Course: CS569
Name: Kai Shi
Competitive Milestone 2

Improved Random Test Generation

Random Testing

Random testing is a good way to test programs. Random testing is also called "fuzzing"[1]. Random testing is highly effective in industry testing development. However, it will cause some problems. It will generate some meaningless cases in limited testing time, which will lower the testing efficiency. Also, some branches are easy to be run and some are not. So, before half of testing time, some branches are run many times, and some are still unexecuted. Then, in rest half time, situation still unchanged, which will cause some of branches are tested too many times but some of them are only tested 1 or 2 times. This will be a problem of random tester. Therefore, I plan to improve this situation of random test in this project. If we divide testing time into two parts: situation1 and situation 2. Situation 1 is normal testing, and situation2 will make some changes.

New way of generating sequences

I divided first input <TIMEOUT> into two phases. Phase1 runs random testing normally. In phase2 first finding the median coverageCount and the branches tested less than this median will be tested in phase 2. These branches have been tested more than median times will not be executed. This algorithm will avoid only test some of branches and ignore some of others.
First, I spend t/2 time to run randomAction() as normal. After doing this, I just print the coverage. Then in the rest t/2 time, calling a function findMid(). Then, I used the idea from BFS algorithm to append the current state into Queue list if it's in the belowMid list, which returns from findMid() function. Then, adding queue list into curren statements. Last, still using randomAction() to test current statements. Also, printing the how many statements are blow median coverage out of total statements. It should be half size of the total statements.

Phase 1:
```
start = time.time()
print "Testing fo half ", TIMEOUT/2, "time..."
while start is less than Timeout/2:
        Doing randomAction();
        Computing CoverageCount;
printCoverage()
```

Phase 2:

start2 = time.time()
while current time - start2 < TIMEOUT/2:
    findMid() → This step will return a Mid coverage and queue list of branches below Mid
    Appending the current state into Queue[];
    Adding queue list into curren statements;
    Doing randomAction();
    Computing new CoverageCount;
printCoverage()

Printing other results, bug reports, etc.

For the findMid() function. First sorting the results of phase1 by their coverageCount. Then, let ss be the half length of sorted list. Then, coverageCount[s] means current state's executed times, and coverageCount[sortedCov[ss]] means the tested times Median branches we found. If coverageCount[s]< coverageCount[sortedCov[ss]], add current state s into belowMid list.

```
def findMid():
  global belowMid
  sortedCov = sorted(coverageCount.keys(),key = lambda x : coverageCount[x])
  ss = len(sortedCov)/2
  for s in sortedCov:
    if coverageCount[s] < sortedCov[ss]:
      belowMid.add(s)
    else:
      break
```

Compairing to the milestone1, I add sut internalreport at last and I changed 5th input. When FAULT is 1, my tester can save bugs as separate files.
Here is the result of tester2 coverage:

```
TSTL BRANCH COUNT: 223
TSTL STATEMENT COUNT: 163
29 FAILED
TOTAL ACTIONS 3498
TOTAL RUNTIME 30.0012872032
```

Her is the result of tester1 coverage:

```
TSTL BRANCH COUNT: 216
TSTL STATEMENT COUNT: 155
566 FAILED
TOTAL ACTIONS 92227
TOTAL RUNTIME 30.0298240185
```

The branches new algorithm covers has slight improvement.

Next Plan

I have tried to use the (FAR) algorithm instead of improving random testing. But I cannot generate the new vector sequence. So, I still use current algorithm, finding the belowMid. Next time, I will also continue to improve the branches coverage of current algorithm. I plan to divide TIMEOUT into 3 parts.

Reference:

[1] W.Howden,“SystemsTestingandStatisticalTestDataCov- erage”, COMPSAC ‘97, Washington, DC, August 1997, p. 500-504.