Jim Crowell  
CS354: ProgLang; Kennington  
WF, 1030-1145

Editor’s note: We are working on our concision, here, after an editorial on assignment 1.

**108: 2.12 - A & B**

**G → S $$**

**S → A M**

**M → S | epsilon**

**A → a E | b A A**

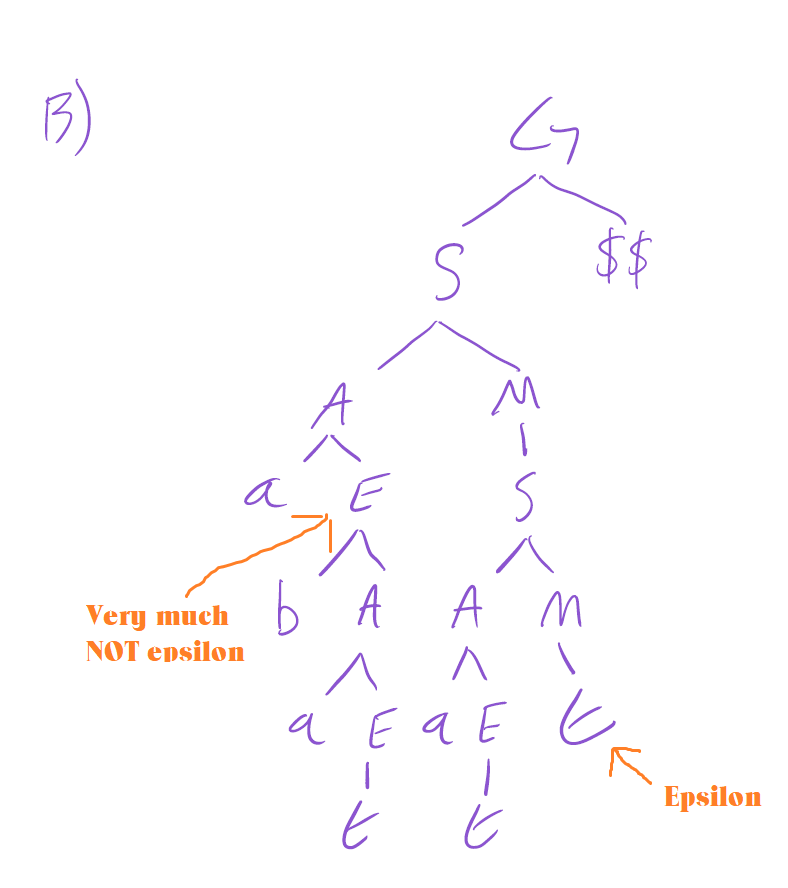
**E → a B | b A | epsilon**

**B → b E | a B B**

**(a) Describe in English the language that the grammar generates (e.g., "this makes a bunch of a's and b's that.....")**

This is a simple (meaning no loops/jumps), language over the characters {a, b}, which also accepts empty string inputs (specifically classed as M’s or E’s). It separates input into "parts of speech", so to speak: G's (which is the entire unit, what would be the "program" in our usual context, denoted by the end marker, "$$"), which is made up of S's, which are themselves made up of A and M tokens. M's can contain S's or empty elements. A tokens are ‘a’ characters paired with E tokens, OR ‘b’ characters paired with double A tokens. E's are composed of an ‘a’ followed by B tokens, OR ‘b’ followed by A tokens, and can also consist of nothing (empty strings/tokens). B tokens are a ‘b’ paired with an E token, or an ‘a’ paired with dual B's.

I think.

