CS502: Compiler Design

Fall 2022 (Due: November 8th, 2022)

Assignment A4: Are you alive?

---

1 Assignment Objective

Use JavaCC and JTB to write an intraprocedural IDFA that performs live-variable analysis.

2 Pheobe goes to the dentist again!

Pheobe was having a toothache and was afraid to go to the dentist because of the curse. After everyone reassured her nothing bad is going to happen she did go and now wants to make sure everyone she knows is alive! Help poor Pheobe in finding out the same.

3 Detailed Specification

You are provided with a grammar file `FriendsTJ.jj`. Our task is to first identify the live variables (local declarations and parameters) at each point of the program, and print them at the occurrences of the /* PRINTLIVEVARIABLES */ query, in lexicographically sorted order. In order to perform the analysis you may use:

- **CFG generator:** We have implemented a CFG generator for you. *You can thank us later!*
  - `CommonUtils.getMethodCFG(declStmt)` - Use this API to get the startNode of the CFG for the method with MethodDeclaration statement "declStmt".
- **Predecessor/Successor Chains:** Once you have a node in the CFG use the APIs:
  - `node.getSuccessorNodes()` - to get Successors of a node (returns them as a list).
  - `node.getPredecessorNodes()` - to get Predecessors of a node (returns them as a list).
- **Visualization tool:** In-order to visualize the CFG for a method you may use the below APIs.
  - `CommonUtils.createDotFile("<filename>", cfgStartNode)` - to generate a graphviz dot file.
  - `dot -Tpng <filename>.dot -o <filename>.png` - Use this command to generate a png file of the CFG.

3.1 Public Testcase

```
class Test1 {
    public static void main(String[] arg) {
        Foobar obj;
        int ret;
        int m_arg;
        arg=10;
        obj=new Foobar();
        ret = obj.Compute(m_arg);
    }
}
class Foobar {
    public int Compute(int num) {
        int num_a;
        boolean ptTemp1;
        int ptTemp2;
        int ptTemp3;
        int ptTemp4;
        int ptTemp5;
        Foobar ptTemp6;
        int ptTemp7;
        int ptTemp8;
        int ptTemp9;
        int ptTemp10;
        ptTemp2 = 0;
        ptTemp10 = 10;
        ptTemp7 = ptTemp10 - ptTemp2;
        ptTemp9 = ptTemp10 + ptTemp2;
        /* PRINTLIVEVARIABLES */                    // OUTPUT : num,ptTemp2
        ptTemp1 = num <= ptTemp2;
        if (ptTemp1) {
            ptTemp3 = 1;
            num_a = ptTemp3;
            /* PRINTLIVEVARIABLES */                // OUTPUT : num_a
        } else {
            ptTemp6 = this;
            ptTemp8 = 1;
            ptTemp7 = num - ptTemp8;
            ptTemp5 = ptTemp6.Compute(ptTemp7);
            ptTemp4 = num * ptTemp5;
            num_a = ptTemp4;
            /* PRINTLIVEVARIABLES */              // OUTPUT : num_a
        }
        /* PRINTLIVEVARIABLES */                  // OUTPUT : num_a
        return num_a; } }
```

3.2 Notes

There are some general notes and assumptions that you can make regarding the test cases.

- Main method will not contain any livenessqueries hence you need not handle the same.
- There will be no invalid function calls i.e. calls to undeclared functions.
- While using the CFG API, ensure to check the type of the node. There are 4 types of nodes in a CFG.
  - STARTNODE/ENDNODE - These are dummy nodes to help you identify the start and end points in the CFG. Note that they do not contain any statement info. Beware of NPE when you access node (*getNode()*) info of such nodes.
  - INTERMEDIATENODE - Such nodes contain actual statement info from the grammar. Process these nodes to perform the IDFA.
  - RETURNNODE - As per the grammar we have an explicit return statement inside each method. This CFGNode handles the identifiers inside return statements.

  Use the API: getType() in CFGNode class to find the type of each node.

3.3 Evaluation

Your submission must be named as `rollnum-a4.zip`, where `rollnum` is your roll-number in small letters. Upon unzipping the submission, we should get a directory named `rollnum-a4`. The main class inside this directory should be named `Main.java`. Your program should read from the standard input and print to the standard output. You can leave all the visitors and syntax-tree nodes as it is, but remember to remove all the `.class` files and jar files.

We would run the following commands as part of the automated evaluation process:

- `javac Main.java`
- `java Main < test > out`

If the contents of `out` match with the expected output for the testcase, you would get marks for the corresponding testcase.

4 Plagiarism Warning

You are allowed to discuss publicly on class, but are supposed to do the assignment completely individually. We would be using sophisticated plagiarism checkers, and if similarity is found, the penalty used in the course would be as follows:

- First instance: 0 marks in the assignment
- Second instance: Grade reduction.
- Third instance: F grade and report to disciplinary committee.

-*-*-*- Do the assignment honestly; enjoy learning the course. -*-*-*-