

# Assignment 1

CSE 755

Due: October 6, by **11:00 am**

1. (10 pts) Consider a generalization of the context-free grammar for binary numbers:

$$\begin{aligned}\langle S \rangle &::= \langle B \rangle . \langle B \rangle \\ \langle B \rangle &::= \langle B \rangle \langle D \rangle \\ \langle B \rangle &::= \langle D \rangle \\ \langle D \rangle &::= 1 \\ \langle D \rangle &::= 0\end{aligned}$$

The starting non-terminal is  $\langle S \rangle$ . This new grammar allows us to express binary numbers that are not integers. For example, the binary number  $10.001$  corresponds to the value  $2.125$ , which of course is  $1 \times 2^1 + 0 \times 2^0 + 0 \times 2^{-1} + 0 \times 2^{-2} + 1 \times 2^{-3}$ .

Show an attribute grammar that computes for  $\langle S \rangle$  the value of the binary number represented by  $\langle S \rangle$ . Do *not* change the context-free grammar, just add the appropriate attributes, evaluation rules, and conditions. For each attribute, specify its value domain (e.g., integer, boolean, etc.)

2. (5 pts) Consider the following grammar:

$$\begin{aligned}\langle S \rangle &::= \langle E \rangle \\ \langle E \rangle &::= \text{id}_{i,j} \mid ( \langle E \rangle ) \mid \langle E \rangle_1 + \langle E \rangle_2 \mid \langle E \rangle_1 * \langle E \rangle_2\end{aligned}$$

(Don't worry about the fact that the grammar is ambiguous — this is irrelevant for the problem being considered.) This grammar represents expressions for matrices — for example,  $( \mathbf{A}_{3,2} + \mathbf{B}_{3,2} ) * \mathbf{C}_{2,5}$ . Here  $\mathbf{A}$  is a matrix with 3 rows and 2 columns (same for  $\mathbf{B}$ ) and  $\mathbf{C}$  is a matrix with 2 rows and 5 columns. The addition operator  $+$  adds two matrices, but only if they have the same number of rows, and also the same number of columns. For example,  $\mathbf{A}_{3,2} + \mathbf{B}_{3,2}$  is a valid expression but  $\mathbf{A}_{3,2} + \mathbf{C}_{2,5}$  is not. The multiplication operator  $*$  multiplies two matrices, but only if the number of columns in the first one is equal to the number of rows in the second one. For example,  $\mathbf{A}_{3,2} * \mathbf{C}_{2,5}$  is a valid expression but  $\mathbf{A}_{3,2} * \mathbf{B}_{3,2}$  is not.

Write an attribute grammar (based on this context-free grammar) that generates all and only expressions in which the operands of the two operators satisfy the correctness conditions described above. (Essentially, this is a simple *type checker*.) Assume that each  $\text{id}_{i,j}$  has two pre-defined attributes “rows” and “columns” corresponding to the values of  $i$  and  $j$ , respectively.

3. (10 pts) Suppose we extend the grammar from the previous question to allow the “transpose” operator ( $A^T$  is a matrix in which the columns are the rows of matrix  $A$ ); in our language we will use the terminal symbol  $\textcircled{}$  to denote this unary operator. We will also include the “multiply with a scalar value” binary operator ( $\alpha A$  in mathematics, where  $\alpha$  is a scalar). This operator will be denoted by the terminal symbol  $\#$ . The first operand must be a matrix with one row and one column (i.e., it will contain one single element  $\alpha$ ). For the matrix that is the second operand, each matrix element will be multiplied by  $\alpha$  to get the corresponding element of the resulting matrix.

