Syntax Analysis
(Part 3)

CSE 415: Compiler Construction

LL(1) Grammar Example

- P1: $S \rightarrow if\ (a)\ S\ else\ S \mid while\ (a)\ S \mid begin\ SL\ end$
  P2: $SL \rightarrow S\ S'$
  P3: $S' \rightarrow ;\ SL \mid \epsilon$
- {if, while, begin, end, a, (,), ;} are all terminal symbols
- Clearly, all alternatives of P1 start with distinct symbols and hence create no problem
- P2 has no choices
- Regarding P3, dirsymb(;SL) = {;}, and dirsymb(ϵ) = {end}, and the two have no common symbols
- Hence the grammar is LL(1)

LL(1) Table Construction Example 1

LL(1) Parsing Table for the original grammar

	if	id	else	a	$
S′	S′ → S$			S′ → S$	
S	S → if id S S → if id S else S			S → a	

Original Grammar

Grammar is not LL(1)

S′ → S$
S → if id S |
  if id S else S |
  a

tokens: if, id, else, a

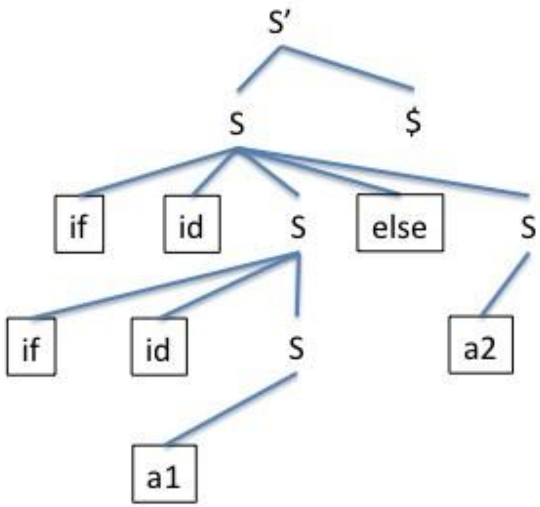
dirsymb(S$) = {if,a}; dirsymb (a) = {a}
dirsymb(if id S) = {if}
dirsymb(if id S else S) = {if}

$$dirsymb(if\ id\ S) \bigcap dirsymb(a) = \varnothing$$

$$dirsymb(if\ id\ S\ else\ S) \bigcap dirsymb(a) = \varnothing$$

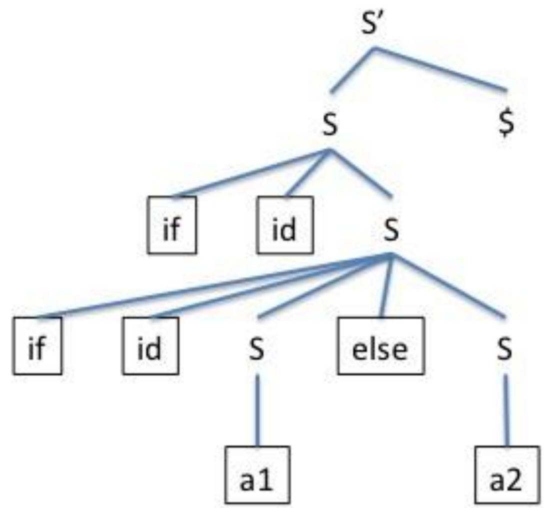
$$dirsymb(if\ id\ S) \bigcap dirsymb(if\ id\ S\ else\ S) \neq \varnothing$$

LL(1) Table Construction Example 1



string: if id (if id a1) else a2

parentheses are not part of the string



string: if id (if id a1 else a2)

parentheses are not part of the string

LL(1) Table Construction Example 2

Original Grammar

S′ → S$
S → if id S |
  if id S else S |
  a

LL(1) Parsing Table for modified grammar

	if	else	a	$
S′	S′ → S$		S′ → S$	
S	S → if id S S1		S → a	
S1		S1 → ε S1 → else S		S1 → ε

dirsymb(S$) = {if, a}; dirsymb (a) = {a}
dirsymb(if id S S1) = {if}
dirsymb(else S) = {else}
dirsymb(ε) = {else, $}

Grammar is not LL(1)

