Compilers (CS31003)

Syntax Analysis or Parsing

Lecture 6

A grammar

$G = < T, N, S, P >$ is a (context-free) grammar where:

T	:	Set of terminal symbols
N	:	Set of non-terminal symbols
S	:	$S \in N$ is the start symbol
P	:	Set of production rules

Every production rule is of the form: $A \rightarrow \alpha$, where $A \in N$ and $\alpha \in (N \cup T)^*$.

Symbol convention:

$a, b, c, \cdots$	Lower case letters at the beginning of alphabet	$\in T$
$x, y, z, \cdots$	Lower case letters at the end of alphabet	$\in T^+$
$A, B, C, \cdots$	Upper case letters at the beginning of alphabet	$\in N$
$X, Y, Z, \cdots$	Upper case letters at the end of alphabet	$\in (N \cup T)$
$\alpha, \beta, \gamma, \cdots$	Greek letters	$\in (N \cup T)^*$

A grammar

- $G_I = (\{E\}, \{+, *, (,), Id\}, P_I, E)$
- P_I $E \rightarrow E + E \mid E * E \mid (E) \mid Id$

Id+Id

Left most derivation:

unambiguous grammar

$E \Rightarrow E + E \Rightarrow Id + E \Rightarrow Id + Id$

Right most derivation

$E \Rightarrow E + E \Rightarrow E + Id \Rightarrow Id + Id$

A grammar

- $G_1=(\{E\}, \{+,*,(,),Id\}, P_1, E)$
- P_1 $E \rightarrow E + E \mid E * E \mid (E) \mid Id$

Id*Id+Id

ambiguous grammar

Left most derivation:

$E \Rightarrow E + E \Rightarrow E * E + E \Rightarrow Id * E + E \Rightarrow Id * Id + E \Rightarrow Id * Id + Id$

$E \Rightarrow E * E \Rightarrow Id * E \Rightarrow Id * E + E \Rightarrow Id * Id + E \Rightarrow Id * Id + Id$

Right most derivation

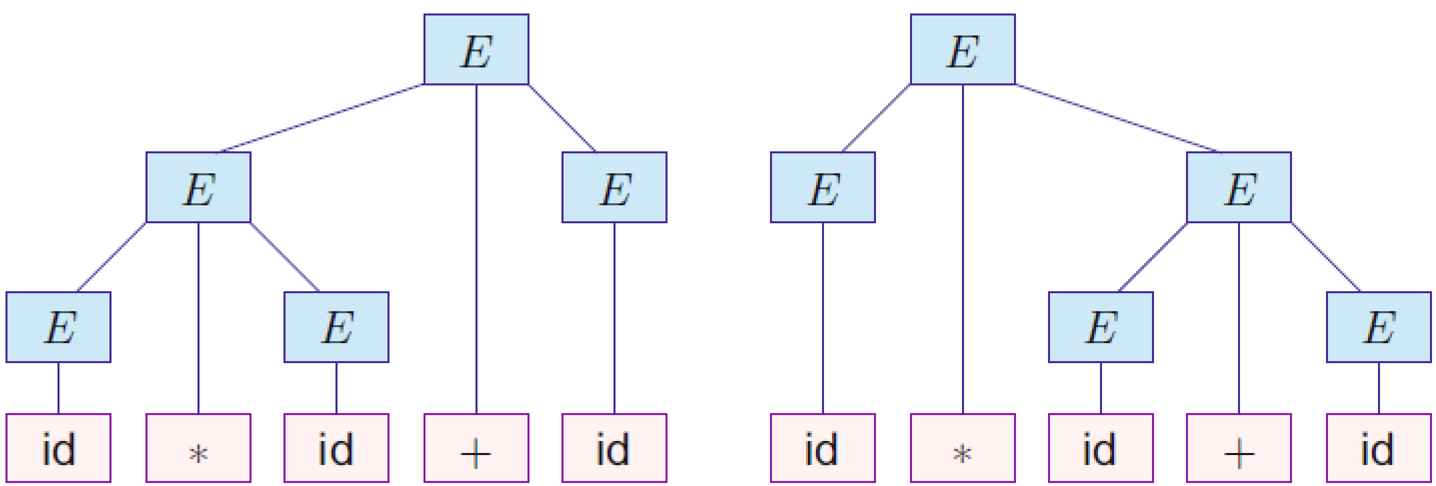
$E \Rightarrow E * E \Rightarrow E * E + E \Rightarrow E * E + Id \Rightarrow E * Id + Id \Rightarrow Id * Id + Id$

