ECS 240 Programming Languages

Winter 2023

Programming Assignment 3

About This Assignment

- This assignment is about data-flow analysis.
- This assignment can be worked on in a group of at most three.
- Information about using CSIF computers, such as how to remotely login to CSIF computers from home and how to copy files to/from the CSIF computers using your personal computer, can be found here.
- Unzip the file `pa3-handout.zip` that you downloaded from Canvas.
- Create a file `team.txt` that contains a full name and email address of each member of your team (⟨name⟩, ⟨email⟩); one member per line.
- Zip your solution, including `build.sh` and `run.sh`, into `pa3-solution.zip` and upload them to Canvas by the due date. Do not include binaries and other intermediate output.
- Your solution will be evaluated on a set of inputs that are different from the ones we have provided.

1 Reaching Definitions Analysis

In this assignment, you will implement reaching definitions analysis.

Input

The input file describes the control-flow graph (CFG) using the following input file format:

- The problem declaration line "`p` $V\ B\ E\ s\ x$" declares that there are V variables, B basic blocks, and E edges in the CFG. Variables are labeled from 1 to V, and basic blocks are labeled from 1 to B. $s, x \in [1, B]$ are entry and exit. There is only one problem declaration line in the input.

- Basic block declaration lines describe the basic blocks in the CFG. There are total B number of these lines. Each line is either "`b` i", "`b` $i\ l$", or "`b` $i\ l\ r_1\ \ldots\ r_n$" ($1 \leq i \leq B$, $0 \leq l \leq V$, and $1 \leq r_k \leq V$). All basic blocks have at most one statement. If "`b` i", basic block i is empty. If "`b` $i\ l$", basic block i contains a definition that assigns a constant to the variable l (l is never a zero for this case). If "`b` $i\ l\ r_1\ \ldots\ r_n$" and l is not zero, basic block i contains a definition, $l = r_1 + \ldots + r_n$;. If l is zero, basic block i contains a print statement, `print`$(r_1, \ldots, r_n)$;.

  For example,

  - "`b 1`" declares that the basic block 1 is empty.
  - "`b 2 1`" declares that the basic block 2 is $v_1 = c$, where c is some constant.
  - "`b 3 1 2 4`" declares that the basic block 3 is $v_1 = v_2 + v_4$;.
  - "`b 4 0 5 6`" declares that the basic block 4 is `print`(v_5, v_6);.

- Edge declaration lines declares the edges between the basic blocks. There are total E number of these lines. "`e` $i_1\ i_2$" ($1 \leq i_1, i_2 \leq B$) declares that there is an edge from i_1 to i_2 ($i_1 \rightarrow i_2$).

- "`c`..." is a comment line, which can be ignored.

Output

The output file contains B lines of "`rdout` $b\ i_1\ \ldots\ i_n$" ($1 \leq b, i_k \leq B$). Each line describes the set of definitions that reach $OUT[b]$. Each definition is identified by the basic block id that contains the definition. If there are no definitions reaching $OUT[b]$, "`rdout` b" must be output.

An example input is shown below:

```
c This is a comment.
p 2 5 5 1 5
c bb1: v1 = c;
b 1 1
e 1 2
```

```
c bb2: ;
b 2
e 2 3
e 2 5
c bb3: v2 = v1;
b 3 2 1
c bb4: v1 = c;
e 3 4
b 4 1
e 4 2
c bb5: print(v1, v2);
b 5 0 1 2
```

An expected output for the above input is:

```
rdout 1 1
rdout 2 1 3 4
rdout 3 1 3 4
rdout 4 3 4
rdout 5 1 3 4
```

- Implement your solution in `reachingdef/` and update `run.sh` script to execute your solution. You may choose any programming language to implement your solution as long as your code runs on CSIF machines. The `run.sh` scripts take two arguments: the input file containing the program statements, and the name of the output file. We have provided some test inputs and expected outputs.
  - After running `./run.sh ./tests/p1.txt output.txt` in the `reachingdef/` directory, the `output.txt` should be the same as `./tests/expected1.txt` (the order of the lines in the output may differ).
  - After running `./run.sh ./tests/p2.txt output.txt` in the `reachingdef/` directory, the `output.txt` should be the same as `./tests/expected2.txt` (the order of the lines in the output may differ).
  - After running `./run.sh ./tests/p3.txt output.txt` in the `reachingdef/` directory, the `output.txt` should be the same as `./tests/expected3.txt` (the order of the lines in the output may differ).
- Update `build.sh` if your solution requires a build process before running `run.sh`.

(Continued.)

2 Faint Variables Analysis

In this assignment, you will implement faint variables analysis. A variable v is faint at a program point p if along every path from p to the exit, v is either not used before being defined or is used to define a faint variable. See Sections 4.1 and 4.2.1 of *Data flow analysis: theory and practice* for more details. (The errata for the book can be found at `https://www.cse.iitb.ac.in/~uday/dfaBook-web/errata.pdf`.)

Input

The input format is the same as the reaching definitions analysis.

Output

The output file contains B lines of "`fvin` b v_1 ... v_n" ($1 \leq b \leq B, 1 \leq v_k \leq V$). Each line describes the set of faint variables at $IN[b]$. If there are no faint variables at $IN[b]$, "`fvin` b" must be output.

An example input is shown below:

```
c This is a comment.
p 3 5 4 1 5
c bb1: ;
b 1
e 1 2
c bb2: v1 = c;
b 2 1
e 2 3
c bb3: v1 = v1 + v2;
b 3 1 1 2
e 3 4
c bb4: v2 = v3;
b 4 2 3
e 4 5
c bb5: print(v1);
b 5 0 1
```

An expected output for the above input is:

```
fvin 1 1 3
fvin 2 1 3
fvin 3 3
fvin 4 2 3
fvin 5 2 3
```

- Implement your solution in `faintvar/` and update `run.sh` script to execute your solution. You may choose any programming language to implement your solution as long as your code runs on CSIF machines. The `run.sh` scripts take two arguments: the

input file containing the program statements, and the name of the output file. We have provided some test inputs and expected outputs.

  - After running `./run.sh ./tests/p1.txt output.txt` in the `faintvar/` directory, the `output.txt` should be the same as `./tests/expected1.txt` (the order of the lines in the output may differ).
  - After running `./run.sh ./tests/p2.txt output.txt` in the `faintvar/` directory, the `output.txt` should be the same as `./tests/expected2.txt` (the order of the lines in the output may differ).
  - After running `./run.sh ./tests/p3.txt output.txt` in the `faintvar/` directory, the `output.txt` should be the same as `./tests/expected3.txt` (the order of the lines in the output may differ).

- Update `build.sh` if your solution requires a build process before running `run.sh`.