

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

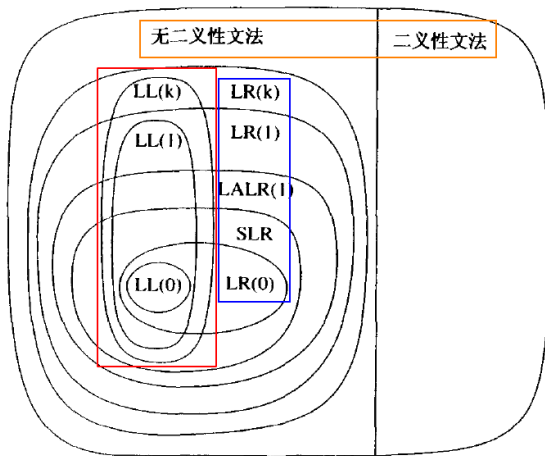
2023 年 03 月 31 日



构建语法分析树: 自顶向下 *vs.* 自底向上

$\langle \text{Stmt} \rangle$			
if ($\langle \text{Expr} \rangle$)	$\langle \text{Stmt} \rangle$
if ($\langle \text{Expr} \rangle$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$)	$\langle \text{Stmt} \rangle$
if ($\langle \text{Id} \rangle$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$)	$\langle \text{Stmt} \rangle$
if (x $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$)	$\langle \text{Stmt} \rangle$
if (x > $\langle \text{Expr} \rangle$)	$\langle \text{Stmt} \rangle$
if (x > $\langle \text{Num} \rangle$)	$\langle \text{Stmt} \rangle$
if (x > 9)	$\langle \text{Stmt} \rangle$
if (x > 9	{ $\langle \text{StmtList} \rangle$ }	
if (x > 9	{ $\langle \text{StmtList} \rangle$ $\langle \text{Stmt} \rangle$ }	
if (x > 9	{ $\langle \text{Stmt} \rangle$ $\langle \text{Stmt} \rangle$ }	
if (x > 9	{ $\langle \text{Id} \rangle = \langle \text{Expr} \rangle$; $\langle \text{Stmt} \rangle$ }	
if (x > 9	{ x = $\langle \text{Expr} \rangle$; $\langle \text{Stmt} \rangle$ }	
if (x > 9	{ x = $\langle \text{Num} \rangle$; $\langle \text{Stmt} \rangle$ }	
if (x > 9	{ x = 0 ; $\langle \text{Stmt} \rangle$ }	
if (x > 9	{ x = 0 ; $\langle \text{Id} \rangle = \langle \text{Expr} \rangle$; }	
if (x > 9	{ x = 0 ; y = $\langle \text{Expr} \rangle$; }	
if (x > 9	{ x = 0 ; y = $\langle \text{Expr} \rangle$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$; }	
if (x > 9	{ x = 0 ; y = $\langle \text{Id} \rangle$ $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$; }	
if (x > 9	{ x = 0 ; y = y $\langle \text{Optr} \rangle$ $\langle \text{Expr} \rangle$; }	
if (x > 9	{ x = 0 ; y = y + $\langle \text{Expr} \rangle$; }	
if (x > 9	{ x = 0 ; y = y + $\langle \text{Num} \rangle$; }	
if (x > 9	{ x = 0 ; y = y + 1 ; }	

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

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)

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

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**

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

```
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,  $A \rightarrow X_1 X_2 \cdots X_k$ ;  
2) for (  $i = 1$  to  $k$  ) {  
3)   if (  $X_i$  是一个非终结符号 )  
4)     递归下降调用其它非终结符对应的递归函数  
5)     调用过程  $X_i()$ ;  
6)   else if (  $X_i$  等于当前的输入符号  $a$  )  
7)     读入下一个输入符号;  
   else /* 发生了一个错误 */;  
}
```

