Semantic Fault Localization and Suspiciousness Ranking

Maria Christakis[1], Matthias Heizmann[2] , Muhammad Numair Mansur[1] , Christian Schilling[3], Valentin Wüstholz[4]

[1] MPI-SWS, [2]University of Freiburg, [3]IST Austria, [4]ConsenSys Diligence

Motivation

Program analyzers are increasingly applied to detect errors in real-world software. When detecting an error, static (or dynamic) analyzers often present the user with an error trace (or a failing test case), which shows how an assertion can be violated. Specifically, an error trace refers to a sequence of statements through the program that leads to the error. The user then needs to process the error trace, which is often long for large programs, in order to localize the problem to a manageable number of statements and identify its actual cause. Despite the effectiveness of static program analyzers in detecting errors and generating error traces, users still spend a significant amount of time on debugging.

Trace-Aberrant Statements

Definition

Let τ be a feasible error trace and s an assignment statement of the form $v := \star$ or $v := e$ along τ. Statement s is ***trace-aberrant*** iff there exists an expression e' that may be assigned to variable v such that the trace verifies. We call e' a ***local fix***.

Theorem

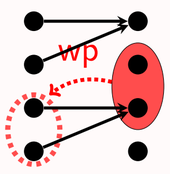
Let $s_1 \dots s_n$ be a feasible error trace and s_i an assignment statement of the form $v := \star$ or $v := e$ along τ. Statement s is *trace-aberrant* iff the following Hoare triple is not valid.

$$\{\mathsf{sp}(true, s_1 \dots s_{i-1})\} \texttt{ v := } \star \texttt{ } \{\mathsf{pre}(true, s_{i+1} \dots s_n)\}$$

Reminder

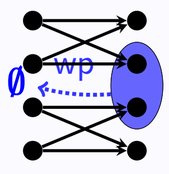
Assignment Statement

$\mathsf{sp}(\varphi, \texttt{v := e}) := \exists v.\ \varphi[v \mapsto v] \wedge v = e[v \mapsto v]$
$\mathsf{wp}(\varphi, \texttt{v := e}) := \varphi[v \mapsto e]$
$\mathsf{pre}(\varphi, \texttt{v := e}) := \varphi[v \mapsto e]$



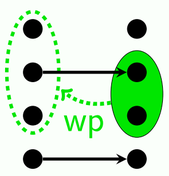
Havoc Statement

$\mathsf{sp}(\varphi, \texttt{v := } \star) := \exists v.\ \varphi$
$\mathsf{wp}(\varphi, \texttt{v := } \star) := \forall v.\ \varphi$
$\mathsf{pre}(\varphi, \texttt{v := } \star) := \exists v.\ \varphi$

