Instability Track for SMT-COMP

Amar Shah[1], Yi Zhou[2], Marijn Heule[1] and Bryan Parno[1]

[1]Carnegie Mellon University, Pittsburgh, PA, USA
[2]Amazon Web Services, New York, NY, USA

Abstract

We propose an Instability Track for SMT-COMP to address the critical issue of solver instability in program verification. SMT solvers often exhibit significant performance variations when solving semantically equivalent queries, creating challenges for industrial adoption of SMT-based verification. We describe Mariposa, our tool that measures solver instability by creating semantics-preserving mutants of SMT queries and performing statistical analysis. The proposed track would evaluate solvers both on their ability to solve verification queries quickly and their stability across query variations. We discuss benchmark sources from Dafny, F*, and Verus verification projects, and propose concrete scoring rubrics for evaluating solver stability.

Keywords

SMT-COMP, Instability, Program Verification, Quantifier Instantiation, Dafny, F*, Verus

A key use case of SMT solvers is automated program verification, solving queries created by tools such as Dafny [1], F* [2], and Verus [3]. These tools have a unique style of SMT encoding, use a mix of theories including non-linear arithmetic, and heavily rely on pattern-based quantifier instantiation techniques.

However, on these queries, SMT solvers suffer from *instability*, where small semantics-preserving changes (such as renaming variables or reordering assertions) can have large effects on the solver's performance. Instability is an obstacle to industrial adoption of SMT-based verification. Both Galois and Amazon have highlighted instability as a serious challenge for verification [4, 5]. Many other large-scale verification projects cite SMT instability as a key pain [6, 7, 8, 9, 10, 11, 12].

Measuring Instability. We have built a tool, Mariposa [13], that measures instability by creating many semantics-preserving mutants of an SMT query and running the solver on each mutant. It performs a statistical test to decide if a query is *unstable.* Oversimplifying, it concludes that (1) if most of the mutants are solvable the query is *stable*; (2) if some mutants are solvable and some are not then the query is *unstable*; and (3) if most mutants are not solvable, then the query is *unsolvable.* The *Instability Track* would reward a solver both for solving a large number of queries quickly and for being stable on the queries it does solve.

Mariposa can perform three different semantics-preserving changes to a query: renaming variables, reordering assertions, and changing the random seed given to the solver. Mariposa found that 2.6% of SMT queries from Dafny and F* verification projects are unstable [13]. Later work found that 1% of SMT queries from Verus verification projects are unstable [14].

These numbers are worrying as large verification projects generate many thousands of queries, and each unstable query inhibits developer productivity. Manually investigating and repairing instability takes away from time the developers could be spending writing new code or proofs. Additionally, we report instability from finalized projects. We expect the amount of instability during development to be higher.

Benchmarks. Mariposa contributed a set of instability benchmarks from various large-scale Dafny and F* verification projects [13]. We would also contribute a benchmark suite from ten different large-scale system verification projects written using Verus [3]. These have been submitted to the 2025

SMT 2025: 23rd International Workshop on Satisfiability Modulo Theories, August 10–11, 2025, Glasgow, UK
amarshah@cmu.edu (A. Shah); yizhou7@alumni.cmu.edu (Y. Zhou); marijn@cmu.edu (M. Heule); parno@cmu.edu (B. Parno)

SMT competition [15], but are scattered across four different benchmark families (UFDTLIA, UFDTNIA, UFDTBVLIA, and UFDTBVNIA). These queries contains uninterpreted functions, quantifers, algebraic data types, linear arithmetic and in certain cases non-linear arithmetic and bit-vectors. We hope to solicit benchmarks from other verification tools.

Scoring Metrics. We discuss several potential scoring metrics for an instability track. We start with some definitions. If we run Mariposa on a query q, it tells us that the query is solvable, unstable, or unsolvable. We define S as the set of solvable queries, U as the set of unstable queries, and N as the set of unsolvable queries. We use T for the timeout and for each query q, we define t_q as the time taken to solve q. We define a very simple scoring metric:

A. **Solved Score:** The solver with the greatest $|S|$ "wins."

This rewards solvers that can *stably* solve more queries. However it has two potential issues:

1. This metric does not consider the amount of time on each solvable query. For instance, we may want to reward solvers that solve queries quickly.
2. A solver is punished equally for failing to solve a query and for being unstable on a query. However, since this is the instability track, we may want to punish unstable queries more.