Left-Factored Grammar

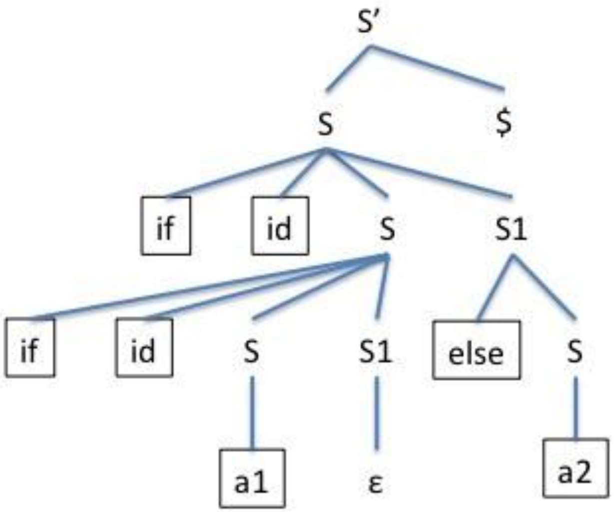
S′ → S$
S → if id S S1 | a
S1 → ε | else S

tokens: if, id, else, a

$$dirsymb(if\ id\ S\ S1) \bigcap dirsymb(a) = \varnothing$$

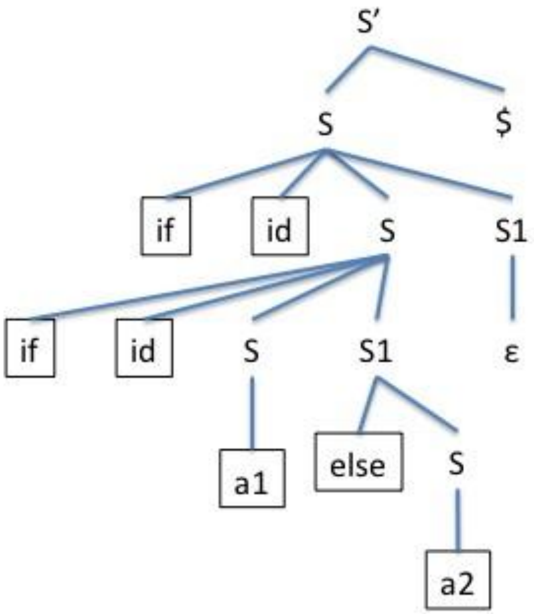
$$dirsymb(\varepsilon) \bigcap dirsymb(else\ S) \neq \varnothing$$

LL(1) Table Construction Example 2



string: if id (if id a1) else a2

parentheses are not part of the string



string: if id (if id a1 else a2)

parentheses are not part of the string

LL(1) Table Construction Example 3

S' → S$
S → aAS | c
A → ba | SB
B → bA | S
Grammar is LL(1)

LL(1) Parsing Table

	a	b	c	$
S'	S' → S$		S' → S$	
S	S → aAS		S → c	
A	A → SB	A → ba	A → SB	
B	B → S	B → bA	B → S	

first(S) = {a,c}
first(A) = {a,b,c}
first(B) = {a,b,c}

$$dirsymb(aAS) \cap dirsymb(c) = \varnothing$$

$$dirsymb(ba) \cap dirsymb(SB) = \varnothing$$

$$dirsymb(bA) \cap dirsymb(S) = \varnothing$$

follow(S) = {a,b,c,$}
follow(A) = {a,c}
follow(B) = {a,c}

dirsymb(S$) = {a,c}
dirsymb(aAS) = {a}
dirsymb(c) = {c}

dirsymb(ba) = {b}
dirsymb(SB) = {a,c}
dirsymb(bA) = {b}
dirsymb(S) = {a,c}

Elimination of Useless Symbols

- Given a grammar $G = (N, T, P, S)$, a non-terminal X is *useful* if $S \Rightarrow^* \alpha X \beta \Rightarrow^* w$, where, $w \in T^*$
  Otherwise, X is useless
- Two conditions have to be met to ensure that X is useful
  1. $X \Rightarrow^* w$, $w \in T^*$ (X derives some terminal string)
  2. $S \Rightarrow^* \alpha X \beta$ (X occurs in some string derivable from S)

Example: $S \rightarrow AB \mid CA, \; B \rightarrow BC \mid AB, \; A \rightarrow a, \; C \rightarrow aB \mid b, \; D \rightarrow d$

$$A \rightarrow a, \; C \rightarrow b, \; D \rightarrow d, \; S \rightarrow CA$$
$$S \rightarrow CA, \; A \rightarrow a, \; C \rightarrow b$$

Elimination of Left Recursion

- A *left-recursive* grammar has a non-terminal A such that $A \Rightarrow^{+} A\alpha$
- Top-down parsing methods (LL(1) and RD) cannot handle left-recursive grammars
- Left-recursion in grammars can be eliminated by transformations
- A simpler case is that of grammars with *immediate left recursion*, where there is a production of the form $A \rightarrow A\alpha$

Two productions $A \rightarrow A\alpha \mid \beta$ can be transformed to

$A \rightarrow \beta A', A' \rightarrow \alpha A' \mid \epsilon$

In general, a group of productions:

