CS243 Homework 2

Winter 2021

Due: February 3, 2021

Directions:

- This is a programming assignment. Download the starter code from the course website, complete the task (see "Your Task" below), and submit your code using Gradescope.
- You may work alone or in pairs. If you are working in a pair, be sure to add your partner to your group on Gradescope.

1 Introduction

Your assignment this week is to implement a basic dataflow framework for the JoeQ system. We will provide the interfaces that your framework must support. You will write the iterative algorithm for any analysis matching these interfaces, and also phrase Reaching Definitions and Faint Variable in terms that an implementation of the solver can understand.

We have setup an environment for the programming assignments found here. Download and unpack this archive on your own machine, or in your home directory on a Stanford machine like myth. **Make sure your code compiles and runs on myth**.

2 Starter Code

We have supplied you with starter code that bundles JoeQ along with files for you to fill in. This directory is organized as follows:

`hw2/build/`	build directory (will be overwritten, not part of zip)
`hw2/lib/joeq.jar`	packaged JoeQ
`hw2/lib/rt.jar`	Java 1.5 Object.class, for compatibility
`hw2/Makefile`	Makefile for compiling your code
`hw2/src/examples`	Examples of using JoeQ
`hw2/src/flow`	The dataflow framework and interfaces
`hw2/src/submit`	The code you will submit (Only make modifications here!)
`hw2/src/test`	Tests for your solutions

3 Getting Started

To run the PrintQuads example from JoeQ, do the following:

```
myth:~$ unzip hw2.zip
myth:~$ cd hw2
myth:~$ source setup.sh
myth:~/hw2$ make
myth:~/hw2$ ./run.sh examples.PrintQuads examples.ExprTest
```

To run `MySolver` using the `ConstantProp` analysis over a test program called `test.Test`, invoke the `run.sh` script on these four classes:

```
myth:~/hw2$ ./run.sh flow.Flow submit.MySolver flow.ConstantProp test.Test
```

Until you write `submit.MySolver`, this will not compute anything useful.

4 Faint Variable Analysis

Definition 1. *A variable is faint if it is not live (dead) or it is only used in computing faint variables.*

The lecture introduced the concept of live variable, which can be used to eliminate dead code in a program. Consider the following code:

```
    int foo() {
1.    x = 1;
2.    y = x + 2;
3.    z = x + y;
4.    return y;
    }
```

In this example only x and y are live; therefore it is safe to eliminate instruction 3. However, there are many cases where live-variable analysis fails to eliminate dead code. Consider the following programs:

Case 1

```
     void foo() {
1.     x = 1;
2.     y = x + 2;
3.     z = x + y;
4.     return;
     }
```

Case 2

```
     bool bar();
     int foo() {
       while(bar()) {
1.       x = x + 1;
2.       y = y + 1;
       }
3.     return y;
     }
```

In Case 1, x and y in instruction 1 and 2 are live because it is used in instruction 3. z is not live and can be eliminated. If you apply the liveness analysis again after removing instruction 3, you will find that y is actually dead and can be eliminated. In the third

iteration, you will find that x is also dead. A better dataflow analysis should be able to eliminate the three instructions in one pass.

In Case 2, x and y are both live because they are used in statements that define them. However, since the final value of x is never used, instruction 1 could in fact be eliminated.

In this assignment, we assume that faintness of LHS variable will only propagate to RHS variable if the operator is either `Operator.Move` or `Operator.Binary`. For example:

```
ADD_I  T12 int, R4 int, R5 int
```

`ADD I` is a binary operator; therefore, if T12 is faint then R4 and R5 are faint. Function calls may look like:

```
INVOKEVIRTUAL_I%  T8 int, M.Class1.Method(I)I, (R0 Class1, R1 int)
```

`INVOKEVIRTUAL_I` is neither binary or move; therefore, faintness is not propagated. Function calls, writes to fields, and returns are all examples of things that do not propagate faintness, and can cause a variable to not be faint.

5 Your Task

1. Fill in the `MySolver` class so it implements the interface `Flow.Solver`. Use it to run the constant propagation and liveness algorithms we provide.
2. Fill in `ReachingDefs` so that it executes reaching definitions analysis.
3. Fill in `Faintness` so it executes faint variable analysis.
4. Test your faintness implementation by adding test methods to `TestFaintness.java`. Make sure you test your implementation thoroughly. Note that the supplied `TestFaintness.out` is only a very basic test.

6 Submission

To submit your assignment, follow the following steps:

1. Type `make submission` in the homework directory.
2. Use `scp` to download `submission.zip` from the server.
3. Upload this file to the Gradescope HW2 assignment.
4. If you discover a bug after submitting (and before the due date), simply run the submit again. Gradescope will take the latest version.

7 Testing your code

The test package contains a `Test` class that you can use to test your implementation of `MySolver` and `ReachingDefs`. The `hw2/src/test/` directory contains `Test.cp.out`, `Test.lv.out`, and `Test.rd.out` which are output files for running the `ConstantProp`, `Liveness`, and `ReachingDefs` analyses over the `Test` class, and similarly for `TestTwo`. `diff` the output of running your `MySolver` with these out files to test your implementation.

8 Hints

- Get started early. This is a sizable project, and you need to familiarize yourself with JoeQ at the same time. If you wait until the last minute, you will not be able to finish.
- Look at the `ConstantProp` and `Liveness` classes. You will find techniques that will be useful to you when you formulate your own analyses.
- Read the JoeQ documentation carefully, and do not hesitate to look at the full javadocs if you get stuck.
- JoeQ has been built for Java 1.5, and does not support many of the Java features introduced since. It will also refuse to load Java code built for a newer JVM.

  In the starter core, we provide a subset of the Java Runtime that is enough to run JoeQ on the test code using a modern Java VM (1.7 or 1.8). With that code, you might get an error like:

  Warning: vm vendor Oracle Corporation is not yet supported, trying default sun13_

  You can ignore this error. It is sent to standard error, so it will not interfere with `diff`-ing your output for correctness.

  If you see errors of the form

  `java.lang.UnsupportedClassVersionError`: unsupported version 52.0

  you should install the Java 1.5 VM from the Oracle website.

  Your code will be tested with Java 1.5, and that version of Java is available on myth. You can enable it with:

  ```
  export PATH=/usr/class/cs243/jvm/bin:$PATH
  ```

- Study the `QuadIterator` interface carefully. It does a great deal of the work for you. But please take note: if you construct a backward `QuadIterator`, you must iterate over it with `hasPrevious()` and `previous()`.
- `DataflowObjects` are mutable; when in doubt, copy them. When writing reaching definitions, be sure to implement an `equals()` method, or your code will loop forever.

  Note: the grading code has a strict timeout of 4 seconds, to detect code that loops.

- The iterator returned by `Quad.successors()` (resp. `Quad.predecessors()`) generates `null` to indicate that the current object has the exit node (resp. entry node) as a successor (resp. predecessor).