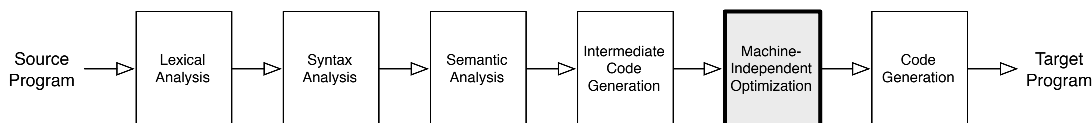
HW 9: Dataflow Analysis and Optimizations

CSCI 434T
Fall, 2007

Overview



As you write the TAC and x86 generators, we are going to begin exploring machine independent optimizations. Like the earlier topics, this is a great example of how a theoretical model can provide insight into how to think about and implement the fairly sophisticated analyses and transformations necessary to generate efficient code.

In this particular case, the theoretical foundations will be dataflow analysis. In general terms, dataflow analyses compute facts that must be true at the beginning and end of each basic block in a Control Flow Graph for a procedure body. The reading for the week focuses on several specific instances of dataflow analysis, as well as a short introduction to lattice theory, the mathematics behind this general technique. Next week, we will develop a general framework for dataflow problems.

Readings

- Dragon 9 – 9.3

Exercises

1. Nothing to write for this question, but be prepared to give me a status update on PA 3.
2. For the Control Flow Graph (CFG) in Fig 9.10:
   (a) Identify the loops.
   (b) Statements (1) and (2) in B_1 are both copy statements, in which a and b are given constant values. For which uses of a and b can we perform copy propagation and replace these uses of variables by uses of a constant. Do so, wherever possible, and show the resulting CFG. Do any statements become Dead Code?
   (c) Identify any global common subexpressions for each loop, and eliminate them wherever possible. Show the resulting CFG.
   (d) Using the CFG from (b), Identify any induction variables for each loop. Be sure to take into account any constants introduced in (b). Can strength reduction and/or induction variable elimination be applied? If so, show the resulting CFG. If not, describe one or two instructions that, if added the flow graph, would result in an opportunity for strength reduction.
   (e) Does the CFG from part (c) contain any loop-invariant computations to which code motion can be applied? If not, describe one or two instructions that, if added to the flow graph, would result in an opportunity for code motion.
3. For the CFG in Fig 9.10, compute the following:
   (a) Reaching Defs:
      - The *gen* and *kill* sets for each block. I usually represent this information in a table like the one I've started below:

Block	*gen*	*kill*
B_1	(1), (2)	(8), (10), (11)
B_2		
B_3		
B_4		
B_5		
B_6		

- The IN and OUT sets for each block. I always find it easiest to actually write the IN and OUT sets on the CFG while working through the algorithm. I've included a version on the next page that you can use for this purpose if you like. And feel free to print a few more copies to use for the other parts of this problem.

(b) Available Expressions:

