CS639A: Program Analysis, Verification And Testing
Users Online : 1

Assignment-5: Program Synthesis using Symbolic Execution

Submitted on 27/10/2021 22:59

Instructions

- You are given an extra 10 minutes after due time to submit your assignment.
- However, please note that any submissions made after the due time are marked as late submissions.

Assignment-5: Program Synthesis using Symbolic Execution

Question:

Problem Statement

In this assignment, you are given two programs **P1** and **P2.** You are to find constant assignments to variables in **P1** (giv
using -c flag) such that it becomes semantically equivalent to **P2**.

Details

A possible plan for the solution can be as follows:

Run the symbolic execution engine to generate testData.json.
Implement the ***checkEq()*** function in ***symbSubmission.py*** file. This function must returns assignments to **constparams** stored in **testData.json** such that the two programs equal for given test cases.

Implement the marked function in **`Submission/symbSubmission.py`** file for this assignment.

```
1. def checkEq()
```

Running Symbolic execution

Points to note.

1. Symbolic execution needs initial seed values for the program inputs. **Specify using '-d' or '--params' flag.**

2. Symbolic execution needs initial seed values for the constants to be inferred. **Specify using '-c' or '--constparams' flag.**

3. Refer to **sExecution.py, sExecutionInterface.py, z3Solver.py** (KachuaCore/) file for better understanding.

4. You may want to install z3-solver ($ **pip install z3-solver**) to solve the generated constraints. Tutorial for z3-solver is at https://ericpony.github.io/z3py-tutorial/guide-examples.htm

Running the symbolic execution: (From KachuaCore folder)

CS639A: Program Analysis, Verification And Testing
Users Online : 1

Checking for Program Equivalence

Objective: Generate values for constparams such that P1 and P2 are semantically equivalent.

Running checkEq() : (from Submission folder)

```
$ python3 symbSubmission.py -b eqtest2.kw -e '["x", "y"]'

-b : pickle file of ir of eqtest2.tl. -e : Output variables.
```

Example

Input programs:

P1 (eqtest1.tl):

```
:y = :x
if :x <= 42 [
  :y = :y + :c1
]
:y = :y + :c2
```

P2 (eqtest2.tl):

```
:y = :x
if :x <= 42 [
  :y = :y + 40
] else [
  :y = :y + 22
]
```

```
$ ./kachua.py -t 100 -se example/eqtest1.tl -d '{":x": 5, ":y": 100}' -c '{":c1": 1, ":c2": 1}'
```

testData.json is generated when **P1** is symbolic executed using **-se** flag as shown above. testData.json contains test cas coverage, path conditions, final expressions (symbolic) at the end of execution for that test case.

```
$ python3 symbSubmission.py -b eqtest2.kw -e '["x", "y"]'
```

Using these data you are to generate values for :c1 and :c2 such that for all test cases (ignoring the first test case) both **P1** and **P2** have same value, for variables mentioned with **-e** flag (x and y here), at the end of execution.

For test case :(:x=45, :y=23) the constrains generated is **Implies(And(x==45,y==23) , x+c2==45+22).** This query will generate c2=22 and c1=x (x meaning any value).

You are to generate the constraints such that for all test cases **P1** and **P2** are equivalent for variables indicated using -

CS639A: Program Analysis, Verification And Testing
Users Online : 1

Minimum Kachua version to use is **v3.2**.

Deliverables

The source code of your implementation, **including the KachuaCore directory of the version you used**.
A brief report (less than 5-pages) describing your implementation, assumptions and limitations. (Understand the difference between the tool's limitation and a bug: any error or missing feature that is caught during evaluation is a bu unless it is listed under the "limitations" section of your tool. Please do include Kachua version used in the submission)
A set of test cases (at least 5) with the expected output. **(tests folder, KachuaCore)**

The quality of all the above would affect your marks. The quality of all the above would affect your marks.

Submission Format

Your submission **MUST** be in the following format:

The submission should be a **zip** file.
The zip file should be named as **assignment_"number"_"Roll-of-student"**.
Zip the content of the **source** as is and submit. **Don't refactor the base code or move the files around.** (KachuaCore Folder, Submission Folder, README)

Please note that your submission will **NOT** be graded if you do not follow the format. Furthermore, we will use the **Readme** file provided by you to build and run your code. Therefore, please make sure that the Readme is clear. We cannot grade your submission if we cannot run it on our system.

Some important comments

Before doing anything "extra" (which might fetch bonus marks), first, complete the basic expectations from your implementation.
Program analysis tools are expected to display their results in a user-friendly manner; a user would never like to use a tool that simply spits out a bunch of numbers. So, display the results from your tool suitably.
Discussion is healthy, copying is not. You are encouraged to discuss the assignments, but you must implement the assignments individually. If any two students are found with "similar" pieces of code, both of them will be failed (with n concern as to who was the source).

Uploaded Files:

assignment_5_21111037.zip

Grades:

Marks: 90

Demo was good.

Did not provide test cases.

Feedback: