Efficient and Extensible Datalog

Aaron Bembenek[1], Stephen Chong[1], and Marco Gaboardi[2]

[1] Harvard University [2] University of Dundee

Privacy Tools for Sharing Research Data

A National Science Foundation Secure and Trustworthy Cyberspace Project

with additional support from the Sloan Foundation and Google, Inc.

Motivation and Initial Objectives

We were primarily motivated by the lack of a good, open-source, Java-based implementation of the logic programming language Datalog. We wanted to have such an implementation as the backend for other projects within Privacy Tools.

Accordingly, our primary goal for this project was to create a Java-based Datalog engine. Ideally, the engine would also:

1. Take advantage of multiple cores during query evaluation
2. Be easily extensible with new evaluation strategies or even new language features
3. Provide a clean (and easily extensible) Java interface so that it could efficiently be integrated within other JVM-based applications

As secondary goals, we also hoped to:

1. Create pedagogical material to accompany our implementation, such as a tutorial
2. Implement new language features, such as types or limited function symbols

Current Status

We have built an engine that seamlessly **supports over half-a-dozen query evaluation strategies**, including:

- Bottom-up techniques that derive all facts
- Top-down techniques that derive only the facts necessary to answer a particular query
- Program rewriting techniques that transform programs in response to a query so that bottom-up evaluation is as efficient as top-down evaluation

We have experimented with different **concurrent evaluation techniques** to improve scalability, both by adapting single-threaded algorithms and creating new algorithms more suitable for concurrent evaluation.

We have also implemented a language extension that allows for hypothetical reasoning known as **DatalogHypothetical.**

Finally, we have created a simple **graphical user interface**, which is suitable for use in an undergraduate programming language course.

- Developer can load, edit and save Datalog source code
- Datalog programs can be dynamically loaded and queried with a built-in interpreter to give the developer real-time feedback

Datalog Overview [1,3,6]

Basics:

- Logic programming language born from the intersection of database theory and logic
- Truly declarative (unlike Prolog), which allows programmers to focus on **what** they want to compute, not **how** they have to compute it
- Supports recursively defined rules (unlike query languages based in relational algebra)
- P-complete, so every Datalog computation is guaranteed to terminate in time polynomial in the size of the input database

Grammar:

$X \in Var$	$t \in Term$	$t ::= X \mid a$
$p \in PredSym$	$A \in Atom$	$A ::= p(t,...,t)$
$a \in Const$	$C \in Clause$	$C ::= A \leftarrow A,..., A.$

A definition: an atom is ground if all its terms are constants

A restriction: every variable in the head of a clause must be in the body

Semantics:

A Datalog program is a set of rules $R \subseteq Clause$. The denotation of a Datalog program is a function from sets of ground atoms to sets of ground atoms. The input set is often thought of as a database of facts DB.

$[\![R]\!]DB = \{A \mid R, DB \vdash A\}$

1. $R, DB \vdash A$ if $A \in DB$
2. $R, DB \vdash A$ if there is a rule $A' \leftarrow A{\downarrow}1, ..., A{\downarrow}k.$ in R and a ground substitution θ such that $A = A'\theta$ and $R, DB \vdash A{\downarrow}i\,\theta$ for $1 \le i \le k$

Intuitively, the clause $A' \leftarrow A{\downarrow}1, ..., A{\downarrow}k.$ corresponds to the logical formula

$\forall X{\downarrow}1, ..., X{\downarrow}m.\, A{\downarrow}1 \wedge \cdots \wedge A{\downarrow}k \Rightarrow A'$

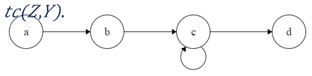
where $A{\downarrow}1, ..., A{\downarrow}m$ are the variables in $A{\downarrow}1, ..., A{\downarrow}k$.

Example: Graph Transitive Closure

Suppose we want to compute graph transitive closure. We can represent a directed graph as a set of binary $edge$ predicates. We can then compute transitive closure with the rules:

$tc(X,Y) \leftarrow edge(X,Y).$
$tc(X,Y) \leftarrow edge(X,Z), tc(Z,Y).$

Take this graph:



$DB = \{edge(a,b), edge(b,c), edge(c,c), edge(c,d)\}$

Taking R to be the above rules, we then have:

$[\![R]\!]DB = DB \cup \{tc(a,b), tc(a,c), tc(a,d), tc(b,c), tc(b,d), tc(c,c), tc(c,d)\}$

Parallelizing Datalog

Motivation: Logic programming languages have traditionally had poor performance, limiting their use in practice.

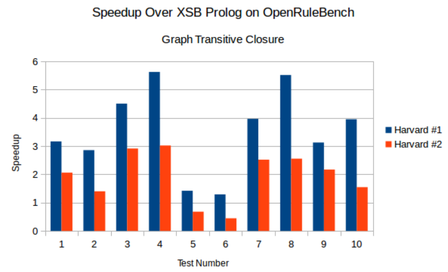
Limitation: Since Datalog is P-complete [5], it theoretically does not parallelize well (assuming P≠NC). However, our hope was that typical Datalog programs would respond well to parallelization.

