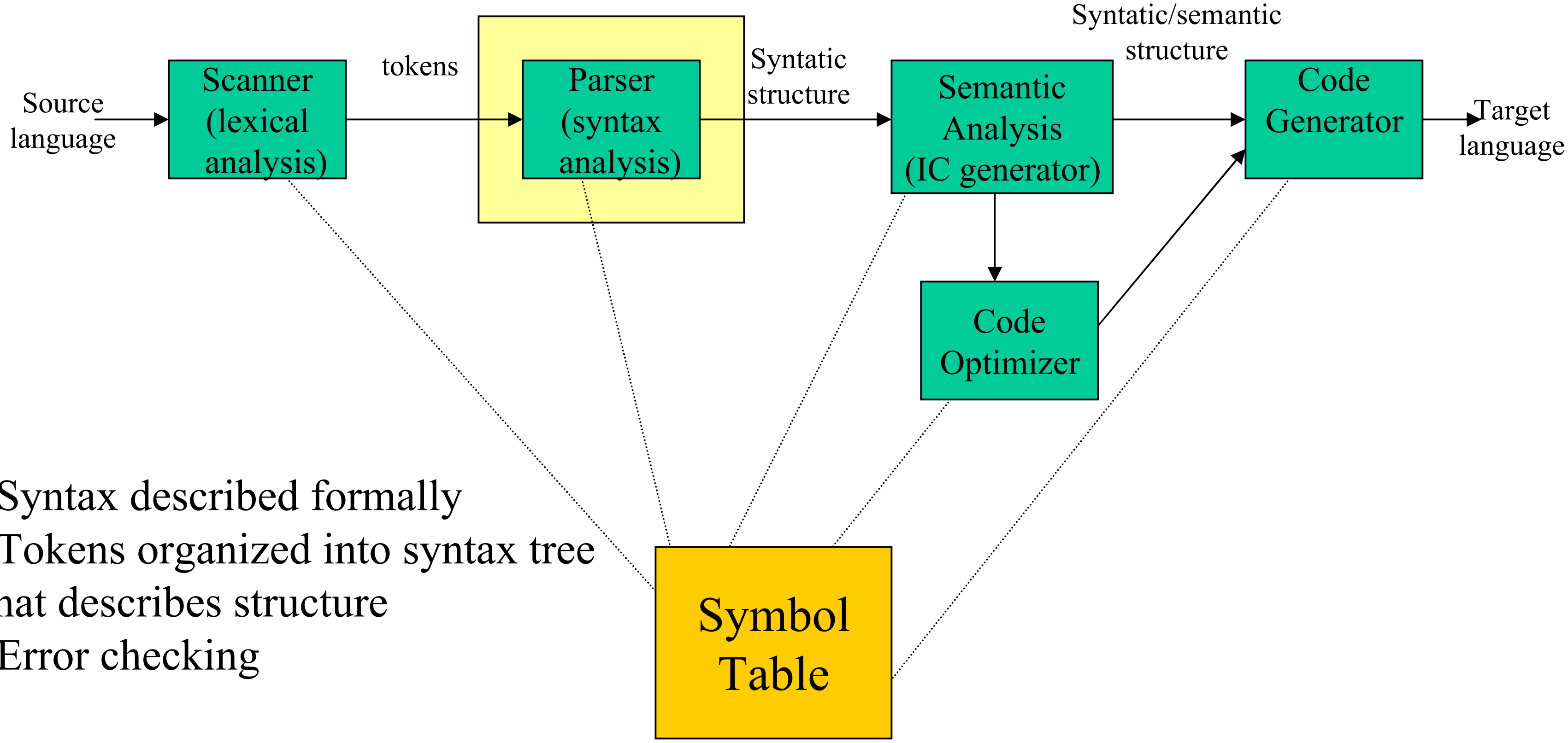
Lecture 3: Parsing

CS 540

George Mason University

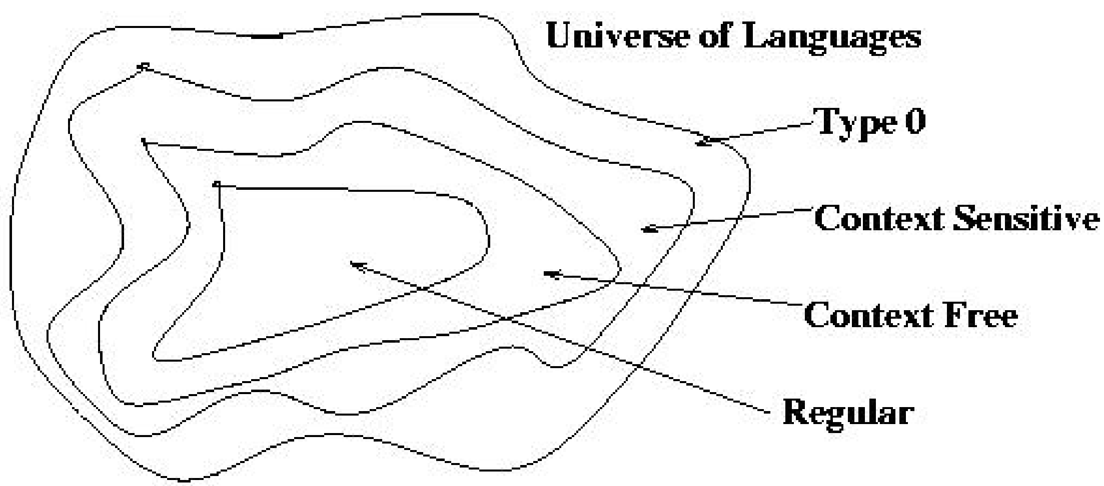
Static Analysis - Parsing



- 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:

   $A \rightarrow R_1 \ldots R_n$ ($n >= 0$) where A in V_n,

   R in $V_n \cup V_t$

CFG Examples

Indicates a production

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

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

$D \rightarrow 0 \mid \ldots \mid 9$

Shorthand for multiple productions

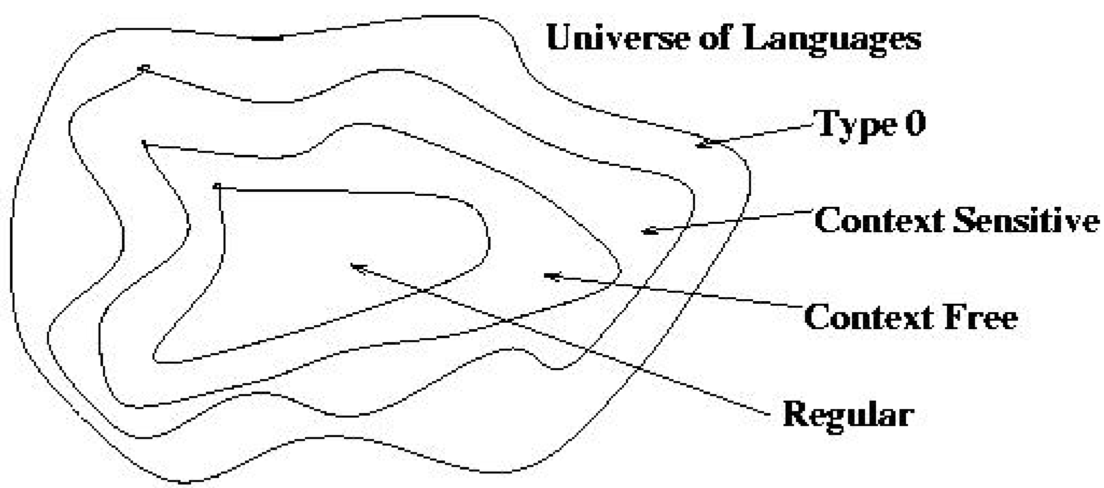
$V_t=\{(,)\}$, $V_n = \{L\}$, $s = \{L\}$

