CS569 (Spring 2016) --- Project

Competitive Milestone 1

Wen-Yin Wang

05/02/2016

Background

In the previous project proposal, I was planning to implement my novel test generation algorithm based on the reference papers, however, the paper I chosen is implement a tiny SQL program in JAVA and it is hard to implement in TSTL. So, I decided to switch my algorithm to random testing with some modifications. I already read through the TSTL paper and the relative codes that provided by instructor on Github website. As I mentioned in previous proposal, I started my implementation on random testing. In this time, I am focusing on built the roughly structure, means that I tend to implement the basic settings instead of implement my novel algorithm.

Current implement

The requirement of this project is that my algorithm using the TSTL API must include SUT modules. The command line must contains seven arguments: timeout, seed, depth, width, faults, coverage, and running. So, the first step in my algorithm is catching the arguments from the command line. As instructor's example code, I used a function called parse_args() that adopting argparse module to parse command line arguments and grapping the arguments through sys.argv[1:]. After parsing the command line, I used a function called make_config() to set up the argument's name and content that could be used easily in following program. Then, I used print() to display the configurations not only to clarify the settings but also to ensure they are correct. The second step is using the data we collected from command line. For argument 'seed', it used as a time seed in random function. For argument 'coverage', if it is 1 then call sut.report() to file named "coverage.out". For argument 'depth', it used in a for loop to ensure that it only execute as many times as 'depth'. Within the for loop of 'depth', there has another for loop of argument 'width' to ensure that there execute at most 'width' times at each level. Before the loop started, I set up two variables that 'start' to hold the start time and 'elapsed' to hold the current time minus start time, which is the execution time. The program checks the 'elapsed' in the loops, if it exceeds the 'timeout', then break the loops and report the failure message. If the argument 'faults' is true, the program will save the discovered failure into a file called 'failure1.test'. If the argument 'running' is true, the program will show the information on the branch coverage. That is the implementation for the command line arguments. In the random testing part, I simply call the function randomAct() which using randomEnabled() in SUT module now, and checking the act using safely() in SUT module. Also, in every act, the program will collect current coverage.

Future works

The next step of my implementation is revising codes. I will revise my program from easy random testing to more interesting random testing. I will add a new function to change the actions instead of randomEnabled() and a function to handle failures.

Also, I will check my codes for static model checking. I expect that my program will be completed over 70% in the next project assignment.

Pseudocode

```
def randomAct():
        act = sut.randomEnabled(config.seed)
        ok = sut.safely(act)
        if not ok:
                if config.faults = 1:
                        write to file named "failure1.test"
                sut.failure()
        return ok

def main():
        make config from command line

        sut = sut.sut()

        start = time.time()
        for i = 1 to config.depth:
                sut.restart()
                for j = 1 to config.width:
                        randomAct()
                        elapsed = time.time() - start
                        if config.running = 1:
                                sut.newBranches()
                                sut.newStatements()
                        if elapsed > config.timeout:
                                break
                        collect current coverage

        if config.coverage = 1:
                sut.report("coverage.out")
```