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

$$S \rightarrow F$$

$$S \rightarrow (S + F)$$

$$F \rightarrow a$$

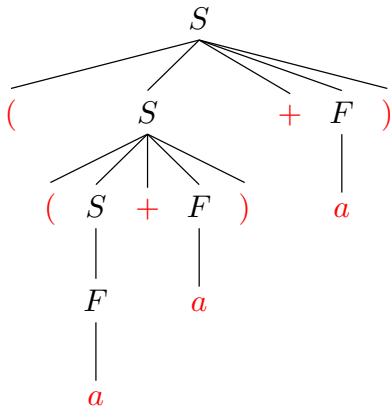
$$w = ((a + a) + a)$$

演示递归下降过程

$$S \rightarrow F$$

$$S \rightarrow (S + F)$$

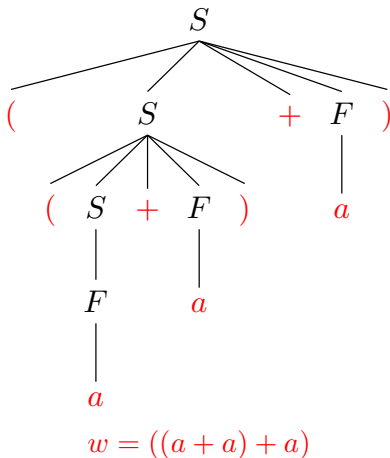
$$F \rightarrow a$$



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

演示递归下降过程

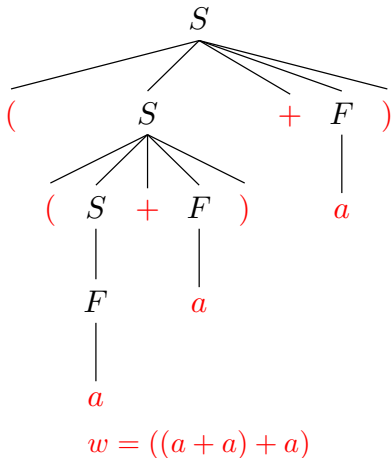
$$\begin{aligned} S &\rightarrow F \\ S &\rightarrow (S + F) \\ F &\rightarrow a \end{aligned}$$



每次都选择语法分析树**最左边**的非终结符进行展开

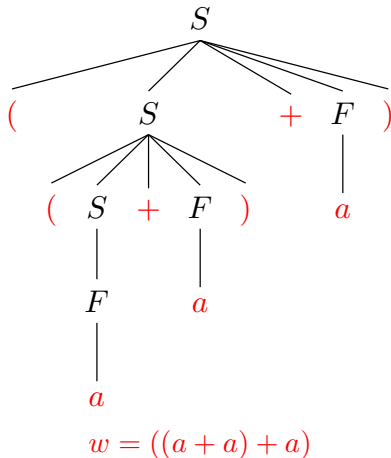
同样是展开非终结符 S ,
为什么前两次选择了 $S \rightarrow (S + F)$, 而第三次选择了 $S \rightarrow F$?

$S \rightarrow F$
$S \rightarrow (S + F)$
$F \rightarrow a$



同样是展开非终结符 S ,
为什么前两次选择了 $S \rightarrow (S + F)$, 而第三次选择了 $S \rightarrow F$?

$S \rightarrow F$
$S \rightarrow (S + F)$
$F \rightarrow a$



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

$S \rightarrow F$
$S \rightarrow (S + F)$
$F \rightarrow a$

		()	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)

$S \rightarrow F$
$S \rightarrow (S + F)$
$F \rightarrow a$

	()	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

$$S \rightarrow F$$
$$S \rightarrow (S + F)$$
$$F \rightarrow a$$

	()	a	+	\$
S	2		1		
F			3		

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

```
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'})
```

递归下降的、预测分析实现方法

$$S \rightarrow F$$

$$S \rightarrow (S + F)$$

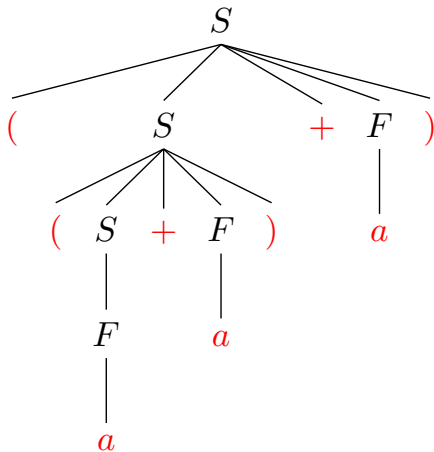
$$F \rightarrow a$$

	()	a	+	\$
S	2		1		
F			3		

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

```
1: procedure F()
2:   if token = 'a' then
3:     MATCH('a')
4:   else
5:     ERROR(token, {'a'})
```

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
```

```
;
```

```
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?

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

Definition ($\text{FIRST}(\alpha)$ 集合)

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

$$\text{FIRST}(\alpha) = \{t \in T \cup \{\epsilon\} \mid \alpha \xRightarrow{*} t\beta \vee \alpha \xRightarrow{*} \epsilon\}.$$

如何计算给定文法 G 的预测分析表?