$L \rightarrow (L) L$

$L \rightarrow \varepsilon$

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	$\alpha \rightarrow \beta$



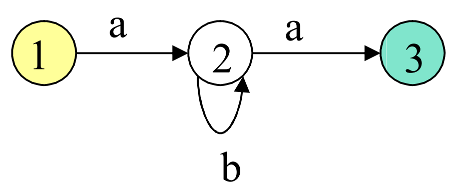
Any regular language can be expressed using a CFG

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 $(S_i,\varepsilon) = S_k$, create production $A_i \rightarrow A_k$
  - If S_i is a final state, create production $A_i \rightarrow \varepsilon$
  - 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



$A_1 \rightarrow a\ A_2$

$A_2 \rightarrow b\ A_2$

$A_2 \rightarrow a\ A_3$

$A_3 \rightarrow \varepsilon$

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.

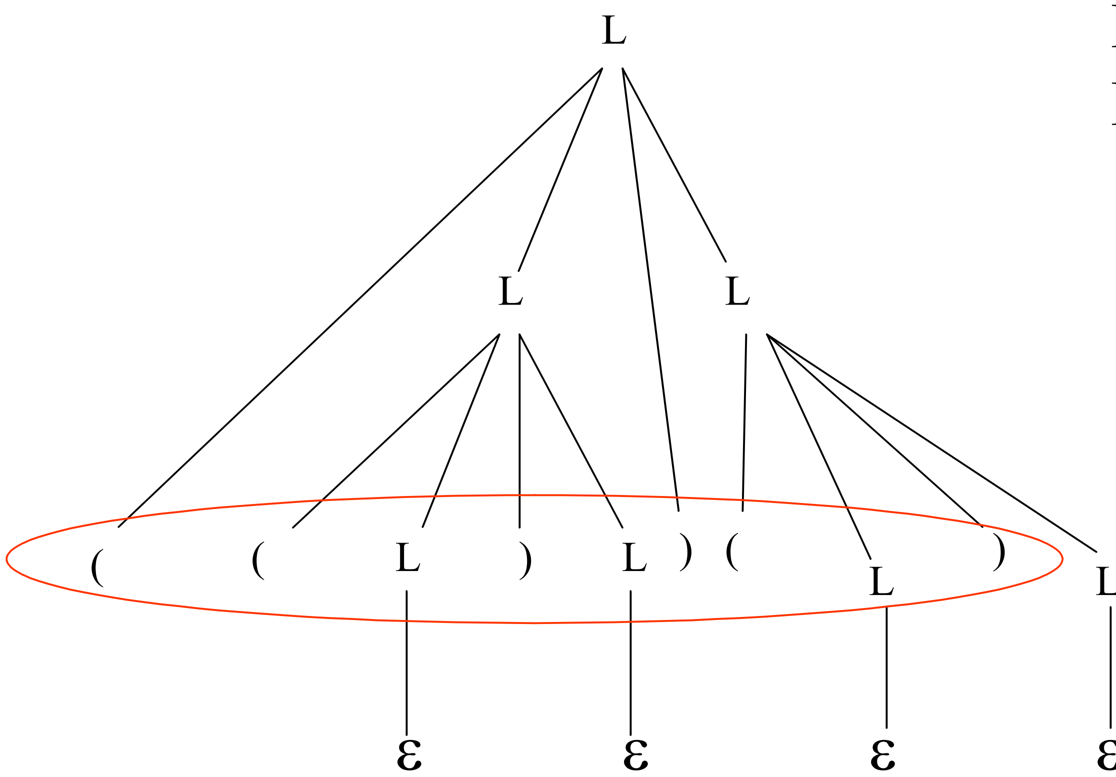
Parse Trees

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 $A \rightarrow B_1 \dots B_k$

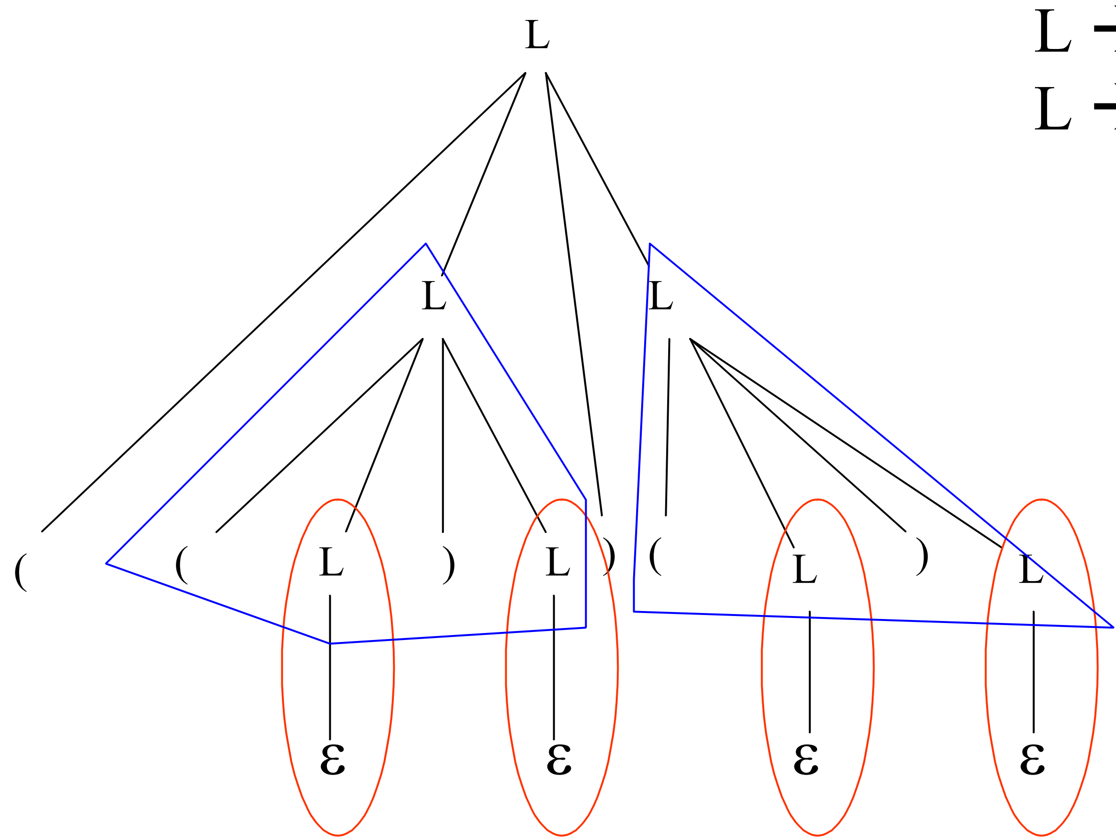
Parse Tree for (())()

$L \rightarrow (L) L$
$L \rightarrow \varepsilon$



Parse Tree for (())()

$L \rightarrow (L) L$
$L \rightarrow \varepsilon$



Parsing

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

Single Step Derivation

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

$\alpha\ A\ \beta \Rightarrow \alpha\ \gamma\ \beta$ is a **single step derivation**.

