Avoid known bugs generated by TSTL Random Tester

Swathi Sri Vishnu Priya Rayala (932-696-872)

The main idea of the algorithm for QuickCheck [2] is to generalize each bug as it is found, to a 'bug pattern', then adapt test case generation so that test cases matching an existing bug pattern are never generated again. The main contributions of [2] are:

• A fully automatic method to avoid provoking already known bugs at test case generation time, and to avoid bug slippage when failed tests are minimized.

• Experimental results showing that this method can, in some cases, reduce the number of tests needed to find a set of bugs quite dramatically, and even find new bugs in well-studied software.

Current Work:

So, I want to implement the same kind of algorithm in TSTL. A screenshot of the implementation where the sequences are captured in the list *actionsList* is shown below –

```
def prettyListTest(test, columns=30):
        i = 0
        actionsList = []
        for (s,_,_) in test:
            parameters = 0
            steps = "# STEP " + str(i)
            #print "each", sut.prettyName(s)
            if "." in sut.prettyName(s):
                statement = sut.prettyName(s).split(".",1)[1]
                #print "statement", statement
                methodName = find_between("."+statement, ".", "(")
                #print "methodName", methodName
                if find_between(statement, "(", ")") != "":
                        parameters = find_between(statement, "(", ")").count(",") + 1
                        #print "count", parameters
                #print "parameter", parameters
            else:
                methodName = "A"
            #actionsList.append(sut.prettyName(s).ljust(columns - len(steps),' '))
            actionsList.append(methodName + "-" + str(parameters))
            i += 1
        return actionsList
```

Whenever a new bug is encountered, the sequence of its operations is saved in a list. So for the next time when a new test is generated, its sequence of operations are verified with the already existing list of test sequences captured. In this way, only unique tests are generated.

I'm successful in the execution of this. When I ran tester1.py and tester2.py for 300 seconds, below are the counts for bugs found and coverage improvement –

Tester1.py results

27 bugs found (not necessarily unique)

Branch count is 225

Statement count is 164

Tester2.py results

53 unique bugs found

Branch count is 231

Statement count is 166

Thus there is a significant improvement with the implement of new algorithm in TSTL.

For example (as shown below), I have 2 test cases T1 and T2, which are identical but differ in variable names only. Now, my implementation identifies them as same test and ignores generating such tests if one test of that kind is already saved.

T1

```
val0 = 18                    # STEP 0
avl1 = avl.AVLTree()         # STEP 1
avl1.delete(val0)            # STEP 2
```

T2

```
val0 = 18                    # STEP 0
avl0 = avl.AVLTree()         # STEP 1
avl0.delete(val0)            # STEP 2
```

Future Work:

My next part is to capture the sequence of operations for a failed test using the method ***actionClass()*** and implement with TSTL. Currently, my algorithm saves the sequence of failed test and compares it with the newly generated test sequence. If the new test sequence matches the existing test sequence in the order of function calls and assignment operations, then it is declared as "same test" otherwise it is a "new failed test case". So in the later part of my execution, the same thing is implemented using

actionClass() method to generate the sequence of operations in a test. This would reduce the overhead in computing the sequences.

References

[1] Algorithms to identify failure pattern, Bhuwan Krishna Som, Poudel
Url: http://www.diva-portal.org/smash/get/diva2:655645/FULLTEXT01.pdf
[2] Find more bugs with QuickCheck, John Hughes, Ulf Norell, Nicholas Smallbone
Url: http://publications.lib.chalmers.se/records/fulltext/232554/local_232554.pdf
[3] TSTL: A Language and Tool for Testing, Alex Groce, Jervis Pinto, Pooria Azimi, Pranjal Mittal
Url: http://www.cs.cmu.edu/~agroce/issta15.pdf