Part 2 for CS569 project

Kun Chen

chenk4@oregonstate.edu

Algorithm Description:

In github [3], Prof. Alex have written an algorithm which utilized random and BFS algorithm. The concept of this algorithm is very concise and but has low efficiency. It created random state and shuffle the execution order of actions, and the do the BFS search.

When size of layers is very large, and if we want to go deeper in depth, it is rather time consuming. So I make modification of this algorithm: in every layer, we mainly go to visit the largest sequence of children which has the same parent. By this way, in all visited layers, we greedily have the largest sequence of states which come from the same ancestors. I think the bugs seems to be found more often in this large sequence than others. This idea origin from the Beam search algorithm [2], where k actions are executed rather than all enabled actions are required to be implemented.

Future Improvement:

In the present work, the the program is terminated when time runs out or all layers are visited. But we only search the largest sequence of states from the first layer to the last layer by depth. Next time if there is still time left after searching the largest sequence, we can go back to the root and continue to keep track the second largest sequences. Continued, if there are still have time, we can go to the third largest sequence and so on. By this way, the opportunities to find more bugs increase and all given time can be utilized better.

In another aspect, coverage based test has been studied in the class. And I am interested in this part and I am considering to add the functions and concept of coverage to this modified algorithm in the next step.

API and SYS Input:

API used in SUT:

newBranches(), allBranches(), backtrack(), enabled(), safely(), reduce(), test(), fails(), failure(), state(), backtrack(), internalReport(), prettyPrintTest()

SYS Input: as the requirement for part 1, we need to put several parameters such as timeout, seed, depth, width, etc as input. This part of work has been added to test1.py too

Bugs report:

TSTL(the template scripting testing language) is used to help find bugs in an avlTree library, called avlbug1.py. This library exists bugs. The tstl file called avlefficient.tstl is used here. And at this time, tester.py written by authors are implemented.

The first bug is founded in the depth 14 very quickly as follows:

```
DEPTH 14 QUEUE SIZE 33 VISITED SET 5537
Note:: There is a FailureDEPTH
Start Reducing
val0 = 11                                                    # STEP 0
val3 = 8                                                     # STEP 1
avl0 = avl.AVLTree()                                         # STEP 2
avl0.insert(val0)                                            # STEP 3
avl0.insert(val3)                                            # STEP 4
val3 = 10                                                    # STEP 5
avl0.insert(val3)                                            # STEP 6
(<type 'exceptions.AttributeError'>, AttributeError("'NoneType' object has no attribute
'right'",), <traceback object at 0x103b7d680>)
```

There are also other bugs reported, but these bugs are very similar to the above one, so the detail of them are ignored.

Of course, if you used bfsrandom.py instead of tester.1py, you will find the bfsrandom.py works very slowly to go from depth to depth. The improved algorithm can find bugs in TSTL and has higher efficiency the one in reference [3].

Reference:

1) Alex Groce, Jervis Pinto, Pooria Azimi, Pranjal Mittal. TSTL. A Language and Tool for Testing (Demo). ISSTA'15, 2015

2) Alex Groce, Jervis Pinto. A Little Language for Testing. NFM'15, 2015

3) https://github.com/agroce/tstl