Examples:

$L + D \Rightarrow L - D + D$ $\qquad L \rightarrow L - D$

$(L) (L) \Rightarrow ((L) L) (L)$ $\qquad L \rightarrow (L) L$

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

Derivations

Definition: A sequence of the form:

$w_0 \Rightarrow w_1 \Rightarrow \ldots \Rightarrow w_n$

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

L	production L → (L) L
⇒ (L) L	production L → ε
⇒ ( ) L	production L → ε
⇒ ( )	

$L \Rightarrow^* ()$

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

$L \Rightarrow^{*} ((\,))\,(\,)$

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

- L(G), the language generated by grammar G is {w in V_t^*: s $\Rightarrow^*$ w, 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 → (L) L
⇒ (**L**) L	production L → (L) L
⇒ ((**L**) L) L	production L → ε
⇒ (( ) **L**) L	production L → ε
⇒ (( )) **L**	production L → ε
⇒ (( )) (**L**) L	production L →(L) L
⇒ (( )) ( ) **L**	production L → ε
⇒ (( )) ( )	

Rightmost Derivation for (())()

L	production L → (L) L
⇒ (L) **L**	production L → (L) L
⇒ (L) (L) **L**	production L → ε
⇒ (L) (**L**)	production L → ε
⇒ (**L**) ( )	production L → (L) L
⇒ ((L) **L**) ( )	production L → ε
⇒ ((**L**)) ( )	production L → ε
⇒ (( )) ( )	

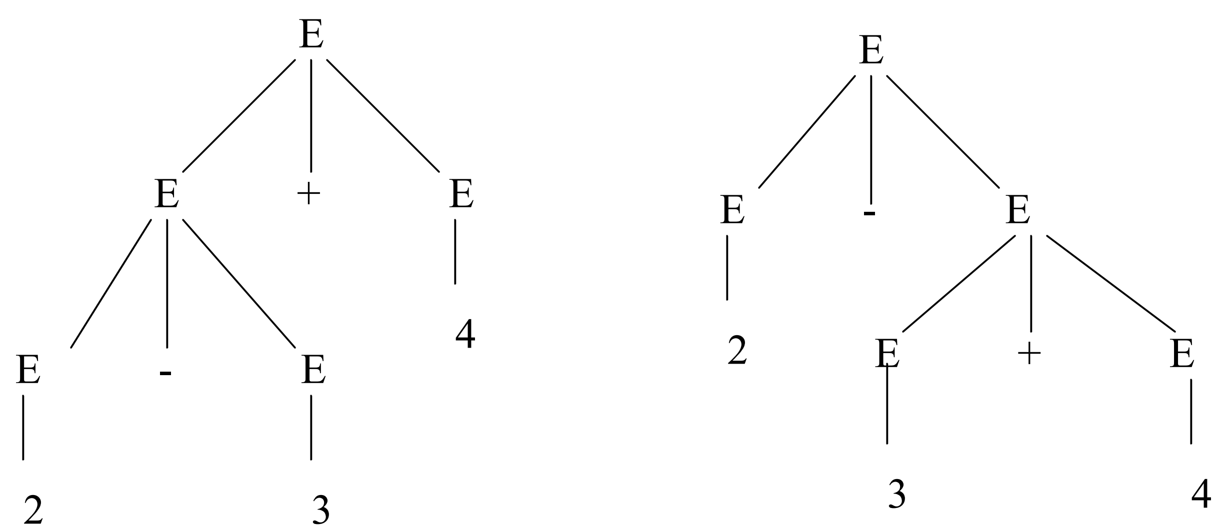
Ambiguity

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

$E \rightarrow E + E$

$E \rightarrow E - E$

$E \rightarrow 0 \mid \ldots \mid 9$



$2 - 3 + 4$

- Two leftmost derivations

E	$\Rightarrow$	E + E	E	$\Rightarrow$	E – E
	$\Rightarrow$	E – E + E		$\Rightarrow$	2 – E
	$\Rightarrow$	2 – E + E		$\Rightarrow$	2 – E + E
	$\Rightarrow$	2 – 3 + E		$\Rightarrow$	2 – 3 + E
	$\Rightarrow$	2 – 3 + 4		$\Rightarrow$	2 – 3 + 4

- An ambiguous grammar can sometimes be made unambiguous:

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

$T \rightarrow 0 \mid \ldots \mid 9$

enforces the correct associativity

- Precedence can be specified as well:

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

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

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

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

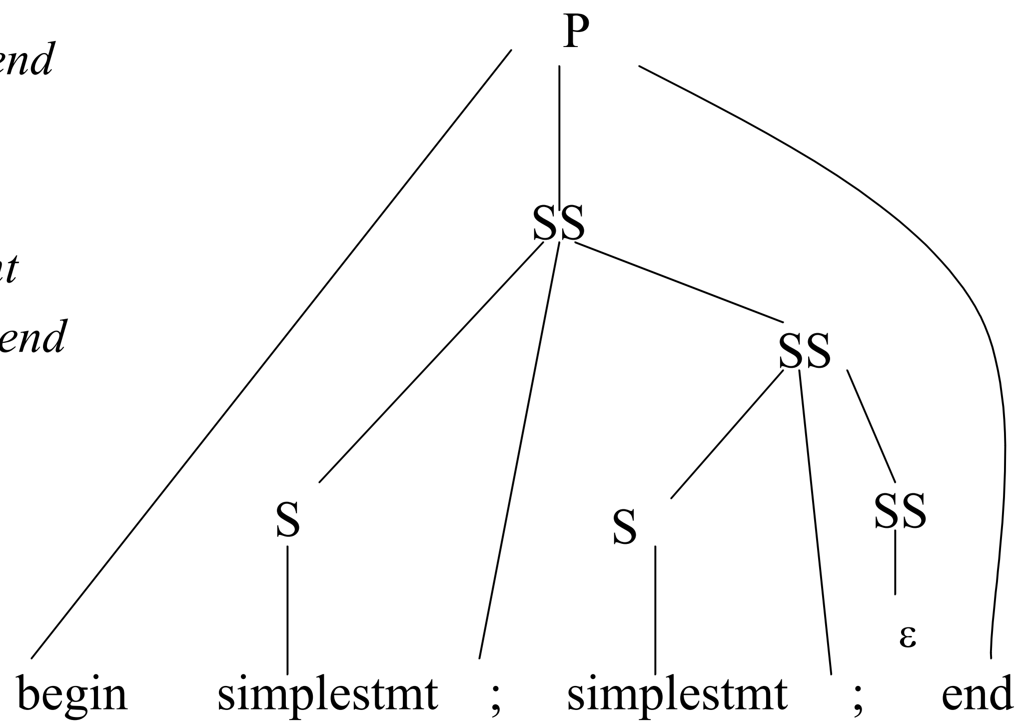
P → *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Top Down (LL) Parsing

