DiffGen: Automated Regression Unit-Test Generation

Kunal Taneja ktaneja@ncsu.edu

Tao Xie xie@csc.ncsu.edu

Problem

Problem:

✓ Software programs continue to evolve throughout their lifetime

✓ Existing test suite is often insufficient to cover changed behavior to guard against unintended changes

Solution:

An approach and a tool, **DiffGen**, for generating regression unit tests

✓ Given two versions, it instruments the code to insert new branches

✓ If these branches are covered, behavioral differences are exposed

✓ **DiffGen** uses a structural test generation tool to generate tests for covering these branches

Approach

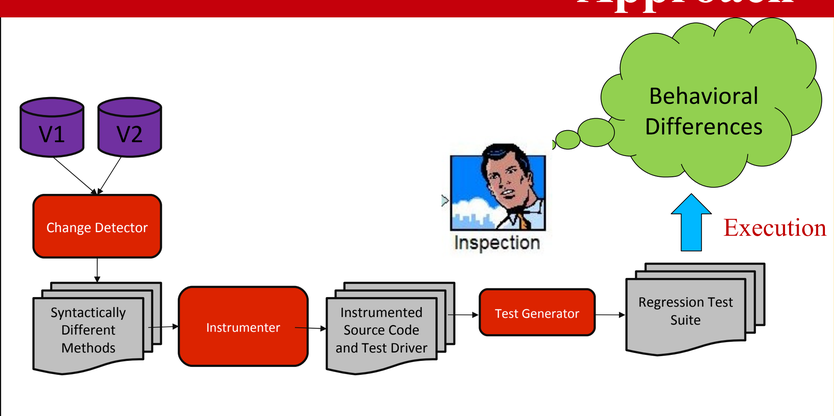

Figure 1: High-Level Overview of **DiffGen**

DiffGen includes four components:

✓**Change Detector** detects textually different methods

✓**Instrumenter** instruments the source code and synthesizes a test driver

✓**Test Generator** generates tests for the synthesized test driver. When executed, the generated tests expose behavioural differences

Example

```
class BSTOld implements set{
        Node node;
        int size;
        static class Node{
            MyInput value;
            Node left;
            Node right;
        }
        public BSTOld() {.....}

        public boolean insert(MyInput m){
            Node t = root;
            while(true){
                if(t.compareTo(m.key)>0)  // if(t.compareTo(m.key)>=0
                    if(t.right == null){
                            t.right = new Node(m);
                            break;
                    }
                    else
                            t = t.right;
                else
                    if(t.left == null){
                        t.left = new Node(m);
                        break;
                    }
                    else
                                t = t.left;
                    .....
            }
        }
        public void remove(MyInput m){.....}
        public void contains(MyInput m){.....}
        .....
}
```

Figure 2: The BSTOld class as in an old version. In a new version, The highlighted line is changed to the one shown in the comment.

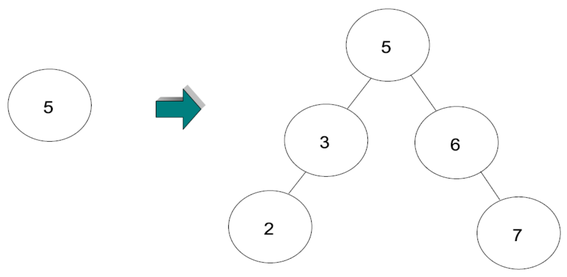

Figure 3: The BST object states (for both versions) before and after nodes with Keys 3, 6, 2, and 7 are inserted, respectively.

