COM S 342
Homework 1

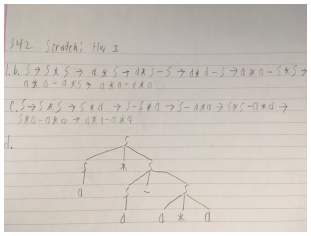
Christian Shinkle

September 6, 2017

1. (a) The terminals are $-, *, a$. The non-terminal is S.

   (b)

   (c)

   (d) 

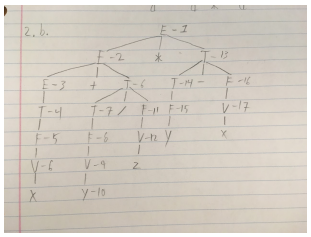
   (e) No, the left derviation will produce a different parse tree then the right derviation.

   (f) Three strings that don't belong to the language are:

   - $a * b$
   - $a * c$
   - $d - a$

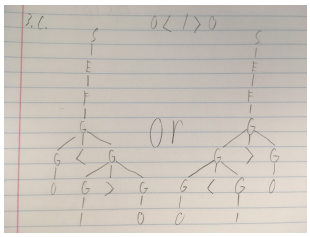
2. (a) The grammar is ambiguous because there are two disticnt parse that can be created using the production rule $E \rightarrow E + T | E * T | T$. If you are given a string $x + y * z$, you can create a derivation by either deriving the $x + y$ first or the $y * z$ first.

   (b)

   (c) 

3. (a) The associativity of @, !, and $<$ is $<$ first, followed by !, then @ last because $<$ is derived from G and G is derived from F, which also derives !, and F is derived from E, which also derives @. In layman's terms, you can't make an @ until you have made all the !, and you can't make a ! until you have made all the $<$.

(b) $<$ will always take precedence over ! and ! will always take precedence over @ because, again, $<$ is derived from G and G is derived from F and F is derived from E.

(c) 

4. The production rules of the grammar where S is the start variable:
   $V \rightarrow V@X|V.X|X$
   $X \rightarrow X \times Y|Y$
   $Y \rightarrow Z \cup Y|Z \cap Y|Z$
   $Z \rightarrow \neg Z|a|b|c|true|false$

5. For Python:

   - The "compound_stmt" and the "async_stmt" can accept the "for_stmt."
   - Consider the following script:
     1. i = 5;
     2. while i != 0:
     3. print(i);
     4. i -= 1;
     5. print("Blast off!");

     Line 1 is a simple_stmt derived from a small_stmt derived from a expr_stmt. This is how assignment statements are handled.
     Line 2 is a compound_stmt derived from a while_stmt which uses a "test" followed by a ':' followed by a suite.
     Line 3 is the first suite which is a NEWLINE character, an INDENT character, and a stmt. The stmt is derived from a compound_stmt derived from a funcdef that print 'i'.
     Line 4 is the second suite which is a simple_stmt derived from a small_stmt derived from expr_stmt derived from an augassign which uses '-=' to decrement 'i' by one.
     Line 5 is the same derivation as line 3 except there is no suite with a NEWLINE character and INDENT character, the derivations starts at a compount_stmt.