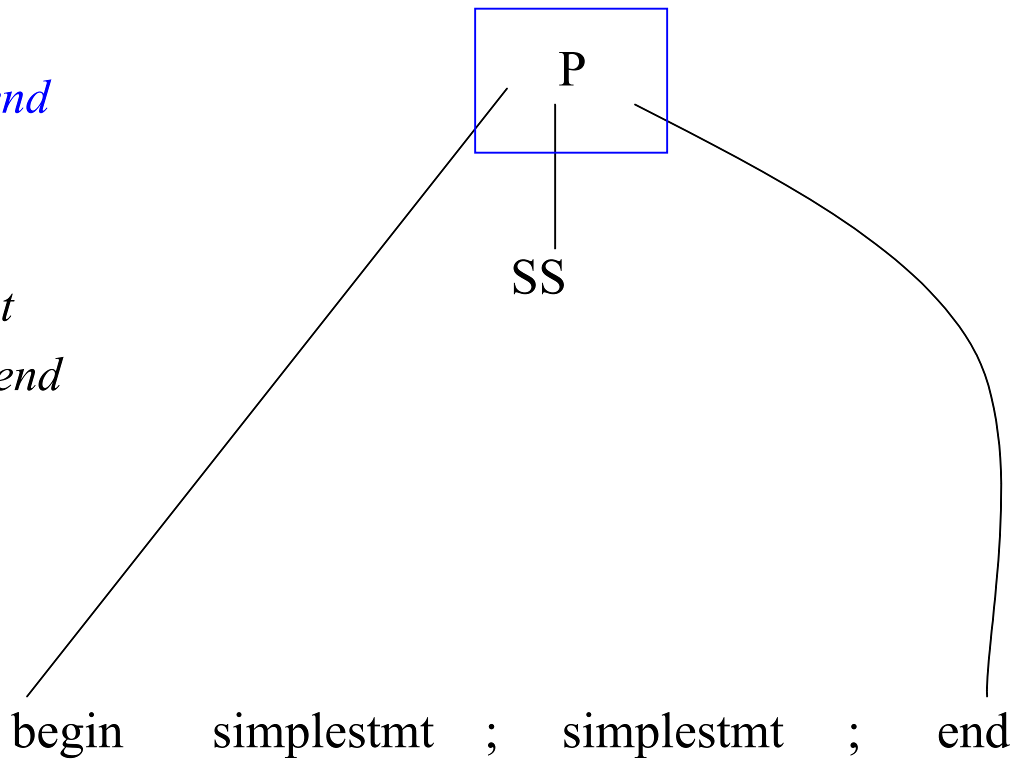
P → *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Top Down (LL) Parsing

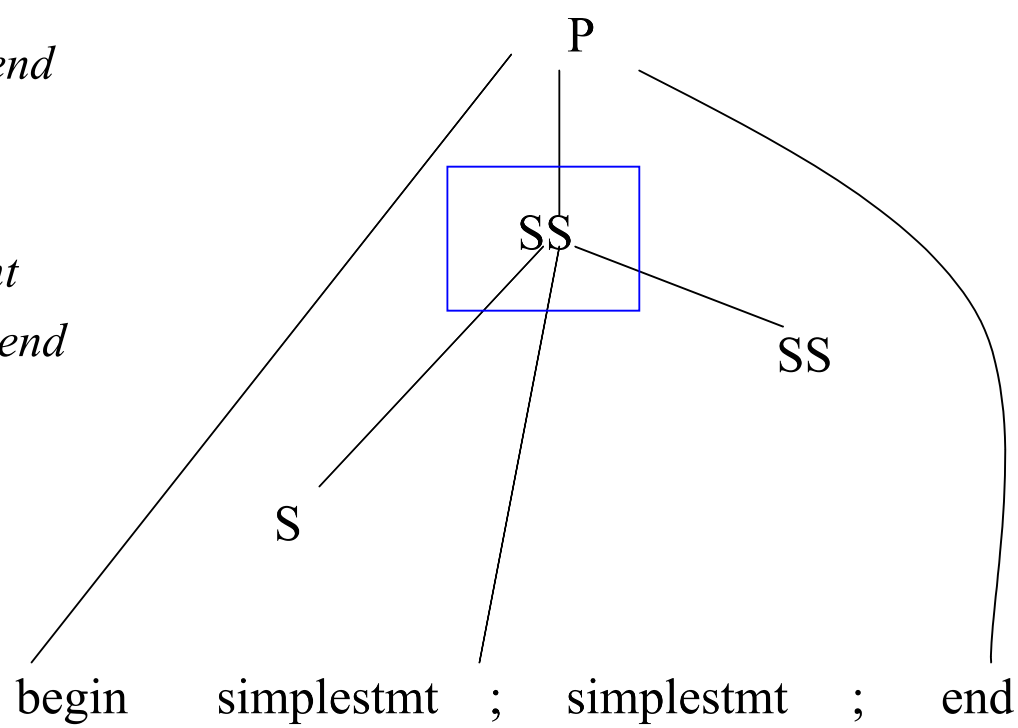
P→ *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Top Down (LL) Parsing

