Deriving Test Oracles for Verification Infrastructure

Maria Christakis[1] [0000−0002−2649−1958]

TU Wien, Vienna, Austria
maria.christakis@tuwien.ac.at

Abstract. Program analyzers are increasingly trusted as infrastructure for software correctness: they prove safety properties, find bugs, discharge verification conditions, and support automated-reasoning pipelines. Yet analyzers are themselves complex software systems, built from abstract domains, fixpoint engines, symbolic executors, solvers, frontends, encodings, and heuristics. Fully verifying modern analyzers is rarely realistic. This raises a complementary question: how can we systematically test the tools that verify our programs?

This article describes an escalation of techniques for addressing the oracle problem in analyzer testing. When mathematical contracts are available, specification-based testing can check critical analyzer components directly. In the absence of such specifications, program generation can produce analyzable programs whose properties are known by construction, although typically within a restricted fragment of the input space. When ground truth is unavailable, differential testing can compare analyzers against one another. When such comparisons are unavailable, metamorphic testing can derive expected outputs by transforming inputs in ways that preserve or predictably change their semantics. Finally, interrogation testing makes metamorphic testing adaptive: instead of asking isolated questions, it cross-examines analyzers and solvers using their own previous answers to expose contradictions.

We close by discussing how these ideas extend beyond analyzers to other systems that make semantic claims, including zero-knowledge systems, secure multiparty computation compilers, and machine-learning models.

Keywords: Program analysis · Oracle problem · Interrogation testing.

1 From Specifications to Interrogation

A natural response to bugs in analyzers is to ask whether the analyzers themselves should be verified. In principle, this is the cleanest solution: a verified analyzer would come with a proof that its answers are sound with respect to a formal semantics. In practice, however, modern analyzers combine frontends, intermediate representations, abstract domains, fixpoint engines, symbolic execution, solvers, memory models, optimizations, and reporting mechanisms. The implemented system is usually too large, too heterogeneous, or too rapidly evolving to be verified end-to-end.

This makes testing a necessary complement to verification. The difficulty is the oracle problem: for many interesting programs, we do not know in advance whether the analyzer should report safe, unsafe, or unknown. A proof of safety may be correct, or it may be an unsound false negative; an alarm may reveal a real bug, or it may be a loss of precision. Even auxiliary artifacts, such as invariants or counterexamples, may be inconsistent with the claimed result.

The following techniques address different oracle-construction obstacles in analyzer testing. When a component has a mathematical contract, specification-based testing can check the implementation directly, as in our work on numerical abstract domains [1]. Without such specifications, program generation can produce analyzable programs whose relevant properties are known by construction, though typically within a restricted fragment of the input space [12, 5]. When ground truth is unavailable, differential testing can compare analyzers and treat disagreement as evidence of soundness or precision issues [10].

Metamorphic testing addresses the absence of ground truth or comparable analyzers: it transforms analyzer inputs in ways that preserve or predictably change their semantics, so that the expected output for the transformed input can be derived from the original one. We used this idea for Datalog engines, first through transformations that preserve or predictably change query results, and later through dependency-aware transformations that exploit rule dependencies [11, 13]. Interrogation testing addresses limitations of such one-step transformations. It uses richer information from analyzer answers and a knowledge base of past queries to guide future ones. This makes the testing process adaptive: instead of asking isolated questions, it records previous queries and answers, uses them to generate follow-up tests, and searches for contradictions. We used this methodology to expose soundness and precision issues in program analyzers [8] and CHC solvers, a foundational layer beneath many verification tools [9].

2 Beyond Program Analyzers

The need to derive useful test oracles is not unique to program analyzers. It arises whenever a system makes a semantic claim that is hard to check directly. A zero-knowledge system claims that a proof attests to a valid computation without revealing private data. A secure multiparty computation compiler claims that the generated protocol implements the source program. A machine-learning model is often expected to satisfy semantic relations over multiple input-output pairs. In each case, the test must derive an oracle for the claim being checked.