$E \Rightarrow E + E \Rightarrow E + Id \Rightarrow E * E + Id \Rightarrow E * Id + Id \Rightarrow Id * Id + Id$

A grammar

- $G_1=(\{E\}, \{+,*,(,),Id\}, P_1, E)$
- P_1 $\quad E \rightarrow E + E \mid E * E \mid (E) \mid Id$

ambiguous grammar



Two grammars

- $G_0=(\{E,T,F\}, \{+,*,(,),Id\}, P_0, E)$
- P_0 $\quad E \rightarrow E + T \mid T$

  $T \rightarrow T * F \mid F$

  $F \rightarrow (E) \mid Id$

- $G_1=(\{E\}, \{+,*,(,),Id\}, P_1, E)$
- P_1 $\quad E \rightarrow E + E \mid E * E \mid (E) \mid Id$

Parsing Fundamentals

Derivation	Parsing	Parser	Remarks
Left-most	Top-Down	Predictive: Recursive Descent, LL(1)	No Ambiguity No Left-recursion Tool: Antlr
Right-most	Bottom-Up	Shift-Reduce: SLR, LALR(1), LR(1)	Ambiguity okay Left-recursion okay Tool: YACC, Bison

Recursive Descent Parser

$$
\begin{array}{lll}
S & \rightarrow & c\ A\ d \\
A & \rightarrow & ab \mid a
\end{array}
$$

```
int main() {
    l = getchar();
    S(); // S is a start symbol

    // Here l is lookahead. If l = $, it represents the end of the string
    if (l == '$')
        printf("Parsing Successful");
    else printf("Error");
}
S() { // Definition of S, as per the given production
    match('c');
    A();
    match('d');
}
A() { // Definition of A as per the given production
    match('a');
    if (l == 'b') { // Look-ahead for decision
        match('b');
    }
}
match(char t) { // Match function - matches and consumes
    if (l == t) { l = getchar();
    }
    else printf("Error");
}
```

Check with: cad$ ($S \Rightarrow cAd \Rightarrow cad$), cabd$ ($S \Rightarrow cAd \Rightarrow cabd$), caad$

Recursive Descent Parser

$$S \rightarrow c\,A\,d$$
$$A \rightarrow aAb \mid a$$

```
int main() {
    l = getchar();
    S(); // S is a start symbol.

    // Here l is lookahead. if l = $, it represents the end of the string
    if (l == '$')
        printf("Parsing Successful");
    else printf("Error");
}
S() { // Definition of S, as per the given production
    match('c');
    A();
    match('d');
}
A() { // Definition of A as per the given production
    match('a');
    if (l == 'a') { // Look-ahead for decision
        A();
        match('b');
    }
}
match(char t) { // Match function - matches and consumes
    if (l == t) { l = getchar();
    }
    else printf("Error");
}
```

Check with: cad\$ ($S \Rightarrow cAd \Rightarrow cad$), cabd\$, caabd\$ ($S \Rightarrow cAd \Rightarrow caAbd \Rightarrow caabd$)

Recursive Descent Parser

$$E \rightarrow a\,E'$$
$$E' \rightarrow +\,a\,E' \mid \epsilon$$

```
int main() {
    l = getchar();
    E(); // E is a start symbol.
    // Here l is lookahead. If l = $, it represents the end of the string
    if (l == '$') printf("Parsing Successful");
    else printf("Error");
}
E() { // Definition of E, as per the given production
    match('a');
    E'();
}
E'() { // Definition of E' as per the given production
    if (l == '+') { // Look-ahead for decision
        match('+');
        match('a');
        E'();
    }
    else return (); // epsilon production
}
match(char t) { // Match function - matches and consumes
    if (l == t) { l = getchar();
    }
    else printf("Error");
}
```

Check with: a$ ($E \Rightarrow aE' \Rightarrow a$), a+a$ ($E \Rightarrow aE' \Rightarrow a + aE' \Rightarrow a + a$), a+a+a$ ($E \Rightarrow aE' \Rightarrow a + aE' \Rightarrow a + a + aE' \Rightarrow a + a + a$)

Recursive Descent Parser

$$E \rightarrow E + E \mid a$$

```
int main() {
    l = getchar();
    E(); // E is a start symbol.

    // Here l is lookahead. if l = $, it represents the end of the string
    if (l == '$')
        printf("Parsing Successful");
    else printf("Error");
}
E() { // Definition of E as per the given production
    if (l == 'a') { // Terminate ? -- Look-ahead does not work
        match('a');
    }

    E();             // Call ?
    match('+');
    E();
}
match(char t) { // Match function - matches and consumes
    if (l == t) { l = getchar();
    }
    else printf("Error");
}
```

Check with: a+a$, a+a+a$

Top-Down parsing

- Action A: Selection of an alternative for the actual leftmost nonterminal and attachment of the right side of the production to the actual tree fragment.