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

Definition ($\text{FIRST}(\alpha)$ 集合)

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

$$\text{FIRST}(\alpha) = \{t \in T \cup \{\epsilon\} \mid \alpha \xRightarrow{*} t\beta \vee \alpha \xRightarrow{*} \epsilon\}.$$

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

如果**它们对应的 $\text{FIRST}(\alpha_i)$ 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?

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

Definition ($\text{FOLLOW}(A)$ 集合)

对于任意的 (产生式的左部) 非终结符 $A \in N$:

$$\text{FOLLOW}(A) = \{t \in T \cup \{\$\} \mid \exists s. S \xRightarrow{*} s \triangleq \beta A t \gamma\}.$$

如何计算给定文法 G 的预测分析表?

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

Definition ($\text{FOLLOW}(A)$ 集合)

对于任意的 (产生式的左部) 非终结符 $A \in N$:

$$\text{FOLLOW}(A) = \{t \in T \cup \{\$\} \mid \exists s. S \xRightarrow{*} s \triangleq \beta A t \gamma\}.$$

考虑产生式 $A \rightarrow \alpha$,

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

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

先计算每个符号 X 的 $\text{FIRST}(X)$ 集合

1: **procedure** $\text{FIRST}(X)$

先计算每个符号 X 的 $\text{FIRST}(X)$ 集合

```
1: procedure FIRST( $X$ )  
2:   if  $X \in T$  then  
3:     FIRST( $X$ ) =  $X$ 
```

▷ 规则 1: X 是终结符

先计算每个符号 X 的 $\text{FIRST}(X)$ 集合

```
1: procedure FIRST( $X$ )  
2:   if  $X \in T$  then                                ▷ 规则 1:  $X$  是终结符  
3:     FIRST( $X$ ) =  $X$   
  
4:   for  $X \rightarrow Y_1 Y_2 \dots Y_k$  do                    ▷ 规则 2:  $X$  是非终结符  
5:     FIRST( $X$ )  $\leftarrow$  FIRST( $X$ )  $\cup$  {FIRST( $Y_1$ )  $\setminus$  { $\epsilon$ }}  
6:     for  $i \leftarrow 2$  to  $k$  do  
7:       if  $\epsilon \in L(Y_1 \dots Y_{i-1})$  then  
8:         FIRST( $X$ )  $\leftarrow$  FIRST( $X$ )  $\cup$  {FIRST( $Y_i$ )  $\setminus$  { $\epsilon$ }}
```

先计算每个符号 X 的 $\text{FIRST}(X)$ 集合

```
1: procedure FIRST( $X$ )
2:   if  $X \in T$  then                                ▷ 规则 1:  $X$  是终结符
3:     FIRST( $X$ ) =  $X$ 
4:   for  $X \rightarrow Y_1 Y_2 \dots Y_k$  do                    ▷ 规则 2:  $X$  是非终结符
5:     FIRST( $X$ )  $\leftarrow$  FIRST( $X$ )  $\cup$  {FIRST( $Y_1$ )  $\setminus$  { $\epsilon$ }}
6:     for  $i \leftarrow 2$  to  $k$  do
7:       if  $\epsilon \in L(Y_1 \dots Y_{i-1})$  then
8:         FIRST( $X$ )  $\leftarrow$  FIRST( $X$ )  $\cup$  {FIRST( $Y_i$ )  $\setminus$  { $\epsilon$ }}
9:       if  $\epsilon \in L(Y_1 \dots Y_k)$  then                ▷ 规则 3:  $X$  可推导出空串
10:      FIRST( $X$ )  $\leftarrow$  FIRST( $X$ )  $\cup$  { $\epsilon$ }
```

先计算每个符号 X 的 $\text{FIRST}(X)$ 集合

```
1: procedure  $\text{FIRST}(X)$ 
2:   if  $X \in T$  then                                ▷ 规则 1:  $X$  是终结符
3:      $\text{FIRST}(X) = X$ 
4:   for  $X \rightarrow Y_1 Y_2 \dots Y_k$  do                    ▷ 规则 2:  $X$  是非终结符
5:      $\text{FIRST}(X) \leftarrow \text{FIRST}(X) \cup \{\text{FIRST}(Y_1) \setminus \{\epsilon\}\}$ 
6:     for  $i \leftarrow 2$  to  $k$  do
7:       if  $\epsilon \in L(Y_1 \dots Y_{i-1})$  then
8:          $\text{FIRST}(X) \leftarrow \text{FIRST}(X) \cup \{\text{FIRST}(Y_i) \setminus \{\epsilon\}\}$ 
9:       if  $\epsilon \in L(Y_1 \dots Y_k)$  then                ▷ 规则 3:  $X$  可推导出空串
10:       $\text{FIRST}(X) \leftarrow \text{FIRST}(X) \cup \{\epsilon\}$ 
```

