CSE 6341, Written Assignment 4
Due Wednesday, November 8, 11:59 pm (8 points)

This assignment contains 3 questions, for a total of 8 points. Your submissions should be uploaded via Carmen. Create your answers using latex, Word, or plain text. If the answers are too difficult to write in an editor, feel free to write them by hand on a piece of paper and then upload a legible scan or photo of it. You can submit up to 24 hours after the deadline; if you do so, your score will be reduced by 10%. If you submit more than 24 hours after the deadline, the submission will not be accepted.

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

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 σ_a = [x→Neg]:

```java
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 | id1 = id2
```

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`.

Using the style of the typechecking attribute grammar from slides 67-74, define an attribute grammar that uses a global table to (1) infer the types of variables, and (2) reject programs in which a variable cannot have a unique inferred type. For simplicity, assume that the program does not have used-before-initialized variables; however, multiple assignments to the same variable are possible. Use only attributes <assignList>.tbl, <assign>.tbl, and id.lexval.