Report: Datalog with Recursive Aggregation for Incremental Program Analyses

Tamás Szabó
itemis AG / JGU Mainz
Germany

Gábor Bergmann*
Budapest University of Technology and Economics
MTA-BME Lendület Research Group on Cyber-Physical Systems
Hungary

Sebastian Erdweg
JGU Mainz
Germany

Markus Voelter
independent / itemis AG
Germany

This abstract is a summary of our OOPSLA'18 paper *"Incrementalizing Lattice-Based Program Analyses in Datalog"* [6]. Original download URL: `https://dl.acm.org/doi/10.1145/3276509`.

Extended Abstract

A static analysis is a tool that reasons about the runtime behavior of a computer program without actually running it. This way static analyses can help to catch runtime errors already at development time before the code goes to production, thereby saving significant costs on maintenance and the mitigation of potential software failures. To this end, static analyses are widely used in many areas of software development. For example, Integrated Development Environments (IDEs) use type checkers or data-flow analyses to provide continuous feedback to developers as they modify their code.

Datalog is a logic programming language that sees a resurgence in the static analysis community. There are many examples of static analyses specified in Datalog; ranging from network analysis on Amazon-scale systems [1] to inter-procedural points-to analysis of large Java projects [5]. The benefit of using Datalog for static analyses is that the declarative nature of the language allows quick prototyping of modular and succinct analysis implementations. Moreover, the execution details of a Datalog program is left to a *solver*, and solvers are free to optimize the execution in many ways. Our goal is to *incrementalize* the Datalog solver, in order to provide continuous feedback with analyses in IDEs.

In response to a program change, an incremental analysis reuses the previously computed results and updates them based on the *changed* code parts. In many applications, incrementality has been shown to bring significant performance improvements over from-scratch re-computation. There also exist incremental Datalog solvers, but they are *limited in expressive power*: They only support standard Datalog with powersets, and there is no support for *recursive aggregation* over custom lattices. This is a problem because static analyses routinely use custom lattices and aggregation (under monotonic aggregation semantics [4]), e.g. when analyzing a subject program with a loop and computing a fixpoint over a lattice using the least upper bound operator. Incrementalizing such computations is challenging, as aggregate results may recursively depend on themselves. It is true that certain finite lattices are expressible with powersets, so incrementalization techniques for standard Datalog may seemingly apply, but Madsen et al. explain that such encodings often incur prohibitive computational cost, while infinite lattices cannot even be expressed with powersets [3].

*The second author was partially supported by the János Bolyai Research Scholarship of the Hungarian Academy of Sciences and by the ÚNKP-19-4 New National Excellence Program of the Ministry For Innovation and Technology.

Submitted to:
ICLP 2020 Sister Track

In this paper, we present an incremental analysis framework called IncA that supports recursive aggregation over custom lattices. IncA uses Datalog extended with lattices for analysis specification. IncA shields analysis developers from the details of efficient execution, as it automatically incrementalizes program analyses in the face of program changes. We develop a novel incremental solver called DRed_L. DRed_L extends the well-known DRed algorithm [2] to support the standard semantics introduced by Ross and Sagiv for recursive monotonic aggregation [4]. Our key idea is that instead of decomposing a program change into deleted and inserted tuples, DRed_L works with antimonotonic and monotonic changes according to the partial order of the chosen lattice. We have formally proven that DRed_L is correct and yields the exact same result as running a Datalog program from scratch. Although we apply DRed_L to incrementalize program analyses, our contributions are generally applicable to any Datalog program using recursive aggregation over custom lattices.

We evaluate the applicability and performance of IncA with real-world static analyses. We implement a lattice-based points-to analysis that reasons about the potential target heap objects of variables, and we implement a number of static analyses that reason about the potential values of string-typed variables. We benchmark the analyses on open-source code bases with sizes up to 70 KLoC, and we synthesize program changes. We find that IncA consistently delivers good performance: After an initial analysis that takes a few tens of seconds, the incremental updates times are on the millisecond ballpark. The price of incrementalization is the extra memory use. We find that the memory consumption of IncA can grow large (up to 5GB), but it is not prohibitive for applications in IDEs.

References

[1] John Backes, Sam Bayless, Byron Cook, Catherine Dodge, Andrew Gacek, Alan J. Hu, Temesghen Kahsai, Bill Kocik, Evgenii Kotelnikov, Jure Kukovec, Sean McLaughlin, Jason Reed, Neha Rungta, John Sizemore, Mark A. Stalzer, Preethi Srinivasan, Pavle Subotic, Carsten Varming & Blake Whaley (2019): *Reachability Analysis for AWS-Based Networks*. In: *Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II*, pp. 231–241, doi:10.1007/978-3-030-25543-5_14. Available at `https://doi.org/10.1007/978-3-030-25543-5_14`.

[2] Ashish Gupta, Inderpal Singh Mumick & V. S. Subrahmanian (1993): *Maintaining Views Incrementally*. In: *Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data*, SIGMOD '93, ACM, New York, NY, USA, pp. 157–166, doi:10.1145/170035.170066. Available at `http://doi.acm.org/10.1145/170035.170066`.

[3] Magnus Madsen, Ming-Ho Yee & Ondřej Lhoták (2016): *From Datalog to Flix: A Declarative Language for Fixed Points on Lattices*. In: *Proceedings of the 37th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '16, ACM, New York, NY, USA, pp. 194–208, doi:10.1145/2908080.2908096. Available at `http://doi.acm.org/10.1145/2908080.2908096`.

[4] Kenneth A. Ross & Yehoshua Sagiv (1992): *Monotonic Aggregation in Deductive Databases*. In: *Proceedings of the Eleventh ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, PODS '92, ACM, New York, NY, USA, pp. 114–126, doi:10.1145/137097.137852. Available at `http://doi.acm.org/10.1145/137097.137852`.

[5] Yannis Smaragdakis & Martin Bravenboer (2011): *Using Datalog for Fast and Easy Program Analysis*. In: *Proceedings of the First International Conference on Datalog Reloaded*, Datalog'10, Springer-Verlag, Berlin, Heidelberg, pp. 245–251, doi:10.1007/978-3-642-24206-9_14. Available at `http://dx.doi.org/10.1007/978-3-642-24206-9_14`.

[6] Tamás Szabó, Gábor Bergmann, Sebastian Erdweg & Markus Voelter (2018): *Incrementalizing Lattice-Based Program Analyses in Datalog*. *Proc. ACM Program. Lang.* 2(OOPSLA), doi:10.1145/3276509. Available at `https://doi.org/10.1145/3276509`.