不断应用上面的规则, 直到每个 $\text{FIRST}(X)$ 都不再变化

先计算每个符号 X 的 $\text{FIRST}(X)$ 集合

```
1: procedure FIRST( $X$ )  
2:   if  $X \in T$  then                                ▷ 规则 1:  $X$  是终结符  
3:     FIRST( $X$ ) =  $X$   
  
4:   for  $X \rightarrow Y_1 Y_2 \dots Y_k$  do                    ▷ 规则 2:  $X$  是非终结符  
5:     FIRST( $X$ )  $\leftarrow$  FIRST( $X$ )  $\cup$  {FIRST( $Y_1$ )  $\setminus$  { $\epsilon$ }}  
6:     for  $i \leftarrow 2$  to  $k$  do  
7:       if  $\epsilon \in L(Y_1 \dots Y_{i-1})$  then  
8:         FIRST( $X$ )  $\leftarrow$  FIRST( $X$ )  $\cup$  {FIRST( $Y_i$ )  $\setminus$  { $\epsilon$ }}  
9:       if  $\epsilon \in L(Y_1 \dots Y_k)$  then                ▷ 规则 3:  $X$  可推导出空串  
10:      FIRST( $X$ )  $\leftarrow$  FIRST( $X$ )  $\cup$  { $\epsilon$ }
```

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

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

$$\alpha = X\beta$$

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

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

$$(1) X \rightarrow Y$$

$$(2) X \rightarrow a$$

$$(3) Y \rightarrow \epsilon$$

$$(4) Y \rightarrow c$$

$$(5) Z \rightarrow d$$

$$(6) Z \rightarrow XYZ$$

$$(1) X \rightarrow Y$$

$$(2) X \rightarrow a$$

$$(3) Y \rightarrow \epsilon$$

$$(4) Y \rightarrow c$$

$$(5) Z \rightarrow d$$

$$(6) Z \rightarrow XYZ$$

$$\text{FIRST}(X) = \{a, c, \epsilon\}$$

$$\text{FIRST}(Y) = \{c, \epsilon\}$$

$$\text{FIRST}(Z) = \{a, c, d\}$$

$$\text{FIRST}(XYZ) = \{a, c, d\}$$

为每个非终结符 X 计算 $\text{FOLLOW}(X)$ 集合

1: **procedure** FOLLOW(X)

为每个非终结符 X 计算 FOLLOW(X) 集合

```
1: procedure FOLLOW( $X$ )  
2:   for  $X$  是开始符号 do  
3:     FOLLOW( $X$ )  $\leftarrow$  FOLLOW( $X$ )  $\cup$   $\{\$$ 
```

▷ 规则 1: X 是开始符号

为每个非终结符 X 计算 $\text{FOLLOW}(X)$ 集合

-
- 1: **procedure** FOLLOW(X)
 - 2: **for** X 是开始符号 **do** ▷ 规则 1: X 是开始符号
 - 3: $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup \{\$\}$
 - 4: **for** $A \rightarrow \alpha X$ **do** ▷ 规则 2: X 是某产生式右部的最后一个符号
 - 5: $\text{FOLLOW}(X) \leftarrow \text{FOLLOW}(X) \cup \text{FOLLOW}(A)$

