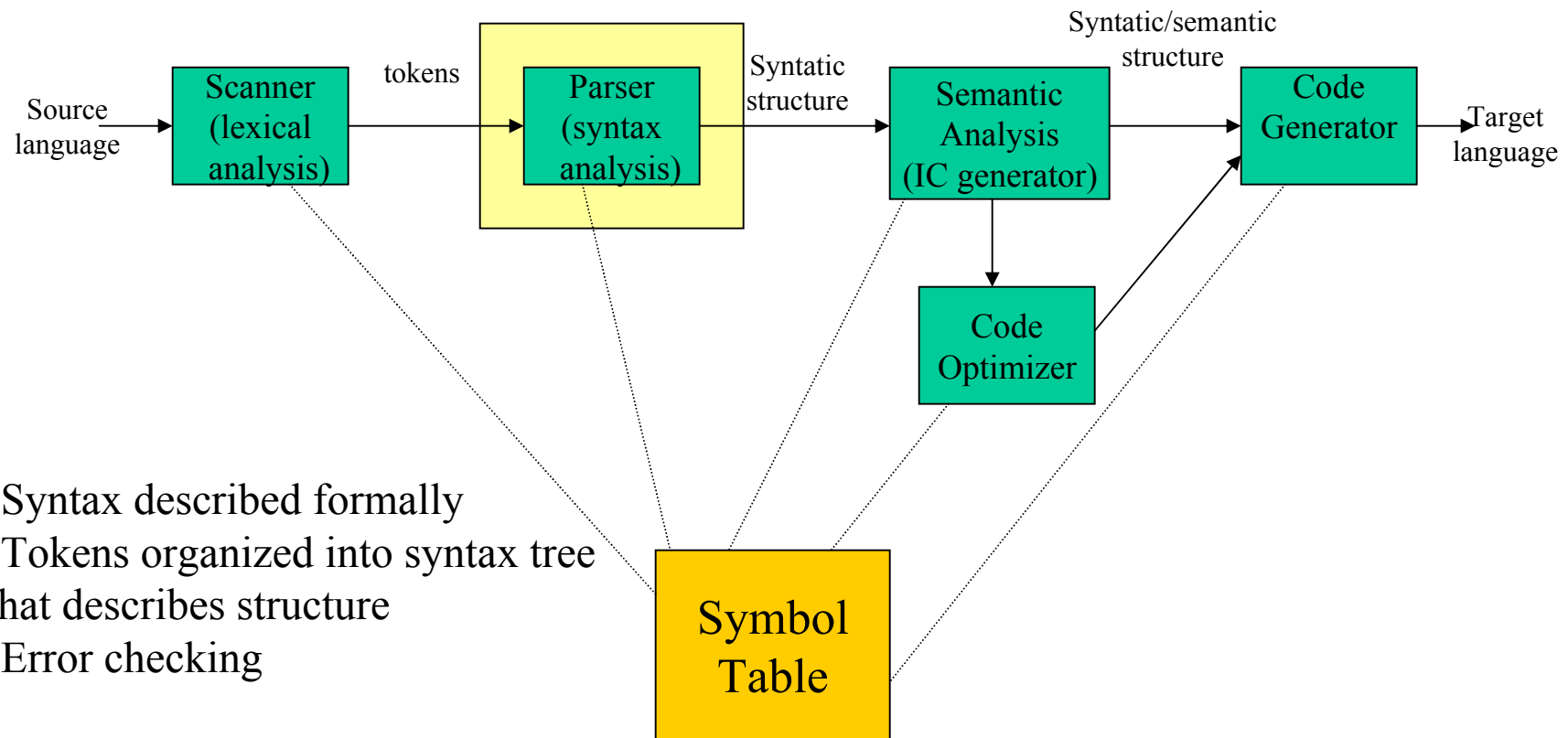


Lecture 3: Parsing

CS 540

George Mason University

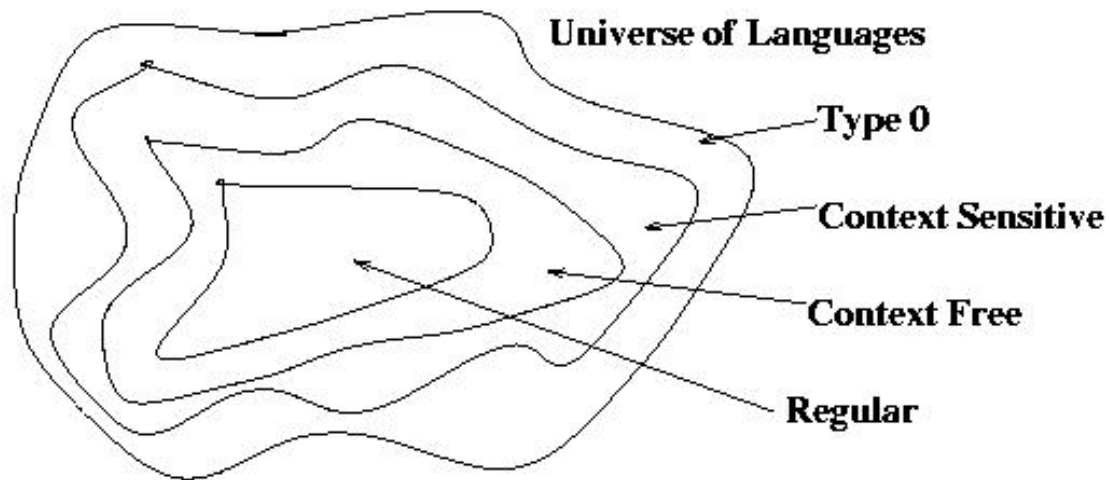
Static Analysis - Parsing



- Syntax described formally
- Tokens organized into syntax tree that describes structure
- Error checking

Static Analysis - Parsing

We can use context free grammars to specify the syntax of programming languages.



Context Free Grammars

Definition: A **context free grammar** is a formal model that consists of:

1. A set of **tokens** or terminal symbols V_t
2. A set of nonterminal symbols V_n
3. Start symbol S (in V_n)
4. Finite set of productions of form:

$$A \rightarrow R_1 \dots R_n \quad (n \geq 0) \text{ where } A \text{ in } V_n, \\ R \text{ in } V_n \cup V_t$$

CFG Examples

Indicates a production

$V_t = \{+, -, 0..9\}, V_n = \{L, D\}, s = \{L\}$

$L \rightarrow L + D \mid L - D \mid D$

$D \rightarrow 0 \mid \dots \mid 9$

Shorthand for multiple productions

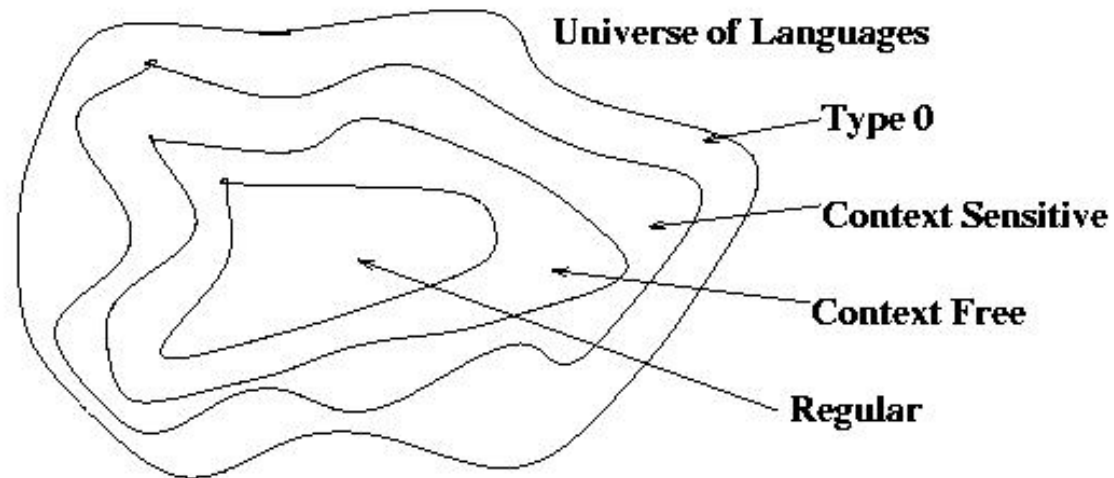
$V_t = \{(\,)\}, V_n = \{L\}, s = \{L\}$

