ASSIGNMENT 5: PROGRAM SYNTHESIS USING SYMBOLIC EXECUTION

KACHUA VERSION: 3.7

EXECUTION FLOW OF THE TOOL:

- User inputs- the turtle file path of program with constant parameter, parameter seed inputs , constant parameter seed input and timeout in seconds as command line argument to the ***Kachua.py***, file is then opened and parsed to make the ***Intermediate Representation (IR).***
- ***IR,* timeout, constant parameter seed input** and **parameter seed input**. are passed as arguments to the **symbolicExecutionM*ain()*** in the ***sExecution.py by Kachua.py*** to run symbolic execution on the turtle program
- After performing Symbolic Execution on given turtle program, test data generated is stored as **testData.json** in **Submission** folder.
- User then inputs Reference Turtle Program path to convert it into **binary IR** and store it as **optimized.kw** in **KachuaCore** folder.
- User then executes the **symbSubmission.py** with command line arguments as path of **optimized.kw** and list of **output variables.**
- **IR** stored in **optimized.kw** is loaded and **checkEq()** with arguments as **IR** and **command line arguments** received is called.
- **testData.json** is read and stored in **testData** variable and then for each test case in **testData**, reference turtle program is executed with input values from test case and output value is stored and later used to create **constraints** and added to **solver**
- Seed input for Symbolic Execution of Reference Turtle Program is taken as input from user and a new set of test cases are generated and input and corresponding output of turtle program is stored.
- Generated **input values** from test cases are matched with **constraints** from **testData** to form new set of solver constraints which are added to the **z3 solver.** This step is done using **checker()**
- Then using **s.s.check()**, it is checked whether a set of values for **constant parameters** exist which satisfy all the constraints if **sat** then a **model** is generated using **s.s.model()** which contains **constant parameter values** which are displayed in terminal. If **unsat** then **'Two programs are not Equivalent'** is displayed in Terminal

NOTE:

1. **mytestx_2 is the turtle program with constant parameter**
2. **mytestx_1 is the reference turtle program**

HARSH AGARWAL
21111030
MTECH CSE 2021-23
PROGRAM ANALYSIS, VERIFICATION AND TESTING

STEPS TO RUN THE TOOL:

1. Check if python packages like ***z3solver, regular expression (re) , time*** and other packages and dependencies mentioned in ***Readme.md*** are installed properly, if not you can install it by running command "***pip <space> install <space> <package name>***" in any terminal or appropriate commands from ***Readme.md***
2. Open any terminal (***Command Prompt, Windows PowerShell, Git Bash***, etc.) and move into the directory where **k*achua.py*** is present
3. Type "***. /kachua.py <space> -t <space>'timeout in seconds'<space> --se <space> file path/turtle file name.tl*** **<space> -d <space> 'parameter seed input' <space> -c <space> 'constant parameter seed input' "** to run symbolic execution on turtle program with constant parameters

   For eg, to run one of the test cases provided type: "**./kachua.py -t 100 -se ./tests/mytest2_2.tl -d '{":a": 14 }' -c '{":c1": 1, ":c2": 1, ":c3" : 1}'** "
4. Press Enter
5. ***Generated Test Data is displayed in terminal***
6. Type **"./kachua.py <space> -O <space> file path/turtle file name.tl"** to convert reference turtle program into its binary IR

   For eg , type **"./kachua.py -O ./tests/mytest2_1.tl"**
7. Press Enter
8. Binary IR is stored as **optimized.kw** in **KachuaCore** folder
9. Open another terminal (**Command Prompt, Windows PowerShell, Git Bash**, etc.) and move into the directory where **symbSubmission.py** is present
10. Type **" python3 <space> symbSubmission.py <space> -b <space> ../KachuaCore/optimized.kw <space> -e <space> list of output variables "** to calculate constant parameter values

    For eg , Type **"python3 symbSubmission.py -b ../KachuaCore/optimized.kw -e '["a" , "b"]' "**
11. Press Enter
12. Enter seed values for symbolic execution of Reference Turtle Program
13. **Calculated Constant Parameters values or 'Two programs are not Equivalent' will be displayed in Terminal**

HARSH AGARWAL
21111030
MTECH CSE 2021-23
PROGRAM ANALYSIS, VERIFICATION AND TESTING

IMPLEMENTATION LOGIC:

- **symbolicExecutionmod():** This function has been taken from sExecution.py in KachuaCore folder and some minor modifications are made to run symbolic execution on Reference Turtle program. Takes IR , Parameter list with values (seed input) and timeLimit as argument.

