




TracerX: Pruning Dynamic Symbolic Execution with Deletion and Weakest Precondition Interpolation (Competition Contribution)

Arpita Dutta¹ , Rasool Maghareh² , Joxan Jaffar^{3(✉)} ,
Sangharatna Godbole⁴ , and Xiao Liang Yu⁵

National University of Singapore, Singapore, Singapore^{1,3,5}

Huawei Canada Research Centre, Toronto, Canada²,

³ National Institute of Technology Warangal, Hanamkonda, India

{arpita,joxan,xiaoly}@comp.nus.edu.sg^{1,3,5}

rasool.maghareh@huawei.com², sanghu@nitw.ac.in⁴

Abstract. Dynamic Symbolic Execution (DSE) is an important method for the testing of programs. The major advantage of DSE is its path-by-path exploration of the program execution space. However, this often leads to the path explosion problem. To address this issue, a method of abstraction learning has been used. The key step here is the computation of an interpolant to represent the learned abstraction. In Test-Comp 2024, we use two different approaches of interpolant generation viz., Deletion Interpolation and Weakest Precondition Interpolation. The former is our more stable and mature system and briefly discussed in [8]. In this paper, we present the latter approach which is the heart of TracerX. In general, the Weakest Precondition (WP) is the ideal (most general) interpolant. However, WP is intractable to compute and is exponentially disjunctive. A major challenge is to obtain a conjunctive approximation of the WP. Therefore, we generate an approximation of the WP.

Keywords: Dynamic Symbolic Execution, Interpolation, Weakest Precondition

1 Test-Generation Approach

DSE is an important method for program testing. The main challenge in symbolic execution (SE) is path explosion. The method of *abstraction learning* [10] has been used to address this by generating the interpolants to represent the learned abstraction. The core feature in abstraction learning is the subsumption of paths whose traversals are deemed to no longer be necessary due to similarity with already-traversed paths. Despite the overhead of computing interpolants, the *pruning* of the symbolic execution tree (SET) that interpolants provide often brings significant overall benefits. An *interpolant* of a program point (state) is an *abstraction* of it which ensures the safety of the subtree rooted at that state. Thus, upon encountering another state of the same program point, if the context

*J. Jaffar—Jury Member Test-Comp 2024.

of the state implies the interpolant formula, then continuing the execution from the new state will not lead to any error. Consequently, we can prune the subtree rooted in the new state [6,7].

The heart of TracerX is the use of interpolation to address the path explosion problem in DSE. The use of interpolation to address the path explosion problem in DSE was first implemented in the TRACER system [9]. While TRACER was able to perform bounded verification and testing on many examples, it could not accommodate industrial programs which often dynamically manipulate heap memory. TracerX combines the state-of-the-art DSE technology used in KLEE [5] with the pruning technology in TRACER to address this issue. We presented the software architecture of TracerX in [8]. The default interpolation algorithm used by TracerX is the Deletion Interpolation and it was first developed under TRACER [9].

Since the last Test-Comp, we have designed another interpolation algorithm i.e., *Weakest precondition (WP)* interpolation. The Deletion algorithm generates interpolant as a subset of the incoming context (which is the strongest postcondition on the path to the assume condition), while the WP algorithm generates interpolants from the weakest precondition of a path in the program. Hence, the WP interpolation algorithm provides a more general interpolant which can have a higher chance of subsuming more subtrees in SET.

The ideal (most general) interpolant is the WP of the target, which is the condition that must be satisfied in order to get the target satisfied. For example, consider the following piece of code:

```
assume (not (b1  $\wedge$   $\neg$  b2  $\wedge$   $\neg$  b3))
if (b1) x += 3 else x += 2
if (b2) x += 5 else x += 7
if (b3) x += 9 else x += 14
{x <= 24}
```

The WP before the first if-statement is:

$$b1 \longrightarrow (\neg b2 \wedge b3 \wedge x \leq 7) \vee (b2 \wedge x \leq 4)$$

$$\neg b1 \longrightarrow x < 3$$

Here, WP is expressed as a disjunction of two conditions. This means that either of the two conditions can be satisfied for the target to be reached.

Unfortunately, WP is intractable to compute, which means it is difficult or impossible to find an exact solution for it. One way to approximate WP is to use a conjunctive approximation, which involves expressing the WP as a conjunction of simpler conditions. This can help to make the WP more tractable, but it may also introduce some imprecision to the quality of interpolants (by under approximation). However, this will not effect the soundness of the tool.