For each of these issues we propose a fix:

B. **PAR-2I Score**: Inspired by the PAR-2 score used in the SAT and SMT competitions, we propose the PAR-2I score. The PAR-2I score is the sum of the time taken to solve each solvable query plus two times the timeout for each unstable or unsolvable query. Specifically, the score is $\sum_{q \in S} t_q + 2 \cdot T \cdot |U \cup N|$. The solver with the lowest such score "wins."

C. **Instability Score:** The instability score is similar to the solved score, but "punishes" unstable queries more than unsolvable queries. Specifically the score is $|S| - |U|$ and the solver with the highest such score "wins."

The PAR-2I score is a fix for issue 1 and the instability score is a fix for issue 2. We could also consider a hybrid approach that combine these two methods. The exact scoring mechanism will be finalized based on community feedback.

Other Features. There are two interesting eccentricities of these SMT queries that could be incorporated into the track.

- **Goal-Axiom Divide.** Program verification tools like Dafny, F*, and Verus have a clear divide between the goal assertion and the axiom assertions. The goal captures the property we want to prove. For instance, this could be an assertion, ensures clause, or loop invariant at the source level. The axioms correspond to ambient facts at the source level from other functions, user-created lemmas, or language-level definitions.

  This can be recognized by the SMT track by having queries annotate the goal assertion.

- **Fast Unknowns vs. Timeouts.** Mariposa deems a query unstable for a solver if it cannot solve some of the semantically equivalent queries produced. However, there are two types of unstable queries: *fast unknowns* are when the solver quickly returns unknown and *timeouts* are when the solver uses its entire time budget without finding a solution. We believe that this corresponds to different issues inside the SMT solver: the former is when quantified axioms are underinstantiated, while the latter are when quantified axioms are overinstantiated. Figure 1 shows the distribution of failure modes in unstable queries, showing a clear divide between fast unknowns and timeouts.

  This can be recognized in SMT track by noting the number of fast unknowns and timeout failures for each solver. This would allow solvers to be scored on both of these metrics.

