Advanced Compiler Techniques

Homework #2

Due: Oct 20, 2009 (before class)

! Exercise 9.2.4: Suppose V is the set of complex numbers. Which of the following operations can serve as the meet operation for a semilattice on V?

a) Addition: $(a + ib) \wedge (c + id) = (a + b) + i(c + d)$.

b) Multiplication: $(a + ib) \wedge (c + id) = (ac - bd) + i(ad + bc)$.

c) Componentwise minimum: $(a + ib) \wedge (c + id) = \min(a, c) + i \min(b, d)$.

d) Componentwise maximum: $(a + ib) \wedge (c + id) = \max(a, c) + i \max(b, d)$.

! Exercise 9.2.5: We claimed that if a block B consists of n statements, and the ith statement has gen and kill sets gen_i and $kill_i$, then the transfer function for block B has gen and kill sets gen_B and $kill_B$ given by

$$kill_B = kill_1 \cup kill_2 \cup \cdots \cup kill_n$$

$$\begin{aligned} gen_B &= gen_n \cup (gen_{n-1} - kill_n) \cup (gen_{n-2} - kill_{n-1} - kill_n) \cup \\ &\cdots \cup (gen_1 - kill_2 - kill_3 - \cdots - kill_n). \end{aligned}$$

Prove this claim by induction on n.

! Exercise 9.2.10: The astute reader will notice that in Algorithm 9.11 we could have saved some time by initializing OUT$[B]$ to gen_B for all blocks B. Likewise, in Algorithm 9.14 we could have initialized IN$[B]$ to gen_B. We did not do so for uniformity in the treatment of the subject, as we shall see in Algorithm 9.25. However, is it possible to initialize OUT$[B]$ to e_gen_B in Algorithm 9.17? Why or why not?

! Exercise 9.3.3: We argued that Algorithm 9.25 converges if the framework is monotone and of finite height. Here is an example of a framework that shows monotonicity is essential; finite height is not enough. The domain V is $\{1, 2\}$, the meet operator is min, and the set of functions F is only the identity (f_I) and the "switch" function ($f_S(x) = 3 - x$) that swaps 1 and 2.

a) Show that this framework is of finite height but not monotone.

b) Give an example of a flow graph and assignment of transfer functions so that Algorithm 9.25 does not converge.

[S1] (DFA on Value Range) In many cases knowing the range of variables is beneficial. For instance, knowing that variables *a* and *b* are between 0 and 127 may allow us to represent both variables within one byte instead of two words, thereby providing a more compact representation for certain data structures.

Suppose you are analyzing a program consisting of the following types of statements:

- `a = <const>`
- `a = b`
- `a = b + <const>`
- `a = b + c`

where all variables and constants are integers.

Your task is to formulate a dataflow problem called **VarRange** that would allow one to approximate the range of any given variable at any point in the program.

The range is to be represented by an interval **[*x*, *y*]** where both *x* and *y* are constants. Assume that **MAX** is the biggest representable integer and we are dealing with **positive** numbers (including zero) only.

a) What are the top and bottom elements of the lattice for the dataflow framework formulation of VarRange?

b) What is the JOIN ($\vee$) operator for VarRange?

c) What is the partial order ($\leq$) relation induced by the $\vee$ operator?

d) Assume for simplicity that each basic block consists of at most one statement. Define the transfer function for VarRange.

e) Is the transfer function you defined above monotonic? Please Explain.

f) Is the transfer function you defined above distributive? Please Explain.

g) What is the range for variable ***a*** [on EXIT] as computed by your algorithm for the CFG below?