1.1 TracerX-WP: Approximation of Weakest Precondition

TracerX-WP implements the algorithm which approximates the ideal WP by defining two components: *path interpolants* and *tree interpolants*. In this section, we briefly explain how these two components are computed and used to generate an approximation of the weakest precondition.

A path interpolant is a formula that represents the WP of a path. It starts from the end of the path (target formula) and works backward to the beginning of the path, using the rules of logic to compute a formula that if satisfiable then

target formulas will also be satisfiable. We consider a path to be a sequence of *assignments* and *assume* statements executed in a specific order.

An *assignment* instruction assigns a value to a variable. Interpolant of an assignment instruction is a logical formula that describes the effect of the assignment. For example, having the assignment instruction “ $x := z + 2$ ”, and a target “ $x \leq 15$ ”, the interpolant is described as $WP(inst, target) : x \leq 13$.

For an *assume* instruction (B), consider the incoming context $\{C\}$ as the precondition and $\{\omega\}$ as the target. An interpolant is a formula that represents the logical relationship between the variables in the context $\{C\}$ and the conditions in B . To find the interpolant, we compute the coarse partition (minimum number of partitions) of $\{C\}$ such that $var(C_i) * var(C_j)$ s.t. $i \neq j$ (* is intuitively the “separating conjunction” from separation logic [12]) as shown in Eq.1:

$$\boxed{\{C_1 * C_2 * C_3 * \dots * C_n\} \text{ assume}(B) \{\omega_1 * \omega_2 * \omega_3 * \dots * \omega_m\}} \quad (1)$$

We partition C_i into three groups. Constraints are replaced using the rules below:

- **Target independent:** The C_i which are separate from B and ω .
Action: Replace C_i with *true*, i.e. remove C_i .
- **Guard independent:** Consider $C_{gi} \equiv C_i$ s.t. $C_i * B$; and, $\omega_{gi} \equiv \omega_j$ s.t. $B * \omega_j$.
Action: Replace C_{gi} by ω_{gi} .
- **Remainder of the C_i :** We do not capture exact WP for this group.
 e.g. $\{z == 5\}$ assume($x > z - 2$) $\{x > 0\}$ (Here, $z > 2$ is the WP)
Action: No change to C_i , i.e. keep C_i .

A tree interpolant is a formula that corresponds to all the branches of a subtree within the SET. It is computed as the *conjunction* of the path interpolants between the root of the tree and each leaf node. Tree interpolants can be used to prove the correctness of subtrees in the SET, by showing that a certain property holds for all possible paths or branches in the subtree.

2 Software Architecture

The software architecture of TracerX-WP is presented in Fig. 1. The core feature of TracerX-WP is its interpolation engine which generalizes the context of a node. TracerX-WP works at the level of LLVM bitcode, the intermediate language of the widely used LLVM compiler infrastructure [11]. It provides an interpreter that can execute almost arbitrary code represented in LLVM IR, both concretely and symbolically. TracerX-WP has a modular and extensible architecture. It provides a variety of different search heuristics (e.g., Random and DFS) to explore the program state space.

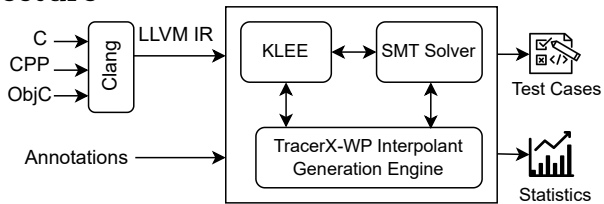


Fig. 1. TracerX-WP Framework

3 Strengths and Weaknesses

In Test-Comp 2024 [4], we participated with two different approaches to prune subtrees viz., Deletion Interpolation and WP Interpolation. We represent the former system as TracerX and the latter as TracerX-WP. TracerX secured a score of 4020 for the 11042 tasks with a CPU time of 694.44 hours and 722.22 hours of wall time. Whereas, TracerX-WP obtained a score of 1480 for 11042 tasks with equal CPU time and wall time of 472.22 hours. The memory used by TracerX and TracerX-WP are 19 TB and 10 TB. The total coverage obtained by TracerX and TracerX-WP are 402000 and 148000 for 11042 tasks respectively.