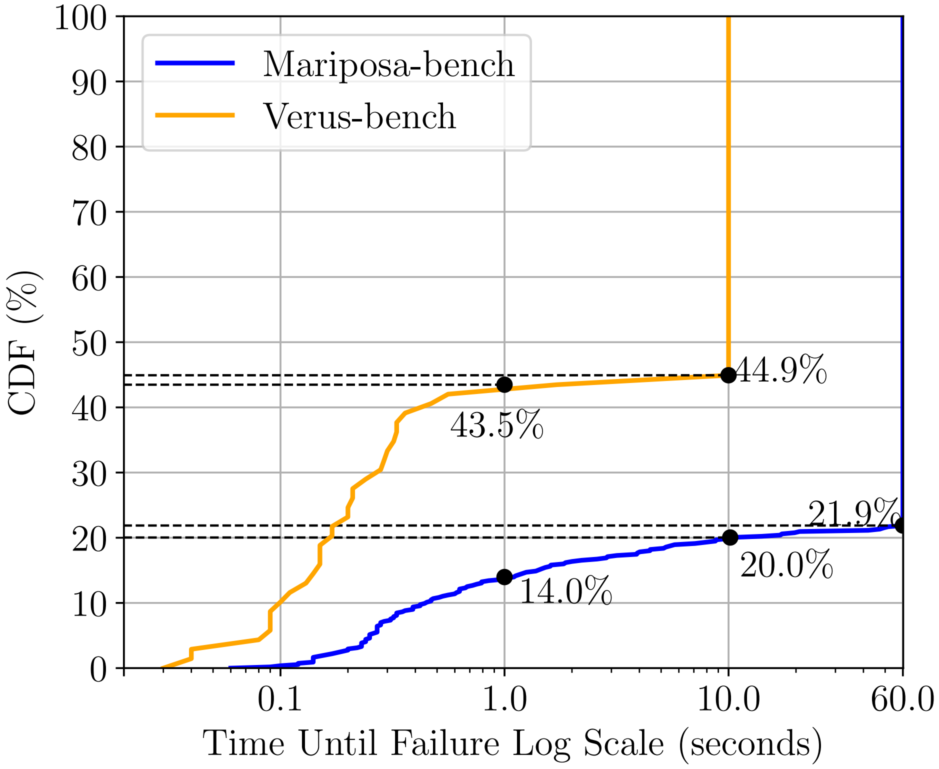

Figure 1: Distribution of failure modes in unstable mutants on a benchmark set from Mariposa [13] and a benchmarks set from the Rust verifier Verus [3]. The Verus benchmarks were run with a timeout of 10 seconds and the Mariposa benchmarks were run with a timeout of 60 seconds, corresponding to the default timeouts of the respective tools. The distribution is bimodal—most mutants either fail quickly (10 seconds for Mariposa or 1 second for Verus) or they timeout. Image is from Zhou et al. [14]

Solutions. There has been some work trying to fix instability. SHAKE automatically prunes the solver's context by removing redundant axioms in a pre-processing step [16]. Cazamariposas analyzes the quantifier instantiation graph of a query to either remove unnecessary quantifier instantiations or add missed instantiations [14]. Other work converts queries to a normal form to mitigate the variation in runtimes between syntactically different, but semantically equivalent, queries [17]. Dafny developers have proposed a method for instantiating universal quantifiers for axioms at the source level [18].

All of this work improves stability, but does not address key underlying issues, and instability is still prevalent when we apply these fixes. More work is required. We believe that much of this work can be done at the SMT-solver level and thus a stability track at the SMT competition could be immensely helpful for encouraging solver developers to improve stability.

Drawbacks. Many program verification tools are built with Z3 in mind, so other solvers may perform worse initially. However, we hope that introducing an instability track can incentivize all solvers to perform better on verification queries and encourage verification tools to use different solvers.

Another drawback is that measuring instability requires running many different mutants of a query, which could be computationally expensive. However, this is mitigated since program verification tools require much shorter timeouts (usually 60 seconds or less) compared to many other SMT queries.

Conclusions. In conclusion, we argue that instability is a critical issue for SMT-based program verification. We believe this is a problem that could be addressed at the solver level. We hope that an instability track at the SMT competition could be a step in this direction.

Acknowledgements We would like to thank the anonymous reviewers for their helpful feedback. We further acknowledge Dominic Winterer and the organizers of the SMT-COMP for the suggestion to present this at the SMT workshop. We thank all of the contributors to the Mariposa project and the Dafny, F*, and Verus developers for contributions to the benchmark suite.

This work was supported in part by the National Science Foundation (NSF) under grant 2224279, funding from AFRL and DARPA under Agreement FA8750-24-9-1000, and the Future Enterprise Security initiative at Carnegie Mellon CyLab (FutureEnterprise@CyLab).

Declaration on Generative AI. The authors have not employed any Generative AI tools.

References

[1] K. R. M. Leino, Dafny: An automatic program verifier for functional correctness, in: Logic for Programming, Artificial Intelligence, and Reasoning (LPAR), 2010.

[2] N. Swamy, C. Hriţcu, C. Keller, A. Rastogi, A. Delignat-Lavaud, S. Forest, K. Bhargavan, C. Fournet, P.-Y. Strub, M. Kohlweiss, J.-K. Zinzindohoue, S. Zanella-Béguelin, Dependent Types and Multi-Monadic Effects in F*, in: Proceedings of the ACM Symposium on Principles of Programming Languages (POPL), 2016.

[3] A. Lattuada, T. Hance, C. Cho, M. Brun, I. Subasinghe, Y. Zhou, J. Howell, B. Parno, C. Hawblitzel, Verus: Verifying Rust programs using linear ghost types, in: Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), 2023.

[4] M. Dodds, Formally Verifying Industry Cryptography, IEEE Security and Privacy Magazine (2022). doi:10.1109/MSEC.2022.3153035.

[5] N. Rungta, A billion SMT queries a day (invited paper), in: S. Shoham, Y. Vizel (Eds.), Proceedings of the International Conference on Computer Aided Verification (CAV), 2022.

