CodeQuest: Scalable Source Code Queries with Datalog

Elnar Hajiyev, Mathieu Verbaere, and Oege de Moor

Programming Tools Group,
Oxford University Computing Laboratory,
Wolfson Building, Parks Road, Oxford OX1 3QD, UK
{Elnar.Hajiyev, Mathieu.Verbaere, Oege.de.Moor}@comlab.ox.ac.uk,
http://progtools.comlab.ox.ac.uk/projects/codequest/

Abstract. Source code querying tools allow programmers to explore relations between different parts of the code base. This paper describes such a tool, named *CodeQuest*. It combines two previous proposals, namely the use of logic programming and database systems.

As the query language we use *safe Datalog*, which was originally introduced in the theory of databases. That provides just the right level of expressiveness; in particular recursion is indispensable for source code queries. Safe Datalog is like Prolog, but all queries are guaranteed to terminate, and there is no need for extra-logical annotations.

Our implementation of Datalog maps queries to a relational database system. We are thus able to capitalise on the query optimiser provided by such a system. For recursive queries we implement our own optimisations in the translation from Datalog to SQL. Experiments confirm that this strategy yields an efficient, scalable code querying system.

1 Introduction

Understanding source code is vital to many tasks in software engineering. Source code querying tools are designed to help such understanding, by allowing programmers to explore relations that exist between different parts of the code base. Modern development environments therefore provide querying facilities, but these are usually fixed: one cannot define new relationships that are particular to the project in hand.

It can be very useful, however, to define such project-specific queries, for instance to enforce coding style rules (*e.g.* naming conventions), to check correct usage of an API (*e.g.* no call to a GUI method from an enterprise bean), or to ensure framework-specific rules (*e.g.* in a compiler, every non-abstract AST class must override the *visitChildren* method). Apart from such checking tasks, we might want new ways of navigating beyond the fixed set of relations provided in a development environment. When cleaning up a piece of legacy software, it is for example useful to know what methods are never called (directly or indirectly) from the *main* method. A good querying tool allows the programmer to define all these tasks via simple, concise queries. Note that none of these examples is easily

D. Thomas (Ed.): ECOOP 2006, LNCS 4067, pp. 2–27, 2006.

implemented with today's dominant code querying tool, namely *grep*. Built-in querying and navigating facilities of Eclipse, widely used by the IDE users, are limited to a fixed number of certain queries.

The research community has long recognised the need for flexible code queries, and many solutions have been proposed. We shall discuss this previous work in detail in Sect. 6. For now it suffices to say that two crucial ideas have emerged from that earlier research: a logical query language like Prolog to formulate queries, and a relational database to store information about the program.

All these earlier attempts, however, fall short on at least one of three counts: the system is not scalable to industrial-size projects, or the query language is not sufficiently expressive, or the queries require complex annotations to guarantee efficiency. Scalability is typically not achieved because no query optimisation is used, and/or all data is held in main memory. Expressiveness requires recursive queries, to inspect the graph structures (the type hierarchy and the call graph, for example) that are typically found in code queries. Yet the use of recursion in SQL and XQuery is cumbersome, and in Prolog recursion over graphs often leads to non-termination. In Prolog that problem may be solved via tabling plus mode annotations, but such annotations require considerable expertise to get right.

1.1 Contributions

This paper solves all these deficiencies, and it presents a code querying tool that is scalable, expressive and purely declarative. We achieve this through a synthesis of the best ideas of the previous work on code querying. To wit, our contributions are these:

- The identification of *safe Datalog* (a query language originating in database theory) as a suitable source code query language, in the sweet spot between expressiveness and efficient implementation.
- The implementation of Datalog via an optimising compiler to SQL, which is in turn implemented on a relational database system. Our compiler performs a specialised version of the well-known 'magic sets' transformation, which we call 'closure fusion'.
- A method of incrementally updating the database relations when a compilation unit is changed.
- A comprehensive set of experiments, with two different commercial database systems (Microsoft SQL Server and IBM DB2) as a backend for our query compiler, to show the scalability of our approach. We also demonstrate that for this application, a special implementation of recursion outperforms the built-in recursion provided by these database systems.
- Detailed comparison with other state-of-the-art code querying tools, in particular JQuery (an Eclipse plugin tailored for code queries) [2, 24, 34] and XSB (a general optimising compiler for tabled Prolog [3,39]), demonstrating that on small projects our approach is competitive, and on large projects superior.