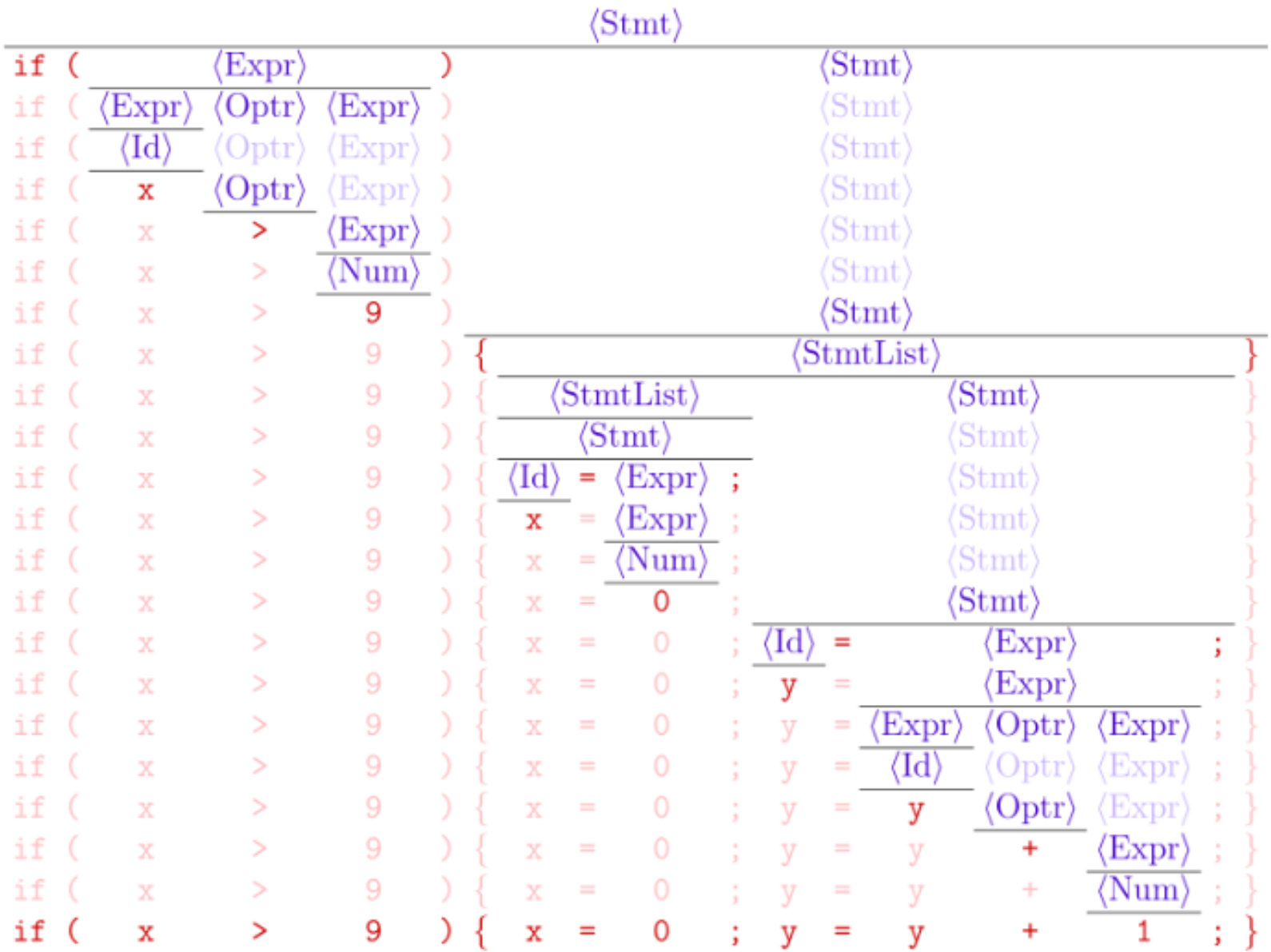
二、语法分析
(7. 递归下降的 $LL(1)$ 语法分析器)