- Action B: Comparison of terminal symbols to the left of the leftmost nonterminal with the remaining input.

$$
\begin{array}{lll}
S \rightarrow E & E' \rightarrow + E \mid \varepsilon & T' \rightarrow * T \mid \varepsilon \\
E \rightarrow T\, E' & T \rightarrow F\, T' & F \rightarrow (E) \mid \text{Id}
\end{array}
$$

Id+Id*Id

Top-Down parsing

$S \to E \qquad E' \to +E \mid \varepsilon \qquad T' \to *T \mid \varepsilon$

$E \to T\,E' \qquad T \to F\,T' \qquad F \to (E) \mid$ Id

Id+Id*Id

Curse or Boon 1: Left-Recursion

A grammar is left-recursive iff there exists a non-terminal A that can derive to a sentential form with itself as the leftmost symbol. Symbolically,

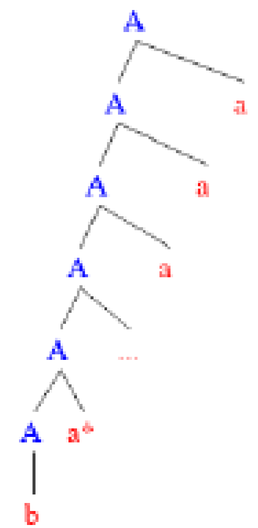
$$A \quad \Rightarrow^+ \quad A\alpha$$

We cannot have a recursive descent or predictive parser (with left-recursion in the grammar) because we do not know how long should we recur without consuming an input

Curse or Boon 1: Left-Recursion

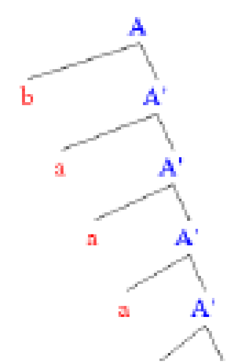
Note that,
$$\begin{array}{lcl} A & \rightarrow & A\alpha \\ A & \rightarrow & \beta \end{array}$$
leads to:

$$\begin{array}{lllllll} A\ \$ & \Rightarrow & A\alpha\ \$ & \Rightarrow & A\alpha\alpha\ \$ & \Rightarrow & A\alpha\alpha\alpha\ \$ \quad \cdots \\ & \Rightarrow & A\alpha^*\ \$ & \Rightarrow & \beta\alpha^*\ \$ & & \end{array}$$



Removing left-recursion
$$\begin{array}{lcll} A & \rightarrow & \beta A' & \\ A' & \rightarrow & \alpha A' & \mid \quad \epsilon \end{array}$$
leads to:

$$\begin{array}{lllllll} A\ \$ & \Rightarrow & \beta A'\ \$ & \Rightarrow & \beta\alpha A'\ \$ & \Rightarrow & \beta\alpha\alpha A'\ \$ \quad \cdots \\ & \Rightarrow & \beta\alpha^* A'\ \$ & \Rightarrow & \beta\alpha^*\ \$ & & \end{array}$$



Left-Recursion: Example

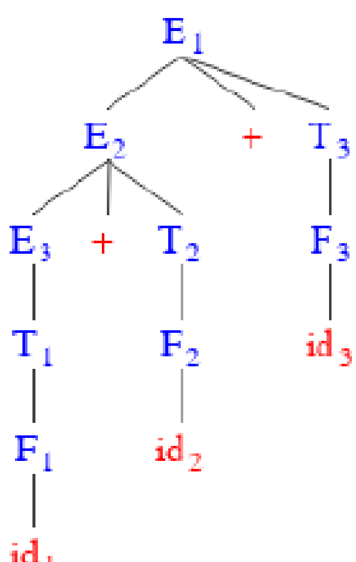
Grammar G_1 before Left-Recursion Removal