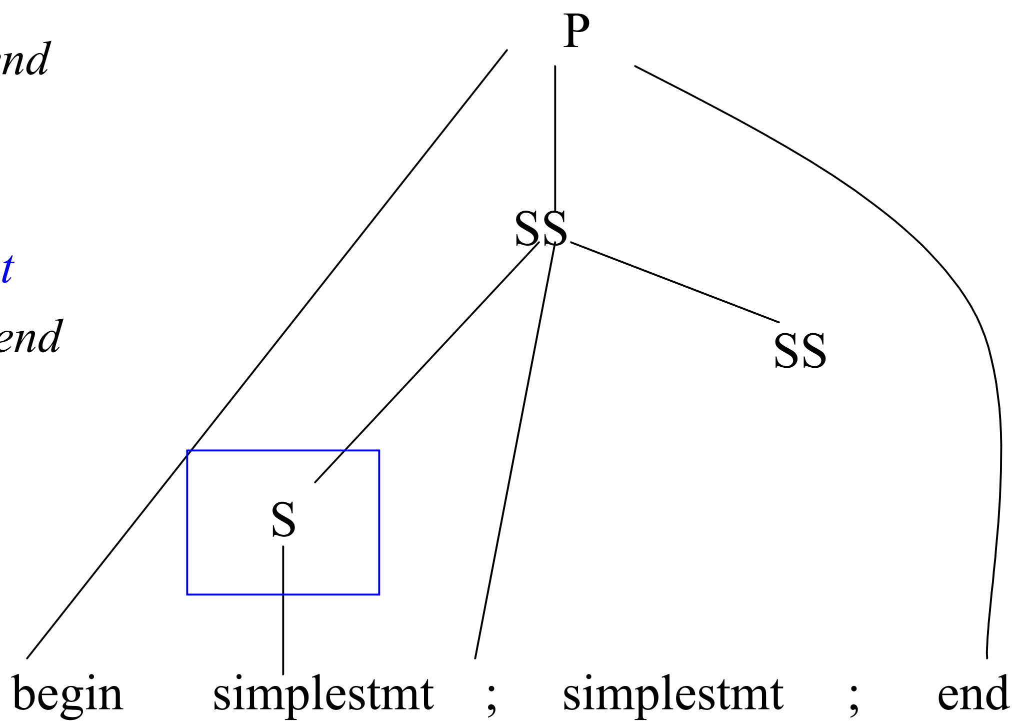
P→ *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Top Down (LL) Parsing

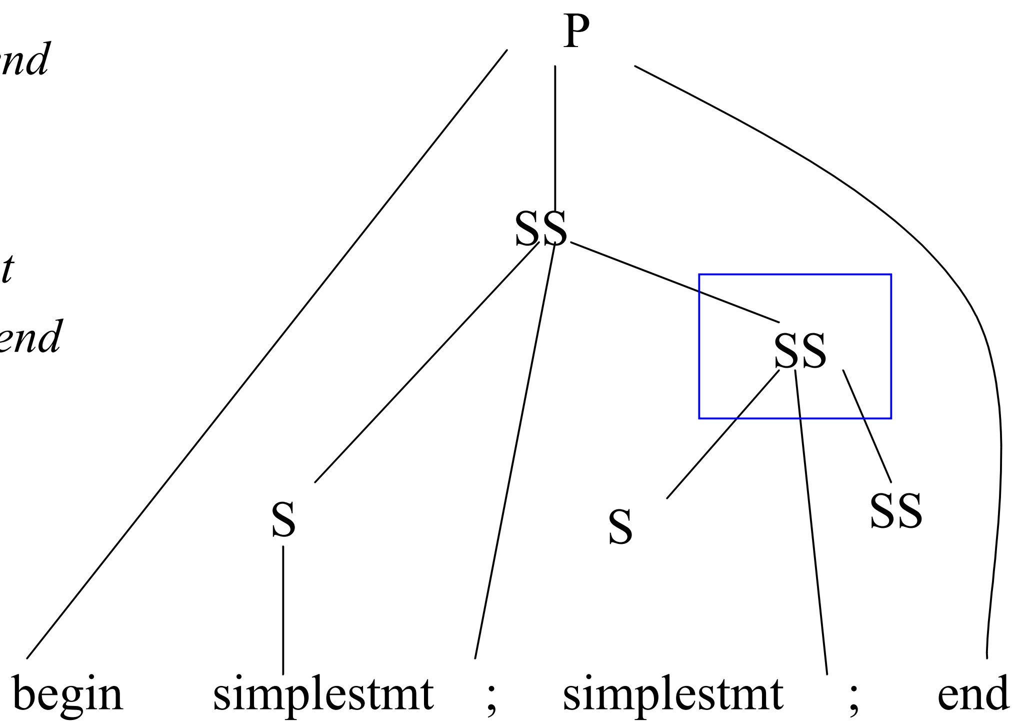
P→ *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Top Down (LL) Parsing

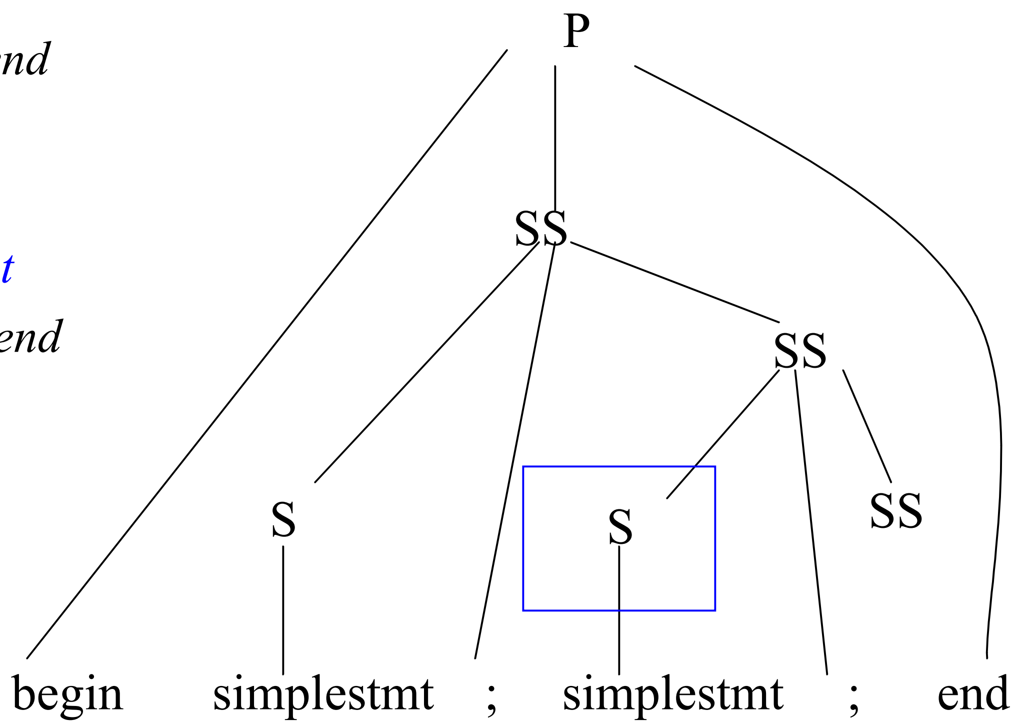
P→ *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Top Down (LL) Parsing

