

Project Milestone 1 for CS569: Static Analysis and Model Checking

for Dynamic Analysis

Xiao Han
May 5, 2016

I. Background

In the project proposal, I write about “Feedback-directed Random Test Generation” [1], which is raised by Pacheco, Carlos; Shuvendu K. Lahiri; Michael D. Ernst; and Thomsa Ball. The authors find a new way to improve the efficacy of random testing which mentions in their paper. The technique that authors present is improving random test generation by taking care of the feedback obtained from the input test cases that the generator created before [1]. If some of the input test cases, which have been tested, make no sense, the generator should avoid create a new test case with the same values.

II. Algorithm

The authors give the algorithm of Feedback-directed Random Test. There are three sequences: error sequence, non-error sequence, and new sequence. Error sequence and non-error sequence are used to store the cases that have been tested. New sequence is used to store the new case that generator creates. First of all, the generator will create a test case and put it into new sequence. Second, checking whether the new test case, which is stored in the new sequence, has been tested before. If the new test case has been tested then create a new case. Third, executing the new sequence and check the feedback. If the feedback has no error then store the new sequence into non-error sequence. If the feedback has error then store the new sequence into error sequence. This algorithm will avoid lots of non-sense test cases.

III. My tester1.py

This is the basic implementation of a novel test generation algorithm using the TSTL API. My program meets the basic requirement and support all of required command line. Define a function which is called `parse_args()`. I’m using the same function as `randomtester.py` [2]. This function will add command line parameters arguments and set input `sys.argv` into this parameters. Define a function which is called `make_config(pargs, parser)`. This function is also the same as `randomtester.py` [2]. This function will return a dictionary which will let me using command line easily by calling `config.xxx`. Define a function which is called `check_action()`. This function will get a random action by using `sut.randomEnabled(random seed)` and check whether the action is safely by using `sut.safely(action)`. If command line running is 1, then give the output as required. If the action is not ok then there is a bug. Output the failure message. If command line fault is 1 then save different failure messages into different files. My `tester1.py` using the algorithm of “Feedback-directed Random Test Generation”. There are three list: `news`, `error`, and `nonerror`. `news` will using to store `sut.newStatements()`. `error` will store the `sut.currStatements()` which find a bug. `nonerror` will store `sut.currStatments()` which don’t contain a bug. When the algorithm generate a new

statements, the algorithm will check whether the statements have been tested. If statements have been tested then generate a new statements. My tester1.py allow user give a time how long will the tester run.

IV. Future works

For milestone 2, I will implementation more parameters and SUTs. In the same time, I will try to let my algorithm becomes more efficient.

Reference:

- [1] Pacheco, Carlos; Shuvendu K. Lahiri; Michael D. Ernst; Thomas Ball (May 2007). "Feedback-directed random test generation". ICSE '07: Proceedings of the 29th International Conference on Software Engineering (IEEE Computer Society): 75–84.
- [2] <https://github.com/agroce/tstl>