[6] J. Backes, P. Bolignano, B. Cook, C. Dodge, A. Gacek, K. Luckow, N. Rungta, O. Tkachuk, C. Varming, Semantic-based automated reasoning for AWS access policies using SMT, in: Formal Methods in Computer Aided Design (FMCAD), 2018. doi:10.23919/FMCAD.2018.8602994.

[7] A. Chakarov, J. Geldenhuys, M. Heck, M. Hicks, S. Huang, G. A. Jaloyan, A. Joshi, R. Leino, M. Mayer, S. McLaughlin, A. Mritunjai, C. P. Claudel, S. Porncharoenwase, F. Rabe, M. Rapoport, G. Reger, C. Roux, N. Rungta, R. Salkeld, M. Schlaipfer, D. Schoepe, J. Schwartzentruber, S. Tasiran, A. Tomb, E. Torlak, J. Tristan, L. Wagner, M. Whalen, R. Willems, J. Xiang, T. J. Byun, J. Cohen, R. Wang, J. Jang, J. Rath, H. T. Syeda, D. Wagner, Y. Yuan, Formally verified cloud-scale authorization, in: International Conference on Software Engineering (ICSE), 2025. URL: https://www.amazon.science/publications/formally-verified-cloud-scale-authorization.

[8] J. W. Cutler, C. Disselkoen, A. Eline, S. He, K. Headley, M. Hicks, K. Hietala, E. Ioannidis, J. Kastner, A. Mamat, D. McAdams, M. McCutchen, N. Rungta, E. Torlak, A. M. Wells, Cedar: A new language for expressive, fast, safe, and analyzable authorization, in: Proceedings of the ACM Conference on Object-Oriented Programming Systems, Languages, and Applications (OOPSLA), 2024. doi:10.1145/3649835.

[9] A. Ferraiuolo, A. Baumann, C. Hawblitzel, B. Parno, Komodo: Using verification to disentangle secure-enclave hardware from software, in: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2017. doi:10.1145/3132747.3132782.

[10] C. Hawblitzel, J. Howell, J. R. Lorch, A. Narayan, B. Parno, D. Zhang, B. Zill, Ironclad Apps: End-to-end security via automated full-system verification, in: Proceedings of the USENIX Symposium on Operating Systems Design and Implementation (OSDI), 2014.

[11] A. Lattuada, T. Hance, J. Bosamiya, M. Brun, C. Cho, H. LeBlanc, P. Srinivasan, R. Achermann, T. Chajed, C. Hawblitzel, J. Howell, J. Lorch, O. Padon, B. Parno, Verus: A practical foundation for systems verification, in: Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), 2024.

[12] A. Reitz, A. Fromherz, J. Protzenko, Starmalloc: Verifying a modern, hardened memory allocator, Proc. ACM Program. Lang. (2024). doi:10.1145/3689773.

[13] Y. Zhou, J. Bosamiya, Y. Takashima, J. Li, M. Heule, B. Parno, Mariposa: Measuring SMT instability in automated program verification, in: Proceedings of the Formal Methods in Computer-Aided Design (FMCAD), 2023.

[14] Y. Zhou, A. Shah, Z. Lin, M. Heule, B. Parno, Cazamariposas: Automated instability debugging in SMT-based program verification, in: Proceedings of the Conference of Automated Deduction (CADE), 2025. To appear.

[15] A. Shah, Adding benchmarks from Verus (a Rust verification tool), GitHub Pull Request #12, 2025. URL: https://github.com/SMT-LIB/benchmark-submission/pull/12, merged on April 17, 2025.

[16] Y. Zhou, J. Bosamiya, J. Li, M. Heule, B. Parno, Context pruning for more robust SMT-based program verification, in: Proceedings of the Formal Methods in Computer-Aided Design (FMCAD) Conference, 2024.

[17] D. Amrollahi, M. Preiner, A. Niemetz, A. Reynolds, M. Charikar, C. Tinelli, C. Barrett, Using normalization to improve SMT solver stability, 2025. URL: https://arxiv.org/abs/2410.22419. arXiv:2410.22419.

[18] T. Bordis, K. R. M. Leino, Free facts: An alternative to inefficient axioms in Dafny, in: Formal Methods: 26th International Symposium (FM), 2024. doi:10.1007/978-3-031-71162-6_8.