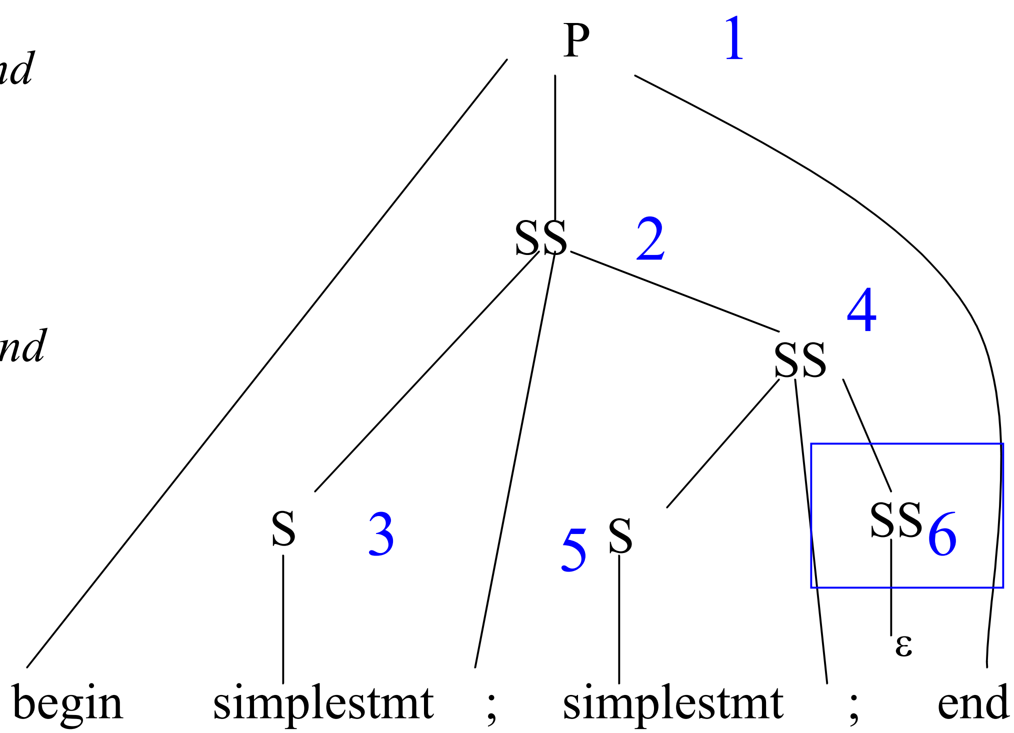
P → *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Bottomup (LR) Parsing

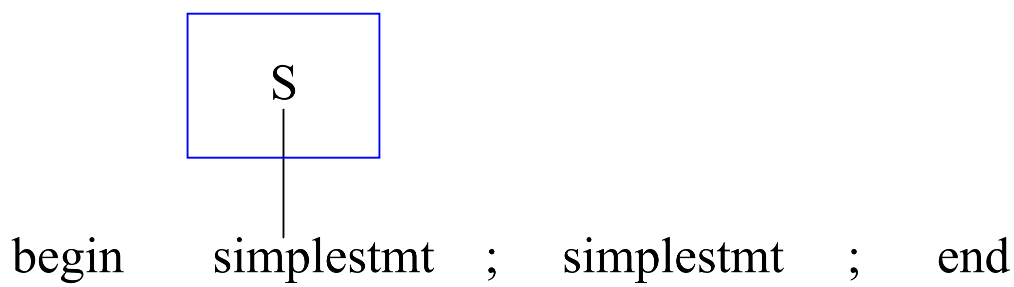
P → *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Bottomup (LR) Parsing

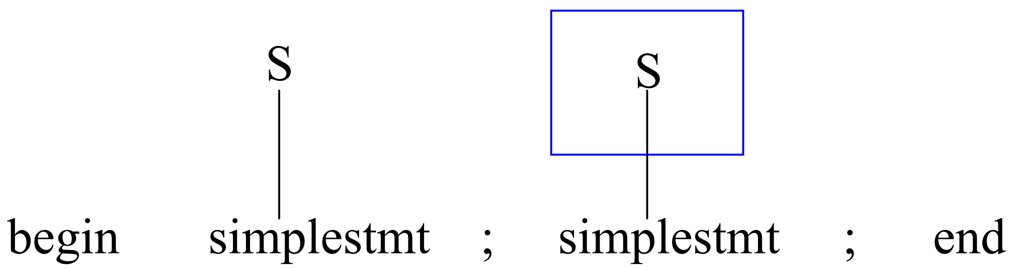
P→ *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Bottomup (LR) Parsing

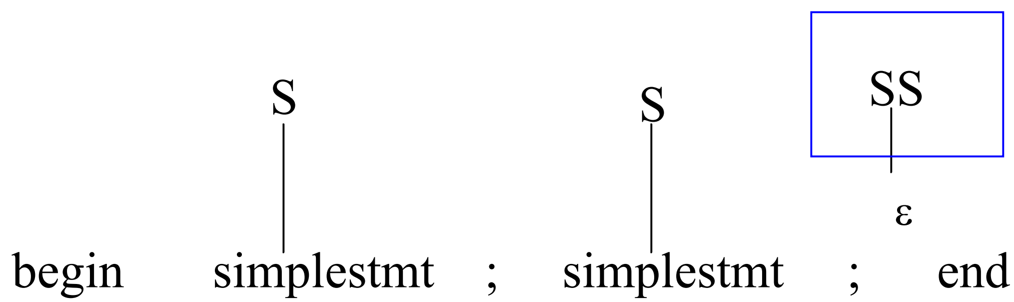
P→ *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Bottomup (LR) Parsing

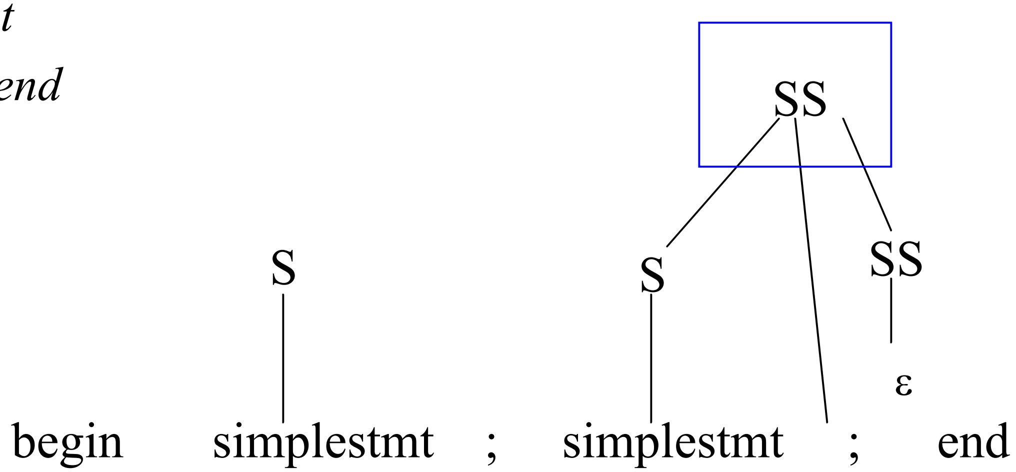
P→ *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Bottomup (LR) Parsing

