ECS 240 Programming Languages

Fall 2020

Programming Assignment 2

About This Assignment

- This assignment is about points-to 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 `pa2-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 `pa2-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 Andersen's Algorithm

In this assignment, you will implement Andersen's algorithm for flow-insensitive points-to analysis.

Input

The input file describes the set of program statements using the following input file format:

- The first line is the problem-declaration line `p` N S, which declares a there are N variables, and S statements. Note that the variables are named from 1 to N. There is a single problem-declaration line in the file.
- The subsequent S lines each describe the program statement using the 4-tuple `s` d_1 v_1 d_2 v_2, where v_1 and v_2 are program variables with $1 \leq v_1, v_2 \leq N$, and d_1 and d_2 represent the number of dereference operators with $0 \leq d_1$ and $-1 \leq d_2$.

  For example,

  - The program statement `1 = &2` is represented as the line `s 0 1 -1 2`.
  - The program statement `*1 = &2` is represented as the line `s 1 1 -1 2`.
  - The program statement `**1 = &2` is represented as the line `s 2 1 -1 2`.
  - The program statement `1 = 2` is represented as the line `s 0 1 0 2`.
  - The program statement `1 = *2` is represented as the line `s 0 1 1 2`.
  - The program statement `***1 = **2` is represented as the line `s 3 1 2 2`.
- `c`... is a comment line, which can be ignored.

Note that Andersen's algorithm can only handle statements of the form `x = &y`, `x = y`, `*x = y`, and `x = *y`; i.e., at most one dereference or address-of operator per statement. Thus, you need to introduce temporary variables and new statements to convert the given input statements into a form that Andersen's algorithm can handle. However, the output points-to pairs should only pertain to the original N variables.

Output

Each line of the output file should be `pt` u v, where $1 \leq u, v \leq N$, iff variable u may point to variable v according to Andersen's algorithm. The order of the lines in the output file does not matter.
An example input is shown below:

```
c This is a comment.
p 3 4
c 1 = &2
s 0 1 -1 2
c 2 = &3
s 0 2 -1 3
```

```
c 3 = &1
s 0 3 -1 1
c 2 = 1
s 0 2 0 1
c This is another comment.
```

An expected output for the above input is:

```
pt 1 2
pt 2 3
pt 3 1
pt 2 2
```

- Implement your solution in `andersen` 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 `andersen` 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 `andersen` 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/p3.txt output.txt` in the `andersen` directory, the `output.txt` should be the same as `./tests/expected1.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`.

2 Precise Flow-Insensitive Points-to Analysis with Well-Defined Types

In this assignment, you will implement Chakravarthy's polynomial-time algorithm for precise flow-insensitive points-to analysis with well-defined types.

Input

The input file describes the set of program statements using the following input file format:

- The first line is the problem-declaration line `p` N S, which declares a there are N variables, and S statements. Note that the variables are named from 1 to N. There is a single problem-declaration line in the file.
- The subsequent N lines of the form `t` u i, where $1 \leq u \leq N$ and $0 \leq i$, denote the type of the variable; viz., `t 1 0` implies the variable `1` is of type `int`, `t 1 1` implies the variable `1` is of type `int *`, `t 1 2` implies the variable `1` is of type `int **`, and so on.
- The subsequent S lines each describe the program statement using the 4-tuple `s` d_1 v_1 d_2 v_2, where v_1 and v_2 are program variables with $1 \leq v_1, v_2 \leq N$, and d_1 and d_2 represent the number of dereference operators with $0 \leq d_1$ and $-1 \leq d_2$.

  For example,

  - The program statement `1 = &2` is represented as the line `s 0 1 -1 2`.
  - The program statement `*1 = &2` is represented as the line `s 1 1 -1 2`.
  - The program statement `**1 = &2` is represented as the line `s 2 1 -1 2`.
  - The program statement `1 = 2` is represented as the line `s 0 1 0 2`.
  - The program statement `1 = *2` is represented as the line `s 0 1 1 2`.
  - The program statement `***1 = **2` is represented as the line `s 3 1 2 2`.
- `c` ... is a comment line, which can be ignored.

Output

Each line of the output file should be `pt` u v, where $1 \leq u, v \leq N$, iff variable u may point to variable v according to Chakravarthy's algorithm. The order of the lines in the output file does not matter.

An example input is shown below:

```
c This is a comment.
p 8 7
t 1 3
t 2 3
t 3 2
t 4 2
```

```
t 5 1
t 6 1
t 7 0
t 8 0
s 0 1 -1 3
s 0 2 -1 4
s 0 3 -1 5
s 0 4 -1 6
s 0 5 -1 7
s 0 6 -1 8
s 2 1 2 2
c This is another comment.
```

An expected output for the above input is:

```
pt 1 3
pt 2 4
pt 3 5
pt 4 6
pt 5 7
pt 5 8
pt 6 8
```

- Implement your solution in `typed` 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 `typed` 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 `typed` 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/p3.txt output.txt` in the `typed` directory, the `output.txt` should be the same as `./tests/expected1.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`.