Jahir Montes

Homework 3

1. (5) a) the name(s) of people who have contributed to your solution of this homework, and b) their contribution (briefly). If you worked by yourself, the answer of this question would be ”N.A.” b) (5) Please make your writing for the homework easy to read. Acknowledge this
    a. N.A.
    b. Acknowledged
2.) a) Give two examples of semantic rules (NOTE: semantic rules are NOT the assignment expressions used in the attribute grammar. See slides for what are semantic rules.). b) Why do we need semantic rules?
    a.
        i. Every identifier has to be declared before use
        ii. Subroutine calls use the correct number of parameters
    b. We need semantic rules to give meaning to programs. Syntax is not enough to describe the meaning behind code.
3. (40) Consider the following context free grammar for an arithmetic expression containing operation – only:
    a.

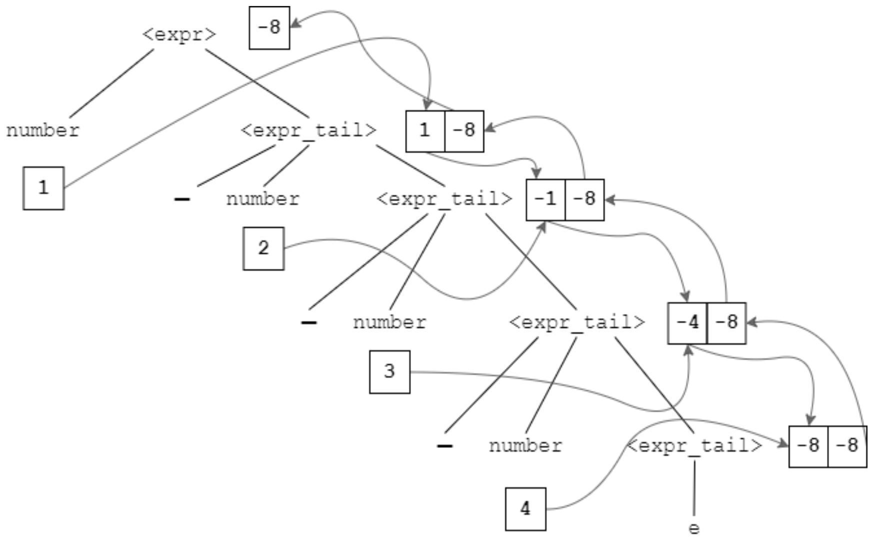
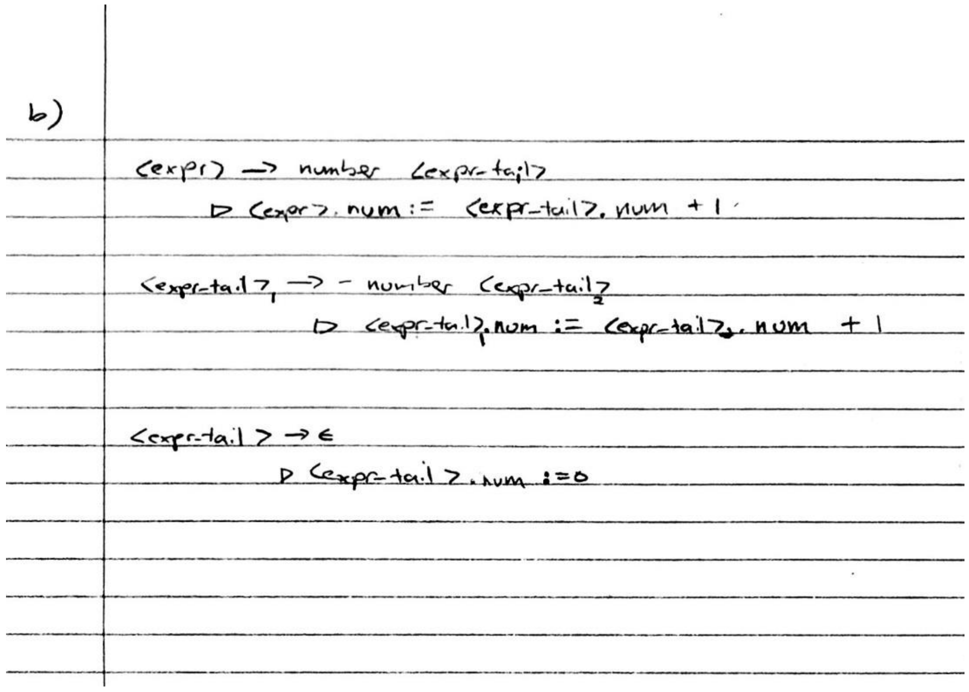


Figure 1: Decorated Parse Tree. The right boxes represent ".value" and the boxes on the left represent ".st"



b)

$\langle expr \rangle \rightarrow$ number $\langle expr\text{-}tail \rangle$

$\triangleright \langle expr \rangle.num := \langle expr\text{-}tail \rangle.num + 1$

$\langle expr\text{-}tail \rangle_1 \rightarrow$ - number $\langle expr\text{-}tail \rangle_2$

$\triangleright \langle expr\text{-}tail \rangle_1.num := \langle expr\text{-}tail \rangle_2.num + 1$

$\langle expr\text{-}tail \rangle \rightarrow \epsilon$

$\triangleright \langle expr\text{-}tail \rangle.num := 0$

4. Consider programs consisting of two parts
   a. CFG
      i. $< P > \rightarrow < fund > < func >$
      ii. $< fund > \rightarrow id(< paramlist >)$
      iii. $< func > \rightarrow id(< arglist >)$
      iv. $id \rightarrow letter$
      v. $< type > \rightarrow bool \mid int$
      vi. $< paramlist > \rightarrow < type >\ id\ < paramlist > \mid \epsilon$
      vii. $< arglist > \rightarrow id < arglist > \mid \epsilon$
   b. AG
      i. $< P > \rightarrow < fund > < func >$
         1. $func.name := fund.name$
         2. $func.number := fund.number$
      ii. $< fund > \rightarrow id(< paramlist >)$

1. $fund.number \coloneqq paramlist.number$
2. $fund.name \coloneqq id.name$

iii. $<paramlist>_1 \rightarrow <type> \ id \ <paramlist>_2$

1. $paramlist_1.number \coloneqq paramlist_2.number + 1$

iv. $<paramlist> \rightarrow \epsilon$

1. $paramlist.number \coloneqq 0$

c. Via the decoration of the parse tree, we can compare the number values of both the function call and declaration. In the given program, the function call only has one argument which is one argument short of the required amount. The parse tree can help check that the number of arguments is incorrect because, according to the parse tree, $func.number \coloneqq fund.number$. Since both *func* and *fund* should have the same *number* attributes, one can easily check that the given program would not be represented by the decoration of the parse tree; therefore, the given program would not be valid.