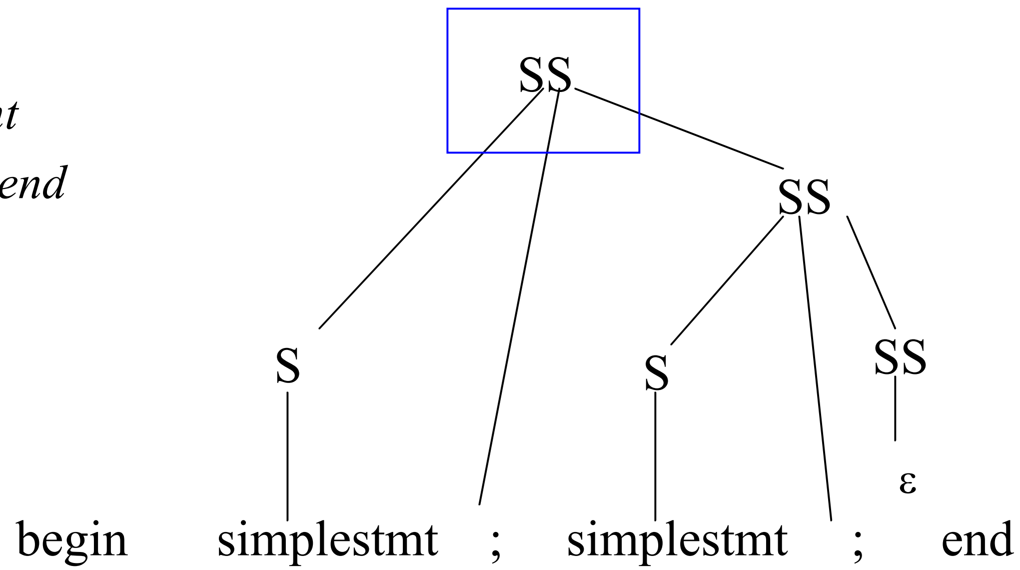
P→ *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *begin* SS *end*



Bottomup (LR) Parsing

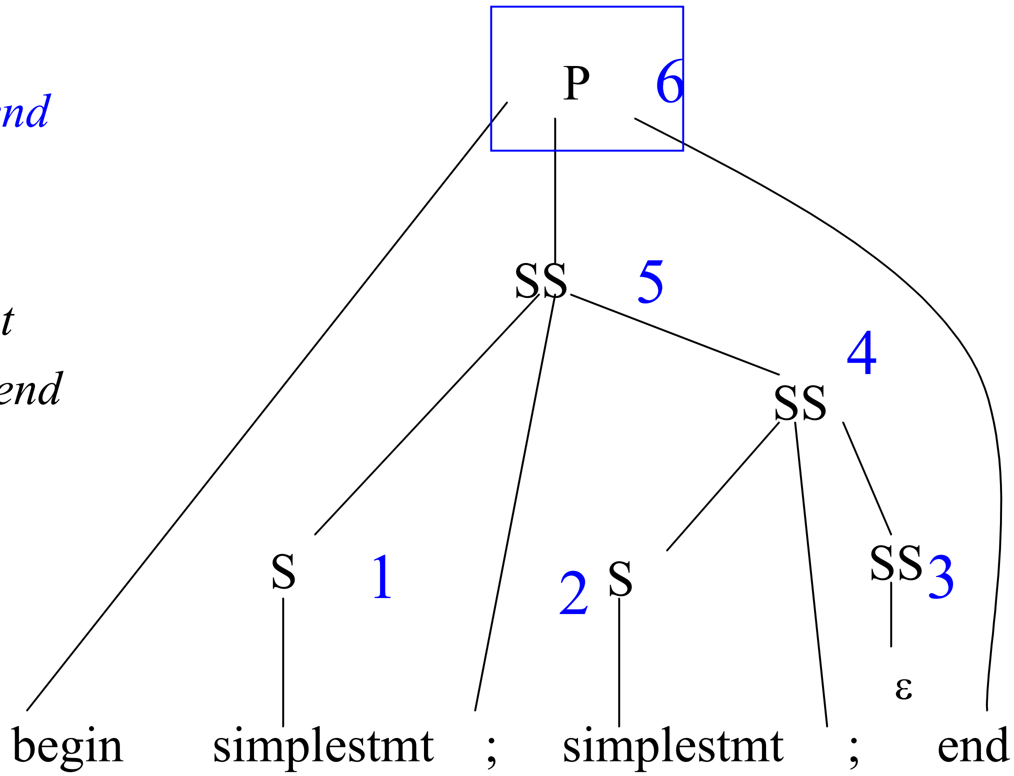
P → *begin* SS *end*

SS → S *;* SS

SS → ε

S → *simplestmt*

S → *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

FIRST(α) is the set of all terminal symbols that can begin some sentential form in a derivation that starts with α

$$\alpha \Rightarrow \ldots \Rightarrow a\ \beta$$

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

  S → simple | begin S end

  FIRST(S) = {simple, begin}

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

$$\mathrm{FIRST}(\varepsilon) = \{ \varepsilon \}$$

$$\mathrm{FIRST}(A\alpha) = \begin{cases} A & \text{if } A \text{ is a terminal} \\ \mathrm{FIRST}(A) \cup \mathrm{FIRST}(\alpha) & \text{if } A \Rightarrow^{*} \varepsilon \\ \mathrm{FIRST}(A) & \text{otherwise} \end{cases}$$

Computing FIRST sets

Initially FIRST(A) (for all A) is empty

1. For productions $A \rightarrow c\ \beta$, where c in V_t

   Add { c } to FIRST(A)

2. For productions $A \rightarrow \varepsilon$

   Add { ε } to FIRST(A)

3. For productions $A \rightarrow \alpha\ B\ \beta$, where $\alpha \Rightarrow^* \varepsilon$ and NOT $(B \Rightarrow^* \varepsilon)$

   Add FIRST(αB) to FIRST(A)

   same thing as $\varepsilon \notin$ FIRST(B)

4. For productions $A \rightarrow \alpha$, where $\alpha \Rightarrow^* \varepsilon$

   Add FIRST(α) and { ε } to FIRST(A)

Example 1

- S → a S e
- S → B
- B → b B e
- B → C
- C → c C e
- C → d

Start with the 'simplest' non-terminal

- FIRST(C) = {c,d}
- FIRST(B) =
- FIRST(S) =

Example 1

- S → a S e
- S → B
- B → b B e
- B → C
- C → c C e
- C → d

- FIRST(C) = {c,d}
- FIRST(B) = {b,c,d}
- FIRST(S) =

Now that we know FIRST(C) …

Example 1

- S → a S e
- S → B
- B → b B e
- B → C
- C → c C e
- C → d

- FIRST(C) = {c,d}
- FIRST(B) = {b,c,d}
- FIRST(S) = {a,b,c,d}

Example 2

- $P \rightarrow \underline{i} \mid \underline{c} \mid \underline{n}\ T\ S$
- $Q \rightarrow P \mid a\ S \mid d\ S\ c\ S\ T$
- $R \rightarrow \underline{b} \mid \underline{\varepsilon}$
- $S \rightarrow e \mid R\ n \mid \varepsilon$
- $T \rightarrow R\ S\ q$