The major reason for the lower score of TracerX-WP is that the implementation of TracerX-WP is experimental. It crashed due to not supporting some expression types during interpolant computation. Also, in TracerX-WP, test cases with ‘.ktest’ extension are converted into ‘.xml’ format after the symbolic execution engine has finished the exploration while TracerX generates the tests during the exploration. This resulted in the unavailability of test cases for the programs with timeout status in the coverage computation. Moreover, the configuration we used in the ‘BenchExec’ tool-info for TracerX-WP missed the support for 64-bit architecture. As a result, TracerX-WP was not able to run the tests in some categories like **ReachSafety-Hardware**, and **SoftwareSystems-BusyBox-MemSafety**. The fix for the above mentioned issues is conceptually straight forward but it requires substantial amount of work. Since, we need to modify the data structures used in our system. In subsequent versions, we will come-up more stable system with all fixes and additional features.

In a comparison of TracerX with Symbiotic and Fizzer which won the bronze for the third place in *Cover-Error* and *Cover-Branches* tracks respectively, TracerX has almost equal scores in 13 out of 16 (with at most difference of 3 tasks) and 15 out of 23 categories. TracerX has better results than Fizzer in some categories like **ReachSafety-BitVectors**, **ReachSafety-Hardware**, and **ReachSafety-Combinations**. These observations show the potential of TracerX approach and we hope to get higher scores in the future Test-Comp competitions.

4 Setup and Configuration

The steps to configure and running of TracerX are similar to KLEE [5] with some extra command-line arguments. The argument `-solver-backend=z3` should be provided to run TracerX with Deletion Interpolation. Along with `-wp-interpolant` option is required to invoke WP Interpolation. For detailed information, please see the integrated `--help` option.

5 Software Project and Contributors

Information about TracerX with self-contained binary is publicly available at <https://tracer-x.github.io/>. Also, the source code can be accessed from [GitHub](#). The authors of this paper and other colleagues have contributed to and developed TracerX at [NUS, Singapore](#). Authors of this paper acknowledge the direct and indirect support of their students, former researchers, and colleagues.

6 Data-Availability Statement

The binary artifact of TracerX with Deletion Interpolation and Weakest Precondition Interpolation used in Test-Comp 2024 are publicly available at Zenodo [2] and [3] respectively. Also, Test-Comp 2024 [1] provides all the necessary scripts, benchmarks, and tool binaries to reproduce the competition's results.

7 Funding Statement

This research project is partially supported by grant MOE-T2EP20220-0012.

References

1. Test-comp 2024, <https://test-comp.sosy-lab.org/2024/>
2. TracerX with Deletion Interpolation, <https://doi.org/10.5281/zenodo.10200610>
3. TracerX with WP Interpolation, <https://doi.org/10.5281/zenodo.10202605>
4. Beyer, D.: Automatic testing of C programs: Test-Comp 2024. Springer (2024)
5. Cadar, C., Dunbar, D., Engler, D.R., et al.: KLEE: unassisted and automatic generation of high-coverage tests for complex systems programs. In: 8th USENIX Symposium on Operating Systems Design and Implementation, OSDI. pp. 209–224 (2008)
6. Godbole, S., Jaffar, J., Maghareh, R., Dutta, A.: Toward optimal MC/DC test case generation. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis. pp. 505–516 (2021)
7. Jaffar, J., Godbole, S., Maghareh, R.: Optimal MC/DC test case generation. In: 2019 IEEE/ACM 41st International Conference on Software Engineering: Companion Proceedings (ICSE-Companion). pp. 288–289. IEEE (2019)
8. Jaffar, J., Maghareh, R., Godbole, S., Ha, X.L.: TracerX: Dynamic symbolic execution with interpolation (competition contribution). In: Fundamental Approaches to Software Engineering (FASE). vol. 12076, p. 530. Springer (2020)
9. Jaffar, J., Murali, V., Navas, J.A., Santosa, A.E.: TRACER: a symbolic execution tool for verification. In: 24th International Conference on Computer Aided Verification (CAV). pp. 758–766. Springer (2012)
10. Jaffar, J., Santosa, A.E., Voicu, R.: An interpolation method for CLP traversal. In: 15th International Conference on Principles and Practice of Constraint Programming (CP). pp. 454–469. Springer (2009)
11. Lattner, C., Adve, V.: LLVM: A compilation framework for lifelong program analysis & transformation. In: International symposium on code generation and optimization, 2004. CGO 2004. pp. 75–86. IEEE (2004)
12. O'Hearn, P., Reynolds, J., Yang, H.: Local reasoning about programs that alter data structures. In: Computer Science Logic: 15th International Workshop, CSL 2001 10th Annual Conference of the EACSL Paris. pp. 1–19. Springer (2001)

Open Access This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

