Principles of Programming Languages [PLP]

Exercises on Syntax-Directed Definitions

1. Given the following grammar for expressions:

E→E+T	T→T/F
E→E-T	T→F
E→T	F→(E)
T→T*F	F → id

   write the generated string **`a*(b-c)+(b-c)/a`** as a parse tree, as an abstract syntax tree, and as a DAG that is minimal.

2. Given the following attributed grammar

	Production	Semantic Rules
1)	$L \to E\ \mathbf{n}$	$L.val = E.val$
2)	$E \to E_1 + T$	$E.val = E_1.val + T.val$
3)	$E \to T$	$E.val = T.val$
4)	$T \to T_1 * F$	$T.val = T_1.val \times F.val$
5)	$T \to F$	$T.val = F.val$
6)	$F \to (E)$	$F.val = E.val$
7)	$F \to \mathbf{digit}$	$F.val = \mathbf{digit}.\text{lexval}$

   show the annotated parse tree for expression **`(5+8*7)*4n`**.

3. Consider the following attributed grammar:

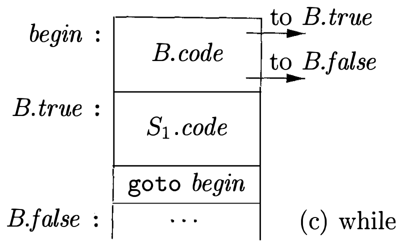
$S \to X\,T$	$T.a := X.b$	$S.b := T.b$
$T \to X\,T_1$	$T.b := T_1.b$	$T_1.a := T.a + X.b$
$T \to \varepsilon$	$T.b := T.a$	
$X \to \mathbf{a}$	$X.b := 1$	
$X \to \mathbf{b}$	$X.b := 2$	

   a. Say, for each attribute, if it is inherited or synthesized.
   b. Is the grammar S-attributed? Is it L-attributed?
   c. Depict the annotated parse tree for string **`bba`**. For each attribute in the tree, depict its value as well as a natural number indicating the order of evaluation of the attributes.

4. Translate into three address code the following program snapshot, using short-circuit code for the boolean expression, and assuming that **`b`** elements are 8 byte wide (**`&&`** denotes lazy conjunction, as in C/Java):

```
i = 0;
while ((i<n) && (b[i]>=0)){
      b[i] = 2*b[i];
}
n = i;
```

5. When generating three address code, it is often desirable to minimize the number of branches. The code layout of a while-loop shown below (left) has two branches per iteration: one to enter the body from the condition **B** and the other to jump back to the code for **B**. Thus it is usually preferable to implement **while (B) S** as if it were **if (B) { repeat S until ! (B) }**. Show what the code layout looks like for this translation, and revise the rule for while-loops shown to the right.



$S \rightarrow$ **while** $(B) S_1$	$begin = newlabel()$ $B.true = newlabel()$ $B.false = S.next$ $S_1.next = begin$ $S.code = label(begin) \mid\mid B.code$ $\mid\mid label(B.true) \mid\mid S_1.code$ $\mid\mid gen('\texttt{goto}'\ begin)$

Generation of three address code for **while (B) S**: Code layout (left) and generation rule (right)

6. The following grammar generates binary numbers with a "decimal" point:

   S → L.L | L L → L B | B B → 0 | 1

   Design an L-attributed SDD to compute B.val, the decimal-number value of an input string. For example, the translation of string 101.101 should be the decimal number 5.625. *Hint*: use an inherited attribute L.side that tells which side of the decimal point a bit is on.

7. Define an L-attributed SDD on a top-down parsable grammar to generate the NFA associated with a regular expression, using Thompson's algorithm sketched in the next figure. Assume that there is a token **char** representing any character, and that **char**.*lexval* is the character it represents. You may assume the existence of a function **new()** that returns a new state, that is, a state never before returned by this function. Use any convenient notation to specify the transitions of the NFA.