

## First and Follow Sets

- The recursive descent, or predictive, parsers we have seen so far have shown that the EBNF alternatives map to *if* statements and repetition maps to a *while* loop.
  - The symbol we have at any stage must uniquely determine which path to follow. Such parsers can handle LL(1) grammars.
- Given a non-terminal symbol, the next symbol on input should *uniquely determine* which alternative of the production to choose. These input symbols are called *director symbols*.
- A production alternative can generate a number of terminal strings. The first symbols of those strings are director symbols for that alternative. To this end, we wish to calculate the set of terminal symbols which form the set of first symbols for each non-terminal in the language.
  - This set of symbols is called the *first set*.
- If a production alternative can generate the empty string, then the symbols that can follow the production also qualify as director symbols. Hence, we are *also* interested in what terminal symbols may follow a nonterminal symbol.
  - This set of terminal symbols is called the *follow set*.
- First and follow sets will be defined more formally a little later. We will consider them informally for the moment.

## A Simple Grammar...

$S ::= a A F$   
 $A ::= B | C$   
 $B ::= D^+$   
 $C ::= E^*$   
 $D ::= x | y | z | \epsilon$   
 $E ::= a | b | c$   
 $F ::= A | \epsilon$

First Sets are *relatively straightforward* – look at each production  
 Follow Sets are *more tricky* – you need to look at *every* production in which a symbol appears...  
 Director symbols for a production alternative are the first symbols for the alternative plus the follow symbols for the (left-hand-side non-terminal), but only if the alternative can generate the empty string

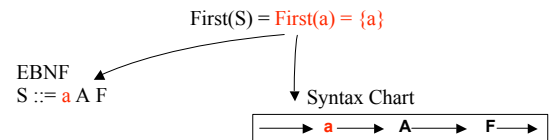
## Determining empty expansions

$S ::= a A F$	$\text{empty}(S) = \text{false}$
$A ::= B   C$	$\text{empty}(A) = \text{empty}(B) \text{ or } \text{empty}(C)$
$B ::= D^+$	$\text{empty}(B) = \text{empty}(D)$
$C ::= E^*$	$\text{empty}(C) = \text{true}$
$D ::= x   y   z   \epsilon$	$\text{empty}(D) = \text{true}$
$E ::= a   b   c$	$\text{empty}(E) = \text{false}$
$F ::= A   \epsilon$	$\text{empty}(F) = \text{true}$

By a simple transitive closure, we get:

$\text{empty}(S) = \text{false}$   
 $\text{empty}(A) = \text{empty}(B) \text{ or } \text{empty}(C) = \text{true}$   
 $\text{empty}(B) = \text{empty}(D) = \text{true}$   
 $\text{empty}(C) = \text{true}$   
 $\text{empty}(D) = \text{true}$   
 $\text{empty}(E) = \text{false}$   
 $\text{empty}(F) = \text{true}$

## First and Follow Sets (Cont.)



First of *any* terminal symbol is a set containing only *that* terminal symbol.

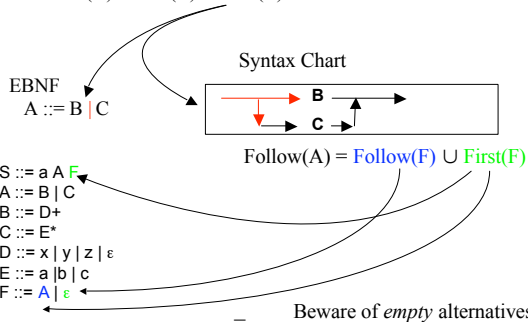
Follow(S) = { $\$$ }  
 “S” is the *start* symbol, so it never appears on the right hand side of a production!  
 $\$$  denotes the end of input

Director(S) = {a}

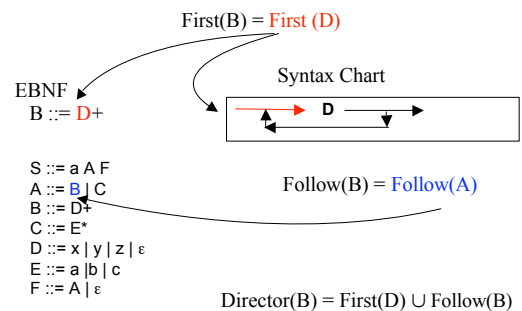
## First and Follow Sets (Cont.)

