CSE 6341, Written Assignment 4
Due Friday, March 22, 11:59 pm (8 points)

Your submissions should be uploaded via Carmen. Create your answers using a text editor and upload the file (e.g., plain text, Word, PDF). Alternatively, you can write your answers by hand and take a photo (or scan), but please ensure that (1) your handwriting is *clear and legible*, and (2) your photo or scan has *high resolution*, to allow the grader to read and understand your submission.

Q1 (2 points): Consider the abstract interpretation defined in slides 1-19. Show the derivation tree for <if (x>0) x=-3*y else x=z*8, σ_a> → σ'_a where $\sigma_a = [x \mapsto Neg, y \mapsto Pos, z \mapsto Pos]$

First determine and show the new abstract state σ'_a. Then show the entire derivation tree for this triple. Every level of the tree should correspond to one of the inference rules (slides 1-19).

Q2 (3 points): Consider the abstract interpretation defined in slides 1-19. Suppose we wanted to "abstractly execute" the following loop, with initial abstract state $\sigma_a = [x \mapsto Pos]$:

while (…) { x=-x; }

The loop condition is not relevant for this question.

Show the abstract state σ_{a1} after 1 iteration of the loop. Show the abstract state σ_{a2} after 2 iterations of the loop. Show the abstract state σ_{a3} after 3 iterations of the loop. You do **not** need to show the derivation trees, just show the abstract states.

As discussed, the final abstract state σ'_a is the merge of the infinite number of intermediate states σ_a, σ_{a1}, σ_{a2}, etc. This merge can be computed by a finite sequence of merge steps. For the example above, show the four merged states σ'_{a0}, σ'_{a1}, σ'_{a2}, σ'_{a3} defined on slide 19.

Q3 (3 points): Consider the following context-free grammar

<program> ::= <assignList>
<assignList> ::= <assign> | <assign> **;** <assignList>$_2$
<assign> ::= **id = intconst** | **id = floatconst** | **id**$_1$ **= id**$_2$

In some languages, instead of asking programmers to declare the types of variables, the compiler attempts to infer types from the code. For example, assignment **x=5** implies that **x** is of type *int*, while assignment **y=3.14** implies that **y** is of type *float*. Further, for **x=5; z=x** the compiler can conclude that both **x** and **z** are of type *int*. Inference is not always possible: for example, **x=5; x=3.14** does not allow a unique type to be associated with **x**.

One way to achieve this *type inferencing* is to construct and analyze a directed graph G=(N,E) as follows. The set of nodes N is the union of three disjoint sets: N = IDS U INTC U FLC. IDS contains a graph node for each **id**.*lexval* that appears in the program. For example, for program **x=5; z=x; x=6** we have IDS = { x, z }. INTC contains a graph node for each **intconst**.*lexval* that appears in the program. For the same example program, we have INTC = { 5, 6 }. Set FLC is defined similarly for **floatconst**.*lexval.*

The set E of graph edges is defined as follows. For every **id = intconst** in the program, E contains an edge from the node for **id** to the node for **intconst** (and similarly for **id = floatconst**). For every **id_1 = id_2** there is an edge from the node for **id_1** to the node for **id_2**. For example, for program **x=5; z=x; x=6** we have E = { x → 5, z → x, x → 6 }.

Part 1. Show graph G for the following program: **a = 8; b = a; c = b; c = 3.14; d = c; b = d**

Part 2. Suppose you are given G for some program. Your goal is to determine, for each node in IDS, one element of set { *int*, *float*, *not-typable* }. For example, if you analyze the graph for **x=5; z=x; x=6; z=2.3**, x would be determined to be *int* and z would be determined to be *not-typable*. Describe at a high level how to compute this information for all nodes n ∈ IDS for any given G. You do **not** need to show detailed algorithms. *Hint:* You solution can use graph properties such as "there exists a path in G from node … to node …".

Explain how your approach would work on the program from Part 1. Show the analysis output for that program.