Complete Shadow Symbolic Execution with Java PathFinder

Authors: Yannic Noller, Minxing Tang, Hoang Lam Nguyen, Timo Kehrer (Germany)

Keywords: *Regression Testing; Symbolic Execution; Symbolic PathFinder*

ACM SIGSOFT October 2019

https://dl.acm.org/doi/10.1145/3364452.33644558

Reading 13.06.2023. Zakharov Vladimir

Meta Data

Conference:	ACM SIGSOFT (Special Interest Group on Software Engineering)
Track:	Software and its engineering
Year:	2019
Number of Authors:	4
Citations:	3
Pages (PDF):	5
Figures:	6
References:	16
Formals:	Absent

What is the study about?

- How to generate tests based on the code diff
- Shadow Symbolic Execution (SSE) -> SSE+

```
1    int foo(int x){
2      int y;
3      if(x < 0){
4-       y = -x;
4+       y = x * x;
5      } else{
6        y = 2 * x;
7      }
8+     y = y+1;
9      if(y > 1){
10       return 0;
11     } else {
12       if(y == 1)
13         assert(false);
14     }
15     return 1;
16   }
```

SSE	SSE+
Detects that error with the path **x = -1** was fixed	Detects that error with the path **x = -1** was fixed
Detects that error is fixed by the change, but misses the new error produced with **x = 0**	Found one more bug with **x = 0**

What is the study about?

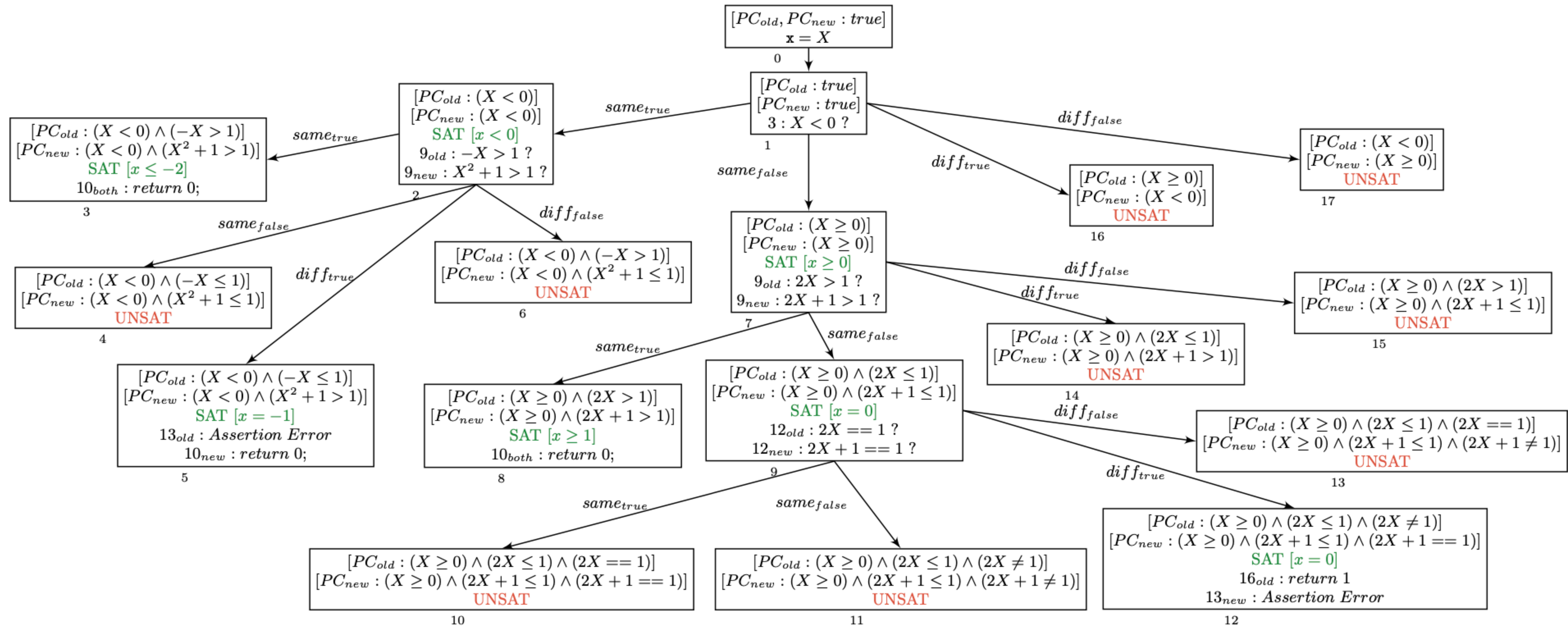

Table of Content

1. Introduction
2. Background and motivation
   2.1. Symbolic Execution
   2.2. Shadow Symbolic Execution
   2.3. Need for Further Research
3. Approach
4. Evaluation
   4.1. Results and Analysis
5. Conclusion and Discussion