$A \rightarrow A\alpha_1 \mid A\alpha_2 \mid \ldots \mid A\alpha_m \mid \beta_1 \mid \beta_2 \mid \ldots \mid \beta_n$

can be transformed to

$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \ldots \mid \beta_n A', A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \ldots \mid \alpha_m A' \mid \epsilon$

Elimination of Left Recursion Example 1

$A \rightarrow A\alpha \mid \beta \Rightarrow A \rightarrow \beta A', A' \rightarrow \alpha A' \mid \epsilon$

- The following grammar for regular expressions is ambiguous:
  $E \rightarrow E + E \mid E\, E \mid E * \mid (E) \mid a \mid b$
- Equivalent left-recursive but unambiguous grammar is:
  $E \rightarrow E + T \mid T,\ T \rightarrow T\, F \mid F,\ F \rightarrow F * \mid P,\ P \rightarrow (E) \mid a \mid b$
- Equivalent non-left-recursive grammar is:
  $E \rightarrow T\, E',\ E' \rightarrow +T\, E' \mid \epsilon,\ T \rightarrow F\, T',\ T' \rightarrow F\, T' \mid \epsilon,$
  $F \rightarrow P\, F',\ F' \rightarrow *F' \mid \epsilon,\ P \rightarrow (E) \mid a \mid b$

Elimination of Left Recursion Example 2

Left-Recursive Grammar for Statement List

S' → SL $
SL → SL S | S
S → a

LL(1) Parsing Table for Left-Recursive Grammar

	a
S'	S' → SL $
SL	SL → SL S SL → S
S	S → a

dirsymb(SL $) = {a}
dirsymb (a) = {a}
dirsymb(SL S) = {a}
dirsymb(S) = {a}

Grammar is not LL(1)

$$dirsymb(SL\ \ S) \bigcap dirsymb(S) \neq \varnothing$$

Right-Recursive Grammar for Statement List

S' → SL $
SL → S A
A → S A | ε
S → a

LL(1) Parsing Table for Right-Recursive Grammar

	a	$
S'	S' → SL $	
SL	SL → S A	
A	A → S A	A → ε
S	S → a	

dirsymb(SL $) = {a}
dirsymb (a) = {a}
dirsymb(S A) = {a}
dirsymb(ε) = {$}

Grammar is LL(1)

$$dirsymb(S\ A) \bigcap dirsymb(\varepsilon) = \varnothing$$

Left Factoring

- If two alternatives of a production begin with the same string, then the grammar is not LL(1)
- Example: $S \rightarrow 0S1 \mid 01$ ……. *is not LL(1)*
  - After left factoring: $S \rightarrow 0S', S' \rightarrow S1 \mid 1$ …. *is LL(1)*
- General method: $A \rightarrow \alpha\beta_1 \mid \alpha\beta_2 \Rightarrow A \rightarrow \alpha A', A' \rightarrow \beta_1 \mid \beta_2$
- Another example: a grammar for logical expressions is given below

  $E \rightarrow T \; or \; E \mid T \,,\; T \rightarrow F \; and \; T \mid F \,,$

  $F \rightarrow not \; F \mid (E) \mid true \mid false$

  - This grammar is not LL(1) but becomes LL(1) after left factoring

    $E \rightarrow TE', E' \rightarrow or \; E \mid \epsilon, T \rightarrow FT', T' \rightarrow and \; T \mid \epsilon,$

    $F \rightarrow not \; F \mid (E) \mid true \mid false$

Grammar Transformation May not Help

Original Grammar

$S' \rightarrow S\$$
$S \rightarrow$ if id S |
if id S else S |
a

LL(1) Parsing Table for modified grammar

	if	else	a	$
S'	$S' \rightarrow S\$$		$S' \rightarrow S\$$	
S	$S \rightarrow$ if id S $S1$		$S \rightarrow$ a	
S1		$S1 \rightarrow \varepsilon$ $S1 \rightarrow$ else S		$S1 \rightarrow \varepsilon$

dirsymb(S$) = {if, a}; dirsymb (a) = {a}
dirsymb(if id S S1) = {if}
dirsymb(else S) = {else}
dirsymb(ε) = {else, $}

Grammar is not LL(1)

Left-Factored Grammar

$S' \rightarrow S\$$
$S \rightarrow$ if id S $S1$ | a
$S1 \rightarrow \varepsilon$ | else S

tokens: if, id, else, a

$$dirsymb(if\ id\ S\ S1) \bigcap dirsymb(a) = \varnothing$$

$$dirsymb(\varepsilon) \bigcap dirsymb(else\ S) \neq \varnothing$$

Choose $S1 \rightarrow$ *else S* instead of $S1 \rightarrow \epsilon$ on lookahead *else*. This resolves the conflict. Associates *else* with the innermost *if*