为每个非终结符 X 计算 $\text{FOLLOW}(X)$ 集合

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

为每个非终结符 X 计算 $\text{FOLLOW}(X)$ 集合

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

不断应用上面的规则, 直到每个 $\text{FOLLOW}(X)$ 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)
```

$$(1) X \rightarrow Y$$

$$(2) X \rightarrow a$$

$$(3) Y \rightarrow \epsilon$$

$$(4) Y \rightarrow c$$

$$(5) Z \rightarrow d$$

$$(6) Z \rightarrow XYZ$$

$$(1) X \rightarrow Y$$

$$(2) X \rightarrow a$$

$$(3) Y \rightarrow \epsilon$$

$$(4) Y \rightarrow c$$

$$(5) Z \rightarrow d$$

$$(6) Z \rightarrow XYZ$$

$$\text{FOLLOW}(X) = \{a, c, d, \$\}$$

$$\text{FOLLOW}(Y) = \{a, c, d, \$\}$$

$$\text{FOLLOW}(Z) = \emptyset$$

如何根据FIRST 与 FOLLOW 集合计算给定文法 G 的预测分析表?

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 $A \rightarrow \alpha$ 与终结符 t , 如果

$$t \in \text{FIRST}(\alpha) \quad (1)$$

$$\alpha \xRightarrow{*} \epsilon \wedge t \in \text{FOLLOW}(A) \quad (2)$$

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

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 $A \rightarrow \alpha$ 与终结符 t , 如果

$$t \in \text{FIRST}(\alpha) \quad (1)$$

$$\alpha \xRightarrow{*} \epsilon \wedge t \in \text{FOLLOW}(A) \quad (2)$$

则在表格 $[A, t]$ 中填入 $A \rightarrow \alpha$ (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。

$$t \in \text{FIRST}(\alpha) \quad (1)$$

$$\epsilon \in \text{FIRST}(\alpha) \wedge t \in \text{FOLLOW}(A) \quad (2)$$

当下的选择未必正确，但“你别无选择”。

$$(1) X \rightarrow Y$$

$$(2) X \rightarrow a$$

$$(3) Y \rightarrow \epsilon$$

$$(4) Y \rightarrow c$$

$$(5) Z \rightarrow d$$

$$(6) Z \rightarrow XYZ$$

$$\text{FIRST}(X) = \{a, c, \epsilon\}$$

$$\text{FIRST}(Y) = \{c, \epsilon\}$$

$$\text{FIRST}(Z) = \{a, c, d\}$$

$$\text{FIRST}(XYZ) = \{a, c, d\}$$

$$\text{FOLLOW}(X) = \{a, c, d, \$\}$$

$$\text{FOLLOW}(Y) = \{a, c, d, \$\}$$

$$\text{FOLLOW}(Z) = \emptyset$$

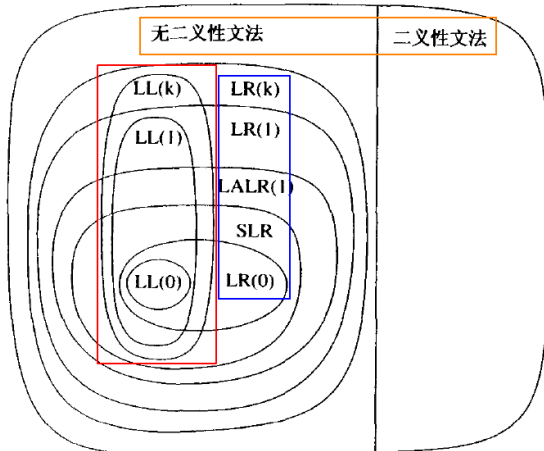
	<i>a</i>	<i>c</i>	<i>d</i>	\$
<i>X</i>	1, 2	1	1	1
<i>Y</i>	3	3, 4	3	3
<i>Z</i>	6	6	5, 6	

$LL(1)$ 语法分析器

L : 从左向右 (left-to-right) 扫描输入

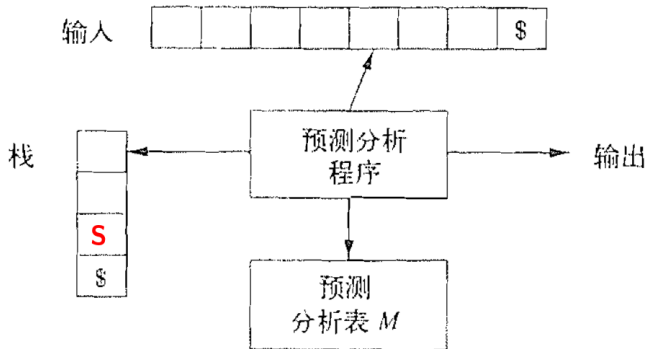
L : 构建最左 (leftmost) 推导

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 $LL(0)$?

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 ($M[X, a] = X \rightarrow Y_1 Y_2 \cdots Y_k$) {

输出产生式 $X \rightarrow Y_1 Y_2 \cdots Y_k$;

弹出栈顶符号;

将 Y_k, Y_{k-1}, \dots, Y_1 压入栈中, 其中 Y_1 位于栈顶。

}

令 X = 栈顶符号;

}

不是 $LL(1)$ 文法怎么办?

不是 $LL(1)$ 文法怎么办?

改造它

不是 $LL(1)$ 文法怎么办?

改造它

消除左递归

提取左公因子

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

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

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

$$F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$$

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**

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

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

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

$$F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$$

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**

$$E \rightarrow E + E \mid E - E \mid E * E \mid E / E \mid (E) \mid \text{id} \mid \text{num}$$

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

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