We will also allow the “assignment” operator  $:=$ , always appearing in the following form:  $\text{id}_{i,j} := \langle E \rangle$ . The assignment is valid only if the right-hand-side expression evaluates to a matrix with  $i$  rows and  $j$  columns. As in the C language, an assignment expression  $\text{id}_{i,j} := \langle E \rangle$  produces a value which is the matrix value being assigned to  $\text{id}$ . Thus, such an assignment expression can appear as a subexpression of other expressions.

$$\begin{aligned} \langle S \rangle &::= \langle E \rangle \\ \langle E \rangle &::= \text{id}_{i,j} \mid ( \langle E \rangle ) \mid \langle E \rangle_1 + \langle E \rangle_2 \mid \langle E \rangle_1 * \langle E \rangle_2 \mid \langle E \rangle_1 \textcircled{ } \mid \langle E \rangle_1 \# \langle E \rangle_2 \mid \text{id}_{i,j} := \langle E \rangle_1 \end{aligned}$$

Extend your solution from the previous question to handle these new language features. In addition, provide a concrete small example of a valid parse tree in which *each* production from above is applied at least once (i.e., there is at least one addition, multiplication, etc.). For this parse tree, show the values of all attributes and conditions at all parse tree nodes.

4. (5 points)

Consider the following context-free grammar with a starting non-terminal  $\langle prog \rangle$ :

$$\begin{aligned} \langle prog \rangle &::= \langle classlist \rangle \\ \langle classlist \rangle &::= \langle decl \rangle ; \langle classlist \rangle_1 \mid \langle decl \rangle \\ \langle decl \rangle &::= \text{class } \langle name \rangle \{ \langle body \rangle \} \mid \\ &\quad \text{class } \langle name \rangle_1 : \langle name \rangle_2 \{ \langle body \rangle \} \\ \langle name \rangle &::= \text{id} \\ \langle body \rangle &::= \dots \end{aligned}$$

The language defined by this grammar resembles a subset of C++. The program consists of a sequence of *class declarations*. Each class declaration consists of the keyword `class`, followed by the name of the class, an optional name of a *superclass* (represented by  $\langle name \rangle_2$  in the second production for  $\langle decl \rangle$ ), and the body of the class. Each class has either a single superclass, or no superclass at all. Assume that each `id` node in the parse tree has an attribute *lexval* containing the actual string value corresponding to this identifier; the value is already computed by the lexical analyzer.

You can assume that there are no duplicate class declarations. You can assume that the declaration of a superclass always occurs before the declaration of any of its sub-

classes. You can assume that the body of a class does not contain class declarations. Do not check these conditions, just assume that they are satisfied.

An example of a valid program is

```
class A { ... } ;
class B : A { ... } ;
class X { ... } ;
class Y : X { ... } ;
class C : B { ... }
```

Here, for example, class A has no superclass and class B has a superclass (which is A). For each program in this language, we can conceptually define a corresponding *class hierarchy forest*, containing one or more *class hierarchy trees*. Each class is represented by a unique node in the forest. If a class does not have a superclass, its node is the root of a tree. If a class Q is a subclass of P (i.e., we have `class Q : P { ... }`), the node for Q is a child of the node for P.

Define an attribute grammar that computes, for the root node  $\langle prog \rangle$ , an attribute *numnodes* representing the number of nodes in the forest corresponding to the program. Similarly, define an attribute *numedges* which computes the number of edges (i.e., parent-child relationships) in the forest. Do *not* construct the actual forest. Define any other attributes you deem necessary. Show all evaluation rules for all attributes. Do *not* change the underlying context-free grammar. For the example from above, the number of nodes is 5 and the number of edges is 3.

5. (10 points)

Consider the previous question, but this time suppose that it is possible to have duplicate class declarations—that is, the input program may contain multiple declarations for the same class name. Suppose you wanted to design a *semantic checker* to determine whether or not such duplicates exist in a given program.

Define an attribute grammar that computes, for the root node  $\langle prog \rangle$ , a boolean attribute *duplicates* which has the value “true” if and only if the program contains at least one duplicate class declaration.

Define any other attributes you deem necessary. Show all evaluation rules for all attributes. Do *not* change the underlying context-free grammar.