Feedback

Problem Statement

Deeper divergences might be missed in the BSE phase: The BSE phase of [13] aims to find additional test inputs that trigger divergent behavior by exploring the execution tree of the new version starting from each divergence point found in the concolic phase. This implies that each BSE run inherits the path condition prefix from the initial input from the start to the divergence point. For example in Listing 1, for the input x=-1, which fully covers the changed statements, [13] would only generate the test case representing the fixed path and it would miss the path with the new introduced assertion error. The reason for that is that the collected path condition prefix limits the exploration space of BSE. The coll
order to get to
BSE needs to
tion: $(X^2 + 1$
contradicts wi
2 in Figure 1).
introduced ass

The initial input has to cover potential divergence points: Since the concolic phase of [13] only searches for divergences along the path of a concrete input that exercises the patch, divergences along alternative paths will be missed if there is no satisfiable divergence at the branching point. This means that if the concrete input does cover the changed statements, but does not cover the potential divergence point, then [13] is not able to find the *diff* path. In the program in Listing 2 a potential divergence can happen only in line 4 because it is the only condition that depends on a changed variable. The condition in line 3 is never a divergence point because the variables in the condition are not affected by the patch. To discover the divergence, the initial input has to actually reach line 4, meaning that any initial input x and y with $x + y \neq 5$ would miss the divergence because concrete execution would follow only the *false* branch of the conditional statement in line 3 and discard the other paths.

```
1 int bar(int x, int y){
2   int z = change(x,y);
3   if((x+y) == 5){
4     if(z == -100)
5       assert(false);
6   }
7   return 0;
8 }
```

Listing 2: Limitation example.

Feedback

Innovation

In this work, we presented an approach to generate test inputs exposing the divergences between two program versions. We provided a complementary exploration strategy to the existing technique by [13] and implemented SHADOW$_{JPF+}$ as an extension of the SHADOW$_{JPF}$ tool. In order to evaluate the effectiveness of our approach, we performed experiments on 79 generated mutants and compared the results with SHADOW$_{JPF}$. Additionally, we ap-

Feedback

Contribution

The main contributions of this work are:

1. The combination of complete symbolic execution with the idea of four-way forking, as a technique to generate regression tests that expose changed program behavior.
2. The tool SHADOW$_{\text{JPF+}}$ as an extension of the SHADOW$_{\text{JPF}}$.
3. The application of SHADOW$_{\text{JPF+}}$ on various examples, including a patch for the Joda-Time library in order to evaluate its test case generation capabilities. Furthermore, our approach is compared to SHADOW$_{\text{JPF}}$ to assess the effectiveness of our improved search exploration strategy.

Feedback

Logical Correctness

Subjects: We selected the following software artifacts as our experimental subjects from the official SPF repository[1] (with the corresponding LOC): Rational.abs (30), Rational.gcd (40), Rational.simplify (51), WBS.update (234) and WBS.launch (242) and generated in total 79 mutants with the Major mutation framework [8] (similar as [12]) with the following change types: Relational Operator Replacement (ROR), Operator Replacement Unary (ORU), Arithmetic Operator Replacement (AOR) and Statement Deletion (STD). Since Major only generated single mutants

Feedback

Proof of Statements

404: Not found

No proofs

No algorithms (at least in algorithmic programming language)

Weak evaluation

Feedback

Readability

- Structure (sequence)
- Well-defined sections
- Clear and understandable RQ and Contribution
- Formatting
- Code samples
- Tables

Conclusion

Advantages	Disadvantages
Readability	Innovation?
Contribution	No proofs, weak evaluation
Existing Tool	Not complete (requires reading of the references, like SSE)

Accept (good for a workshop)