15-745 Project Proposal: Implementing Dataflow Optimizations for Pegasus in Datalog

Dan Licata
Project Web Site: `http://www.cs.cmu.edu/~drl/course/compilers/`

1 Project Description

Recent work by Lam et al. [1, 2, 3] has investigated the use of the logic-programming language Datalog to code program analyses, exploiting a fast implementation of Datalog using binary decision diagrams. Analyses written in Datalog are more compact and easier to reason about than hand-coded versions, while the BDD-based implementation makes their time performance competitive with hand-coded versions.

However, this approach has not previously been implemented for the Pegasus intermediate representation. It would be a useful contribution to represent Pegasus graphs, and analyses and transformations on them, in Datalog. We therefore propose to connect Whaley and Lam's `bddbddb` implementation of Datalog to the CASH compiler, and use it to implement some dataflow analyses.

In Datalog, programs are specified as sets of inference rules defining predicates, as in Prolog. Prolog interprets inference rules by backward chaining: for example, to show that $P(x)$ holds, find rules of the form $P(x) \leftarrow Q(x)$ and recursively show that $Q(x)$ holds; this terminates when we get to a rule with no antecedent. Datalog, on the other hand, interprets rules by forward chaining and saturation: starting with some base set of facts, apply any rules for which the current set of facts proves all the antecedents; iterate until saturation (no new facts can be discovered). Since dataflow analyses are typically described via iteration to a fixed point, it is very natural to implement them in Datalog.

Binary decision diagrams provide a compact and canonical representation of Boolean functions. A predicate $P(x)$ over a finite domain D can be represented by a Boolean function $D \rightarrow$ Bool, and therefore in turn by a BDD. The operations of forward chaining logic programming can be viewed as relational projections and joins, which have natural and efficient implementations in terms of BDDs.

In this project, I hope to:

- Give a Datalog representation of Pegasus graphs and implement one or more dataflow analyses.
- Interface `bddbddb` to CASH: output a Datalog representation of a Pegasus graph for analysis, and then read the results of an analysis back into CASH.
- Demonstrate that it is feasible to program CASH optimizations in Datalog by running the code on some small benchmarks, comparable to those that we have used to assess the homework assignments.

However, since this is now a one-person project, I may have to make some simplifications from the pre-proposed work. Specifically, I may consider a restricted subset of Pegasus (Mahim suggested that I consider the 15-or-so most-common nodes), and I may implement less-ambitious optimizations than we previously proposed. My first goal is to implement an analysis on the scale of the first homework assignment (conditional constant propagation).

2 Logistics

Plan of Attack

My plan for accomplishing this project consists of the following steps:

1. Devise a Datalog representation of Pegasus graphs suitable for implementing a simple analysis (say, CCP). I am not yet sure whether it will be better to tailor the Datalog representation to each analysis or to give one generic representation.

2. Implement the sample example analysis on this representation.

3. Test the analysis on some hand-coded graphs.

4. Implement the glue code necessary to output the Datalog representation from CASH and read the results of the analysis back in.

5. Run the analysis on some larger programs and see that the running-time is reasonable.

6. (Time permitting) Consider other analyses, which may require an alternate or generalized Datalog representation of Pegasus graphs.

I am hesitant to suggest particular dates for these steps because the first three are the novel (and therefore riskiest) parts of the project. Based on my own logic programming experience and Whaley's work [2], I have a general idea of how these three steps will go, but I cannot make definite claims without having gone through the details.

Milestone

I hope to complete the first two or three steps of the plan by the milestone date. Failing that, I will at least be able to report on why the Datalog representation was less straightforward than I had thought.

Resources Needed

The only software necessary for this project is the open-source `bddbddb` implementation of Datalog and the CASH compiler. I will not need any hardware other than my office machine.

Literature Search

The work by Lam and her students [1, 2, 3] is the most directly related work, as it involves implementing compiler analysis in Datalog using `bddbddb`. In the final project report, I will survey other research using logic programming to implement compiler analyses and optimizations.

Getting Started

So far, I have begun to consider a Datalog representation of Pegasus graphs.

References

[1] M. A. Lam, J. Whaley, V. B. Livshits, M. C. Martin, D. Avots, M. Carbin, and C. Unkel. Context-sensitive program analysis as database queries. In *24th SIGMOD-SIGACT-SIGARTS Symposium on Principles of Database Systems*, June 2005.

[2] J. Whaley, D. Avots, M. Carbin, and M. S. Lam. Using datalog with binary decision diagrams for program analysis. In *Programming Languages and Systems: Third Asian Symposium*,, Tsukuba, Japan, November 2005.

[3] J. Whaley and M. S. Lam. Cloning-based context-sensitive pointer alias analysis using binary decision diagrams. In *ACM SIGPLAN Conference on Programming Language Design and Implementation*, June 2004.