Designing a Datalog Engine for Industrial-Grade Static Analysis

Pavle Subotić[1] and Bernhard Scholz[2]

[1] Microsoft, Serbia
[2] University of Sydney, Australia

Abstract. Datalog has become popular as a specification language for static program analyzers. However, when deployed in real-world applications, classical Datalog engine and vanilla Datalog languages are insufficient for dealing with challenges such as scale and expressiveness for phrasing static analyzers. In this position paper, we present the design of the Datalog engine Soufflé for real-world static program analyzers. We outline several key innovations concerning language design, evaluation techniques and usability.

Introduction

Over the last 30 years, static program analysis has been frequently suggested as a fitting use case for Datalog. In this setup, the Datalog language acts as a concise domain-specific language for specifying the semantics of static analysis with its subset lattice domains. As a result, the Datalog engine becomes a powerful fix-point machinery that computes the least fix-point solution to the static analysis problem. In practice, however, standard Datalog dialects and engines have their limitations in crafting a real-world static analyzer. Industrial-grade static analyzers are a significant engineering undertaking and require to scale to problems within time, memory and precision bounds dictated by an organization's service level agreement (SLA). Standard techniques implementing Datalog engines do not scale compared to static analyzers crafted with non-declarative languages because expressing a static analyzer in simple Horn clauses is too terse and cumbersome to express an industrial-grade static analysis. In this position paper, we outline our experience developing Soufflé [3]: a Datalog engine designed and implemented for static code analysis. We showcase several innovations in language design, evaluation and debugging that have made Soufflé popular for developing a diverse range of real-world static analyses [4, 3, 5, 6].

Language.

We refer the reader to [1] for a detailed description of standard Datalog. Here, we discuss the differences between conventional Datalog use cases in the Database realm and the program analysis use case and what extensions are needed to express analyzers rapidly. Use cases like graph databases typically have vast data sets with a few rules that model queries. In contrast, real-world static analyses require thousands of rules, hundreds of relations with complex recursion, and

access patterns. Thus, Datalog, in this context, resembles a logical specification language that requires a considerable software engineering effort. Therefore, in Soufflé, we have implemented several language features that aid reuse, expressiveness, and flexibility. We have implemented meta-level constructs such as components, Abstract Data Types (ADTs), a type system, and pre-processing directives to enable abstraction and reuse. These mechanisms allow for easier reuse, as has been shown by the Doop static analysis library implemented with the Soufflé language. There are also semantic subtleties in expressing static analyzers in Datalog. For example, in program analysis, we require richer domains to model other things than subset lattices (the only domain standard Datalog would permit) like context sensitivity, paths, states etc. Therefore in Soufflé, we have implemented several domains, including numbers, records, and strings. The operations on these domains require built-in *functors*. As a consequence of allowing Datalog to operate on these domains, we no longer provide termination guarantees. Recently, we have extended Soufflé so that set subsumption can be performed via subsumptive rules, allowing complex lattices with deletions to be expressed. Although we have extended semantics such as the Choice construct [7], we acknowledge that static analyzers must interface with external systems or require code best done in imperative languages such as C. We thus provide external functors where this code can be integrated as a functor in a Datalog rule.

Evaluation.

The standard static analysis workflow allows for the ruleset to be programmed *before* execution. Thus Soufflé compiles Datalog rules to highly specialized concurrent C++ code [3]. This specialization process performs various optimizations at compile time, including index selection [13] and join optimizations [2]. Users can select from a number of indexing data structures [11] including a concurrent BTree [10], Brie [9] or EqRel [12] data structure. Besides the compilation system, Soufflé provides a high-performance interpreter [8] that permits use cases whose rules/programs change on-the-fly. A high-performance interpreter using de-specialized data structures executes programs approximately two orders slower than the compiled version but has no compilation overhead, which can be significant for large Datalog programs. Moreover, several program analysis use cases, such as static analysis in development pipelines, can utilize incremental evaluation to avoid unnecessary re-computations. Soufflé provides a unique *elastic* incremental evaluation mechanism [14] that selectively performs incremental evaluation.

Usability.

As with any large software engineering effort, static analyzers require debugging support. Therefore, Soufflé provides proof annotations [15] that allow developers to construct minimal proof trees to understand better how their ruleset resulted in the existence or non-existence of a tuple.