Director(A) = First(B)  $\cup$  First(C)  $\cup$  Follow(A)

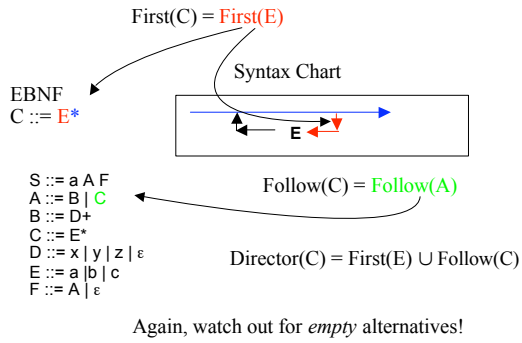
First(A) = First(B)  $\cup$  First(C)



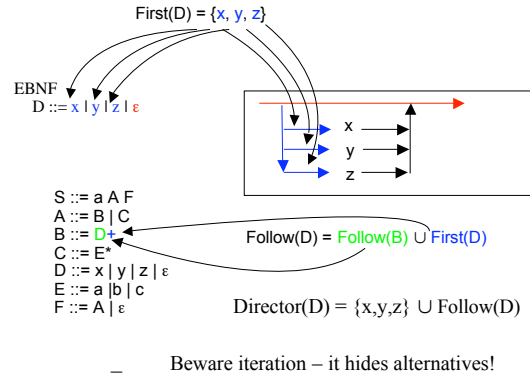
## First and Follow Sets (Cont.)



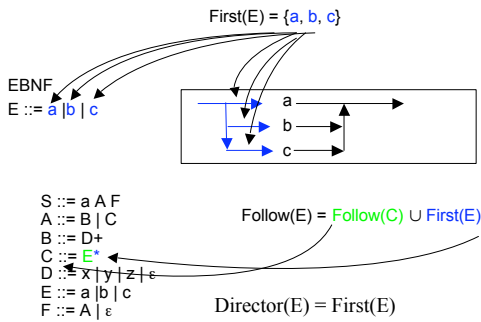
### First and Follow Sets (Cont.)



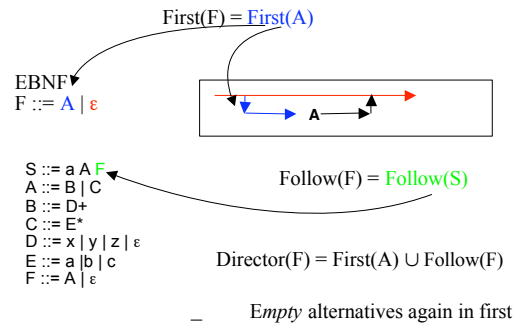
### First and Follow Sets (Cont.)



### First and Follow Sets (Cont.)

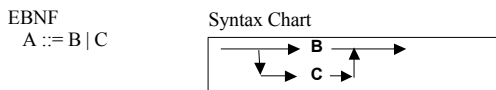


### First and Follow Sets (Cont.)



### Director Set Disjointness Property

- To check for the LL(1) property, write down the first set of each *alternative* in each production
  - For each production, these sets must be disjoint...



Director( $B$ ) = First( $D$ )  $\cup$  Follow( $B$ )  
 = {x, y, z}  $\cup$  Follow( $B$ )  
 = {x, y, z}  $\cup$  follow( $A$ )

Director( $C$ ) = First( $E$ )  $\cup$  Follow( $C$ )  
 = {a, b, c}  $\cup$  Follow( $A$ )

OOPS! Obviously *not* disjoint!  
 Therefore, not LL(1) and we can't deal with this grammar!

### Factoring Grammars..

- We can *sometimes* simply re-arrange the grammar into one that generates an equivalent language to avoid the problem

$A ::= B   C$	First( $A$ ) = first( $B$ ) $\cup$ first( $C$ )
$B ::= ac   bd$	First( $B$ ) = {a, b}
$C ::= ae   bf$	First( $C$ ) = {a, b}

We do this by identifying those terminals which are common (i.e. the intersection of the sets) and factoring these out.

$A' ::= a B'   b C'$	First( $A'$ ) = {a, b}
$B' ::= c   e$	First( $B'$ ) = {c, e}
$C' ::= d   f$	First( $C'$ ) = {d, f}

IMPORTANT: This does not *always* work.  
 Sometimes you need to do this more than once!

## Summary

- When constructing a recursive descent parser for a language, the following steps must be followed:
  - 1) Identify the first and follow sets of each nonterminal symbol.
  - 2) **Prove** that the grammar is LL(1)
    - for each pair of alternatives at a branch point, we need  $\text{director}(A_1) \cap \text{director}(A_2) = \emptyset$
  - 3) If the grammar is not LL(1), then transform it **and goto (1)**.
  - 4) Generate the parser from the grammar.
- The structure of the parser should reflect the structure of the grammar.