

**Written Exercise 1**  
due Friday, February 5, 2016 in class

**Problem 1:** Consider the following ambiguous BNF grammar:

$$\begin{aligned} \langle C \rangle &\rightarrow \langle C \rangle ? \langle C \rangle : \langle C \rangle \mid \langle V \rangle = \langle C \rangle \\ \langle C \rangle &\rightarrow \langle C \rangle \&\& \langle C \rangle \mid \langle C \rangle \parallel \langle C \rangle \mid !\langle C \rangle \mid \langle V \rangle \mid \text{true} \mid \text{false} \mid (\langle C \rangle) \\ \langle V \rangle &\rightarrow x \mid y \mid z \end{aligned}$$

Rewrite the grammar so that it is no longer ambiguous and has the following properties: The operators have the following precedence, from highest to lowest: ( ), !, &&, ||, ?:, =. The !, ?:, and = operators are right associative, and the && and || operators are left associative.

**Problem 2:** Consider the following grammar (yes, it is ambiguous but that is unimportant). The subscripts are used to distinguish otherwise identical non-terminals for the purpose of the questions below.

$$\begin{aligned} \langle \text{start1} \rangle &\rightarrow \langle \text{stmt3} \rangle ; \langle \text{start3} \rangle \\ \langle \text{start2} \rangle &\rightarrow \langle \text{stmt4} \rangle \\ \langle \text{stmt1} \rangle &\rightarrow \langle \text{declare2} \rangle \\ \langle \text{stmt2} \rangle &\rightarrow \langle \text{assign2} \rangle \\ \langle \text{declare1} \rangle &\rightarrow \langle \text{type3} \rangle \langle \text{var} \rangle \\ \langle \text{type1} \rangle &\rightarrow \text{int} \\ \langle \text{type2} \rangle &\rightarrow \text{double} \\ \langle \text{assign1} \rangle &\rightarrow \langle \text{var} \rangle = \langle \text{expression3} \rangle \\ \langle \text{expression1} \rangle &\rightarrow \langle \text{expression4} \rangle \langle \text{op} \rangle \langle \text{expression5} \rangle \\ \langle \text{expression2} \rangle &\rightarrow \langle \text{value4} \rangle \\ \langle \text{op} \rangle &\rightarrow + \mid - \mid * \mid \\ \langle \text{value1} \rangle &\rightarrow \langle \text{var} \rangle \\ \langle \text{value2} \rangle &\rightarrow \langle \text{integer} \rangle \\ \langle \text{value3} \rangle &\rightarrow \langle \text{float} \rangle \\ \langle \text{var} \rangle &\rightarrow \text{a legal name in the language} \\ \langle \text{integer} \rangle &\rightarrow \text{a base 10 representation of an integer} \\ \langle \text{float} \rangle &\rightarrow \text{a base 10 representation of a floating point number} \end{aligned}$$

Suppose our static semantic description has five attributes:

$$\begin{aligned} \text{type} &= \{ \text{integer}, \text{double} \} \\ \text{typetable}(\langle \text{var} \rangle) &= \{ \text{integer}, \text{double}, \text{error} \} \\ \text{inittable}(\langle \text{var} \rangle) &= \{ \text{true}, \text{false}, \text{error} \} \\ \text{typebinding} &= (\langle \text{var} \rangle, \{ \text{integer}, \text{double} \}) \\ \text{initialized} &= (\langle \text{var} \rangle, \{ \text{true}, \text{false} \}) \end{aligned}$$

typetable maps each possible variable name to its declared type, and inittable maps each possible variable name to a boolean indicating whether the variable has been assigned a value. Initially, both typetable and inittable will map all possible variable names to error to indicate that the variables have not been declared in the program. typebinding maps a single variable name to its declared type, and initialized maps a single variable name to whether it has been assigned a value. For each subscripted non-terminal, provide a rule to calculate its type, table, inittable, typebinding, and initialized attributes,

if that non-terminal requires that attribute. Each attribute should either be inherited or synthesized, but not both. For example, here are two such rules:

$$\begin{aligned} \langle \text{value2} \rangle . \text{type} &:= \text{integer} \\ \langle \text{declare1} \rangle . \text{initialized} &:= (\langle \text{var} \rangle, \text{false}) \end{aligned}$$

(Here I am using  $:=$  to create a mapping so you can use  $=$  to mean only mathematical equality.)

**Problem 3:** Suppose we want to enforce the following rules in the grammar from Problem 2:

- (a) The type of the expression must match the type of the variable in all assignment statements.
- (b) A variable must be declared before it can be used.
- (c) A variable must be assigned a value as its first use in the program.

Where in the parse tree should these rules be checked (i.e. at which non-terminals), and write the precise tests that should be done at those non-terminals using the attributes available.

**Problem 4:** Use axiomatic semantics to prove that the postcondition is true following the execution of the program assuming the precondition is true:

```
//Precondition: n      0 and A contains n elements indexed from 0
bound = n;
while (bound > 0) {
    for (i = 0; i < bound-1; i++) {
        if (A[i] > A[i+1]) {
            swap = A[i];
            A[i] = A[i+1];
            A[i+1] = swap;
            t = i+1;
        }
        t = 0;
    }
    bound = t;
}
//Postcondition: A[0]      A[1]      ...      A[n-1]
```

**Problem 5:** Let  $M_{\text{state}}$  be a denotational semantic mapping that takes a syntax rule and a state and produces a state. Define the  $M_{\text{state}}$  mapping for the following three syntax rules assuming we allow side effects, that is, assuming expressions and conditions can change the values of variables.

$$\begin{aligned} \langle \text{assign} \rangle &\rightarrow \langle \text{var} \rangle = \langle \text{expression} \rangle \\ \langle \text{if} \rangle &\rightarrow \text{if } \langle \text{condition} \rangle \text{ then } \langle \text{statement1} \rangle \text{ else } \langle \text{statement2} \rangle \\ \langle \text{while} \rangle &\rightarrow \text{while } \langle \text{condition} \rangle \langle \text{loop body} \rangle \end{aligned}$$

Assume we have the following mappings defined:

$M_{\text{value}}$  takes a syntax rule and a state and produces a numeric value (or an error condition).

$M_{\text{boolean}}$  takes a syntax rule and a state and produces a *true* / *false* value (or an error condition).

$M_{\text{name}}$  takes a syntax rule and produces a name (or an error condition).

*Add* takes a name, a value, and a state and produces a state that adds the pair  $(\text{name}, \text{value})$  to the state.

*Remove* takes a name and a state and produces a state that removes any pair that contains the name as the first element.

You may assume the *Add* and *Remove* mappings do not produce errors.