  Modifications made:
  - constantparams argument removed as reference program has no constant parameters.
  - Code added to store variable name and value at the end of program execution in a dictionary called output
  - Code added to add output dictionary to dictionary data which finally becomes testData
  - changed format in which computed parameter values are stored
  - Instead of storing testData as a json file, it is returned by function
  - Removed 5-6 lines at the end dealing with storing testData as json file and at timeout killing the execution of program. That is exit() was removed as program was not needed to be terminated.

- **checker():** Function to find out which constraint does the input parameter value match and returning sat or unsat. Takes params dictionary and list of constraints as argument and returns sat if parameter values passed in params are able to satisfy all the constraints in the list
  - Initialize a z3 Solver and add all variables in params as symbolic variable to the solver
  - For each constraint in the list replace variable name with its numeric value from params dictionary and add this modified constraint expression to the solver.
  - Using s.s.check(), check if all the constraints added in solver are satisfied or not
  - If satisfied, return ‘sat’ else return ‘unsat’**.**

- **CheckEq():** Primary function that performs computations to compute constant parameter values. Takes command line argument given and IR of reference turtle program as arguments.
  - Reads the testData.json file and stores its content in dictionary testData
  - Initializes a Z3 solver
  - For each test case in testData
    - Reads the parameters and stores them in dictionary p from ‘params’ field of test case dictionary
    - Reads the variable names and stores them in list interest_var from ‘symbEnc’ field of test case dictionary
    - For every variable in interest_var , it is added as symbolic variable in solver by removing ‘:’ from variable name
    - For every constant parameter c in ‘constparams’ field of test case dictionary, we remove c from dictionary p and interest_var.
    - Reference Turtle Program is executed with input parameter values taken from dictionary p and values of variable present in interest_var at the end of execution is displayed and stored.

        - For each expression in ‘symbEnc’ field of test case dictionary, we check if lhs is not equal to rhs then we check if lhs or rhs contains variable which is present in interest_var list, if present then variable name is replaced with numeric value stored in previous step. This modified expression is added to solver in the from ‘lhs == rhs’ as a constraint.
    - Then we perform Symbolic Execution of Reference Turtle program by taking seed input as input from user and then calling symbolicExecutionmod() with right arguments.
    - SymbolicExecutionmod() returns a dictionary structure of testData which contains symbolic execution information plus values of output variables at the end of execution.
    - Then checks for each new test case generated above from symbolic execution of Reference Turtle Program, which test case's set of constraints from symbolic execution of turtle program with constant parameter is satisfied by the parameter value specified in current test case. if a set of constraints is found to be satisfied, then variable name in symbolic expression of that test case is replaced with numeric parameter value and it is added as constraint to z3 Solver in similar way as mentioned earlier.
    - Then using s.s.check(), it is checked if all the constraints in solver are satisfied by some combination of constant parameter values. If satisfied then s.s.check() returns ‘sat’ else ‘unsat’
    - If ‘sat’ is returned then a model is generated using s.s.model() which contains constant parameter values that satisfy all solver constraints and these computed constant parameter value is displayed in terminal.
    - If ‘unsat’ is returned by s.s.check(), then a message ‘Two Programs are not Equivalent’ is displayed in terminal.

LIMITATIONS & ASSUMPTIONS:

- There are no errors raised during execution of the input turtle program.
- All the variables involved are of integer type ie domain of variable values is only integers and not float.
- Good seed inputs are provided for performing symbolic execution on both programs. As quality and variety of test cases generated depends on seed input and Constraints formed to calculate constant parameter values are in turn totally dependent on generated test cases.
- Timeout in seconds is at least enough for symbolic execution to get 100% coverage over inputted turtle program so all paths are traversed, and appropriate test cases are generated
- Value of constant parameters computed is dependent on set of test cases generated and so if testcases are not exhaustive then constant parameters calculated may be wrong.