魏恒峰

hfwei@nju.edu.cn

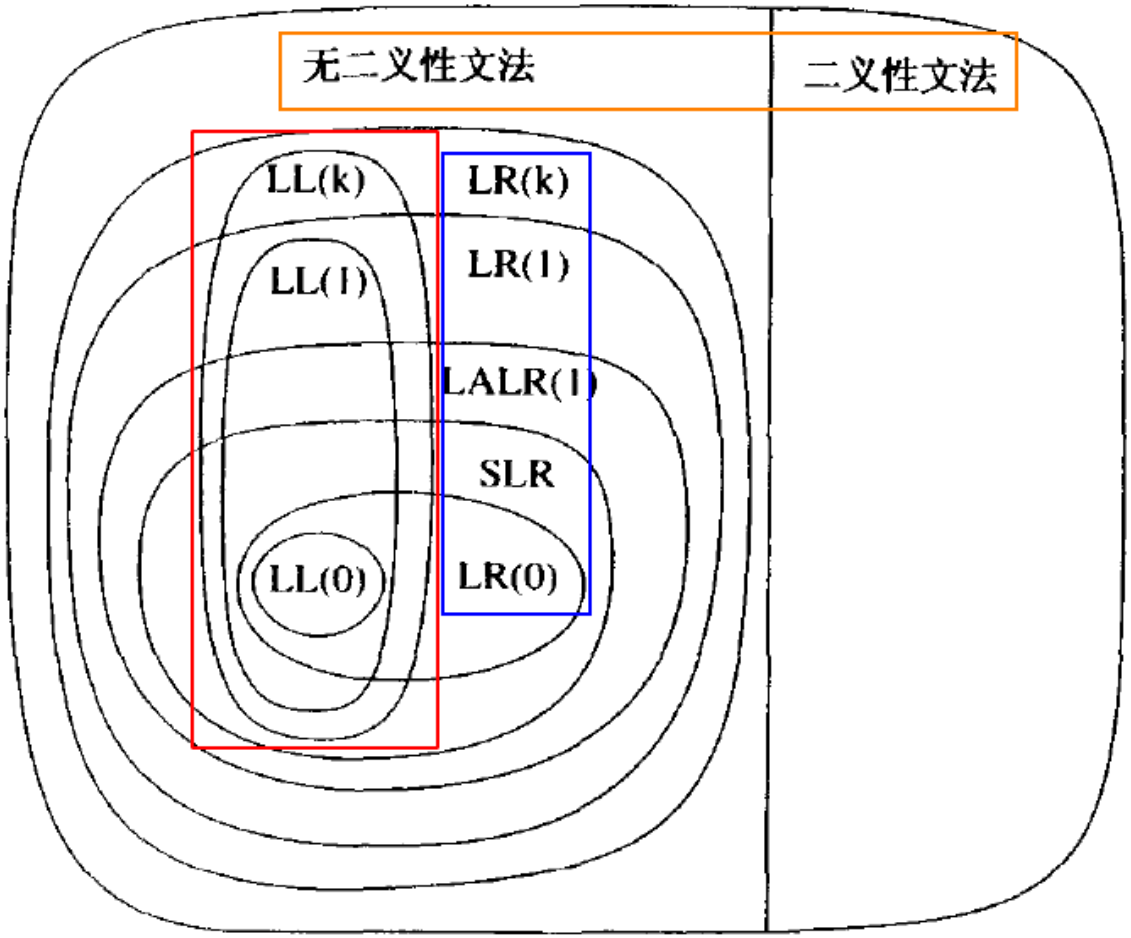
2023 年 03 月 31 日

构建语法分析树：自顶向下 *vs.* 自底向上



只考虑**无二义性**的文法

这意味着, 每个句子对应唯一的一棵语法分析树



今日主题: $LL(1)$ **语法分析器**

Adaptive *LL(*)* Parsing: The Power of Dynamic Analysis

Terence Parr
University of San Francisco
parrt@cs.usfca.edu

Sam Harwell
University of Texas at Austin
samharwell@utexas.edu

Kathleen Fisher
Tufts University
kfisher@eecs.tufts.edu

自顶向下的、

递归下降的、

基于预测分析表的、

适用于$LL(1)$ **文法**的、

$LL(1)$ 语法分析器

自顶向下构建语法分析树

根节点是文法的起始符号 S

每个**中间节点**表示**对某个非终结符应用某个产生式进行推导**

(Q：选择哪个非终结符, 以及选择哪个产生式)

叶节点是词法单元流 $w\$$

仅包含终结符号与特殊的**文件结束符** `$ (EOF)`

每个**中间节点**表示**对某个非终结符应用某个产生式进行推导**

Q：选择哪个非终结符, 以及选择哪个产生式

在推导的每一步, $LL(1)$ 总是选择**最左边的非终结符进行展开**

$LL(1)$: 从左向右读入词法单元

递归下降的典型实现框架

不需要任何参数

```
void A() {先不考虑这里是如何选择产生式的
1)      选择一个 A 产生式，A → X1X2···Xk;
2)      for ( i = 1 to k ) {
3)          if ( Xi 是一个非终结符号 )
4)              调用过程 Xi();
5)          else if ( Xi 等于当前的输入符号 a )
6)              读入下一个输入符号;
7)          else /* 发生了一个错误 */;
        }
    }
```

递归下降调用其它非终结符对应的递归函数

匹配当前词法单元

出现了不期望出现的词法单元

为每个**非终结符**写一个**递归函数**

内部按需调用其它非终结符对应的递归函数，**下降**一层

$$
\boxed{
\begin{aligned}
S &\to F \\
S &\to (S + F) \\
F &\to a
\end{aligned}
}
$$