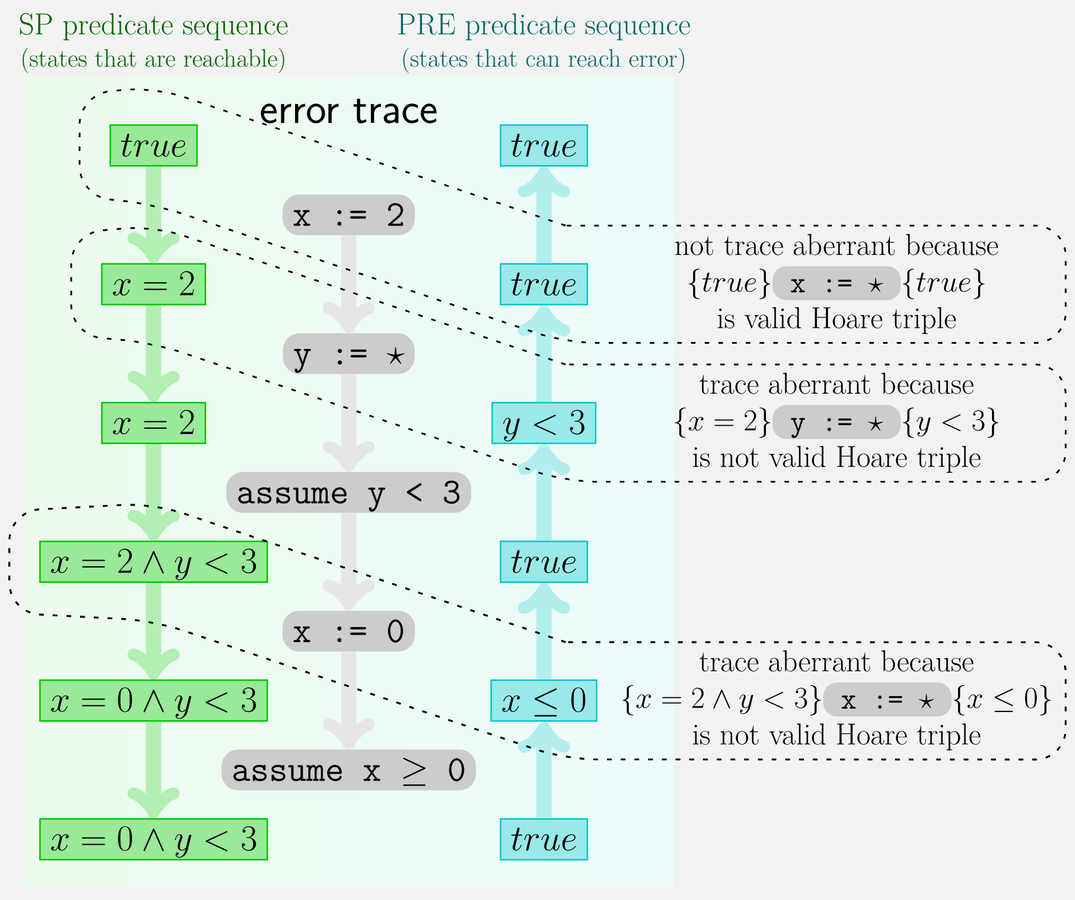


Assume Statement

$\mathsf{sp}(\varphi, \texttt{assume } \psi) := \varphi \wedge \psi$
$\mathsf{wp}(\varphi, \texttt{assume } \psi) := \psi \rightarrow \varphi$
$\mathsf{pre}(\varphi, \texttt{assume } \psi) := \varphi \wedge \psi$



Example



Program-Aberrant Statements

Definition

Let τ be a feasible error trace and s an assignment statement of the form $v := \star$ or $v := e$ along τ. Statement s is ***program-aberrant*** iff there exists an expression e' that may be assigned to variable v such that all traces through s verify.

Deciding program aberrance is difficult because there are infinitely many expressions that can fix the assignment statement and there are infinitely many traces.

Definition

Let $s_1 \dots s_n$ be a feasible error trace and s_i an assignment statement of the form $v := \star$ or $v := e$ along τ. We call a (possibly nondeterministic) local fix e' ***maximally permissive*** if e' evaluated in $\mathsf{sp}(true, s_1 \dots s_{i-1})$ coincides with $\mathsf{wp}(true, s_{i+1} \dots s_n)$ projected to the variable v. We call statement s ***must-program-aberrant*** iff all traces verify after replacing s with an assignment of a maximally permissive local fix.

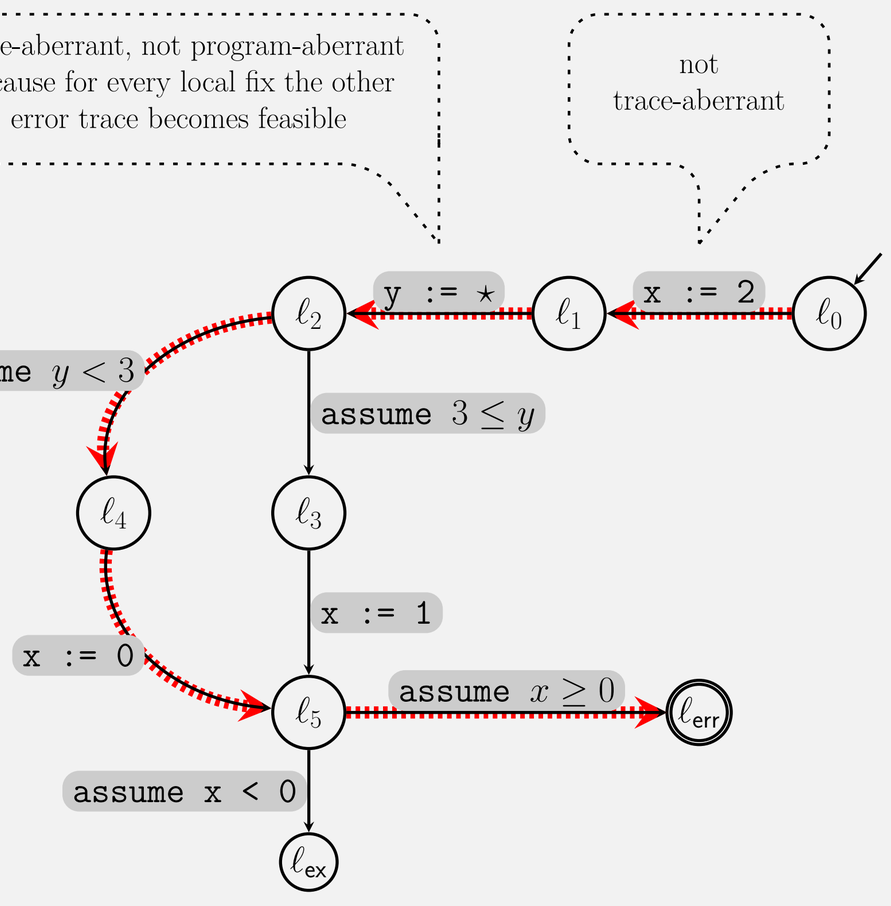
Example

```
1  x := 2;
2  y := ⋆;
3  if (⋆) {
4    assume y < 3;
5    x := 0;
6  } else {
7    assume 3 ≤ y;
8    x := 1;
9  }
10 assert 0 < x;
```

trace-aberrant, not program-aberrant because for every local fix the other error trace becomes feasible

not trace-aberrant

trace-aberrant and must-program-aberrant, the expression that nondeterministically selects some negative value is a maximally permissive local fix



Semantic Suspiciousness Ranking

Idea

Typically, developers do not intentionally write dead code. Give every statement a score that is inversely proportional to how much code would become unreachable if we applied a local fix to the statement.

Algorithm

1. Apply maximally permissive fix to assignment statement
2. Iteratively instrument program and use software verifier to check which lines are still reachable

Paper

Available online