$L \rightarrow (L)L$

$L \rightarrow \varepsilon$

Languages

Regular	$A \rightarrow a B, C \rightarrow \varepsilon$
Context free	$A \rightarrow \alpha$
Context sensitive	$\alpha A \beta \rightarrow \alpha \gamma \beta$
Type 0	$\alpha \rightarrow \beta$



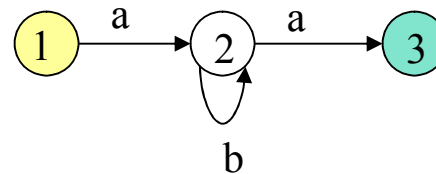
Any regular language can be expressed using a CFG

Starting with a NFA:

- For each state S_i in the NFA
 - Create non-terminal A_i
 - If transition $(S_i, a) = S_k$, create production $A_i \rightarrow a A_k$
 - If transition $(S_i, \epsilon) = S_k$, create production $A_i \rightarrow A_k$
 - If S_i is a final state, create production $A_i \rightarrow \epsilon$
 - If S_i is the NFA start state, $s = A_i$
- *What does the existence of this algorithm tell us about the relationship between regular and context free languages?*

NFA to CFG Example

ab^*a



$A_1 \rightarrow a A_2$

$A_2 \rightarrow b A_2$

$A_2 \rightarrow a A_3$

$A_3 \rightarrow \epsilon$

Writing Grammars

When writing a grammar (or RE) for some language, the following must be true:

1. All strings generated are in the language.
2. Your grammar produces all strings in the language.

Try these:

- Integers divisible by 2
- Legal postfix expressions
- Floating point numbers with no extra zeros
- Strings of 0,1 where there are more 0 than 1

Parsing

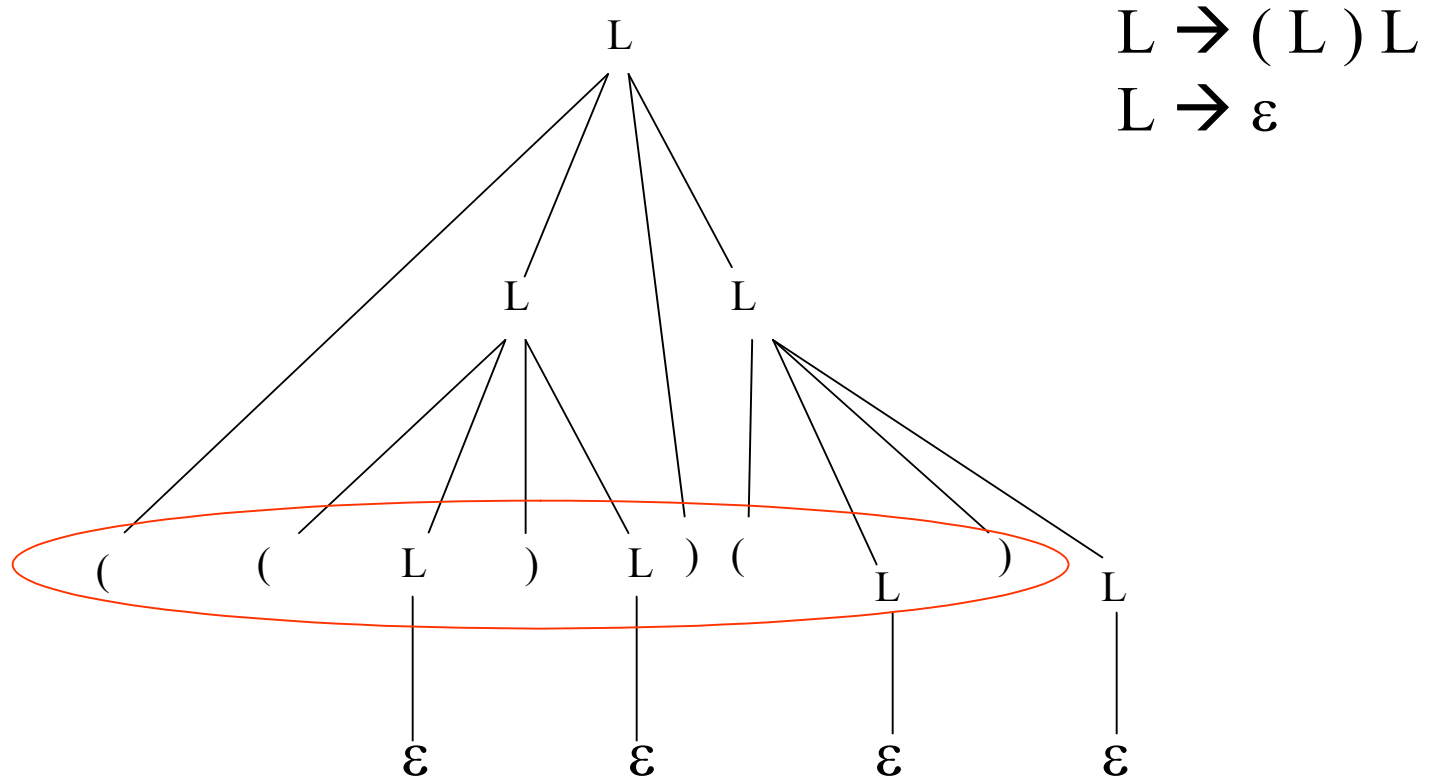
- The task of parsing is figuring out what the **parse tree** looks like for a given input and language.
- If a string is in the given language, a parse tree must exist.
- However, just because a parse tree exists for some string in a given language doesn't mean a given algorithm can find it.

Parse Trees

The parse tree for some string in a language that is defined by the grammar G as follows:

- The root is the start symbol of G
- The leaves are terminals or ϵ . When visited from left to right, the leaves form the input string
- The interior nodes are non-terminals of G
- For every non-terminal A in the tree with children $B_1 \dots B_k$, there is some production $A \rightarrow B_1 \dots B_k$

Parse Tree for $(())()$



Parsing

- Parsing algorithms are based on the idea of *derivations*.
- General Algorithms:
 - LL (top down)
 - LR (bottom up)

Single Step Derivation

Definition: Given $\underline{\alpha} \underline{A} \underline{\beta}$ (with α, β in $(V_n \cup V_t)^*$) and a production $\underline{A} \rightarrow \underline{\gamma}$,
 $\underline{\alpha} \underline{A} \underline{\beta} \Rightarrow \alpha \gamma \beta$ is a **single step derivation**.

Examples:

$$L + D \Rightarrow L - D + D$$

$$L \rightarrow L - D$$

$$(L)(L) \Rightarrow ((L)L)(L)$$

$$L \rightarrow (L)L$$

Greek letters ($\alpha, \beta, \gamma, \dots$) denote a (possibly empty) sequence of terminals and non-terminals.

Derivations

Definition: A sequence of the form:

$$w_0 \Rightarrow w_1 \Rightarrow \dots \Rightarrow w_n$$

is a **derivation** of w_n from w_0 ($w_0 \Rightarrow^* w_n$)

L	production $L \rightarrow (L) L$
$\Rightarrow (L) L$	production $L \rightarrow \varepsilon$
$\Rightarrow () L$	production $L \rightarrow \varepsilon$
$\Rightarrow ()$	

$$L \Rightarrow^* ()$$

If w_i has non-terminal symbols, it is referred to as *sentential form*.

$$L \Rightarrow^* (()) ()$$

L

$\Rightarrow (L) \mathbf{L}$

$\Rightarrow (L) (L) \mathbf{L}$

$\Rightarrow (\mathbf{L}) (L)$

$\Rightarrow ((\mathbf{L}) L) (L)$

$\Rightarrow (() L) (\mathbf{L})$

$\Rightarrow (() \mathbf{L}) ()$

$\Rightarrow (()) ()$

production $L \rightarrow (L) L$

production $L \rightarrow (L) L$

production $L \rightarrow \varepsilon$

production $L \rightarrow (L) L$

production $L \rightarrow \varepsilon$

production $L \rightarrow \varepsilon$

production $L \rightarrow \varepsilon$

- $L(G)$, the language generated by grammar G is $\{w \text{ in } V_t^* : s \Rightarrow^* w, \text{ for start symbol } s\}$
- We've just shown that both $()$ and $((()))$ are in $L(G)$ for the previous grammar.

