Datalog for Static Analysis

Ben Greenman, HOPL 2017

Abstract

Datalog is a domain-specific language for building and querying (mutually recursive) relations. Many static analyses are about computing relations between parts of a program. The use of Datalog to implement scalable and efficient static analyses is a recurring theme.

What You Always Wanted to Know About Datalog (And Never Dared to Ask)

```
@article{cgt-ieee-1989,
  author = {Stefano Ceri and Georg Gottlob and Letizia Tanca},
  title = {What You Always Wanted to Know About {D}atalog
           (And Never Dared to Ask)},
  journal = {{IEEE} Transactions on Knowledge and Data Engineering},
  volume = {1},
  number = {1},
  year = {1989},
  pages = {146 -- 166},
  url = {http://ieeexplore.ieee.org/iel3/69/1663/00043410.pdf}
}
```

This article is a self-contained tutorial on Datalog. The authors present Datalog's syntax and semantics, motivate its use as a query language, survey implementation and optimization strategies, and outline proposals for extending Datalog. In particular, the syntax is first-order Horn clauses (no function symbols, no negation, variables in a rule conclusion must appear positively in the rule's premises). The semantics of a Datalog program is in terms of its *Herbrand Base*: the set of all consequences to the Horn clauses after replacing free variables with atoms from a given database. Typically, users connect a Datalog program to an external database and use the program to formulate a query. Datalog engines can efficently serve such queries by leveraging constants in the query to guide the search for solutions

Significance: this paper is the canonical reference on Datalog. It is an accessible tutorial that covers both the logical foundations of Datalog (in the abstract) and techniques for efficiently implementing Datalog (in the concrete).

On Edge Addition Rewrite Systems And their Relevance to Program Analysis

```
@inproceedings{a-tagt-1994,
  author = {Uwe Assmann},
  title = {On Edge Addition Rewrite Systems And their Relevance
           to Program Analysis},
  booktitle = {TAGT},
  pages = {321 -- 335},
  year = {1994}
}
```

The paper describes three distinct problems in static analysis, then restates the problems in terms of graphs and rewriting rules. Each rule describes how to add edges to the graph representation of a program based on patterns among its subgraphs. Iterating these edge-addition rules to a least fixed point yields a solution to the corresponding analysis problem. Furthermore, the paper identifies properties of the rewrite rules that correspond to efficient techniques for computing a least fixed point. These efficient techniques for solving the graph problems coincide with efficient, manually derived techniques for solving the traditional analysis problems.

Significance: this paper is the first work to actively apply Datalog to static analysis problems. The paper demonstrates that Datalog can concisely express such programs and can lead to efficient algorithms for computing their solution.

Demand Interprocedural Program Analysis Using Logic Databases

```
@inproceedings{r-ald-1994,
  author = {Thomas Reps},
  title = {Demand Interprocedural Program Analysis Using Logic
           Databases},
  booktitle = {CC},
  year = {1994}
}
```

The goal of a program analysis is to answer queries about the structure of a program. An *exhaustive* program analysis first summarizes a given program and then uses the summary to answer queries. A *demand* analysis first receives a query and then summarizes only the relevant parts of the program.

This paper demonstrates that the magic sets transformation[1] for logic programs is effective for converting an *exhaustive* (interprocedural) program analysis into a *demand* analysis. In particular, it reports on an implementation of program slicing built using the CORAL deductive database. The implementation describes exhaustive program slicing in terms of first-order relations (Datalog). CORAL applies magic sets to this description and derives a demand analysis.

Significance: this is the first work to seriously explore the connection between Datalog and program analysis. Instead of just noting the symmetries between Datalog and program analysis, Reps validates the ideas using an (existing) implementation of Datalog. The paper also demonstrates why it can be useful to separate an analysis specification from its implementation. An *exhaustive* implementation of program slicing cannot easily be re-used to build a *demand* implementation; however, the paper demonstrates that magic sets can transform an *exhaustive* specification to a *demand* implementation.

[1] Banchilon, F., Maier, D., Sagiv, Y., and Ullman, J. *Magic sets and other strange ways to implement logic programs*, in PODS 1986.

Context Sensitive Program Analysis as Database Queries

```
@inproceedings{lwlmacu-pods-2005,
  author = {Monica S. Lam and John Whaley and V. Benjamin Livshits
            and Michael C. Martin and Dzintars Avots
            and Michael Carbin and Christopher Unkel},
  title = {Context Sensitive Program Analysis as Database Queries},
  booktitle = {PODS},
  pages = {1 -- 12},
  year = {2005}
}
```

This paper suggests that binary decision diagrams (BDDs) and deductive databases are useful tools for implementing context-sensitive points-to analyses. These tools can efficiently represent and query a large number of call contexts (the authors remark that it is "not unusual" for Java programs to have more than 10^{14} contexts). The authors validate their technique by building a deductive database and a framework for specifying points-to analyses. They report success using the framework to implement five taint analyses for C and Java. Across nine Java web applications, the analyses reported 29 true errors and 12 false positives.

Significance: the authors bddbddb framework is the first system for specifying new static analyses via Datalog.[2] The paper demonstrates that the system is able to express a range of useful analyses, works for multiple languages, and can identify vulnerabilities in practical programs.

[2]Programmers write source code patterns and the framework compiles these patterns to Datalog.

Strictly Declarative Specification of Sophisticated Points-To Analyses

```
@inproceedings{bs-oopsla-2009,
  author = {Martin Bravenboer and Yannis Smaragdakis},
  title = {Strictly Declarative Specification of Sophisticated
           Points-To Analyses},
  booktitle = {OOPSLA},
  pages = {243 -- 262},
  year = {2009}
```

This paper reports on the DOOP framework for points-to analysis of Java programs. DOOP encodes call graph construction and context-sensitive points-to analysis as Datalog rules tuned to a commercial Datalog engine written by LogicBlox. The authors compare DOOP to an existing tool for points-to analysis and find that Doop builds smaller call graphs in less time. Furthermore, DOOP can produce results for levels of context sensitivity at which the other tool times out (2-call + 2 contexts on the heap, and 2 object + 1 context on the heap).

Significance: DOOP is the fastest, most precise, and most scalable points-to analysis among its peers. The paper demonstrates the benefits of using Datalog almost exclusively and re-using a commercial Datalog engine. Using Datalog lessened the gap between the Java language specification and the analysis implementation (encoding the specification in C would have been more difficult). Using the LogicBlox engine reduced the problem of compiling Datalog efficiently to writing rules that the engine could optimize.