- FIRST(P) = $\{i,c,n\}$
- FIRST(Q) =
- FIRST(R) = $\{b,\varepsilon\}$
- FIRST(S) =
- FIRST(T) =

Example 2

- $P \rightarrow i \mid c \mid n\ T\ S$
- $Q \rightarrow \underline{P} \mid \underline{a}\ S \mid \underline{d}\ S\ c\ S\ T$
- $R \rightarrow b \mid \varepsilon$
- $S \rightarrow e \mid R\ n \mid \varepsilon$
- $T \rightarrow R\ S\ q$

- FIRST(P) = $\{i,c,n\}$
- FIRST(Q) = $\{i,c,n,a,d\}$
- FIRST(R) = $\{b,\varepsilon\}$
- FIRST(S) =
- FIRST(T) =

Example 2

- P → i | c | n T S
- Q → P | a S | d S c S T
- R → b | ε
- S → <u>e</u> | <u>R n</u> | <u>ε</u>
- T → R S q

- FIRST(P) = {i,c,n}
- FIRST(Q) = {i,c,n,a,d}
- FIRST(R) = {b,ε}
- FIRST(S) = {e,b,n,ε}
- FIRST(T) =

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

Example 2

- $P \rightarrow i \mid c \mid n\ T\ S$
- $Q \rightarrow P \mid a\ S \mid d\ S\ c\ S\ T$
- $R \rightarrow b \mid \varepsilon$
- $S \rightarrow e \mid R\ n \mid \varepsilon$
- $T \rightarrow \underline{R\ S\ q}$

- FIRST(P) = {i,c,n}
- FIRST(Q) = {i,c,n,a,d}
- FIRST(R) = {b,ε}
- FIRST(S) = {e,b,n,ε}
- 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

- $S \rightarrow a\ S\ e \mid S\ T\ S$
- $T \rightarrow R\ S\ e \mid Q$
- $R \rightarrow r\ S\ r \mid \varepsilon$
- $Q \rightarrow S\ T \mid \varepsilon$

- FIRST(S) =
- FIRST(R) =
- FIRST(T) =
- FIRST(Q) =

Example 3

- $S \rightarrow a\ S\ e \mid S\ T\ S$
- $T \rightarrow R\ S\ e \mid Q$
- $R \rightarrow r\ S\ r \mid \varepsilon$
- $Q \rightarrow S\ T \mid \varepsilon$

- FIRST(S) = $\{a\}$
- FIRST(R) = $\{r, \varepsilon\}$
- FIRST(T) = $\{r, a, \varepsilon\}$
- 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.
- FOLLOW(A) = $\{c \text{ in } V_t \mid S \Rightarrow^+ \ldots Ac \ldots\} \cup \{\$\}$ if $S \Rightarrow^+ \ldots A$
- 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 FIRST(β) – {ε} to FOLLOW(A)

INTUITION: Suppose $B \rightarrow AX$ and 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

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

- FIRST(C) = {c,d,ε}
- FIRST(B) = {b,c,d,ε}
- FIRST(S) = {a,b,c,d,ε}

- FOLLOW(C) =
- FOLLOW(B) =
- FOLLOW(S) = {$}

Using rule #1

Example 4

- $S \rightarrow a\ \underline{S\ e} \mid B$
- $B \rightarrow b\ \underline{B\ C\ f} \mid C$
- $C \rightarrow c\ \underline{C\ g} \mid d \mid \varepsilon$

- FIRST(C) = {c,d,ε}
- FIRST(B) = {b,c,d,ε}
- FIRST(S) = {a,b,c,d,ε}

- FOLLOW(C) = {f,g}
- FOLLOW(B) = {c,d,f}
- FOLLOW(S) = {$,e}

Using rule #2

Example 4

- $S \rightarrow a\ S\ e \mid \underline{B}$
- $B \rightarrow b\ B\ C\ f \mid \underline{C}$
- $C \rightarrow c\ C\ g \mid d \mid \varepsilon$

- FIRST(C) = $\{c,d,\varepsilon\}$
- FIRST(B) = $\{b,c,d,\varepsilon\}$
- FIRST(S) = $\{a,b,c,d,\varepsilon\}$

- FOLLOW(C) = $\{f,g\} \cup$ FOLLOW(B) $= \{c,d,e,f,g,\$\}$
- FOLLOW(B) = $\{c,d,f\} \cup$ FOLLOW(S) $= \{c,d,e,f,\$\}$
- 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,ε}
- FIRST(A) = {a, ε}
- FIRST(S) = {a,b,c,e,f}

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

Example 5

- S → A B C | A D
- A → ε | a A
- B → b | c | ε
- C → D d C
- D → e b | f c

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

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

Example 6

- S $\rightarrow$ (A) | ε
- A $\rightarrow$ T E
- E $\rightarrow$ & T E | ε
- T $\rightarrow$ (A) | a | b | c

- FOLLOW(S) =
- FOLLOW(A) =
- FOLLOW(E) =
- FOLLOW(T) =

- FIRST(T) = {(,a,b,c}
- FIRST(E) = {&, ε }
- FIRST(A) = {(,a,b,c}
- FIRST(S) = {(, ε}

Example 6

- $S \rightarrow (A) \mid \varepsilon$
- $A \rightarrow T\ E$
- $E \rightarrow \&\ T\ E \mid \varepsilon$
- $T \rightarrow (A) \mid a \mid b \mid c$

- FIRST(T) = {(,a,b,c}
- FIRST(E) = {&, ε }
- FIRST(A) = {(,a,b,c}
- FIRST(S) = {(, ε}

- FOLLOW(S) = {$}
- FOLLOW(A) = {) }
- FOLLOW(E) =

FOLLOW(A) = {) }

- FOLLOW(T) =

FIRST(E) ∪ FOLLOW(A) ∪ FOLLOW(E) = {&,)}

Example 7

- E → T E'
- E' → + T E' | ε
- T → F T'
- T' → * F T' | ε
- F → (E) | 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

- $E \rightarrow T\ E'$
- $E' \rightarrow +\ T\ E' \mid \varepsilon$
- $T \rightarrow F\ T'$
- $T' \rightarrow *\ F\ T' \mid \varepsilon$
- $F \rightarrow (\ E\ ) \mid id$

- FIRST(F) = FIRST(T) = FIRST(E) = {(,id}
- FIRST(T') = {*,ε}
- FIRST(E') = {+,ε}

- FOLLOW(E) = {$,)}
- FOLLOW(E') = FOLLOW(E) = {$,)}
- FOLLOW(T) = FIRST(E') ∪ FOLLOW(E) ∪ FOLLOW(E') = {+,$,)}
- FOLLOW(T') = FOLLOW(T) = {+,$,)}
- FOLLOW(F) = FIRST(T') ∪ FOLLOW(T) ∪ FOLLOW(T') = {*,+,$,)}