Leftmost Derivations

- A derivation where the leftmost nonterminal is always chosen
- If a string is in a given language (i.e. a derivation exists), then a leftmost derivation *must* exist
- Rightmost derivation defined as you would expect

Leftmost Derivation for $((()())()$

L	production $L \rightarrow (L)L$
$\Rightarrow (L)L$	production $L \rightarrow (L)L$
$\Rightarrow ((L)L)L$	production $L \rightarrow \varepsilon$
$\Rightarrow ((L)L)L$	production $L \rightarrow \varepsilon$
$\Rightarrow ((L)L)L$	production $L \rightarrow \varepsilon$
$\Rightarrow ((L)L)L$	production $L \rightarrow \varepsilon$
$\Rightarrow ((L)L)L$	production $L \rightarrow (L)L$
$\Rightarrow ((L)L)L$	production $L \rightarrow \varepsilon$
$\Rightarrow ((L)L)L$	

Rightmost Derivation for $((()())()$

L	production $L \rightarrow (L)L$
$\Rightarrow (L)\mathbf{L}$	production $L \rightarrow (L)L$
$\Rightarrow (L)(L)\mathbf{L}$	production $L \rightarrow \varepsilon$
$\Rightarrow (L)(\mathbf{L})$	production $L \rightarrow \varepsilon$
$\Rightarrow (\mathbf{L})()$	production $L \rightarrow (L)L$
$\Rightarrow ((L)\mathbf{L})()$	production $L \rightarrow \varepsilon$
$\Rightarrow ((\mathbf{L}))()$	production $L \rightarrow \varepsilon$
$\Rightarrow (())()$	

Ambiguity

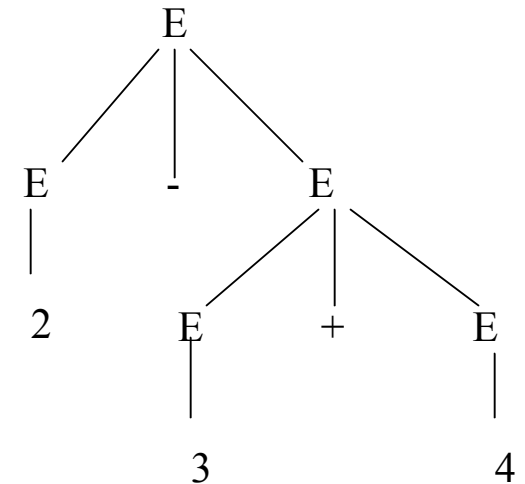
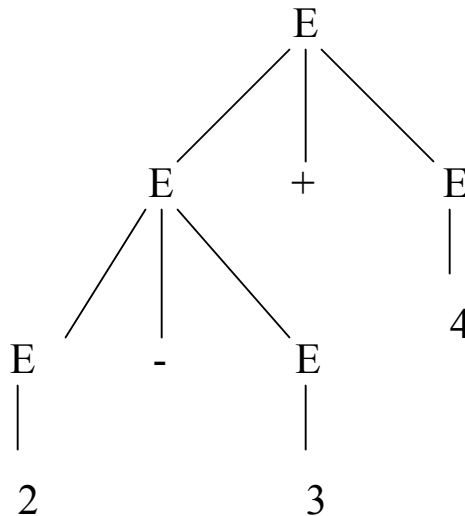
Two (or more) parse trees or leftmost derivations for *some string in the language*

$$E \rightarrow E + E$$

$$E \rightarrow E - E$$

$$E \rightarrow 0 \mid \dots \mid 9$$

2 - 3 + 4



- Two leftmost derivations

$$\begin{aligned} E &\Rightarrow E + E \\ &\Rightarrow E - E + E \\ &\Rightarrow 2 - E + E \\ &\Rightarrow 2 - 3 + E \\ &\Rightarrow 2 - 3 + 4 \end{aligned}$$

$$\begin{aligned} E &\Rightarrow E - E \\ &\Rightarrow 2 - E \\ &\Rightarrow 2 - E + E \\ &\Rightarrow 2 - 3 + E \\ &\Rightarrow 2 - 3 + 4 \end{aligned}$$

- An ambiguous grammar can sometimes be made unambiguous:

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow 0 \mid \dots \mid 9$$

enforces the correct
associativity



- Precedence can be specified as well:

$$E \rightarrow E + T \mid E - T \mid T$$

$$T \rightarrow T * F \mid T / F \mid F$$

$$F \rightarrow (E) \mid 0 \mid \dots \mid 9$$

Input: **begin simplestmt;
simplestmt; end**

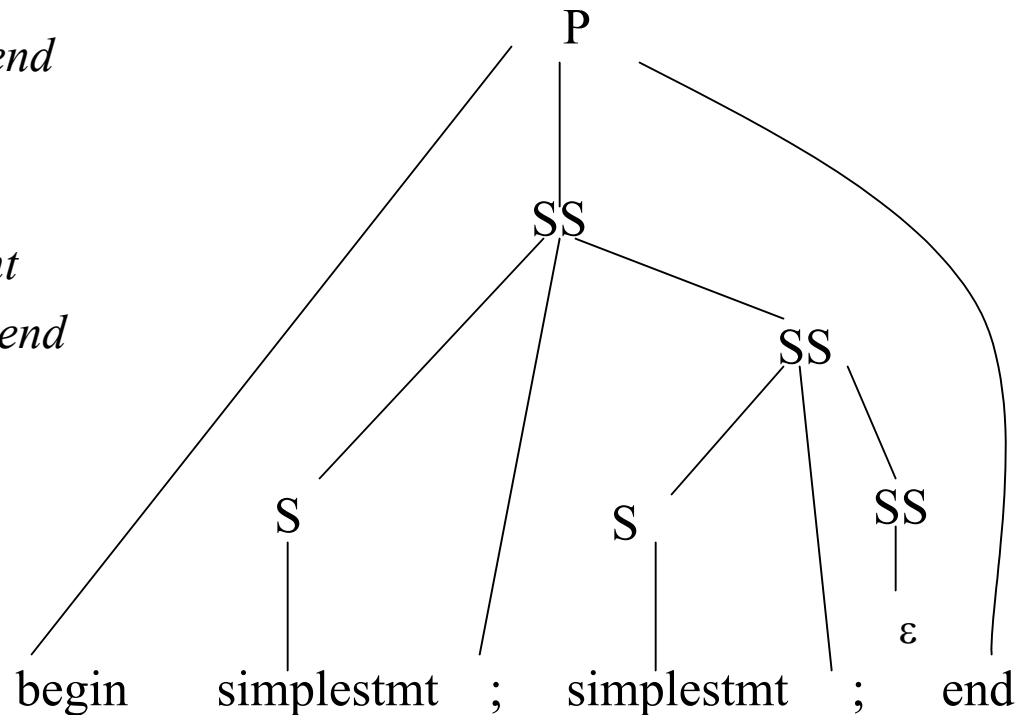
$P \rightarrow \textit{begin SS end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \textit{simplestmt}$