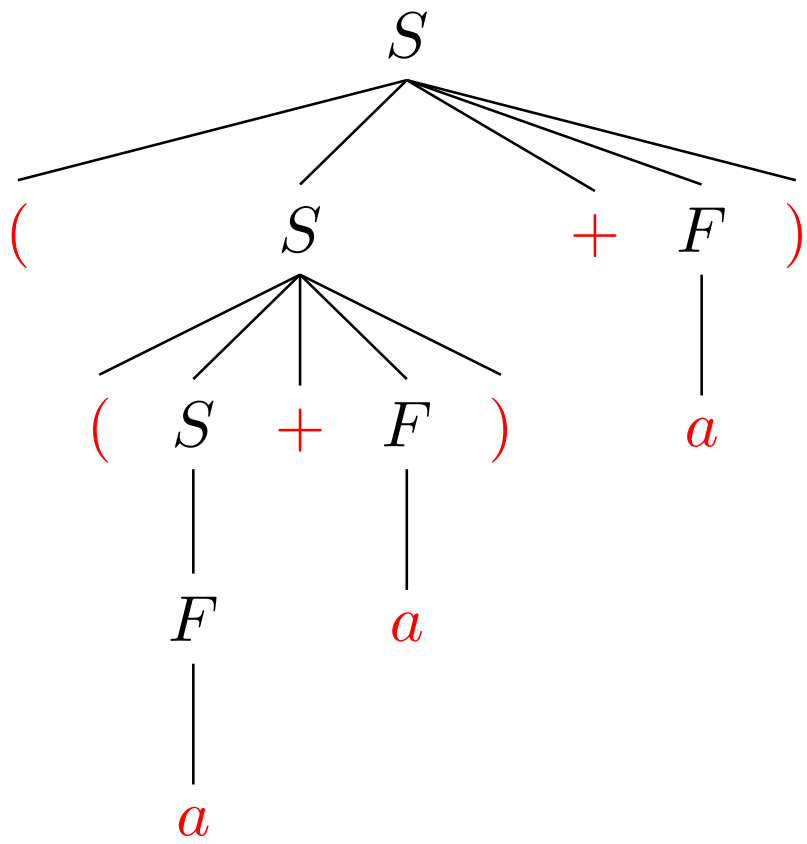
$$w = ((a + a) + a)$$

演示**递归下降**过程

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$



$w = ((a + a) + a)$

每次都选择语法分析树**最左边**的非终结符进行展开

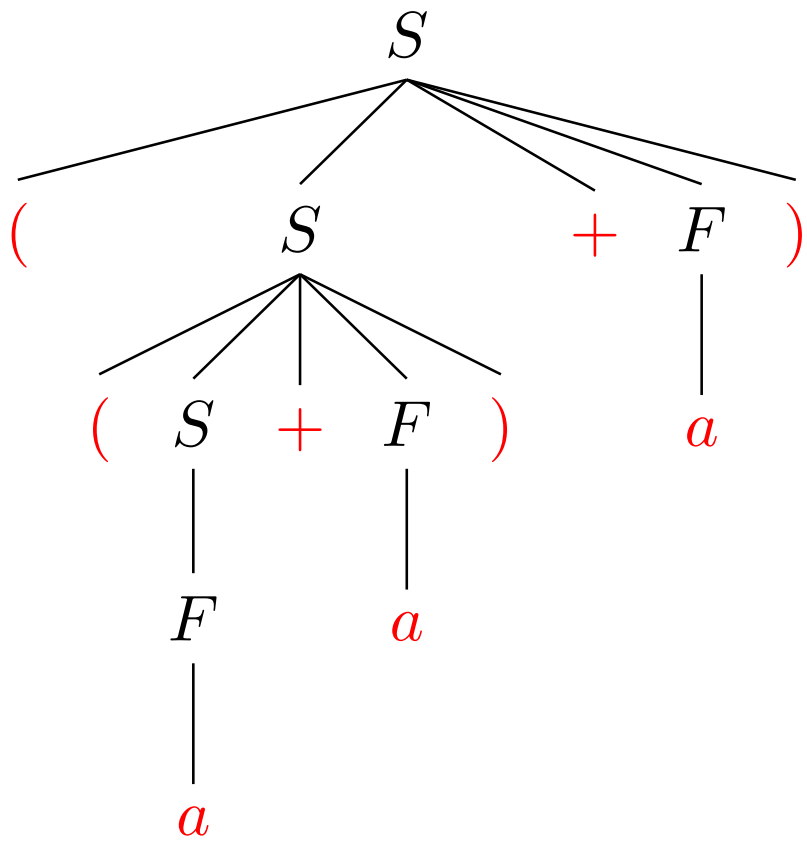
同样是展开非终结符 S,

为什么前两次选择了 $S \to (S+F)$, 而第三次选择了 $S \to F$?

$$S \to F$$

$$S \to (S+F)$$

$$F \to a$$



$w = ((a+a)+a)$

因为它们面对的**当前词法单元**不同

使用**预测分析表**确定产生式

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$

	(	)	a	+	\$
S	2		1		
F			3		

指明了每个**非终结符**在面对不同的**词法单元或文件结束符**时，
该选择哪个**产生式** (按编号进行索引) 或者**报错** (空单元格)

Definition ($LL(1)$ 文法)

如果文法 G 的**预测分析表**是**无冲突**的, 则 G 是 $LL(1)$ 文法。

无冲突: 每个单元格里只有一个产生式 (编号)

