

Homework 1

CSC 7101, Spring 2009

Due: 9 February 2009

1. (10 pts)

Given the following grammar with start symbol $\langle A \rangle$:

$$\begin{aligned} \langle A \rangle & ::= \langle B \rangle \langle C \rangle \\ & \quad \{ \langle B \rangle.i := \langle C \rangle.s1; \} \\ & \quad \{ \langle C \rangle.i := \langle B \rangle.s + 2; \} \\ & \quad \{ \text{print}(\langle C \rangle.s2); \} \\ \langle B \rangle & ::= b \\ & \quad \{ \langle B \rangle.s = 2 * \langle B \rangle.i; \} \\ \langle C \rangle_1 & ::= \langle D \rangle \langle C \rangle_2 \\ & \quad \{ \langle C \rangle_1.s1 := \langle D \rangle.s + \langle C \rangle_2.s1 + 2; \} \\ & \quad \{ \langle C \rangle_1.s2 := \langle C \rangle_2.s2 + 2 * \langle D \rangle.s; \} \\ & \quad \{ \langle C \rangle_2.i := \langle C \rangle_1.i + 1; \} \\ \langle C \rangle & ::= \langle D \rangle \\ & \quad \{ \langle C \rangle.s1 := \langle D \rangle.s + 1; \} \\ & \quad \{ \langle C \rangle.s2 := \langle D \rangle.s + \langle C \rangle.i; \} \\ \langle D \rangle & ::= d \\ & \quad \{ \langle D \rangle.s := 1 \} \end{aligned}$$

State the values of all the attributes in the order in which they are evaluated for the input `bcd`, and state the value printed by the print statement.

Hints: You might want to draw the parse tree and the dependencies between the attributes first. The evaluation corresponds to two traversals of the parse tree, starting with $\langle D \rangle.s$.

2. (10 pts)

Consider the following grammar:

$$\begin{aligned}\langle S \rangle &::= \langle E \rangle \\ \langle E \rangle &::= 0 \mid 1 \mid 0 \langle E \rangle \mid 1 \langle E \rangle\end{aligned}$$

This grammar accepts an arbitrary sequence of zeros and ones as input. Add appropriate attributes and conditions to the grammar to restrict the accepted language so that any consecutive ones in the input must come in multiples of three. For example, the sequence '011100111111' is legal, while '011010' is not. Do not change the context-free grammar, and do not construct the whole string, pass it up to the root, and do the computation there.

3. (10 pts)

Design a context-free grammar for regular expressions. Make this an attribute grammar with a set-valued attribute attached to the start symbol that is the language (set of strings) denoted by the regular expression. A regular expressions can be empty, a symbol, the concatenation of two regular expressions, two regular expressions separated by a vertical bar, a regular expression followed by a star, or a regular expression in parentheses. E.g., for the regular expression '1(1|d)*' your attribute grammar should construct the (infinite) set of all strings consisting of an 1 followed by zero or more occurrences of either 1 or d.

4. (10 pts)

Consider a standard programming language (say, C or Pascal). Suppose we want to identify assignments in which a constant value is assigned to a variable. E.g., for the two assignments

```
x = 4 * 5;
y = 2 * x + 2;
```

we would like to recognize that x and y are assigned the constant values 20 and 42, respectively. (This information would allow us to generate more efficient code for these assignments by calculating the constant value at compile time.) Note that there may be other code in between these two assignments and that the second assignment may be, e.g., inside a loop. We would like to recognize y as constant as long as the value of x used in the assignment to y comes from a unique, constant assignment to x.

Briefly explain how you would design an attribute grammar to identify constant expressions and constant assignments. Explain which attributes you need for expressions and statements to compute this information, and how their values will be synthesized or inherited.

If you believe that constant assignments cannot be recognized by an attribute grammar, explain why not.