$S \rightarrow \textit{begin SS end}$



Top Down (LL) Parsing

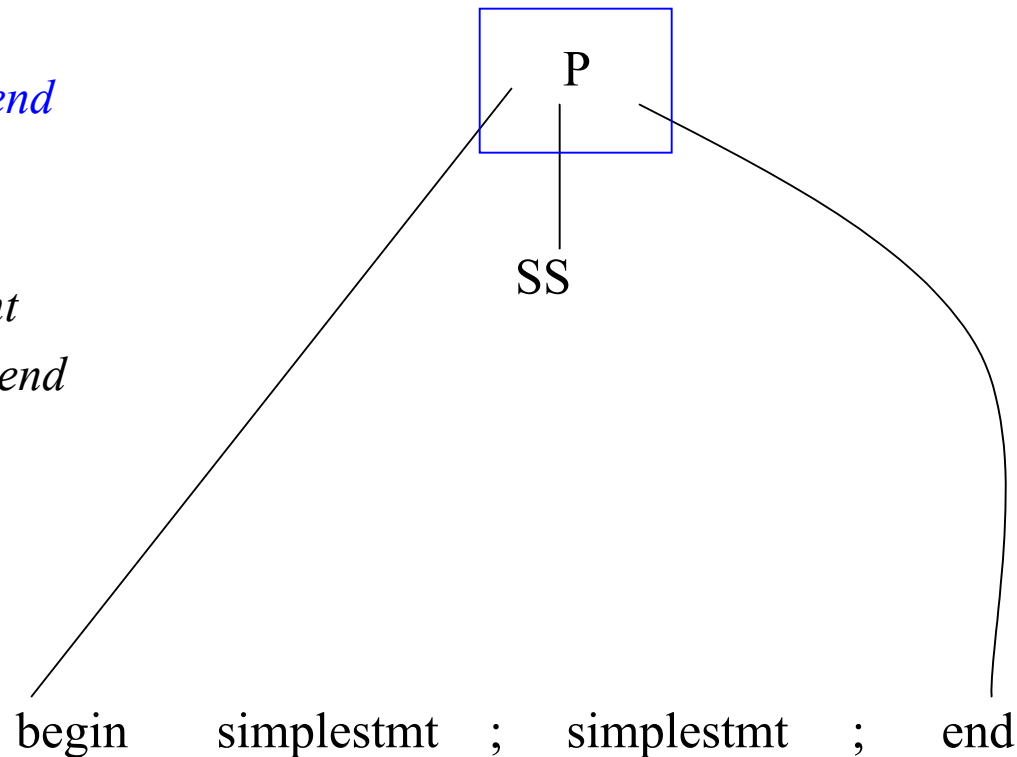
$P \rightarrow \textit{begin SS end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \textit{simplestmt}$

$S \rightarrow \textit{begin SS end}$



Top Down (LL) Parsing

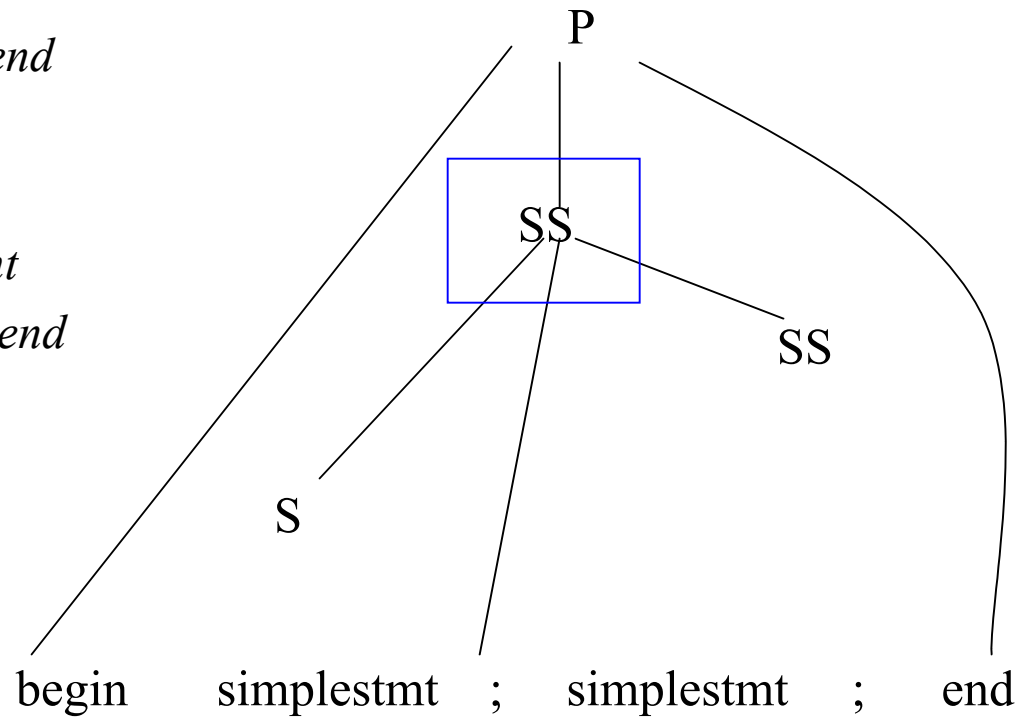
$P \rightarrow \text{begin } SS \text{ end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \text{simplestmt}$

$S \rightarrow \text{begin } SS \text{ end}$



Top Down (LL) Parsing

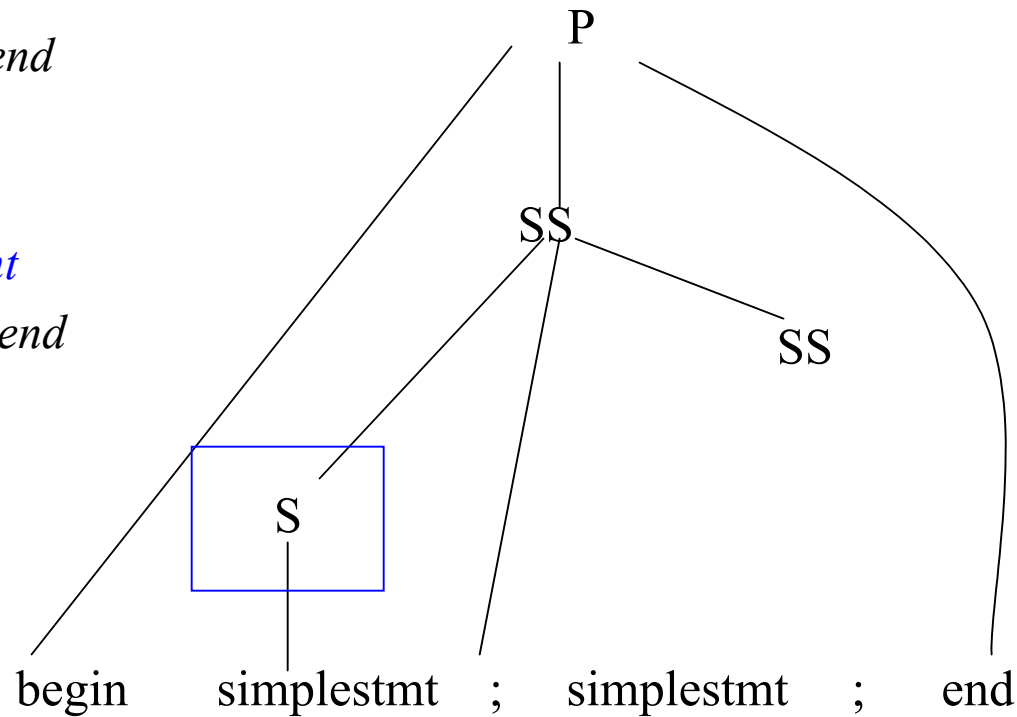
$P \rightarrow \text{begin } SS \text{ end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \text{simplestmt}$

