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 (O).
✓ 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 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

```java
static public int testMe(int x, int[] y)
{
    int j = 1;
    if (x == 90)
    {
        for (int i = 0; i < y.length; i++)
        {
            if (y[i] == 15)
            {
                x++; //x++;
            }
            if (y[i] == 16)
            {
                j = 2;
            }
            if (y[i] == 25)
            {
                return x;
            }
            if (x == 110)
            {
                //x = j + 1;
            } //x = j + 1
            if (x > 110)
            {
                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

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

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
4. Project Web Page: [https://sites.google.com/site/asergrp/projects/regtestgen](https://sites.google.com/site/asergrp/projects/regtestgen)

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