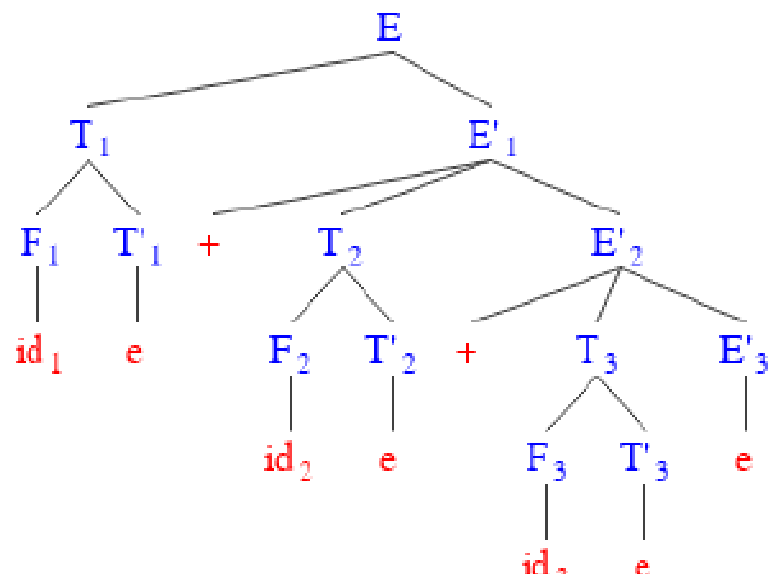
1:	E	$\rightarrow$	$E + T$
2:	E	$\rightarrow$	T
3:	T	$\rightarrow$	$T * F$
4:	T	$\rightarrow$	F
5:	F	$\rightarrow$	(E)
6:	F	$\rightarrow$	id

Grammar G_2 after Left-Recursion Removal

1:	E	$\rightarrow$	$T\ E'$		
2:	E'	$\rightarrow$	$+\ T\ E'$	$\vert$	ϵ
3:	T	$\rightarrow$	$F\ T'$		
4:	T'	$\rightarrow$	$*\ F\ T'$	$\vert$	ϵ
5:	F	$\rightarrow$	(E)		
6:	F	$\rightarrow$	id		

- These are syntactically equivalent. But what happens semantically?
- Can left recursion be effectively removed?
- What happens to Associativity?





Curse or Boon 2: Left-Recursion

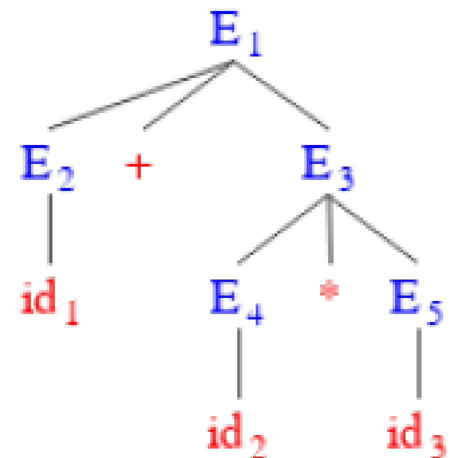
$$\begin{array}{llll} 1: & E & \rightarrow & E + E \\ 2: & E & \rightarrow & E * E \\ 3: & E & \rightarrow & (E) \\ 4: & E & \rightarrow & \text{id} \end{array}$$

- Ambiguity simplifies. But, ...
  - Associativity is lost
  - Precedence is lost
- Can *Operator Precedence (infix $\rightarrow$ postfix)* give us a clue?

Ambiguous Derivation of id + id * id

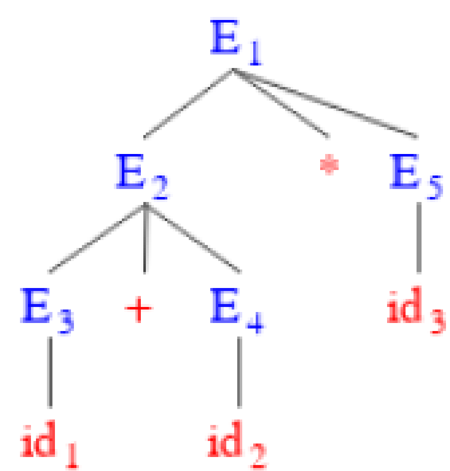
Correct derivation: * has precedence over +

$$
\begin{array}{lcl}
E\ \$ & \Rightarrow & \underline{E + E}\ \$ \\
 & \Rightarrow & E + \underline{E * E}\ \$ \\
 & \Rightarrow & E + E * \underline{\textbf{id}}\ \$ \\
 & \Rightarrow & E + \underline{\textbf{id}} * \textbf{id}\ \$ \\
 & \Rightarrow & \underline{\textbf{id}} + \textbf{id} * \textbf{id}\ \$
\end{array}
$$