$S \rightarrow \text{begin } SS \text{ end}$



Top Down (LL) Parsing

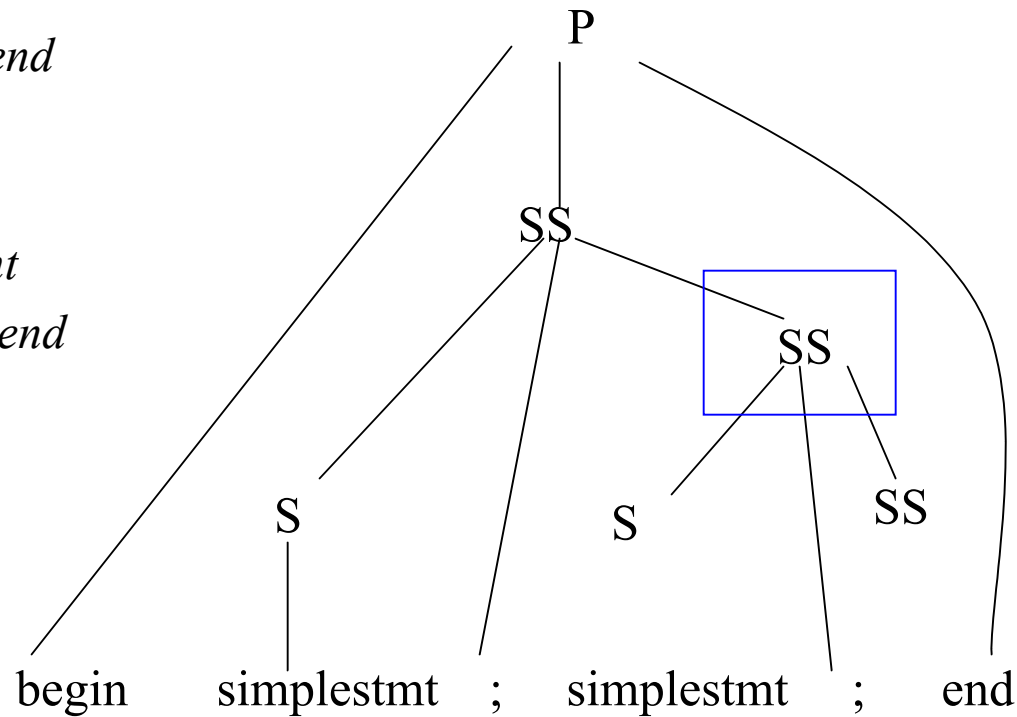
$P \rightarrow \textit{begin SS end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \textit{simplestmt}$

$S \rightarrow \textit{begin SS end}$



Top Down (LL) Parsing

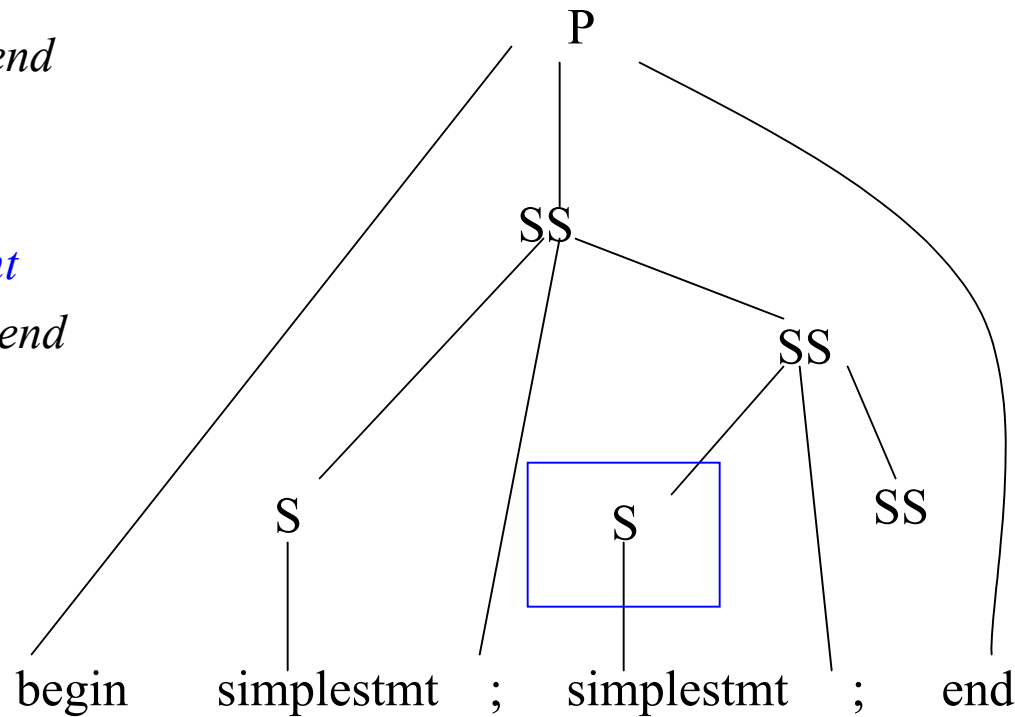
$P \rightarrow \text{begin } SS \text{ end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \text{simplestmt}$

$S \rightarrow \text{begin } SS \text{ end}$



Top Down (LL) Parsing

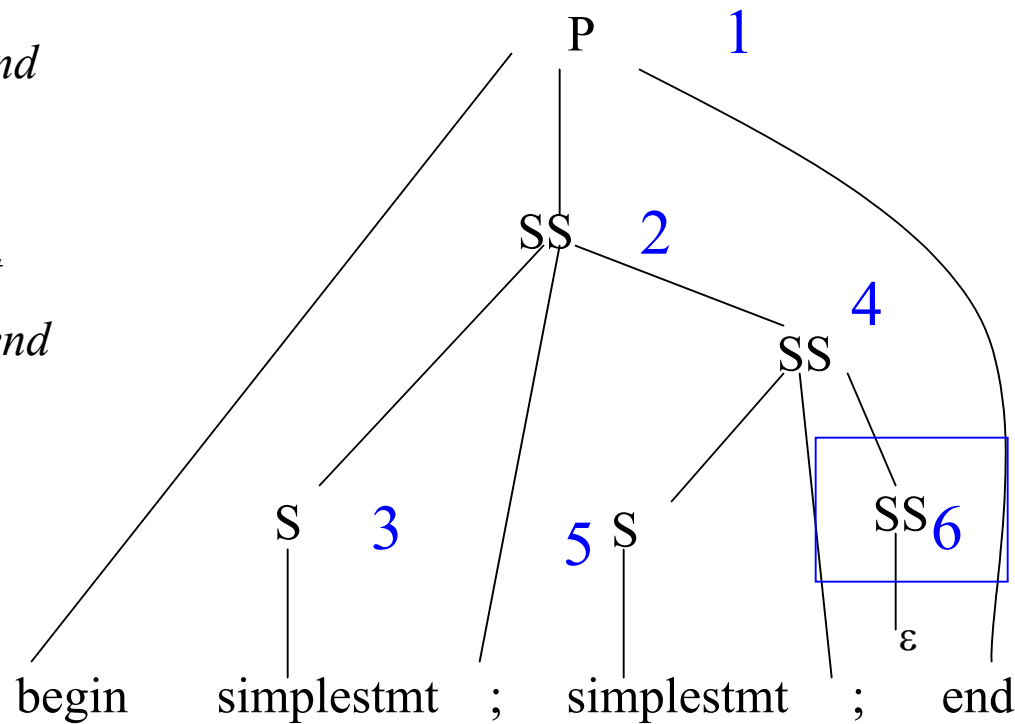
$P \rightarrow \textit{begin SS end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \textit{simplestmt}$

$S \rightarrow \textit{begin SS end}$



Bottomup (LR) Parsing

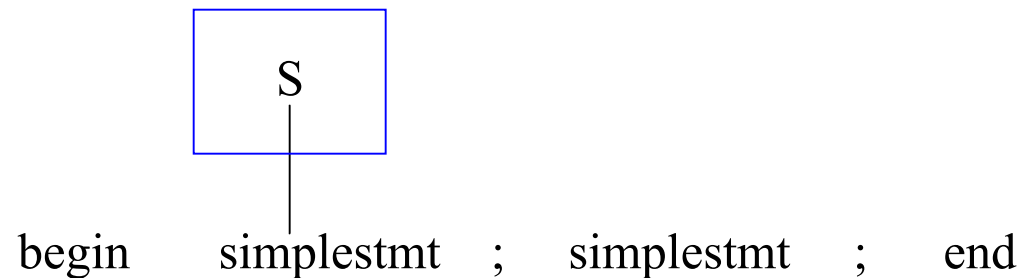
$P \rightarrow \textit{begin SS end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \textit{simplestmt}$