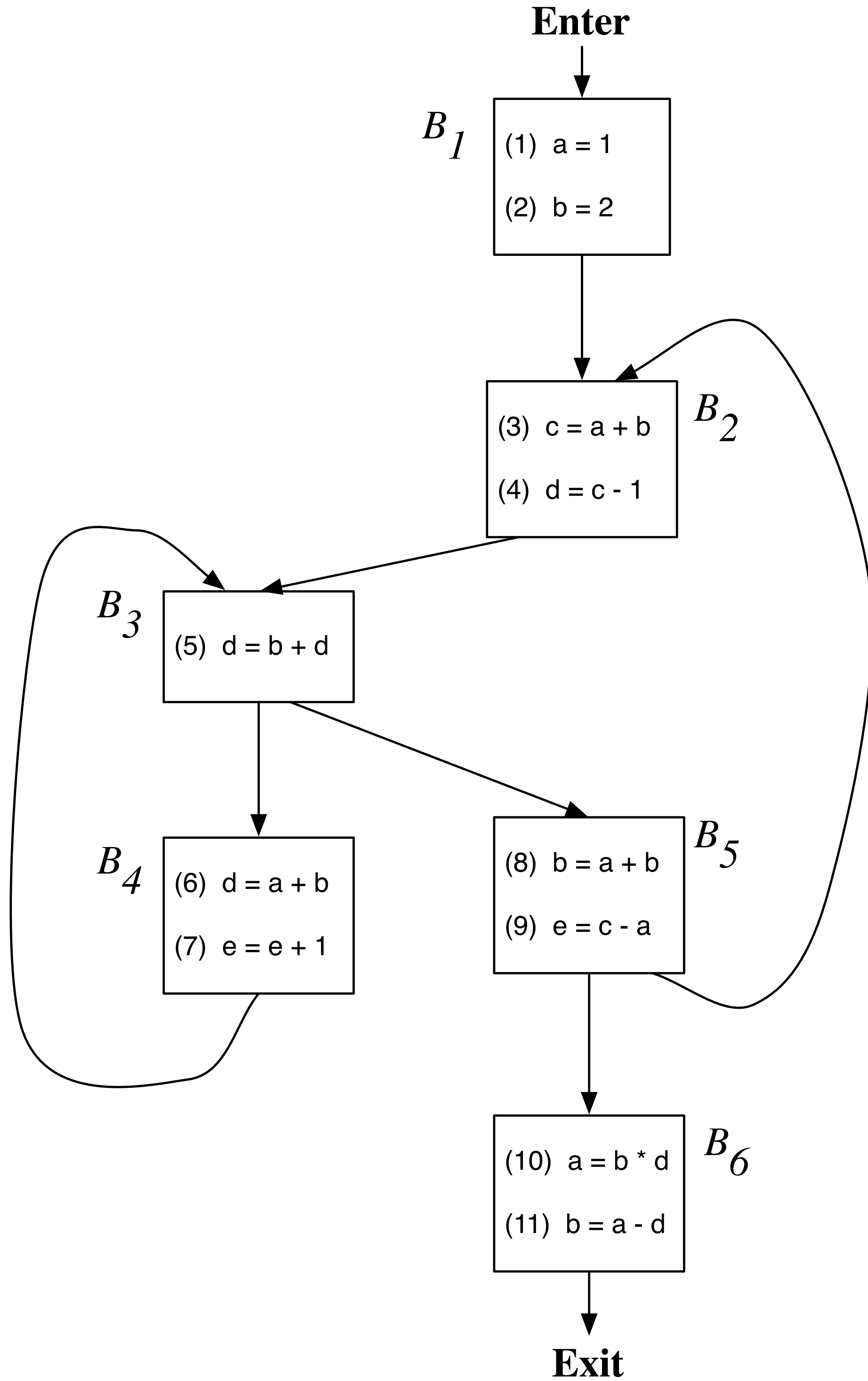
- The *e_gen* and *e_kill* sets for each block. For the *e_kill* sets, you may use a description like "all expressions using a or b as an operand".
- The IN and OUT sets for each block.

(c) Live Variables:

- The *def* and *use* sets for each block.
- The IN and OUT sets for each block.

4. Dragon 9.2.6. You may prove this statement inductively as requested, but I will also accept a clear, precise explanation of the intuition behind why the IN and OUT sets never shrink.)

5. **(Optional)** While you are not required to write up the solutions, please spend a few minutes thinking about Dragon 9.2.7 or 9.2.8, in order to develop an intuition for how to relate dataflow facts back to program behavior. I hope to touch on this a little bit next Monday.


Enter
B1
(1) a = 1
(2) b = 2
B2
(3) c = a + b
(4) d = c - 1
B3
(5) d = b + d
B4
(6) d = a + b
(7) e = e + 1
B5
(8) b = a + b
(9) e = c - a
B6
(10) a = b * d
(11) b = a - d
Exit