Wrong derivation: + has precedence over *

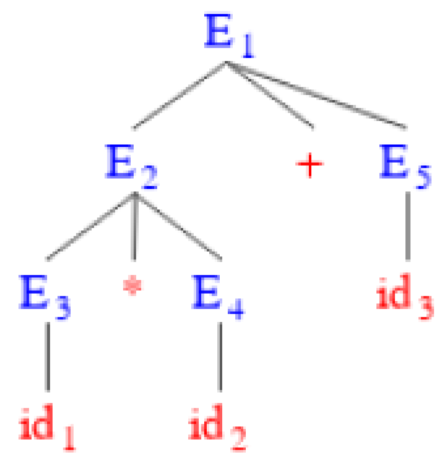
$$
\begin{array}{lcl}
E\ \$ & \Rightarrow & \underline{E * E}\ \$ \\
 & \Rightarrow & E * \underline{\textbf{id}}\ \$ \\
 & \Rightarrow & \underline{E + E} * \textbf{id}\ \$ \\
 & \Rightarrow & E + \underline{\textbf{id}} * \textbf{id}\ \$ \\
 & \Rightarrow & \underline{\textbf{id}} + \textbf{id} * \textbf{id}\ \$
\end{array}
$$



Ambiguous Derivation of id * id + id

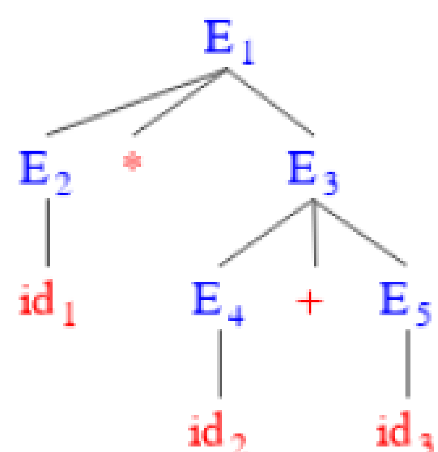
Correct derivation: * has precedence over +

$$
\begin{aligned}
E\,\$ \quad & \Rightarrow \quad \underline{E + E}\ \$ \\
& \Rightarrow \quad E + \underline{\mathbf{id}}\ \$ \\
& \Rightarrow \quad \underline{E * E} + \mathbf{id}\ \$ \\
& \Rightarrow \quad E * \underline{\mathbf{id}} + \mathbf{id}\ \$ \\
& \Rightarrow \quad \underline{\mathbf{id}} * \mathbf{id} + \mathbf{id}\ \$
\end{aligned}
$$



Wrong derivation: + has precedence over *

$$
\begin{aligned}
E\,\$ \quad & \Rightarrow \quad \underline{E * E}\ \$ \\
& \Rightarrow \quad E * \underline{E + E}\ \$ \\
& \Rightarrow \quad E * E + \underline{\mathbf{id}}\ \$ \\
& \Rightarrow \quad E * \underline{\mathbf{id}} + \mathbf{id}\ \$ \\
& \Rightarrow \quad \underline{\mathbf{id}} * \mathbf{id} + \mathbf{id}\ \$
\end{aligned}
$$



Remove: Ambiguity and Left-Recursion

$$
\begin{array}{llll}
1: & E & \rightarrow & E + E \\
2: & E & \rightarrow & E * E \\
3: & E & \rightarrow & (E) \\
4: & E & \rightarrow & \mathbf{id}
\end{array}
$$

Removing ambiguity:

$$
\begin{array}{llll}
1: & E & \rightarrow & E + T \\
2: & E & \rightarrow & T \\
3: & T & \rightarrow & T * F \\
4: & T & \rightarrow & F \\
5: & F & \rightarrow & (E) \\
6: & F & \rightarrow & \mathbf{id}
\end{array}
$$

Removing left-recursion:

$$
\begin{array}{rlll}
1: & E & \rightarrow & T\ E' \\
2|3: & E' & \rightarrow & +\ T\ E' \mid \epsilon \\
4: & T & \rightarrow & F\ T' \\
5|6: & T' & \rightarrow & *\ F\ T' \mid \epsilon \\
7: & F & \rightarrow & (E) \\
8: & F & \rightarrow & \mathbf{id}
\end{array}
$$