Written Assignment 4

CSE 6341

Due: October 15, 2020, by 12:45 pm

This assignment contains 4 questions, for a total of 16 points. It is OK to write/draw by hand and then scan (or take a photo of) your answer, and then upload that scan/photo. 

I will discuss the solutions for this assignment in class on October 15. As a result, late submissions will not be accepted.

Q1 (4 pts): Consider the attribute grammar for code generation discussed in class. Suppose we extend the language with a do-while loop, with the following syntax:

\[ \langle \text{stmt} \rangle ::= \text{do} \langle \text{stmt} \rangle \text{2} \text{while} (\langle \text{cond} \rangle) \]

The semantics of a do-while loop is this: the loop body \( \langle \text{stmt} \rangle \text{2} \) is first executed. Then the condition \( \langle \text{cond} \rangle \) is evaluated. If the condition evaluates to false, the loop completes execution. If the condition evaluates to true, the loop continues with the next loop iteration.

Show the attribute grammar evaluation rules for generating assembly code for do-while loops. Use notation similar to what was used in the lecture notes. Illustrate your solution by showing the complete generated code for the following program:

\[ x:=0; \text{do } \{ x:=x+1; \} \text{ while } (x<5) \]

Use the code generation rules from the lecture notes together with your new rule. Since we have not defined code generation rules for conditional expressions such as \( x<5 \), just use \( ... \) in your example to denote the assembly code for computing the value of \( x<5 \).

Q2 (5 pts): Consider the language from our discussion of operational semantics. Suppose we extend the language with an operator \( ?: \), which is used in C and similar languages. An expression that uses this operator is of the form \( e_1?:e_2:e_3 \). The evaluation semantics is the following. First, \( e_1 \) is evaluated. If its value is not 0, \( e_2 \) is evaluated and its value becomes the value of the entire expression. If the value of \( e_1 \) is 0, expression \( e_3 \) is evaluated and its value becomes the value of the entire expression.

Write down operational semantics inference rules that, in your opinion, are the most appropriate definition of the semantics of an \( ?: \) expression. Illustrate how your proposed rule(s) work by showing the entire derivation tree for

\[ \langle x+y?2*z:x-y,\sigma \rangle \rightarrow 8 \]

where \( \sigma = \{ x \mapsto -1, y \mapsto -2, z \mapsto 4 \} \)
Q3 (2 pts): In class we briefly mentioned the use of operational semantics to prove the correctness of compiler transformations. As an example, consider the following code transformation:

```plaintext
if (be) then { x:=e1; } else { y:=e2; }; x:=e1;
```

is transformed into

```plaintext
if (be) then { x:=e1; } else { y:=e2; x:=e1; };
```

Show an example of concrete code for which this transformation is not semantics-preserving.

Propose a correctness condition—i.e., some code restrictions based on x, y, e1, and e2—which guarantee the correctness of this transformation. Do not try to create a formal proof based on operational semantics: state the correctness condition and do not prove it.

Q4 (5 pts): Consider the abstract interpretation defined in class. Suppose we extend the language with the ?: operator described earlier. Define the abstract version of this operator, working on values Neg, Zero, Pos, and AnyInt. Since the operator takes three operands, break up the definition into several cases based on the abstract value of the first operand, and show separately the definition of the abstract semantics for each of those cases.

Illustrate how your evaluation rules will apply for the following example:

```plaintext
x:=-1; y:=-2; z:=4; w:=x+y?2*z:x-y
```

Show the abstract state after each assignment is executed. Explain briefly how the value of w is computed in the last assignment.