We have derived such oracles in several ways. For zero-knowledge systems [6, 7], we used metamorphic testing and fault injection to check whether implementations accept invalid claims or reject valid ones. For secure multiparty computation compilers [14], we used differential testing against an oracle implementation of the source program. For machine-learning models [3, 2, 4], we used specifications expressing metamorphic relations between inputs and predictions. Across these domains, the common theme is to make correctness claims testable by turning the artifacts that systems expose into oracles.

References

1. Bugariu, A., Wüstholz, V., Christakis, M., Müller, P.: Automatically testing implementations of numerical abstract domains. In: ASE. pp. 768–778. ACM (2018). https://doi.org/10.1145/3238147.3240464
2. Christakis, M., Eniser, H.F., Hoffmann, J., Singla, A., Wüstholz, V.: Specifying and testing k-safety properties for machine-learning models. In: IJCAI. pp. 4748–4757. ijcai.org (2023). https://doi.org/10.24963/IJCAI.2023/528
3. Eniser, H.F., Gros, T.P., Wüstholz, V., Hoffmann, J., Christakis, M.: Metamorphic relations via relaxations: An approach to obtain oracles for action-policy testing. In: ISSTA. pp. 52–63. ACM (2022). https://doi.org/10.1145/3533767.3534392
4. Eniser, H.F., Wüstholz, V., Christakis, M.: Automatically testing functional properties of code translation models. In: AAAI. pp. 21055–21062. AAAI (2024). https://doi.org/10.1609/AAAI.V38I19.30097
5. Fleischmann, M., Kaindlstorfer, D., Isychev, A., Wüstholz, V., Christakis, M.: Constraint-based test oracles for program analyzers. In: ASE. pp. 344–355. ACM (2024). https://doi.org/10.1145/3691620.3695035
6. Hochrainer, C., Isychev, A., Wüstholz, V., Christakis, M.: Fuzzing processing pipelines for zero-knowledge circuits. In: CCS. pp. 783–797. ACM (2025). https://doi.org/10.1145/3719027.3744791
7. Hochrainer, C., Wüstholz, V., Christakis, M.: Arguzz: Testing zkVMs for soundness and completeness bugs. In: Security. USENIX (2026), to appear
8. Kaindlstorfer, D., Isychev, A., Wüstholz, V., Christakis, M.: Interrogation testing of program analyzers for soundness and precision issues. In: ASE. pp. 319–330. ACM (2024). https://doi.org/10.1145/3691620.3695034
9. Kaindlstorfer, D., Isychev, A., Wüstholz, V., Christakis, M.: Interrogation testing of CHC solvers. In: FSE. ACM (2026), to appear
10. Klinger, C., Christakis, M., Wüstholz, V.: Differentially testing soundness and precision of program analyzers. In: ISSTA. pp. 239–250. ACM (2019). https://doi.org/10.1145/3293882.3330553
11. Mansur, M.N., Christakis, M., Wüstholz, V.: Metamorphic testing of Datalog engines. In: ESEC/FSE. pp. 639–650. ACM (2021). https://doi.org/10.1145/3468264.3468573
12. Mansur, M.N., Christakis, M., Wüstholz, V., Zhang, F.: Detecting critical bugs in SMT solvers using blackbox mutational fuzzing. In: ESEC/FSE. pp. 701–712. ACM (2020). https://doi.org/10.1145/3368089.3409763
13. Mansur, M.N., Wüstholz, V., Christakis, M.: Dependency-aware metamorphic testing of Datalog engines. In: ISSTA. pp. 236–247. ACM (2023). https://doi.org/10.1145/3597926.3598052
14. Watzinger, S., Wüstholz, V., Garg, D., Christakis, M.: Cost-effective testing of MPC compilers. In: FSE. ACM (2026), to appear