

CS 515 Fall 2011
 Homework 1 Sample Solution

---- 1.1 ----

```

E -> E + E
    | E * E
    | a | b | c
  
```

The grammar is ambiguous, because there "a+b*c" has two leftmost derivations:
 $E \Rightarrow E+E \Rightarrow a+E \Rightarrow a+E*E \Rightarrow a+b*E \Rightarrow a+b*c$

$E \Rightarrow E*E \Rightarrow E+E*E \Rightarrow a+E*E \Rightarrow a+b*E \Rightarrow a+b*c$

---- 2.1 ----

FIRST (true) = {true}
 FIRST (false) = {false}

FIRST (Opnd) = {true, false}
 FIRST (not Expr) = {not}
 FIRST (exor Expr Expr) = {exor}
 FIRST (and Expr Expr) = {and}
 FIRST (or Expr Expr) = {or}

FIRST (Expr) = {or, and, exor, not} \cup FIRST (Opnd)
 = {or, and, exor, not, true, false}

FOLLOW (Start) = \emptyset
 FOLLOW (Expr) = FIRST (Expr) \cup FOLLOW (Start) \cup {eof}
 = {or, and, exor, not, true, false, eof}

FOLLOW (Opnd) = FOLLOW (Expr) = {or, and, exor, not, true, false, eof}

---- 2.2 ----

The grammar is LL(1) because the FIRST sets for each of the right hand sides of Expr are pairwise disjoint. The same is true for the right hand sides of Opnd. There are no epsilon derivations. That the grammar is LL(1) can also be seen from the transition table:

| | or | and | exor | not | true | false | eof |
|-------|----------------|-----------------|------------------|---------------|-----------|------------|-----|
| Start | S -> E | S -> E | S -> E | S -> E | S -> E | S -> E | |
| Expr | E -> or E E | E -> and E E | E -> exor E E | E -> not E | E -> 0 | E -> 0 | |
| Opnd | | | | | 0 -> true | 0 -> false | |

---- 2.3 ----

```
[ eof S ]      and
[ eof E ]      and
[ eof E E and ] and
[ eof E E ]    and not
[ eof E E not ] and not
[ eof E E ]    and not true
[ eof E 0 ]    and not true
[ eof E true ] and not true
[ eof E ]      and not true or
[ eof E E or ] and not true or
[ eof E E ]    and not true or true
[ eof E 0 ]    and not true or true
[ eof E true ] and not true or true
[ eof E ]      and not true or true false
[ eof 0 ]      and not true or true false
[ eof false ]  and not true or true false
[ eof ]        and not true or true false eof
```

---- 2.4 ----

// If Start completes without error then the parse is OK.

```
procedure Start ()
  token := next_token()
  Expr()
  if token != eof then
    error()

procedure Expr ()
  if token in {"or", "and", "exor"} then
    token := next_token()
    Expr(); Expr()
  else if token = "not" then
    token := next_token()
    Expr()
  else
    Opnd()

procedure Opnd ()
  if token in {"true", "false"} then
    token := next_token()
  else
    error()
```

---- 2.5 ----

Shows call stack and total tokens read *after* next_token, *before* jump

```
[S]          and
[S E]        and not
[S E E]      and not true
[S E E E]    and not true
[S E E E 0]  and not true or
[S E E E]    and not true or
[S E E]      and not true or
[S E]        and not true or
[S E E]      and not true or true
[S E E E]    and not true or true
[S E E E 0]  and not true or true false
[S E E E]    and not true or true false
[S E E]      and not true or true false
[S E E E]    and not true or true false
[S E E E 0]  and not true or true false eof
[S E E E]    and not true or true false eof
[S E E]      and not true or true false eof
[S E]        and not true or true false eof
[S]          and not true or true false eof
```