$$F \rightarrow (E) \mid \text{id} \mid \text{num}$$

$$\text{FIRST}(E + T) \cap \text{FIRST}(T) \neq \emptyset$$

不是 $LL(1)$ 文法

改写成“右递归”文法

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

改写成“右递归”文法

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

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

$$\text{FIRST}(F) = \{ (, \mathbf{id} \}$$

$$\text{FIRST}(T) = \{ (, \mathbf{id} \}$$

$$\text{FIRST}(E) = \{ (, \mathbf{id} \}$$

$$\text{FIRST}(E') = \{ +, \epsilon \}$$

$$\text{FIRST}(T') = \{ *, \epsilon \}$$

$$\text{FOLLOW}(E) = \text{FOLLOW}(E') = \{), \$ \}$$

$$\text{FOLLOW}(T) = \text{FOLLOW}(T') = \{ +,), \$ \}$$

$$\text{FOLLOW}(F) = \{ +, *,), \$ \}$$

改写成“右递归”文法

$$E \rightarrow TE'$$

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

$$T \rightarrow FT'$$

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

$$F \rightarrow (E) \mid \text{id}$$

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

$$\text{FIRST}(F) = \{ (, \text{id} \}$$

$$\text{FIRST}(T) = \{ (, \text{id} \}$$

$$\text{FIRST}(E) = \{ (, \text{id} \}$$

$$\text{FIRST}(E') = \{ +, \epsilon \}$$

$$\text{FIRST}(T') = \{ *, \epsilon \}$$

$$\text{FOLLOW}(E) = \text{FOLLOW}(E') = \{), \$ \}$$

$$\text{FOLLOW}(T) = \text{FOLLOW}(T') = \{ +,), \$ \}$$

$$\text{FOLLOW}(F) = \{ +, *,), \$ \}$$

文件结束符 \$ 的用处

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

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)

$$A \rightarrow A\alpha \mid \beta$$

$$A \rightarrow \beta A'$$

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

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

其中, β_i 都不以 A 开头

$$A \rightarrow \beta_1 A' \mid \beta_2 A' \mid \dots \mid \beta_n A'$$

$$A' \rightarrow \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \alpha_m A' \mid \epsilon$$

间接左递归 (Indirect Left Recursion)

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sb \mid \epsilon$$

$$S \Longrightarrow Aa \Longrightarrow Sba$$

间接左递归 (Indirect Left Recursion)

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sb \mid \epsilon$$

$$S \Longrightarrow Aa \Longrightarrow Sba$$

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

间接左递归 (Indirect Left Recursion)

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sb \mid \epsilon$$

$$S \Rightarrow Aa \Rightarrow Sba$$

$$A_i \rightarrow A_j \alpha \Rightarrow i < j$$

- 1) 按照某个顺序将非终结符号排序为 A_1, A_2, \dots, 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 $A_i \rightarrow A_j \gamma$ 的产生式替换为产生式组 $A_i \rightarrow \delta_1 \gamma \mid \delta_2 \gamma \mid \dots \mid \delta_k \gamma$,
 其中 $A_j \rightarrow \delta_1 \mid \delta_2 \mid \dots \mid \delta_k$ 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) }

图 4-11 消除文法中的左递归的算法

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Sb \mid \epsilon$$

$$S \rightarrow Aa \mid b$$

$$A \rightarrow Ac \mid Aad \mid bd \mid \epsilon$$

$$S \rightarrow Aa \mid b$$

$$A \rightarrow bdA' \mid A'$$

$$A' \rightarrow cA' \mid adA' \mid \epsilon$$

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

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  */
```

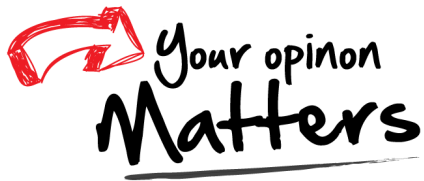
ANTLR 4 可以处理有左公因子的文法

```
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