References

1. Abiteboul, S., Hull, R., Vianu, V.: Foundations of Databases. Addison-Wesley Publishing Company (1995)
2. Arch, S., Hu, X., Zhao, D., Subotic, P., Scholz, B.: Building a join optimizer for soufflé. In: Villanueva, A. (ed.) Logic-Based Program Synthesis and Transformation - 32nd International Symposium, LOPSTR 2022, Tbilisi, Georgia, September 21-23, 2022, Proceedings. Lecture Notes in Computer Science, vol. 13474, pp. 83–102. Springer (2022). https://doi.org/10.1007/978-3-031-16767-6_5, https://doi.org/10.1007/978-3-031-16767-6_5
3. Backes, J., Bayless, S., Cook, B., Dodge, C., Gacek, A., Hu, A.J., Kahsai, T., Kocik, B., Kotelnikov, E., Kukovec, J., McLaughlin, S., Reed, J., Rungta, N., Sizemore, J., Stalzer, M.A., Srinivasan, P., Subotic, P., Varming, C., Whaley, B.: Reachability analysis for aws-based networks. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification - 31st International Conference, CAV 2019, New York City, NY, USA, July 15-18, 2019, Proceedings, Part II. Lecture Notes in Computer Science, vol. 11562, pp. 231–241. Springer (2019). https://doi.org/10.1007/978-3-030-25543-5_14, https://doi.org/10.1007/978-3-030-25543-5_14
4. Bravenboer, M., Smaragdakis, Y.: Strictly declarative specification of sophisticated points-to analyses. SIGPLAN Not. **44**(10), 243–262 (2009). https://doi.org/10.1145/1639949.1640108, http://doi.acm.org/10.1145/1639949.1640108
5. Flores-Montoya, A., Schulte, E.: Datalog disassembly. In: Proceedings of the 29th USENIX Conference on Security Symposium. SEC'20, USENIX Association, USA (2020)
6. Grech, N., Brent, L., Scholz, B., Smaragdakis, Y.: Gigahorse: Thorough, declarative decompilation of smart contracts. In: Proceedings of the 41st International Conference on Software Engineering. p. 1176–1186. ICSE '19, IEEE Press (2019). https://doi.org/10.1109/ICSE.2019.00120, https://doi.org/10.1109/ICSE.2019.00120
7. Hu, X., Karp, J., Zhao, D., Zreika, A., Wu, X., Scholz, B.: The choice construct in the soufflé language. In: Oh, H. (ed.) Programming Languages and Systems - 19th Asian Symposium, APLAS 2021, Chicago, IL, USA, October 17-18, 2021, Proceedings. Lecture Notes in Computer Science, vol. 13008, pp. 163–181. Springer (2021). https://doi.org/10.1007/978-3-030-89051-3_10, https://doi.org/10.1007/978-3-030-89051-3_10
8. Hu, X., Zhao, D., Jordan, H., Scholz, B.: An efficient interpreter for datalog by de-specializing relations. In: Freund, S.N., Yahav, E. (eds.) PLDI '21: 42nd ACM SIGPLAN International Conference on Programming Language Design and Implementation, Virtual Event, Canada, June 20-25, 2021. pp. 681–695. ACM (2021). https://doi.org/10.1145/3453483.3454070, https://doi.org/10.1145/3453483.3454070
9. Jordan, H., Subotic, P., Zhao, D., Scholz, B.: Brie: A specialized trie for concurrent datalog. In: Chen, Q., Huang, Z., Si, M. (eds.) Proceedings of the 10th International Workshop on Programming Models and Applications for Multicores and Manycores, PMAM@PPoPP 2019, Washington, DC, USA, February 17, 2019. pp. 31–40. ACM (2019). https://doi.org/10.1145/3303084.3309490, https://doi.org/10.1145/3303084.3309490
10. Jordan, H., Subotic, P., Zhao, D., Scholz, B.: A specialized b-tree for concurrent datalog evaluation. In: Hollingsworth, J.K., Keidar, I. (eds.) Proceedings of the 24th ACM SIGPLAN Symposium on Principles and Practice of

Parallel Programming, PPoPP 2019, Washington, DC, USA, February 16-20, 2019. pp. 327–339. ACM (2019). https://doi.org/10.1145/3293883.3295719, https://doi.org/10.1145/3293883.3295719
11. Jordan, H., Subotic, P., Zhao, D., Scholz, B.: Specializing parallel data structures for datalog. Concurr. Comput. Pract. Exp. **34**(2) (2022). https://doi.org/10.1002/cpe.5643, https://doi.org/10.1002/cpe.5643
12. Nappa, P., Zhao, D., Subotic, P., Scholz, B.: Fast parallel equivalence relations in a datalog compiler. In: 28th International Conference on Parallel Architectures and Compilation Techniques, PACT 2019, Seattle, WA, USA, September 23-26, 2019. pp. 82–96. IEEE (2019). https://doi.org/10.1109/PACT.2019.00015, https://doi.org/10.1109/PACT.2019.00015
13. Subotic, P., Jordan, H., Chang, L., Fekete, A.D., Scholz, B.: Automatic index selection for large-scale datalog computation. Proc. VLDB Endow. **12**(2), 141–153 (2018). https://doi.org/10.14778/3282495.3282500, http://www.vldb.org/pvldb/vol12/p141-subotic.pdf
14. Zhao, D., Subotic, P., Raghothaman, M., Scholz, B.: Towards elastic incrementalization for datalog. In: Veltri, N., Benton, N., Ghilezan, S. (eds.) PPDP 2021: 23rd International Symposium on Principles and Practice of Declarative Programming, Tallinn, Estonia, September 6-8, 2021. pp. 20:1–20:16. ACM (2021). https://doi.org/10.1145/3479394.3479415, https://doi.org/10.1145/3479394.3479415
15. Zhao, D., Subotic, P., Scholz, B.: Debugging large-scale datalog: A scalable provenance evaluation strategy. ACM Trans. Program. Lang. Syst. **42**(2), 7:1–7:35 (2020). https://doi.org/10.1145/3379446, https://doi.org/10.1145/3379446