**(b) Show a parse tree for the string a b a a  
  
**

**108: 2.13 - A & B**

**stmt → assignment**

**→ subr call**

**assignment → id := expr**

**subr call → id ( arg list )**

**expr → primary expr tail**

**expr tail → op expr**

**→ epsilon**

**primary → id**

**→ subr call**

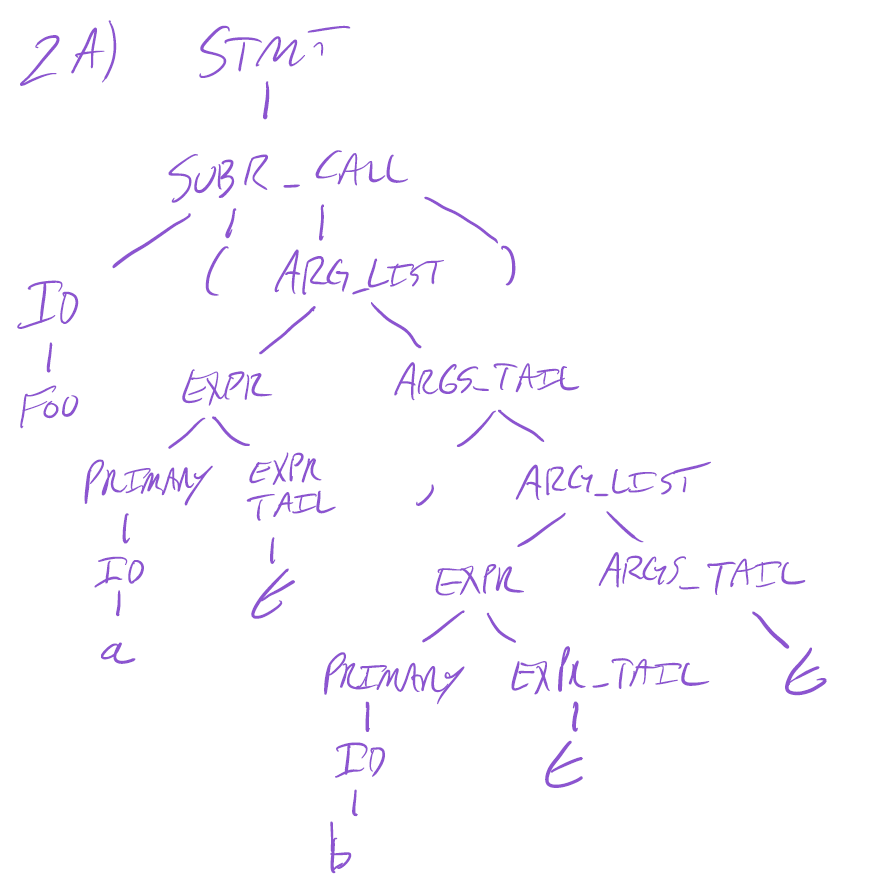
**→ ( expr )**

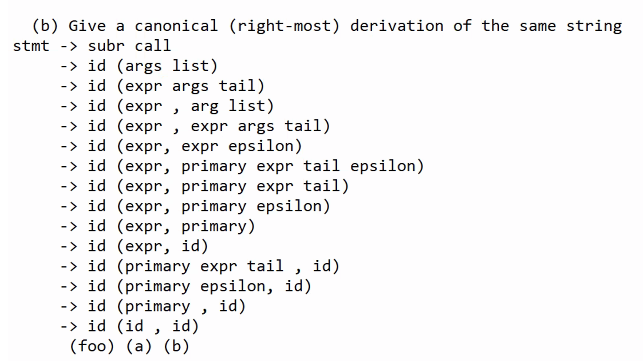
**op → + | - | \* | /**

**arg list → expr args tail**

**args tail → , arg list**

**→ epsilon**

1. **construct a tree for the parse string foo(a, b)**

****

**109: 2.17 - Extend the grammar of Figure 2.25 to include if statements and while loops, along the lines suggested by the following examples:**

**abs := n**

**if n < 0 then abs := 0 - abs fi**

**sum : = 0**

**read count**

**while count > 0 do**

**read n**

**sum := sum + n**

**count := count - 1**

**od**

**write sum**

Addition  
stmt -> IF cond THEN stmt\_list FI

-> WHILE cond DO stmt\_list OD

cond -> Factor boolop Factor

boolop -> <

-> >

-> <=

-> >=

-> ==

-> !=