Approach:

- Leveraged work-stealing thread scheduling
- Adapted a single-threaded semi-naïve evaluation and created a new algorithm tailored for concurrent evaluation

Results:

- 1.2-5X speedup over XSB Prolog (a leading implementation) when using 32 cores on the transitive closure test suite from the OpenRuleBench benchmarks [7]



Next Steps:

- Leverage profiling tools to find opportunities for further optimizations
- Test performance on other Datalog programs (besides transitive closure)

DatalogHypothetical [2,8]

Basics:

- Extends Datalog with the ability to reason hypothetically, i.e., what can we derive if we assume a given fact is in the input database?
- PSPACE-complete, so much more powerful than Datalog (P-complete) and not as powerful as Prolog (Turing-complete)

Grammar:

DatalogHypothetical extends the grammar of Datalog as follows:

$\phi \in Premise$	$\phi ::= A \mid A > A$	
$C \in Clause$	$C ::= A \leftarrow \phi, ..., \phi.$	

Semantics:

Similarly to Datalog, the denotation of a DatalogHypothetical program is a function from sets of ground atoms to sets of ground atoms.

$[\![R]\!]DB = \{A \mid R, DB \vdash A\}$

1. $R, DB \vdash A$ if $A \in DB$
2. $R, DB \vdash A$ if there is a rule $A' \leftarrow \phi{\downarrow}1, ..., \phi{\downarrow}k$ in R and a ground substitution θ such that $A = A'\theta$ and $R, DB \vdash \phi{\downarrow}i\,\theta$ for $1 \le i \le k$
3. $R, DB \vdash A' > A$ if $R, DB \cup \{A'\} \vdash A$

Example: Graduation Eligibility

Say we have a database of students, courses, and the courses students have taken. Using Datalog, we could compute facts about graduation eligibility. For instance, maybe a student is eligible to graduate after taking CS 152 and STAT 110:

$grad(S) \leftarrow student(S), taken(S, cs152), taken(S, stat110).$

But what if we want to reason about who is one (or two) courses away from graduation? DatalogHypothetical to the rescue!

$grad1(S,C) \leftarrow student(S), course(C), taken(S,C) > grad(S).$
$grad2(S,C1,C2) \leftarrow student(S), course(C1), course(C2)$
$taken(S,C1) > grad1(S,C2).$

Potential Applications in Privacy Tools

Modeling Privacy Legislation for DataTags:

- DataTags broadly state permitted and disallowed actions on datasets, always giving "safe" answers, even when a particular action might be allowed on a particular dataset
- We hope to encode relevant portions of privacy legislation in a logic program to allow a finer-grained analysis
- Having our own implementation of Datalog will allow us to create new language features as necessary
- In particular, our DatalogHypothetical implementation might be helpful, as hypotheticals were found to be useful in formalizing the British Nationality Act [4]

Declarative Encoding of DataTags Questionnaires:

- Questionnaires currently encoded in a procedural manner
- Datalog has the potential to encode questionnaires declaratively, which would allow for easier extensibility and maintenance
- Open questions include how to extend Datalog with dynamic user interaction, and how to determine the best order to ask questions
- Having a home-built Datalog implementation will allow us to more easily explore these issues

References

1. Serge Abiteboul, Richard Hull, and Victor Vianu. 1995. *Foundations of Databases*. Addison-Wesley, Reading, MA.
2. Anthony J. Bonner. 1990. Hypothetical datalog: complexity and expressibility. *Theor. Comput. Sci.* 76, 1 (October 1990), 3-51. DOI=10.1016/0304-3975(90)90011-6 http://dx.doi.org/10.1016/0304-3975(90)90011-6
3. S. Ceri, G. Gottlob, and L. Tanca. 1989. What You Always Wanted to Know About Datalog (And Never Dared to Ask). *IEEE Trans. on Knowl. and Data Eng.* 1, 1 (March 1989), 146-166. DOI=10.1109/69.43410 http://dx.doi.org/10.1109/69.43410
4. D.M. Gabbay, U. Reyle, N-Prolog: An extension of Prolog with hypothetical implications. I., The Journal of Logic Programming, Volume 1, Issue 4, 1984, Pages 319-355, ISSN 0743-1066, http://dx.doi.org/10.1016/0743-1066(84)90029-3.
5. Evgeny Dantsin, Thomas Eiter, Georg Gottlob, and Andrei Voronkov. 2001. Complexity and expressive power of logic programming. *ACM Comput. Surv.* 33, 3 (September 2001), 374-425. DOI=10.1145/502807.502810 http://doi.acm.org/10.1145/502807.502810
6. Todd J. Green, Shan Shan Huang, Boon Thau Loo, and Wenchao Zhou. 2013. Datalog and Recursive Query Processing. *Found. Trends databases* 5, 2 (November 2013), 105-195. DOI=10.1561/1900000017 http://dx.doi.org/10.1561/1900000017
7. Senlin Liang, Paul Fodor, Hui Wan, and Michael Kifer. 2009. OpenRuleBench: an analysis of the performance of rule engines. In *Proceedings of the 18th international conference on World wide web* (WWW '09). ACM, New York, NY, USA, 601-610. DOI= http://dx.doi.org/10.1145/1526709.1526790
8. Henning Christiansen and Troels Andreasen. 1996. A Practical Approach to Hypothetical Database Queries. In *International Seminar on Logic Databases and the Meaning of Change, Transactions and Change in Logic Databases* (ILPS '97), Burkhard Freitag, Hendrik Decker, Michael Kifer, and Andrei Voronkov (Eds.). Springer-Verlag, London, UK, UK, 340-355.