$S \rightarrow \textit{begin SS end}$



Bottomup (LR) Parsing

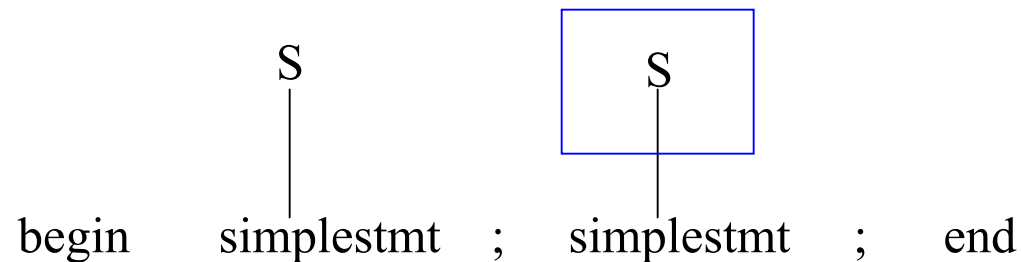
$P \rightarrow \textit{begin SS end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \textit{simplestmt}$

$S \rightarrow \textit{begin SS end}$



Bottomup (LR) Parsing

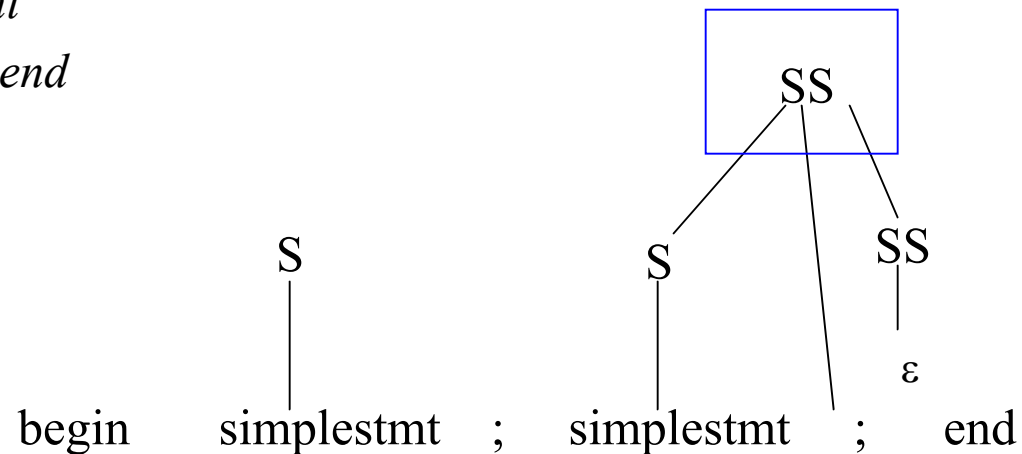
$P \rightarrow \text{begin } SS \text{ end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \text{simplestmt}$

$S \rightarrow \text{begin } SS \text{ end}$



Bottomup (LR) Parsing

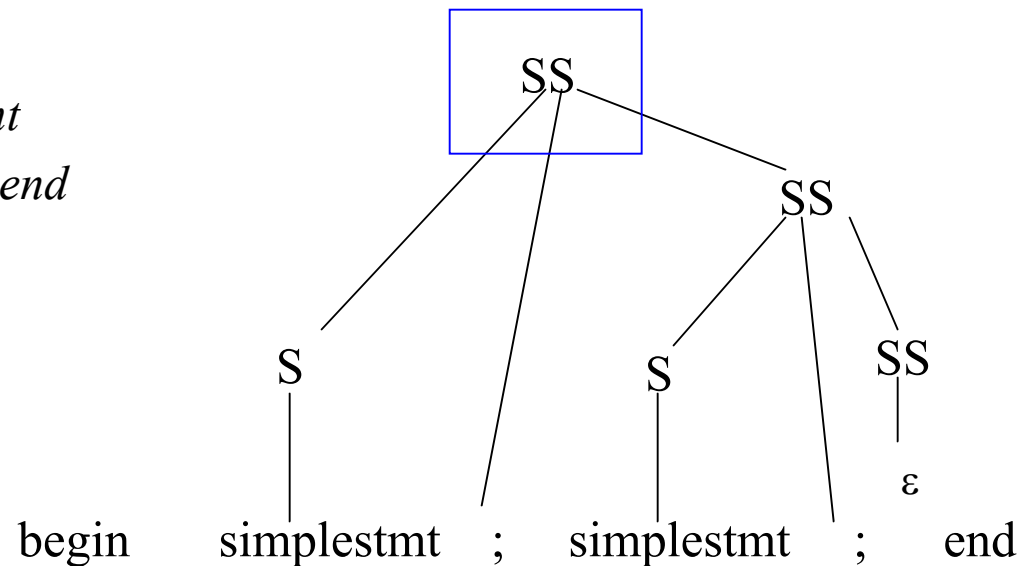
$P \rightarrow \textit{begin SS end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \textit{simplestmt}$

$S \rightarrow \textit{begin SS end}$



Bottomup (LR) Parsing

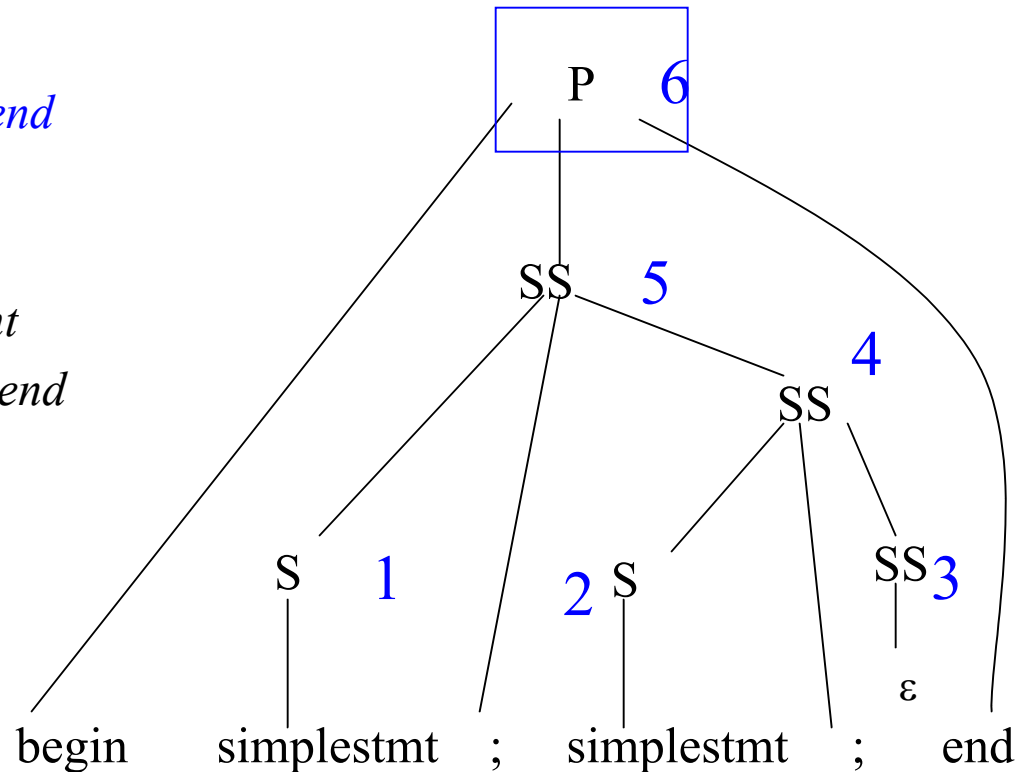
$P \rightarrow \textit{begin SS end}$

$SS \rightarrow S ; SS$

$SS \rightarrow \epsilon$

$S \rightarrow \textit{simplestmt}$

$S \rightarrow \textit{begin SS end}$



Parsing

- General Algorithms:
 - LL (top down)
 - LR (bottom up)
- Both algorithms are driven by the input grammar and the input to be parsed
- Two important sets: FIRST and FOLLOW

FIRST Sets

$\text{FIRST}(\alpha)$ is the set of all terminal symbols that can begin some sentential form in a derivation that starts with α

$$\alpha \Rightarrow \dots \Rightarrow a \beta$$

- $\text{FIRST}(\alpha) = \{a \text{ in } V_t \mid \alpha \Rightarrow^* a\beta\} \cup \{\varepsilon\}$ if $\alpha \Rightarrow^* \varepsilon$