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$

	(	)	a	+	\$
S	2		1		
F			3		

对于当前选择的非终结符,

仅根据输入中当前的词法单元 ($LL(1)$) 即可确定需要使用哪条产生式

递归下降的、预测分析实现方法

$$S \to F$$

$$S \to (S + F)$$

$$F \to a$$

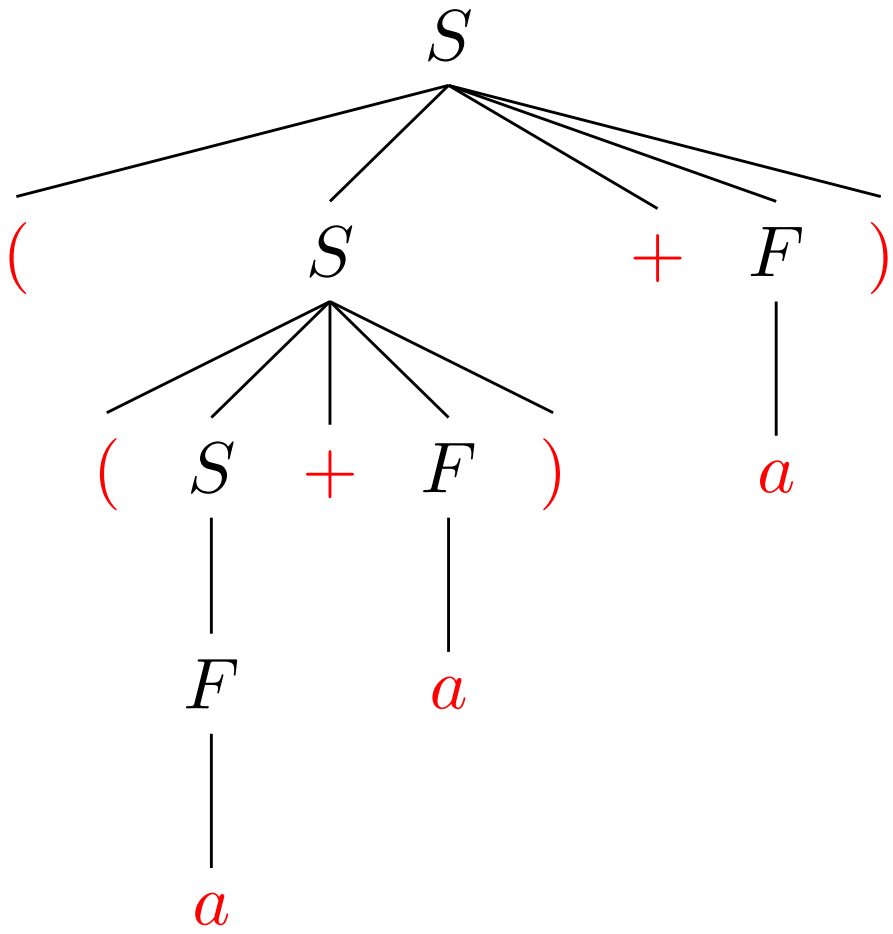
	(	)	a	+	$
S	2		1		
F			3		

```
1: procedure S()
2:     if token = '(' then
3:         MATCH('(')
4:         S()
5:         MATCH('+')
6:         F()
7:         MATCH(')')
8:     else if token = 'a' then
9:         F()
10:    else
11:        ERROR(token, {'(', 'a'})
```

```
1: procedure MATCH(t)
2:     if token = t then
3:         token ← NEXT-TOKEN()
4:     else
```

```
1: procedure F()
2:     if token = 'a' then
```

再次演示**递归下降**过程



$$w = ((a + a) + a)$$

自顶向下的、

递归下降的、

基于预测分析表的、

适用于$LL(1)$ **文法**的、

$LL(1)$ 语法分析器

```
prog : func_call | decl EOF;

func_call : ID '(' arg ')' ;
decl : 'int' ID optional_init ';' ;

arg : 'int' ID optional_init ;
optional_init
    : '=' ID  # Init
    |         # NoInit
    ;
```

```
int x = y;      int x;
f(int x = y)    f(int x)
```

如何计算给定文法 G 的预测分析表?

$\textsc{First}(\alpha)$ 是可从 α 推导得到的句型的**首终结符号**的集合

Definition ($\textsc{First}(\alpha)$ 集合)

对于任意的 (产生式的右部) $\alpha \in (N \cup T)^*$:

$$\textsc{First}(\alpha) = \left\{ t \in T \cup \{\epsilon\} \mid \alpha \overset{*}{\Rightarrow} t\beta \vee \alpha \overset{*}{\Rightarrow} \epsilon \right\}.$$

考虑非终结符 A 的所有产生式 $A \to \alpha_1, A \to \alpha_2, \ldots, A \to \alpha_m$,

如果它们对应的 $\textsc{First}(\alpha_i)$ 集合互不相交,

则只需查看当前输入词法单元, 即可确定选择哪个产生式 (或报错)

如何计算给定文法 G 的预测分析表?