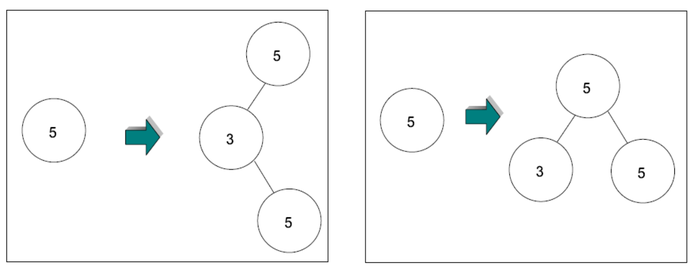

Figure 4: The BST object states before and after nodes with Keys 3 and 5 are inserted, respectively for (a) the old version of class BST and (b) the new version of class BST.

```
public class BSTJUFDriver{
  public void compareInsert(BSTOld oldBST,
    MyInput input){
    BST bstNew = new BST();
    bstNew = copyObject(oldBST);
    boolean b1 = bstOld.insert(input);
    boolean b2 = bstNew.insert(input);
    if(b1 != b2)
            Assert(false);
    if(bstOld.size != bstNew.size)
            Assert(false);
    if(!bstOld.root.equals(bstNew.root))
            Assert(false);
  }
}
```

Branches to be covered to expose behavioural differences

Figure 5: Test driver synthesized for JUnit factory

Evaluation

Research Question:

Can the regression test suite generated by **DiffGen** effectively detect regression faults that cannot be detected by previous state-of-the-art approach **SeparateGen** [1], which generates test suites separately for old and new versions?

Experiments:

✓ Generate mutants for various subjects

✓ Generate tests for each version of mutant and original version of class under test separately (**SeparateGen**)

✓ Generate tests using **DiffGen**

✓ For each pair containing a mutant and original version, compare the detection of behavioral differences using test suites generated by **SeparateGen** and **DiffGen**

IF1: Improvement Factor of **DiffGen** over **SeparateGen:** IF1 = DG-killed/ JUF-unkilled

IF2: Improvement Factor of **DiffGen** over **SeparateGen** excluding mutants with same behavior: IF2 = DG-killed/ (JUF-unkilled – same-behavior)

Table 1: Experimental subjects

class	meths	public	ncnb	Cov
IntStack (IS)	5	5	44	100%
UBStack (UBS)	11	11	106	100%
ShoppingCart (SC)	9	8	70	100%
BankAccount (BA)	7	7	34	100%
BinSearchTree (BST)	13	8	246	100%
BinomialHeap (BH)	22	17	535	87%
DisjSet (DS)	10	7	166	100%
FibonacciHeap (FH)	24	14	468	98%

Table 2: Experimental results

class	#Mutants	#JUF UnKilled	DG Killed	Same Behavior	IF1 %	IF2 %
IS	85	21	0	21	0	-
UBS	187	15	6	7	40	75
SC	18	7	3	4	42.8	100
BA	35	6	0	6	0	-
BST	125	13	4	4	30.8	44.4
BH	281	39	8	19	20.5	40
DS	385	97	15	33	15.5	23.4
FH	339	53	5	43	9.4	50

Experiments on Larger Subject Programs

Experiments:

✓ Subjects and faults taken from Subject Infrastructure Repository [2]

✓ Seeded all available faults for JTopas one at a time

✓ Compared **SeparateGen** and **DiffGen** to detect the seeded faults

F: Number of faults

U: Number of Faulty versions undetected using **SeparateGen**

D: Number of faulty versions detected by **DiffGen** among the ones not detected by **SeparateGen**

Table 3: Experimental results on larger subject programs

Ver	class	LOC	F	U	D
v1	ExtIOException	78	3	0	-
v1	AbstractTokenizer	1672	3	1	1
v1	Token	159	1	0	-
v1	Tokenizer	287	1	0	-
v1	ExtIndexOutOfBoundsException	67	2	0	-
v2	ExtIOException	89	2	0	-
v2	ThrowableMessageFormatter	137	2	0	-
v2	AbstractTokenizer	2966	4	2	2
v2	Token	447	4	0	-
v3	EnvironmentProvider	240	3	1	0
v3	PluginTokenizer	407	1	0	-
v3	StandardTokenizer	1992	8	2	2
v3	StandardTokenizerProperties	2736	4	1	0
Total	13 classes		38	7	5

References

[1] R. B. Evans and A. Savoia. Differential testing: a new approach to change detection. In *Proc. ESEC/FSE, pages 549–552, 2007*

[2] H. Do, S. G. 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.

Supported in part by NSF grant CCF-0725190