Beyond Deductive Datalog: How Datalog is used to perform AI tasks for program analysis

Bernhard Scholz[1], Pavle Subotić[1], and David Zhao[2]

[1] Sonic Research**
[2] RelationalAI

Abstract. Datalog is a popular reasoning engine for use cases such as static program analysis. However, the standard reasoning that Datalog engines provide is insufficient to perform "AI" tasks such as code repair and rule synthesis as typically found in Machine Learning based analyzers. In this paper we discuss how Datalog engines can be leveraged to perform similar tasks using logical reasoning.

Introduction

Datalog has become a popular language for implementing declarative program analyses [2]. 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 piece of powerful fix-point machinery that computes the least fix-point solution to the static program analysis problem. However, as static analyzers have evolved, the standard program analysis workflow is insufficient for mass adoption in software engineering. A static analysis tool is expected to provide additional intelligence beyond flagging potentially erroneous code. For instance, engineers typically want to understand why code has been flagged as potentially erroneous and how it can be fixed, and they even expect analyzers to learn custom analyses from examples.

In recent work, we have explored techniques that leverage Datalog engines such as Soufflé [1] to perform tasks such as provenance [7], input repair [4] and rule synthesis [6]. In the static program analysis context, provenance allows users to understand why the static program analyzer has flagged some code as erroneous. Input repair provides users with proposed fixes to their erroneous code, and rule synthesis can suggest rectifying incorrect analyses or even generating entirely new ones.

Given the popularity of employing black box Large Language Models (LLMs) for such "AI" tasks, we believe this work presents an interesting alternative technique based on logical reasoning. Consequently, unlike LLMs, our techniques are not subject to the hallucinations phenomena [3] and require low resource usage. We finally discuss how Datalog-based program analysis, repair, and synthesis can co-exist with LLM-based techniques.

** Prev. Fantom Research

Program Provenance.

A static analyzer presents users with a list of alarms (potential bugs). The user then has to triage these to determine if they are actionable. To aid triaging, an explanation of why the analyzer believes the alarm to be true must be presented to the user. For Datalog-based static analyzers, we have proposed using succinct proof trees [7]. Since proof trees can be prohibitively large, the work in [7] used proof annotations, i.e., information from the bottom-up evaluation, to compute minimal proof trees (w.r.t. a given metric) in a top-down manner. Given a succinct proof tree, the user can understand the reasoning behind the alarm and better understand where the reasoner may be wrong or have a certificate for the bug found.

Program Repair.

Users can frequently find traces [5], proofs [7], etc., tedious to follow, especially for large programs. It is easier for the user to triage the alarm by being given a fix suggestion, i.e., a way to repair the code so the bug is no longer present. This can be seen in human terms through a code review. A reviewer typically doesn't provide step by step reasoning for why they think there is a bug, instead they typically suggest a new code fragment that will rectify the problem. Similarly, the work in [4] annotates the inputs and rules with symbolic terms. Here Soufflé performs a standard bottom-up fixpoint computation, which can be seen as a type of symbolic execution. Then, using an SMT solver, we can find models that remove the errors by removing, adding or changing the inputs.

Static Analysis Synthesis.

A major reason for the popularity of Datalog-based static analyzers is that they allow users to encode domain-specific bugs. However, despite Datalog's declarative and high-level nature, this type of programming is prohibitive for some users. Given a set of examples, such users could generate an analysis they could use for similar bugs. To this end, the work in [6] uses Soufflé to synthesize Datalog programs from input-output specifications. This approach leverages query provenance [7] to scale the counterexample-guided inductive synthesis (CEGIS) procedure for program synthesis. In each iteration of the procedure, a SAT solver proposes a candidate Datalog program and a Datalog solver evaluates the proposed program to determine whether it meets the desired specification – failure to satisfy the specification results in additional constraints to the SAT solver.

Conclusion and Future Directions.

We have presented several techniques that combine reasoning techniques (e.g., Datalog, SMT) to perform tasks that are these days attributed to ML-based reasoning (e.g., LLMs). In the context of static analysis, our reasoners over-approximate. On the other-hand, they are sound and do not suffer from hallucinations. Many techniques require domain-specific knowledge (e.g., templates) to improve precision. Similarly, LLMs also require prompts from users to improve precision. An interesting line of work is investigating the combination of logical reasoners with ML-based approaches.

References

1. 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
2. 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
3. Ji, Z., Lee, N., Frieske, R., Yu, T., Su, D., Xu, Y., Ishii, E., Bang, Y.J., Madotto, A., Fung, P.: Survey of hallucination in natural language generation. ACM Comput. Surv. **55**(12) (mar 2023). https://doi.org/10.1145/3571730, https://doi.org/10.1145/3571730
4. Liu, Y., Mechtaev, S., Subotic, P., Roychoudhury, A.: Program repair guided by datalog-defined static analysis. In: Chandra, S., Blincoe, K., Tonella, P. (eds.) Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/FSE 2023, San Francisco, CA, USA, December 3-9, 2023. pp. 1216–1228. ACM (2023). https://doi.org/10.1145/3611643.3616363, https://doi.org/10.1145/3611643.3616363
5. Psallidas, F., Leszczynski, M.E., Namaki, M.H., Floratou, A., Agrawal, A., Karanasos, K., Krishnan, S., Subotic, P., Weimer, M., Wu, Y., Zhu, Y.: Demonstration of geyser: Provenance extraction and applications over data science scripts. In: Das, S., Pandis, I., Candan, K.S., Amer-Yahia, S. (eds.) Companion of the 2023 International Conference on Management of Data, SIGMOD/PODS 2023, Seattle, WA, USA, June 18-23, 2023. pp. 123–126. ACM (2023). https://doi.org/10.1145/3555041.3589717, https://doi.org/10.1145/3555041.3589717
6. Raghothaman, M., Mendelson, J., Zhao, D., Naik, M., Scholz, B.: Provenance-guided synthesis of datalog programs. Proc. ACM Program. Lang. **4**(POPL), 62:1–62:27 (2020). https://doi.org/10.1145/3371130, https://doi.org/10.1145/3371130
7. 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