$\textsc{Follow}(A)$ 是可能在某些句型中**紧跟在 A 右边的终结符**的集合

Definition ($\textsc{Follow}(A)$ 集合)

对于任意的 (产生式的左部) 非终结符 $A \in N$:

$$\textsc{Follow}(A) = \left\{ t \in T \cup \{\$\} \mid \exists s.\ S \overset{*}{\Rightarrow} s \triangleq \beta A t \gamma \right\}.$$

考虑产生式 $A \to \alpha$,

如果从 α 可能推导出空串 ($\alpha \overset{*}{\Rightarrow} \epsilon$),

则只有当当前词法单元 $t \in \textsc{Follow}(A)$, 才可以选择该产生式

先计算每个符号 X 的 $\textsc{First}(X)$ 集合

```
1:  procedure FIRST(X)
2:      if X ∈ T then                                  ▷ 规则 1: X 是终结符
3:          First(X) = X
4:      for X → Y_1 Y_2 ... Y_k do                     ▷ 规则 2: X 是非终结符
5:          First(X) ← First(X) ∪ {First(Y_1) \ {ϵ}}
6:          for i ← 2 to k do
7:              if ϵ ∈ L(Y_1 ... Y_{i-1}) then
8:                  First(X) ← First(X) ∪ {First(Y_i) \ {ϵ}}
9:          if ϵ ∈ L(Y_1 ... Y_k) then                 ▷ 规则 3: X 可推导出空串
10:             First(X) ← First(X) ∪ {ϵ}
```

不断应用上面的规则, 直到每个 $\textsc{First}(X)$ 都不再变化 (**不动点!!!**)

再计算每个符号串 α 的 $\textsc{First}(\alpha)$ 集合

$$\alpha = X\beta$$

$$\textsc{First}(\alpha) = \begin{cases} \textsc{First}(X) & \epsilon \notin L(X) \\ (\textsc{First}(X) \setminus \{\epsilon\}) \cup \textsc{First}(\beta) & \epsilon \in L(X) \end{cases}$$

最后, 如果 $\epsilon \in L(\alpha)$, 则将 ϵ 加入 $\textsc{First}(\alpha)$。

(1) $X \to Y$

(2) $X \to a$

(3) $Y \to \epsilon$

(4) $Y \to c$

(5) $Z \to d$

(6) $Z \to XYZ$

$$\textsc{First}(X) = \{a, c, \epsilon\}$$
$$\textsc{First}(Y) = \{c, \epsilon\}$$
$$\textsc{First}(Z) = \{a, c, d\}$$

$$\textsc{First}(XYZ) = \{a, c, d\}$$

为每个非终结符 X 计算 $\textsc{Follow}(X)$ 集合

1: **procedure** $\textsc{follow}(X)$
2: **for** X 是开始符号 **do** ▷ 规则 1: X 是开始符号
3: $\textsc{Follow}(X) \leftarrow \textsc{Follow}(X) \cup \{\$\}$
4: **for** $A \to \alpha X$ **do** ▷ 规则 2: X 是某产生式右部的最后一个符号
5: $\textsc{Follow}(X) \leftarrow \textsc{Follow}(X) \cup \textsc{Follow}(A)$
6: **for** $A \to \alpha X \beta$ **do** ▷ 规则 3: X 是某产生式右部中间的一个符号
7: $\textsc{Follow}(X) \leftarrow \textsc{Follow}(X) \cup (\textsc{First}(\beta) \setminus \{\epsilon\})$
8: **if** $\epsilon \in \textsc{First}(\beta)$ **then**
9: $\textsc{Follow}(X) \leftarrow \textsc{Follow}(X) \cup \textsc{Follow}(A)$

不断应用上面的规则, 直到每个 $\textsc{Follow}(X)$ 都不再变化 (**不动点!!!**)

```
prog : func_call | decl EOF;

func_call : ID '(' arg ')' ;
decl : 'int' ID optional_init ';' ;

arg : 'int' ID optional_init ;
optional_init
    : '=' ID  # Init
    |         # NoInit
    ;
```

```
int x = y;      int x;
f(int x = y)    f(int x)
```

(1) $X \to Y$

(2) $X \to a$

(3) $Y \to \epsilon$

(4) $Y \to c$

(5) $Z \to d$

(6) $Z \to XYZ$

$$\textsc{Follow}(X) = \{a, c, d, \$\}$$
$$\textsc{Follow}(Y) = \{a, c, d, \$\}$$
$$\textsc{Follow}(Z) = \emptyset$$

如何根据First 与 Follow 集合计算给定文法 G 的预测分析表?

对应每条产生式 $A \to \alpha$ 与终结符 t, 如果

$$t \in \textsc{First}(\alpha) \tag{1}$$

$$\alpha \stackrel{*}{\Rightarrow} \epsilon \wedge t \in \textsc{Follow}(A) \tag{2}$$

则在表格 $[A, t]$ 中填入 $A \to \alpha$ (编号)。

Definition ($LL(1)$ 文法)

如果文法 G 的**预测分析表**是**无冲突**的, 则 G 是 $LL(1)$ 文法。

$$t \in \textsc{First}(\alpha) \tag{1}$$

$$\epsilon \in \textsc{First}(\alpha) \wedge t \in \textsc{Follow}(A) \tag{2}$$

当下的选择未必正确, 但 “你别无选择”。

(1) $X \to Y$

(2) $X \to a$

(3) $Y \to \epsilon$

(4) $Y \to c$

(5) $Z \to d$

(6) $Z \to XYZ$

$$\textsc{First}(X) = \{a, c, \epsilon\}$$
$$\textsc{First}(Y) = \{c, \epsilon\}$$
$$\textsc{First}(Z) = \{a, c, d\}$$

$$\textsc{First}(XYZ) = \{a, c, d\}$$

$$\textsc{Follow}(X) = \{a, c, d, \$\}$$
$$\textsc{Follow}(Y) = \{a, c, d, \$\}$$
$$\textsc{Follow}(Z) = \emptyset$$