- Example:

$S \rightarrow \text{simple} \mid \text{begin } S \text{ end}$


$\text{FIRST}(S) = \{\text{simple}, \text{begin}\}$

To compute FIRST across strings of terminals and non-terminals:

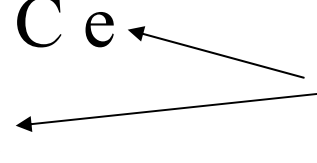
$$\begin{aligned} \text{FIRST}(\varepsilon) &= \{ \varepsilon \} \\ \text{FIRST}(A\alpha) &= \left\{ \begin{array}{l} A \quad \text{if } A \text{ is a terminal} \\ \text{FIRST}(A) \cup \text{FIRST}(\alpha) \\ \quad \text{if } A \Rightarrow^* \varepsilon \\ \text{FIRST}(A) \\ \quad \text{otherwise} \end{array} \right. \end{aligned}$$

Computing FIRST sets

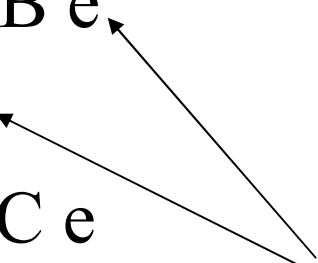
Initially $\text{FIRST}(A)$ (for all A) is empty

1. For productions $A \rightarrow c \beta$, where c in V_t
Add $\{ c \}$ to $\text{FIRST}(A)$
2. For productions $A \rightarrow \varepsilon$
Add $\{ \varepsilon \}$ to $\text{FIRST}(A)$
3. For productions $A \rightarrow \alpha B \beta$, where $\alpha \Rightarrow^* \varepsilon$ and
NOT $(B \Rightarrow^* \varepsilon)$  same thing as
 $\varepsilon \notin \text{FIRST}(B)$
Add $\text{FIRST}(\alpha B)$ to $\text{FIRST}(A)$
4. For productions $A \rightarrow \alpha$, where $\alpha \Rightarrow^* \varepsilon$
Add $\text{FIRST}(\alpha)$ and $\{ \varepsilon \}$ to $\text{FIRST}(A)$

Example 1

- $S \rightarrow a S e$
 - $S \rightarrow B$
 - $B \rightarrow b B e$
 - $B \rightarrow C$
 - $C \rightarrow \underline{c} C e$
 - $C \rightarrow \underline{d}$
- $\text{FIRST}(C) = \{c,d\}$
 - $\text{FIRST}(B) =$
 - $\text{FIRST}(S) =$
- Start with the 'simplest' non-terminal
- 

Example 1

- $S \rightarrow a S e$
 - $S \rightarrow B$
 - $B \rightarrow \underline{b} B e$
 - $B \rightarrow \underline{C}$
 - $C \rightarrow c C e$
 - $C \rightarrow d$
- $\text{FIRST}(C) = \{c, d\}$
 - $\text{FIRST}(B) = \{b, c, d\}$
 - $\text{FIRST}(S) =$
- Now that we know $\text{FIRST}(C) \dots$
- 

Example 1

- $S \rightarrow \underline{a} S e$
- $S \rightarrow \underline{B}$
- $B \rightarrow b B e$
- $B \rightarrow C$
- $C \rightarrow c C e$
- $C \rightarrow d$
- $\text{FIRST}(C) = \{c,d\}$
- $\text{FIRST}(B) = \{b,c,d\}$
- $\text{FIRST}(S) = \{a,b,c,d\}$

Example 2

- $P \rightarrow \underline{i} \mid \underline{c} \mid \underline{n} T S$
- $Q \rightarrow P \mid a S \mid d S c S T$
- $R \rightarrow \underline{b} \mid \underline{\epsilon}$
- $S \rightarrow e \mid R n \mid \epsilon$
- $T \rightarrow R S q$
- $\text{FIRST}(P) = \{i, c, n\}$
- $\text{FIRST}(Q) =$
- $\text{FIRST}(R) = \{b, \epsilon\}$
- $\text{FIRST}(S) =$
- $\text{FIRST}(T) =$

Example 2

- $P \rightarrow i \mid c \mid n T S$
- $Q \rightarrow \underline{P} \mid \underline{a} S \mid \underline{d} S c S T$
- $R \rightarrow b \mid \varepsilon$
- $S \rightarrow e \mid R n \mid \varepsilon$
- $T \rightarrow R S q$
- $\text{FIRST}(P) = \{i, c, n\}$
- $\text{FIRST}(Q) = \{i, c, n, a, d\}$
- $\text{FIRST}(R) = \{b, \varepsilon\}$
- $\text{FIRST}(S) =$
- $\text{FIRST}(T) =$

Example 2

- $P \rightarrow i \mid c \mid n T S$
- $Q \rightarrow P \mid a S \mid d S c S T$
- $R \rightarrow b \mid \varepsilon$
- $S \rightarrow \underline{e} \mid \underline{R n} \mid \underline{\varepsilon}$
- $T \rightarrow R S q$
- $\text{FIRST}(P) = \{i, c, n\}$
- $\text{FIRST}(Q) = \{i, c, n, a, d\}$
- $\text{FIRST}(R) = \{b, \varepsilon\}$
- $\text{FIRST}(S) = \{e, b, n, \varepsilon\}$
- $\text{FIRST}(T) =$

Note:

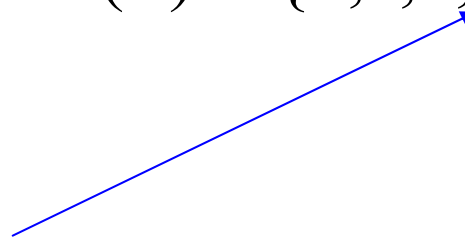
$S \Rightarrow R n \Rightarrow n$ because $R \Rightarrow^* \varepsilon$

Example 2

- $P \rightarrow i \mid c \mid n T S$
- $Q \rightarrow P \mid a S \mid d S c S T$
- $R \rightarrow b \mid \varepsilon$
- $S \rightarrow e \mid R n \mid \varepsilon$
- $T \rightarrow \underline{R S} q$
- $\text{FIRST}(P) = \{i, c, n\}$
- $\text{FIRST}(Q) = \{i, c, n, a, d\}$
- $\text{FIRST}(R) = \{b, \varepsilon\}$
- $\text{FIRST}(S) = \{e, b, n, \varepsilon\}$
- $\text{FIRST}(T) = \{b, c, n, q\}$

Note:

$T \Rightarrow R S q \Rightarrow S q \Rightarrow q$
because both R and $S \Rightarrow^* \varepsilon$



Example 3

- $S \rightarrow a S e \mid S T S$
- $T \rightarrow R S e \mid Q$
- $R \rightarrow r S r \mid \varepsilon$
- $Q \rightarrow S T \mid \varepsilon$
- $\text{FIRST}(S) =$
- $\text{FIRST}(R) =$
- $\text{FIRST}(T) =$
- $\text{FIRST}(Q) =$

Example 3

- $S \rightarrow a S e \mid S T S$
- $T \rightarrow R S e \mid Q$
- $R \rightarrow r S r \mid \varepsilon$
- $Q \rightarrow S T \mid \varepsilon$
- $\text{FIRST}(S) = \{a\}$
- $\text{FIRST}(R) = \{r, \varepsilon\}$
- $\text{FIRST}(T) = \{r, a, \varepsilon\}$
- $\text{FIRST}(Q) = \{a, \varepsilon\}$

FOLLOW Sets

- FOLLOW(A) is the set of terminals (including end of file - \$) that may follow non-terminal A in some sentential form.
- $\text{FOLLOW}(A) = \{c \text{ in } V_t \mid S \Rightarrow^+ \dots Ac\dots\} \cup \{\$\}$ if $S \Rightarrow^+ \dots A$
- For example, consider $L \Rightarrow^+ (())(L)L$
Both ‘)’ and end of file can follow L
- NOTE: ε is *never* in FOLLOW sets

