries that allow data scientists to better understand their datasets. Unfortunately, existing query languages provided by database management systems are often far from ideal for these application domains.

In this tutorial, we describe a set of technologies that assist users in specifying database queries for different application domains. The goal of such systems is to bridge the gap between current query interfaces provided by database management systems and the needs of different usage scenarios that are not well served by existing query languages. We discuss the different interaction modes that such systems provide and the algorithms used to infer user queries. In particular, we focus on a new class of systems built using program synthesis techniques, and furthermore discuss opportunities in combining synthesis and other methods used in prior systems to infer user queries.

From financial transactions to online shopping, we interact with database management systems (DBMSs) on a daily basis. Since the initial development of relational database systems, various query languages such as SQL have been developed for users to interact with the DBMS. Many of these languages proved very effective for what was originally their primary application: business data processing (e.g., generating transaction reports at a financial institution). Unfortunately, many important applications of DBMSs that have emerged in recent decades have proven to be a less than ideal fit for the interaction models supported by traditional DBMSs.

One particularly important extension to the business data processing application space corresponds to applications with complex business logic, such as social network websites, online shopping applications, etc. Unfortunately, traditional query interfaces often make developing such applications difficult. First, the general-purpose languages in which these applications are usually written (e.g., Java or Python) are quite different from the query languages supported by the DBMS. This forces developers to learn a new language—and often a new programming paradigm altogether. For example, an application developer who
is used to thinking about computation over objects stored in the program heap will need to recast her computation in terms of structured relations stored on disks when interacting with a DBMS. Moreover, in addition to being concerned about efficient memory layout for retrieving in-memory objects, she will also need to understand the costs associated with bringing objects into memory from the disk. This “impedance mismatch” \cite{copeland1984} problem has plagued application developers for decades. Today, this mismatch is often addressed by application frameworks known as Object Relational Mapping (ORM) Frameworks that eliminate the need to think in terms of two distinct programming models. Unfortunately, the use of ORMs often imposes significant performance costs \cite{subramanian}. 

There are many reasons for the performance cost of ORMs, but one that is especially significant is that they encourage a programming style where computation that could have been implemented with a single query and a single round trip to the database is instead implemented with several simpler queries connected together with imperative code that manipulates their results. This is problematic because in addition to increasing the number of round trips and the amount of data that needs to be transferred between the application and the DBMS, doing so also increases the cost of the computation, since the DBMS is in much better position to optimize queries compared to a general-purpose code compiler trying to optimize a block of imperative code that happens to implement a relational operation.

As an example, while a relational join between relations \( R \) and \( S \) can be implemented using a nested loop, with each loop processing tuples from the two respective relations fetched from the DBMS, it is much more efficient to implement the join as a single SQL query, as the DBMS can choose the best way to implement the join during query optimization.

In this tutorial we focus on a new approach based on verified lifting \cite{cheung2015} to reduce the performance cost of these application frameworks, allowing programmers to enjoy the benefits of the reduced impedance mismatch. The first step in this technique is to identify places in the application code where the programmer is using...
imperative code to implement functionality that could be implemented as part of a query. The second and most important step is to use program synthesis technology to derive a query that is provably equivalent to the imperative code. Once that is done, the third step involves generating a new version of the code that uses the query in place of the original code.

The technology behind this work was originally published earlier [Cheung et al., 2013]. In this paper, we expand on the content of that original paper in order to make the technology more accessible to researchers without a strong background in program synthesis or verification, as well as to researchers who may not be as familiar with database concepts. In Section 2, we provide a quick primer on query execution and query processing, focusing on key concepts that will help the reader understand the reasons for the performance problems introduced by ORMs. Section 3 provides a comprehensive primer on program synthesis technology, focusing in particular on the techniques that are leveraged by QBS, and putting them in context of other synthesis technologies. Section 4 describes the details of the QBS approach, and finally Section 5 describes the state of the art in terms of applications of synthesis to interact with DBMS systems and promising directions for future work.


References


Alvin Cheung, Samuel Madden, and Armando Solar-Lezama. Sloth: being lazy is a virtue (when issuing database queries). In ACM SIGMOD International Conference on Management of Data, pages 931–942, 2014.


References


References


References


