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 is shown below -

```
while time.time()-start < TIME_BUDGET:
    sut.restart()
    for s in xrange(0,DEPTH):
        action = sut.randomEnabled(rgen)
        ok = sut.safely(action)
        if (not ok):
                S = sut.reduce(sut.test(), lambda x: sut.fails(x) or sut.failsCheck(x))
                if S not in testsCovered:
                        testsCovered.append(S)
                        print "Reduced Test"
                        sut.prettyPrintTest(S)
                else:
                        print "Same test"
                        sut.prettyPrintTest(S)
                        break;
```

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

So, for our algorithm to be successfully implemented, it is important to save the sequence of operations that caused the failure. But, later after implementing the algorithm, I faced a problem with TSTL. I noticed that **TSTL doesn't have equivalence check for the variable names**.

For example (as shown below), I have 2 test cases T1 and T2, which are identical but differ in variable names only.

T1

val0 = 18 # STEP 0

avl1 = avl.AVLTree() # STEP 1

avl1.delete(val0) # STEP 2

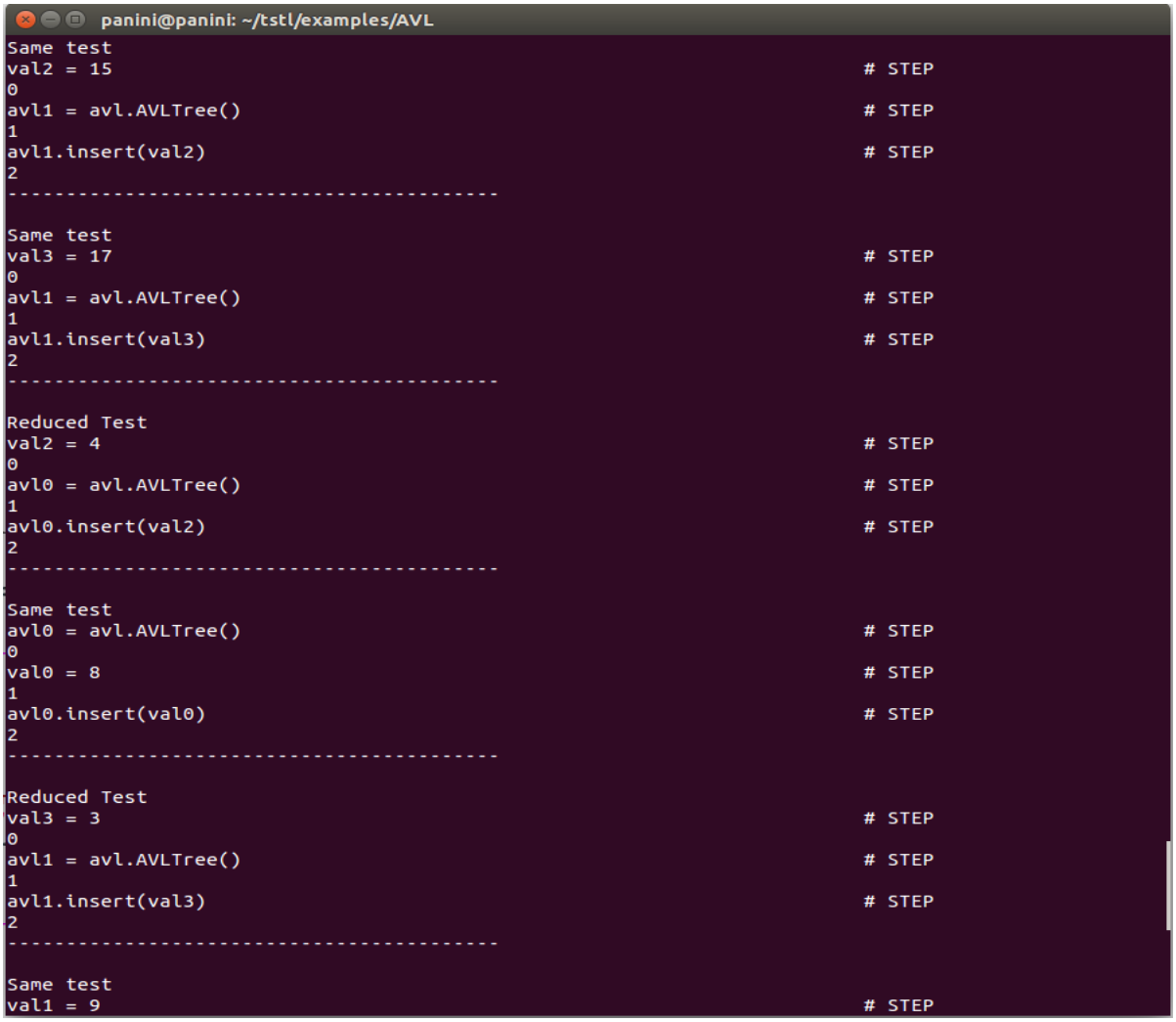
T2

val0 = 18 # STEP 0

avl0 = avl.AVLTree() # STEP 1

avl0.delete(val0) # STEP 2

Below is the sample output screen of my execution



According to the algorithm defined to be implemented, both these tests are same. But TSTL shows these as 2 different tests as they differ in the variable names. Also for now, there is no way to capture the sequence of operations in a particular test case in TSTL.

If we capture the sequence of operations that caused a failed test case, further test cases can be ignored. This would help us in giving more time to test for new sequence of operations which causes a failure.

Future Work:

My next part is to find ways to capture the sequence of operations for a failed test and implement with TSTL. Currently, my algorithm saves the entire failed test and compares it with the newly generated test. If the new test matches the existing test in all aspects along with the variable names, then it is declared as "same test" otherwise it is a "new failed test case". So in the later part of my execution, if the sequence of operations of new test matches with any of the existing sequences, I'll ignore such tests otherwise add it as a new sequence for failed cases.

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