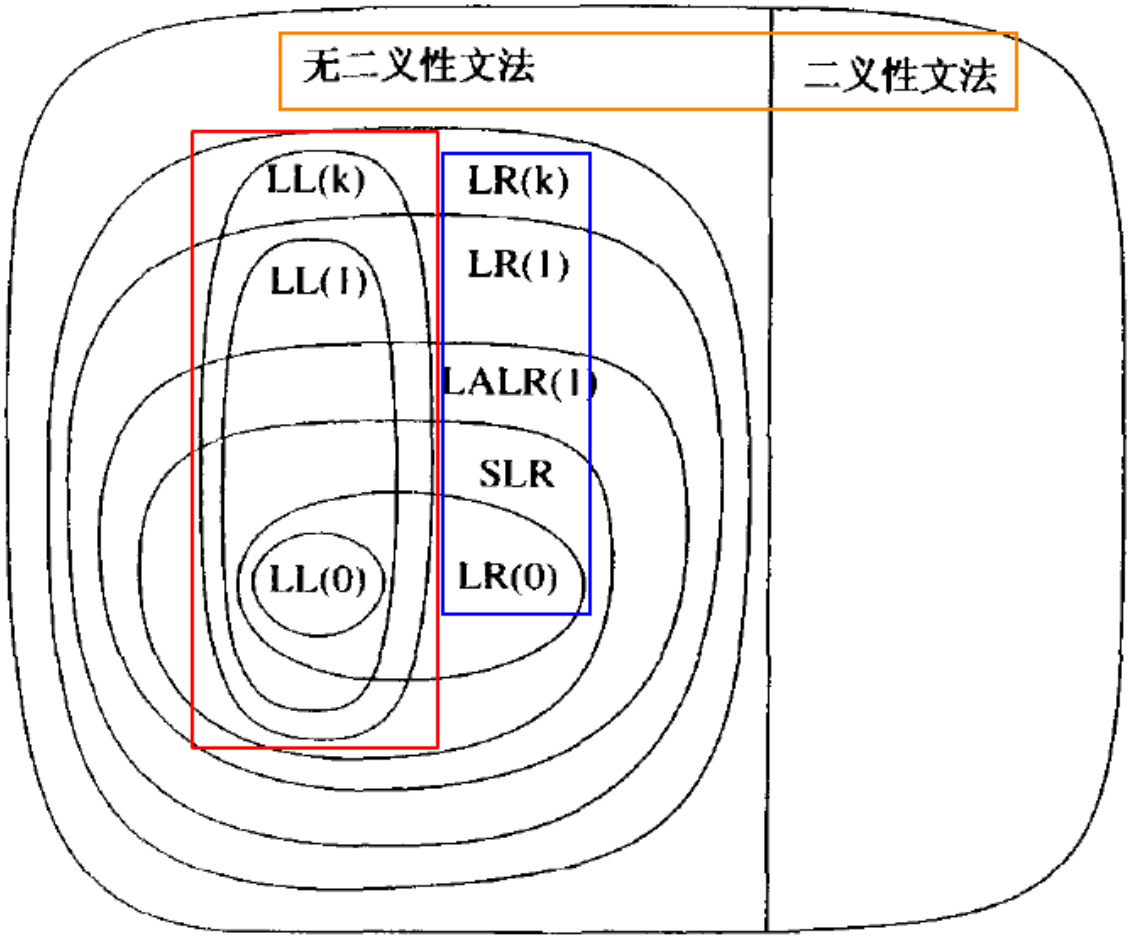
	a	c	d	\$
X	1, 2	1	1	1
Y	3	3, 4	3	3
Z	6	6	5, 6	

