

# Fault Localization using Interpolation

Chanseok Oh<sup>1</sup>, Martin Schäf<sup>2</sup>, Daniel Schwartz-Narbonne<sup>1</sup>, and Thomas Wies<sup>1</sup>

<sup>1</sup> New York University

<sup>2</sup> SRI International

An integral part of all debugging activities is the task of fault localization. Most existing fault localization techniques rely on the availability of high quality test suites because they work by comparing failing and passing runs to identify the cause of an error. This limits their applicability. One alternative are techniques that statically analyze an error trace of the program without relying on additional passing runs to compare against. Particularly promising are novel proof-based approaches that leverage the advances in automated theorem proving. These approaches encode error paths into *error trace formulas*. An error trace formula is an unsatisfiable logical formula. A refutation proof of this formula captures the reason why the execution of the error trace fails. By applying an automated theorem prover to obtain such proofs, the relevant portions of the trace can be automatically identified without relying on the availability of test suites. For example, in our previous work, we have used Craig interpolation to extract formulas from the obtained refutation proof. These so-called *error invariants* provide an explanation of the error for a given position in the trace and can be used for more effective static slicing.

In particular, we encode the error trace into a logical formula whose satisfying assignment exactly corresponds to the states that are observed during the concrete execution of the trace. The encoding uses standard symbolic execution techniques. Namely, we first convert the trace into static single-assignment form, then translate each individual statement into a logical constraint, and finally conjoin all the constraints to a single formula. We call the resulting conjunction the *error trace formula* of the trace. By conjoining the error trace formula with the property that is violated at the end of the trace, we obtain an unsatisfiable formula. Since this property is violated at the end of the execution of the error trace, this formula is contradictory (i.e. UNSAT). We then use an interpolating SMT solver to generate an error invariant for each position in the trace from a refutation proof of the unsatisfiable formula. The computed error invariants have two important properties: (1) they are satisfied by the concrete state that is observed at the respective position in the trace; and (2) any execution of the remainder of the trace starting from that position with a state that satisfies the error invariant will still violate the property at the end of the trace.

The generated error invariants are then propagated along the trace to identify maximal substraces such that the same error invariant holds at both the beginning and the end of the subtrace. From the two properties of error invariants, it follows that the statements between these two positions are irrelevant for understanding the bug and, hence, can be sliced from the trace. We proceed like this for the entire error trace to produce the abstract slice.

While proof-based fault localization techniques are a promising alternative to dynamic techniques, they have practical limitations that have not been addressed in previous work. Precisely capturing the behavior of an erroneous program requires modeling such features as dynamic memory allocation, function calls, inputs, and loops, over traces that can extend to large numbers of instructions. Although mathematical techniques exist to handle these complexities, their implementation in current off-the-shelf theorem provers is often far from robust. For example, heap accesses are traditionally handled using array theories. Unfortunately, most provers lack interpolation procedures for the theory of arrays, making it difficult to automatically infer invariants about heap allocated objects.

To address these shortcomings of purely static approaches, we propose a novel fault localization technique that combines dynamic and static analysis of the error trace. In particular, we note that given an executable error trace, we can determine the concrete value of every variable, and every memory address, using a standard debugger. This information can then be used to dramatically simplify the encoding of the error trace for static analysis. Instead of relying on complex mathematical theories to reason about pointer aliasing, we can directly determine which memory location a pointer references using the debugger. If a trace contains too many statements to analyze using a theorem prover, we can split the trace into subtraces, and use the program states observed during the concrete execution to seed the analysis of the individual subtraces.

We have conducted a case study where we applied our new algorithm to error traces generated from faulty versions of UNIX utils such as `gzip` and `sed`. Our case study indicates that our concolic fault localization scales to real-world error traces and generates useful error diagnoses.