

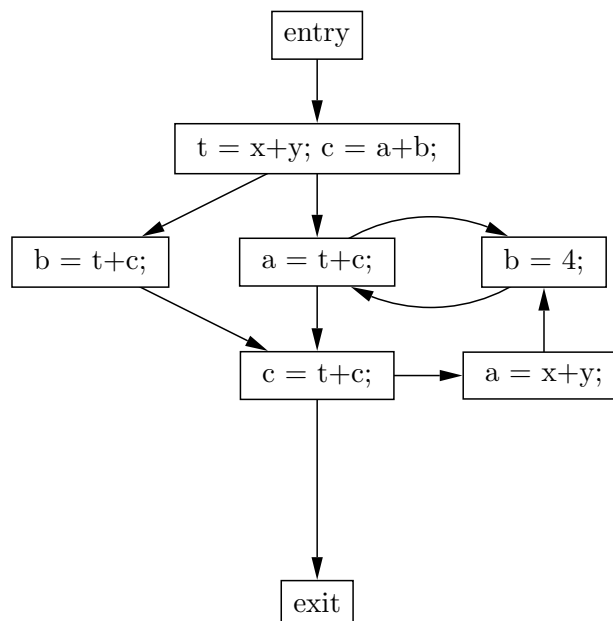
Assignment 1

Dataflow Analysis

Due: January 25, 11:00 am

This is a written assignment, every student must hand in his or her homework. Bring your homework to class on January 25. SCPD students may submit their homework by e-mail via scpd-distribution@lists.stanford.edu or give your homework to the courier.

1. Compute the available expressions (Chapter 9.2.6 in ALSU) on entry and exit for each basic block in the flow graph below.



Solution: The most common error here was assuming that $t + c$ was available after the expression $c = t + c$.

Block	In	Out
Entry	-	\emptyset
B1	\emptyset	$\{x+y, a+b\}$
B2	$\{x+y, a+b\}$	$\{x+y, t+c\}$
B3	$\{x+y\}$	$\{x+y, t+c\}$
B4	$\{x+y\}$	$\{x+y\}$
B5	$\{x+y, t+c\}$	$\{x+y\}$
B6	$\{x+y\}$	$\{x+y\}$
Exit	$\{x+y\}$	-

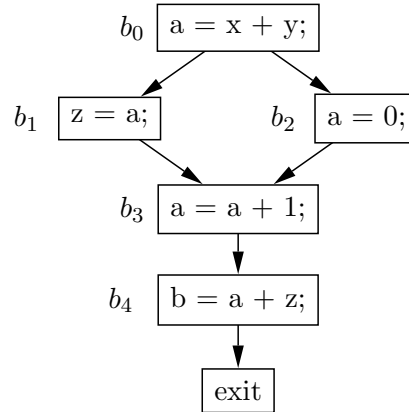
2. Is the following a meet operator? Please answer yes or no. No explanation is necessary.

- a. Maximum value (on integers)
- b. Product (on integers)
- c. Addition (on integers)
- d. Product mod 2 (on the set $\{0, 1\}$)
- e. Addition mod 2 (on the set $\{0, 1\}$)
- f. The GCD function (on integers)

Solution: All the “no” answers failed because they were not idempotent (i.e., $x \wedge x$ does not necessarily equal x).

- a. Yes
- b. No
- c. No
- d. Yes
- e. No
- f. Yes

3. We say that a program point p belongs to a live range of a definition d $u=x+y$ iff the value assigned to u in definition d may be accessed by using variable u at point p , for some path in the flow graph starting at p . Consider for example the code fragment below:



The live range of the definition ‘ $a = x + y$ ’ at point $\text{entrypoint}(b_0)$ is $\{\text{exitpoint}(b_0), \text{entrypoint}(b_1), \text{exitpoint}(b_1), \text{entrypoint}(b_3)\}$. Similarly, the live range of the definition ‘ $a = a + 1$ ’ at point $\text{entrypoint}(b_3)$ is $\{\text{exitpoint}(b_3), \text{entrypoint}(b_4)\}$.

This concept of live ranges can be used to increase flexibility in register assignment. For example, the live ranges of the definitions of a in b_0 and b_3 do not intersect at any points, so it may be possible to use same registers to hold these two values.

Describe an algorithm to find the live range of every definition in a program.

Solution: Compute both reaching definitions as well as liveness for each variable. A program point P is in the live range of a definition $D : x = y + z$ if and only if D reaches point P and x is live at the point P .

4. We wish to define a dataflow analysis that flags the use of a variable if:
- it *may* refer to a variable that has not been initialized.
 - it *must* refer to a variable that has not been initialized.

Formulate such an analysis by filling in the following table (provide only one analysis to handle both cases).

Solution: There are a variety of solutions to this problem. A single dataflow analysis can be used to solve both problems. Define a semi-lattice for a single variable with values \top , DEF , $UNDEF$, and $MAYBEDEF$, with $\top \geq DEF, UNDEF \geq MAYBEDEF$, with the obvious meanings and transfer function. Let the semilattice for the problem be the product of these lattices per variable in the program. Set $OUT[ENTRY]$ to $UNDEF$ for every variable, and set every interior value to \top . Then, issue warning 1 if a variable is used which has value $MAYBEDEF$ at the point of use, and issue warning 2 if a variable is used which has value $UNDEF$ at the point of use.

Other approaches performed two dataflow analyses, one for each warning. Correct solutions exist that use either forward or backward dataflows for each warning. Variants of must-reach (for warning 1) and may-reach (for warning 2) are forward dataflow algorithms that can be used to issue warnings. Likewise, performing liveness analysis and looking at the variables that are live at entry gives lets you issue warning 1. Performing a variant liveness analysis (“must be used” instead of “may be read”) lets you issue warning 2.

5. This question asks you to think about how changes to the initial values in a data flow analysis can affect the result. Recall that an answer to a data flow problem is considered “safe” if it is no bigger than the ideal solution.

Suppose you have defined a forward dataflow algorithm that is distributive and has finite descending chains. You accidentally initialized $\text{OUT}[B]$ to \perp for all nodes other than ENTRY.

- a. Will your algorithm give a safe answer for all flow graphs?
- b. If not, will it give a safe answer for some flow graphs? If it will, give an example.
- c. Will your algorithm give the MOP solution for all flow graphs?
- d. If not, will it give the MOP solution for some flow graphs? If it will, give an example.

Solution:

- a. Yes; the modified values start and remain \leq the MFP values.
- b. Yes, see previous
- c. No
- d. Yes, it will give the MOP on any acyclic flow graph.