$LL(1)$ 语法分析器

L：从左向右 (left-to-right) 扫描输入

L：构建最左 (leftmost) 推导

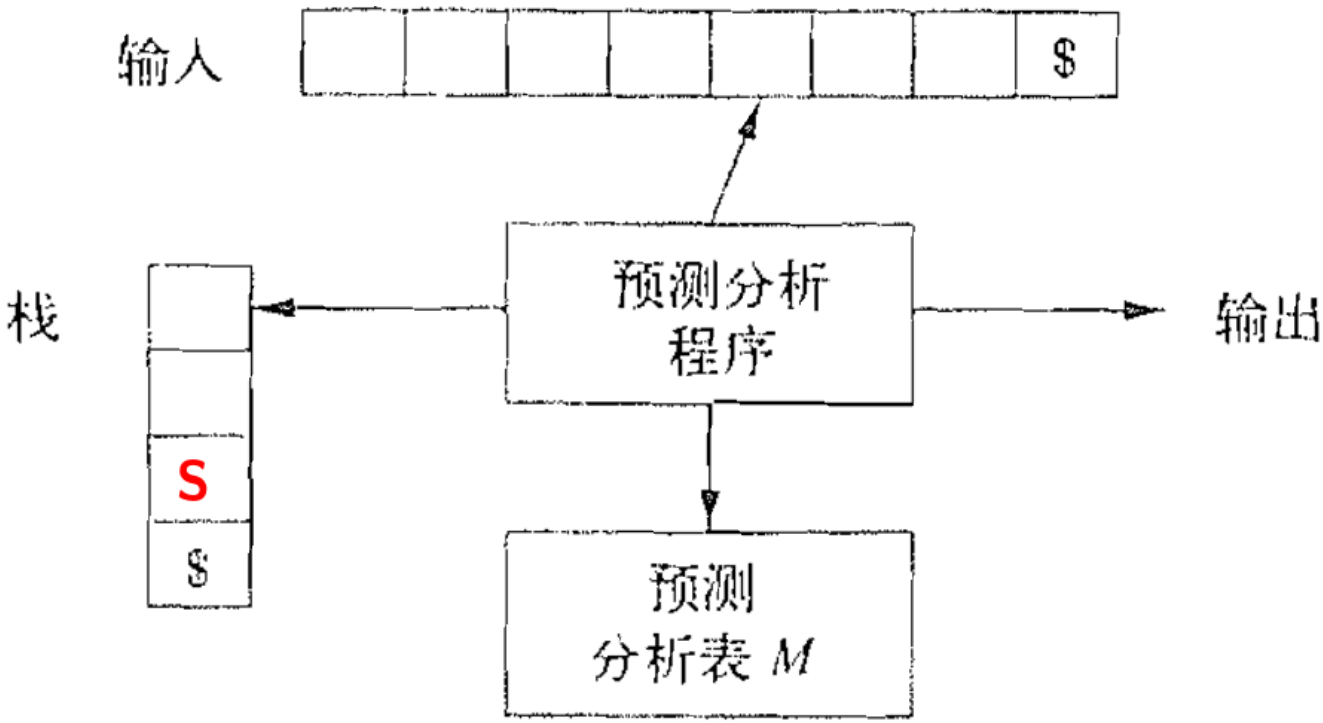
1：只需向前看一个输入符号便可确定使用哪条产生式



What is $LL(0)$?

什么都不看，每个非终结符的展开式是唯一的

非递归的预测分析算法



非递归的预测分析算法

设置 ip 使它指向 w 的第一个符号，其中 ip 是输入指针；
令 X = 栈顶符号；
while ($X \neq \$$) { /* 栈非空 */
    if (X 等于 ip 所指向的符号 a) 执行栈的弹出操作，将 ip 向前移动一个位置；
    else if (X 是一个终结符号) $error()$;
    else if ($M[X, a]$ 是一个报错条目) $error()$;
    else if ($M[X, a] = X \to Y_1 Y_2 \cdots Y_k$) {
        输出产生式 $X \to Y_1 Y_2 \cdots Y_k$;
        弹出栈顶符号；
        将 $Y_k, Y_{k-1}, \ldots, Y_1$ 压入栈中，其中 Y_1 位于栈顶。
    }
    令 X = 栈顶符号；
}

反向压栈

不是 $LL(1)$ 文法怎么办?

改造它

消除左递归

提取左公因子

E 在**不消耗任何词法单元**的情况下, 直接递归调用 E, 造成**死循环**

$$E \to E + E \mid E - E \mid E * E \mid E/E \mid (E) \mid \mathbf{id} \mid \mathbf{num}$$

$$\begin{array}{c} E \to E + T \mid E - T \mid T \\ T \to T * F \mid T/F \mid F \\ F \to (E) \mid \mathbf{id} \mid \mathbf{num} \end{array}$$

$$\textsc{First}(E + T) \cap \textsc{First}(T) \neq \emptyset$$

不是 $LL(1)$ 文法

改写成“右递归”文法

$$E \to TE'$$

$$E' \to +TE' \mid \epsilon$$

$$T \to FT'$$

$$T' \to *FT' \mid \epsilon$$

$$F \to (E) \mid \mathbf{id}$$

非终结符号	输入符号					
	id	+	*	(	)	$
E	$E \to TE'$			$E \to TE'$		
E'		$E' \to +TE'$			$E' \to \epsilon$	$E' \to \epsilon$
T	$T \to FT'$			$T \to FT'$		
T'		$T' \to \epsilon$	$T' \to *FT'$		$T' \to \epsilon$	$T' \to \epsilon$
F	$F \to \mathbf{id}$			$F \to (E)$		

$$\textsc{First}(F) = \{(, \mathbf{id}\}$$

$$\textsc{First}(T) = \{(, \mathbf{id}\}$$

$$\textsc{First}(E) = \{(, \mathbf{id}\}$$

$$\textsc{First}(E') = \{+, \epsilon\}$$

$$\textsc{First}(T') = \{*, \epsilon\}$$

$$\textsc{Follow}(E) = \textsc{Follow}(E') = \{), \$\}$$

$$\textsc{Follow}(T) = \textsc{Follow}(T') = \{+,), \$\}$$

$$\textsc{Follow}(F) = \{+, *,), \$\}$$

文件结束符 $ 的用处

已匹配	栈	输入	动作	
	$E\$$	id + id * id$		
	$TE'\$$	id + id * id$	输出	$E \to TE'$
	$FT'E'\$$	id + id * id$	输出	$T \to FT'$
	id $T'E'\$$	id + id * id$	输出	$F \to$ id
id	$T'E'\$$	+ id * id$	匹配	id
id	$E'\$$	+ id * id$	输出	$T' \to \epsilon$
id	$+ TE'\$$	+ id * id$	输出	$E' \to + TE'$
id +	$TE'\$$	id * id$	匹配	+
id +	$FT'E'\$$	id * id$	输出	$T \to FT'$
id +	id $T'E'\$$	id * id$	输出	$F \to$ id
id + id	$T'E'\$$	* id$	匹配	id
id + id	$* FT'E'\$$	* id$	输出	$T' \to * FT'$
id + id *	$FT'E'\$$	id$	匹配	*
id + id *	id $T'E'\$$	id$	输出	$F \to$ id
id + id * id	$T'E'\$$	$	匹配	id
id + id * id	$E'\$$	$	输出	$T' \to \epsilon$
id + id * id	$\$$	$	输出	$E' \to \epsilon$