Computing FOLLOW(A)

1. If A is start symbol, put \$ in FOLLOW(A)
 2. Productions of the form $B \rightarrow \alpha A \beta$,
Add $\text{FIRST}(\beta) - \{\epsilon\}$ to FOLLOW(A)
-

INTUITION: Suppose $B \rightarrow AX$ and $\text{FIRST}(X) = \{c\}$

$S \Rightarrow^+ \alpha B \beta \Rightarrow \alpha A X \beta \Rightarrow^+ \alpha A c \delta \beta$
= FIRST(X)

3. Productions of the form $B \rightarrow \alpha A$ or

$B \rightarrow \alpha A \beta$ where $\beta \Rightarrow^* \varepsilon$

Add FOLLOW(B) to FOLLOW(A)

INTUITION:

– Suppose $B \rightarrow Y A$

$S \Rightarrow^+ \alpha B \beta \Rightarrow \alpha Y A \beta$

FOLLOW(B)



– Suppose $B \rightarrow A X$ and $X \Rightarrow^* \varepsilon$

$S \Rightarrow^+ \alpha B \beta \Rightarrow \alpha A X \beta \Rightarrow^* \alpha A \beta$

FOLLOW(B)



Assume the first non-terminal is
the start symbol

Example 4

- $S \rightarrow a S e \mid B$
- $B \rightarrow b B C f \mid C$
- $C \rightarrow c C g \mid d \mid \varepsilon$
- $\text{FIRST}(C) = \{c, d, \varepsilon\}$
- $\text{FIRST}(B) = \{b, c, d, \varepsilon\}$
- $\text{FIRST}(S) = \{a, b, c, d, \varepsilon\}$
- $\text{FOLLOW}(C) =$
- $\text{FOLLOW}(B) =$
- $\text{FOLLOW}(S) = \{\$ \}$

Using rule #1

Example 4

- $S \rightarrow a \underline{S} e \mid B$
- $B \rightarrow b \underline{B} \underline{C} f \mid C$
- $C \rightarrow c \underline{C} g \mid d \mid \varepsilon$
- $\text{FOLLOW}(C) = \{f, g\}$
- $\text{FOLLOW}(B) = \{c, d, f\}$
- $\text{FIRST}(C) = \{c, d, \varepsilon\}$
- $\text{FIRST}(B) = \{b, c, d, \varepsilon\}$
- $\text{FIRST}(S) = \{a, b, c, d, \varepsilon\}$
- $\text{FOLLOW}(S) = \{\$, e\}$

Using rule #2

Example 4

- $S \rightarrow a S e \mid \underline{B}$
- $B \rightarrow b B C f \mid \underline{C}$
- $C \rightarrow c C g \mid d \mid \varepsilon$
- $\text{FIRST}(C) = \{c, d, \varepsilon\}$
- $\text{FIRST}(B) = \{b, c, d, \varepsilon\}$
- $\text{FIRST}(S) = \{a, b, c, d, \varepsilon\}$
- $\text{FOLLOW}(C) = \{f, g\} \cup \text{FOLLOW}(B) = \{c, d, e, f, g, \$\}$
- $\text{FOLLOW}(B) = \{c, d, f\} \cup \text{FOLLOW}(S) = \{c, d, e, f, \$\}$
- $\text{FOLLOW}(S) = \{\$, e\}$

Using rule #3

Example 5

- $S \rightarrow A B C \mid A D$
- $A \rightarrow \varepsilon \mid a A$
- $B \rightarrow b \mid c \mid \varepsilon$
- $C \rightarrow D d C$
- $D \rightarrow e b \mid f c$

- $FIRST(D) = \{e, f\}$
- $FIRST(C) = \{e, f\}$
- $FIRST(B) = \{b, c, \varepsilon\}$
- $FIRST(A) = \{a, \varepsilon\}$
- $FIRST(S) = \{a, b, c, e, f\}$

- $FOLLOW(S) =$
- $FOLLOW(A) =$
- $FOLLOW(B) =$
- $FOLLOW(C) =$
- $FOLLOW(D) =$

Example 5

- $S \rightarrow A B C \mid A D$
- $A \rightarrow \varepsilon \mid a A$
- $B \rightarrow b \mid c \mid \varepsilon$
- $C \rightarrow D d C$
- $D \rightarrow e b \mid f c$

- $\text{FOLLOW}(S) = \{\$ \}$
- $\text{FOLLOW}(A) = \{b, c, e, f\}$
- $\text{FOLLOW}(B) = \{e, f\}$
- $\text{FOLLOW}(C) = \{\$ \}$
- $\text{FOLLOW}(D) = \{\$ \}$

- $\text{FIRST}(D) = \{e, f\}$
- $\text{FIRST}(C) = \{e, f\}$
- $\text{FIRST}(B) = \{b, c, \varepsilon\}$
- $\text{FIRST}(A) = \{a, \varepsilon\}$
- $\text{FIRST}(S) = \{a, b, c, e, f\}$

Example 6

- $S \rightarrow (A) \mid \varepsilon$
- $A \rightarrow T E$
- $E \rightarrow \& T E \mid \varepsilon$
- $T \rightarrow (A) \mid a \mid b \mid c$
- $\text{FOLLOW}(S) =$
- $\text{FOLLOW}(A) =$
- $\text{FOLLOW}(E) =$
- $\text{FOLLOW}(T) =$
- $\text{FIRST}(T) = \{(, a, b, c\}$
- $\text{FIRST}(E) = \{\&, \varepsilon\}$
- $\text{FIRST}(A) = \{(, a, b, c\}$
- $\text{FIRST}(S) = \{(, \varepsilon\}$

Example 6

- $S \rightarrow (A) \mid \varepsilon$
- $A \rightarrow T E$
- $E \rightarrow \& T E \mid \varepsilon$
- $T \rightarrow (A) \mid a \mid b \mid c$
- $\text{FIRST}(T) = \{ (, a, b, c \}$
- $\text{FIRST}(E) = \{ \&, \varepsilon \}$
- $\text{FIRST}(A) = \{ (, a, b, c \}$
- $\text{FIRST}(S) = \{ (, \varepsilon \}$
- $\text{FOLLOW}(S) = \{ \$ \}$
- $\text{FOLLOW}(A) = \{) \}$
- $\text{FOLLOW}(E) =$
 $\text{FOLLOW}(A) = \{) \}$
- $\text{FOLLOW}(T) =$
 $\text{FIRST}(E) \cup \text{FOLLOW}(A) \cup$
 $\text{FOLLOW}(E) = \{ \&,) \}$

Example 7

- $E \rightarrow T E'$
 - $E' \rightarrow + T E' \mid \varepsilon$
 - $T \rightarrow F T'$
 - $T' \rightarrow * F T' \mid \varepsilon$
 - $F \rightarrow (E) \mid id$
- FOLLOW(E) =
 - FOLLOW(E') =
 - FOLLOW(T) =
 - FOLLOW(T') =
 - FOLLOW(F) =
- FIRST(F) = FIRST(T) = FIRST(E) =
{(,id}
 - FIRST(T') = {*, ε }
 - FIRST(E') = {+, ε }

Example 7

- $E \rightarrow T E'$
 - $E' \rightarrow + T E' \mid \varepsilon$
 - $T \rightarrow F T'$
 - $T' \rightarrow * F T' \mid \varepsilon$
 - $F \rightarrow (E) \mid id$
- $FOLLOW(E) = \{\$, \}$
 - $FOLLOW(E') = FOLLOW(E) = \{\$, \}$
 - $FOLLOW(T) = FIRST(E') \cup FOLLOW(E) \cup FOLLOW(E') = \{+, \$, \}$
 - $FOLLOW(T') = FOLLOW(T) = \{+, \$, \}$
 - $FOLLOW(F) = FIRST(T') \cup FOLLOW(T) \cup FOLLOW(T') = \{*, +, \$, \}$
- $FIRST(F) = FIRST(T) = FIRST(E) = \{(, id\}$
 - $FIRST(T') = \{*, \varepsilon\}$
 - $FIRST(E') = \{+, \varepsilon\}$