Homework 3 (Coding): Dataflow Analysis (V2)

17-355/17-665/17-819: Program Analysis *(Spring 2023)*
Claire Le Goues, Fraser Brown, Daniel Ramos

Due: Thursday, February 9, 2023 (11:59 pm)

100 points total

Assignment Objectives:

- Implement the key parts of a dataflow analysis for WHILE3ADDR, based on the concepts of flow functions, lattices, and the worklist algorithm.

Recall in class that we introduced the idea of *integer sign analysis*, which tracks whether each integer in a program is positive, negative, or zero. The results of this analysis can be used to optimize a program or to circumvent errors like using a negative index into an array.[1] In this homework, you will implement the fundamental components of integer sign analysis for WHILE3ADDR, including α, $join$, $\neq$, and the worklist algorithm.

Handin Instructions Submit *only* your version of the `src/analysis/df.ml` file to the assignment on Gradescope. Gradescope will run a set of held-out tests as part of its autotester against your code. Your grade will be based primarily on a combination of the tests your code passes, modulo certain manual considerations, described below.

Note that we do *not* plan to look at your GITHUB repository for this assignment. You *must* submit your code files via Gradescope.

1 Setup and Tools

Go to `https://classroom.github.com/a/UnLE6ysO` to clone the hw3 starter code into your own private GITHUB account.

The code is written in `OCaml`, a variant of the functional ML language family. If you have programmed in SML, the syntax should not be too shocking. We also provide pointers to resources on `OCaml` in the README.

The README on GitHub describes how to set up your environment, install dependencies, and compile the code. It provides an overview on the starter code and the facilities it provides. We have also commented the code reasonably extensively. At a high level, we provide parsers for WHILE and WHILE3ADDR, an interpreter for WHILE, a compiler for automatic translation between WHILE and WHILE3ADDR, and a hook into the dataflow analysis that you will be writing. You can therefore run your analysis on WHILE3ADDR programs directly and WHILE programs by

[1] For the purposes of this assignment, we have been ignoring the possibility of integer overflow (i.e., we consider only mathematical integers).

first compiling them into WHILE3ADDR. You can do this assignment without ever playing with the WHILE interpreter; we provide it for completeness and demonstration.

Recitation on Friday, February 3, will involve interacting with this codebase and a brief introduction to programming in OCaml. We encourage you to attend (even more than usual), especially if you aren't familiar with functional ML languages

The only file you need to modify and submit for this assignment is **`src/analysis/df.ml`**. Although it is not strictly required, you should also probably interact with and modify **`test/test_analysis.ml`** to test your analysis. You should not have to modify the Makefile or interact with the build/compile system at all, once you have the project set up and running.

2 Your Task: Sign Analysis Implementation

In this assignment, you will implement integer sign analysis for the WHILE3ADDR language.[2] You will implement the *less precise* analysis we discussed in class. The `domain` of this analysis is defined at the top of `df.ml`: we track whether variables are Negative, Positive, or Zero (or Top/Bottom). The state of the analysis at each program point is thus a mapping from a variable name to its abstract value at that point.

We have provided starter code in `df.ml` that provides these definitions, and implements useful utilities as well as several (...less interesting) parts of the analysis proper. *You* are to implement:[3]

- `alpha`, or α, the function that abstracts concrete to abstract values. (5 points)
- `join_values`, which joins individual abstract values (we lift this to states for you). (10 points)
- The inequality check $\neq$ between abstract states (function `sigma_ne`). (10 points)
- The rest of `flow`, which implements the flow function. We have provided some of the cases; you will do arithmetic operators, and if statements. (15 points)
- The rest of `kildall`.[4] We have set up the initialization, and the beginning of the `work` function that pulls items off the worklist. The heart of `work` should use the node pulled off the worklist to compute new dataflow states and worklist; this is the part you should implement. (*60 points*)

We will also look at your code, and reserve the right to deduct up to 10% for poor coding practice (lack of comments, impossible variable/function names, etc).

Clarification on division Bear in mind that the division in this language is integer division (that is, it returns only the integer portion of the result and discards any fractional result). Carefully consider what happens when dividing two numbers and the result is < 1.

Grading and testing We have provided some initial tests for you in **test/test_analysis.ml**. You can run them using `make test`. You can (AND SHOULD) add your own to them, the code should make it clear how.

[2] We will be analyzing a real language in a later assignment.

[3] We also marked the methods you need to complete with "TODO" comments, to help you find them in the code.

[4] Note that we provided pseudo-code for Kildall's in class; it is a good place to start!

We have also set up Gradescope's autograding functionality to run additional held-out tests that we will use to inform our grading of your solution. You can submit `df.ml` via Gradescope, and it will (fairly quickly) email you a report on how your submission did on the held-out tests. These tests are different from the ones in the repository.

If you experience problems or think you've found a bug, *please contact us ASAP.* We have done our best in setting it up, and we want it to be useful for you, but we may still be ironing out bugs!