句型

图 4-21　对输入 id + id * id 进行预测分析时执行的步骤

直接左递归 (Direct Left Recursion)

$$A \to A\alpha \mid \beta$$

$$A \to \beta A'$$

$$A' \to \alpha A' \mid \epsilon$$

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

其中, β_i 都不以 A 开头

$$A \to \beta_1 A' \mid \beta_2 A' \mid \cdots \mid \beta_n A'$$

$$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \cdots \mid \alpha_m A' \mid \epsilon$$

间接左递归 (Indirect Left Recursion)

$$S \to Ac \mid c$$

$$A \to Bb \mid b$$

$$B \to Sa \mid a$$

$$S \Longrightarrow Ac \Longrightarrow Bbc \Longrightarrow Sabc$$

$$A_i \to A_j\alpha \Longrightarrow i < j$$

1) 按照某个顺序将非终结符号排序为 $A_1, A_2, \ldots, A_n$.
2) for (从 1 到 n 的每个 i) {
3) for (从 1 到 $i-1$ 的每个 j) {
4) 将每个形如 $A_i \to A_j\gamma$ 的产生式替换为产生式组 $A_i \to \delta_1\gamma \mid \delta_2\gamma \mid \cdots \mid \delta_k\gamma$, 其中 $A_j \to \delta_1 \mid \delta_2 \mid \cdots \mid \delta_k$ 是所有的 A_j 产生式
5) }
6) 消除 A_i 产生式之间的立即左递归
7) }

$$
\begin{aligned}
S &\to Ac \mid c \\
A &\to Bb \mid b \\
B &\to Sa \mid a
\end{aligned}
$$

$$
\begin{aligned}
B &\to Sa \mid a \\
&\to (Ac \mid c)a \mid a \\
&\to Aca \mid ca \mid a \\
&\to (Bb \mid b)\mathtt{c}a \mid ca \mid a \\
&\to Bb\mathtt{c}a \mid b\mathtt{c}a \mid ca \mid a
\end{aligned}
$$

$$
\begin{aligned}
S &\to Ac \mid c \\
A &\to Bb \mid b \\
B &\to (bca \mid ca \mid a)B' \\
B' &\to bcaB' \mid \epsilon
\end{aligned}
$$

$$A_i \to A_j\alpha \implies i < j$$

算法要求:

文法中不存在环 (形如 $A \overset{*}{\Rightarrow} A$ 的推导)

文法中不存在 ϵ 产生式 (形如 $A \to \epsilon$ 的产生式)

$$S \to Aa \mid b$$

$$A \to Ac \mid Sb \mid \epsilon$$

提取左公因子 (Left-Factoring)

```
7   /*
8   decl : 'int' ID ';'
9        | 'int' ID '=' ID ';'
10       ;
11  */
12
13  decl : 'int' ID optional_init ';' ;
14  optional_init
15      : '=' ID  # Init
16      |         # NoInit
17      ;
18
19  /*
20  decl : 'int' ID ('=' ID)? ';'
21  */
```

ANTLR 4 可以处理有左公因子的文法

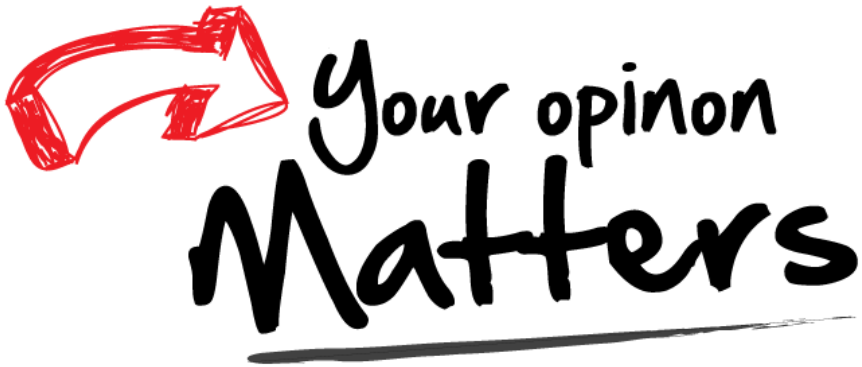
```
stat : 'if' expr 'then' stat
     | 'if' expr 'then' stat 'else' stat
     | expr
     ;
```

```
stat : 'if' expr 'then' stat stat_prime ;
stat_prime : 'else' stat
           |
           ;

expr : ID ;
```

很明显, 提取左公因子无助于消除文法二义性


Thank
You!



Office 926

hfwei@nju.edu.cn