Guided Path Exploration for Regression Test Generation

Kunal Taneja ktaneja@ncsu.edu
Tao Xie xie@csc.ncsu.edu
Nikolai Tillmann nikolait@microsoft.com
Jonathan de Halleux jhalleux@microsoft.com
Wolfram Schulte schulte@microsoft.com

NC STATE UNIVERSITY Computer Science
Microsoft Research

Problem:

✓ Regression test generation aims at generating a test suite that can detect behavioral differences between two versions of a program

✓ Regression test generation can be automated by using Dynamic Symbolic Execution (DSE)

  ✓ It is often expensive for DSE to explore paths in the program to achieve high structural coverage

Solution: *Guided Path Exploration* specifically for finding behavioral differences

  ✓ Pruning paths that cannot help in finding behavioral differences

Approach

✓Adopt the PIE model [1] for finding irrelevant paths that cannot help in finding behavioral differences

  ✓ **PIE model**: A fault can be detected by a test if a faulty statement is executed (**E**), the execution infects the state (**I**), and the infected state propagates to an observable output (**P**)

  ✓ Prune paths that cannot help in satisfying any of **P**, **I**, or **E** condition

Pruning of Branching Nodes

✓ DSE's path exploration realized by flipping branching nodes

✓ Avoid from flipping branches of three categories:

Category E: branching nodes whose the other unexplored branch cannot lead to any changed region

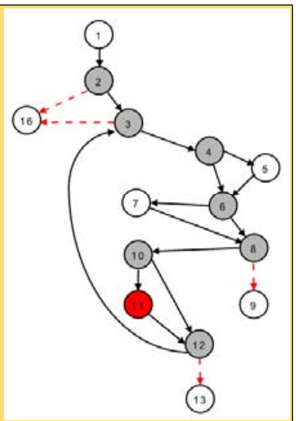
Category I: If a changed region is executed but the program state is not infected, all the branches nodes after the changed region in the current execution path

Category P: Let χ be the statement at which change propagation stops. All the branches nodes after Statement χ in the current execution path

Example

```
    static public int  testMe(int x, int[] y)
    {
1       int j = 1;
2       if (x == 90){
3           for (int i = 0; i < y.Length; i++){
4               if (y[i] == 15)
5                   x++;
6               if (y[i] == 16)
7                   j = 2;
8               if (y[i] == 25)
9                   return x;
10              if (x == 110)
11                  x = j + 2;      //x = 2*j+1
12              if (x > 110)
13                  return x;
14          }
15      }
16      return x;
    }
```

Example Program



Control Flow Graph

Category E: Red dotted branches as after taking these branches, program execution cannot lead to Statement 11.

Category I : If program state not infected after execution of Statement 11 (such as for inputs x: 90, y [20]: {15, 15, 15, ….., 15}), the branches in the execution trace after the execution of Statement 11 (Branch <12,3>).

Category P : The branches in the execution trace after the execution of propagation stopping statement

Program Instrumentation for State Checking

```
   public boolean testMe(int x, int[] y)
   {
       ....
10     if (x == 110) {
11         x = 2 * j + 1;
12         PexStore.ValueForValidation("uniqueName", x);
13     }
       ....
   }
```

Instrumented new version of the program

✓Program instrumented for both versions

✓DSE performed on the modified version

✓As soon as a test is generated, it is executed on the instrumented original version to check whether program state is infected

Preliminary Evaluation

✓Prototype parts of our approach by manually inserting probes in program code to guide Pex [2] to avoid exploring branches in Categories **E** and **I** in the program code

✓Use the tcas program (converted to C#) from the Software Infrastructure Repository (SIR) [3] as our subject

✓Seed the first 11 faults available at SIR one by one to generate 11 new versions of tcas

✓Compare the number of runs of DSE required by the default search strategy in Pex with the number of runs required by our approach for **E**

✓Compare the number of runs required by the default search strategy in Pex with the number of runs required by our approach to achieve **I**

Results

RQ1. On average, our approach requires 12.9% fewer runs (maximum 25%) to achieve **E**

RQ2. On average, our approach requires 11.8% fewer runs (maximum 31.2%) to achieve **I**

Details of results and versions of tcas available at project web page [4]

References

1. J. Voas. PIE: A dynamic failure-based technique. *IEEE Transactions on Software Engineering*, 18(8):717–727, 1992.

2. N. Tillmann and J. de Halleux. Pex-white box test generation for .NET. In *Proc. International Conference on Tests and Proofs*, pages 134–153, 2008.

3. H. Do, S. Elbaum and G. Rothermel, "Supporting Controlled Experimentation with Testing Techniques: An Infrastructure and its Potential Impact", *Empirical Software Engineering: An International Journal*. 10(4):405-435, 2005.

4. Project Web Page: https://sites.google.com/site/asergrp/projects/regtestgen

Automated Software Engineering Research Group @NCSU

Pex

Supported in part by NSF grant CCF-0725190 and ARO grant W911NF-08-1-0443