

语法分析

(5. LR0 语法分析器)

魏恒峰

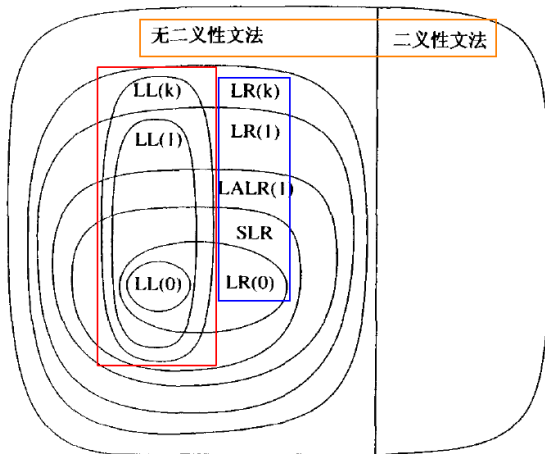
hfwei@nju.edu.cn

2021 年 11 月 30 日



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



今日份主题: **LR 语法分析器**



“打哪指哪”的思维方式

自顶向下的、
递归下降的、
预测分析的、
适用于 $LL(1)$ 文法的、
 $LL(1)$ 语法分析器

$LL(k)$ 的弱点:

在仅看到右部的 k 个词法单元时必须预测要使用哪条产生式

$LL(k)$ 的弱点:

在仅看到右部的前 k 个词法单元时就必须预测要使用哪条产生式

$LR(k)$ 的优点:

看到与正在考虑的这个产生式的整个右部对应的词法单元之后再决定

自底向上的、
不断归约的、
基于句柄识别自动机的、
适用于 LR 文法的、
 LR 语法分析器

自底向上构建语法分析树

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 $\$$

自底向上构建语法分析树

根节点是文法的起始符号 S

每个中间非终结符节点表示使用它的某条产生式进行归约

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 $\$$

自顶向下的“推导”与 自底向上的“归约”

$$E \xRightarrow{\text{rm}} T \xRightarrow{\text{rm}} T * F \xRightarrow{\text{rm}} T * \mathbf{id} \xRightarrow{\text{rm}} F * \mathbf{id} \xRightarrow{\text{rm}} \mathbf{id} * \mathbf{id}$$

$$(1) E \rightarrow E + T$$

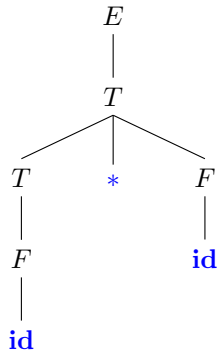
$$(2) E \rightarrow T$$

$$(3) T \rightarrow T * F$$

$$(4) T \rightarrow F$$

$$(5) F \rightarrow (E)$$

$$(6) F \rightarrow \mathbf{id}$$



$$w = \mathbf{id} * \mathbf{id}$$

$$E \Leftarrow T \Leftarrow T * F \Leftarrow T * \mathbf{id} \Leftarrow F * \mathbf{id} \Leftarrow \mathbf{id} * \mathbf{id}$$

“推导” ($A \rightarrow \alpha$) 与 “归约” ($A \leftarrow \alpha$)

$$S \triangleq \gamma_0 \Rightarrow \dots \gamma_{i-1} \Rightarrow \gamma_i \Rightarrow \gamma_{r+1} \Rightarrow \dots \Rightarrow r_n = w$$

$$S \triangleq \gamma_0 \Leftarrow \dots \gamma_{i-1} \Leftarrow \gamma_i \Leftarrow \gamma_{r+1} \Leftarrow \dots \Leftarrow r_n = w$$

自底向上语法分析器为输入构造**反向推导**

LR 语法分析器

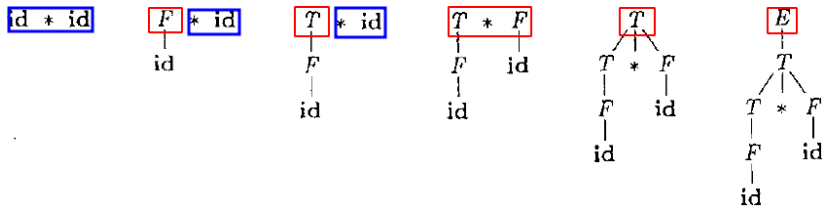
L : 从左向右 (Left-to-right) 扫描输入

R : 构建反向 (Reverse) 最右 (Rightmost) 推导

“反向最右推导”与“从左到右扫描”相一致

LR 语法分析器的状态

在任意时刻, 语法分析树的**上边缘**与**剩余的输入**构成当前句型



$$E \leftarrow T \leftarrow T * F \leftarrow T * id \leftarrow F * id \leftarrow id * id$$

LR 语法分析器使用**栈**存储语法分析树的**上边缘**

它包含了语法分析器目前所知的所有信息

板书演示“栈”上操作

$$(1) E \rightarrow E + T$$

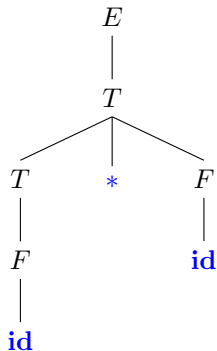
$$(2) E \rightarrow T$$

$$(3) T \rightarrow T * F$$

$$(4) T \rightarrow F$$

$$(5) F \rightarrow (E)$$

$$(6) F \rightarrow \text{id}$$



$w = \text{id} * \text{id}$

两大操作: 移入输入符号 与 按产生式归约

直到栈中仅剩开始符号 S , 且输入已结束, 则成功停止

基于栈的 LR 语法分析器

Q_1 : 何时归约? (何时移入?)

Q_2 : 按哪条产生式进行归约?

基于栈的 LR 语法分析器

$$(1) E \rightarrow E + T$$

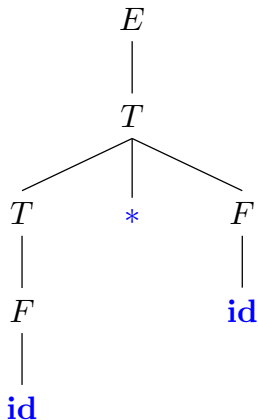
$$(2) E \rightarrow T$$

$$(3) T \rightarrow T * F$$

$$(4) T \rightarrow F$$

$$(5) F \rightarrow (E)$$

$$(6) F \rightarrow \text{id}$$



为什么第二个 F 以 $T * F$ 整体被归约为 T ?

这与栈的当前状态 “ $T * F$ ” 相关

LR (SLR) 分析表指导 LR 语法分析器

状态	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

在当前状态 (编号)下, 面对当前文法符号时, 该采取什么动作

ACTION 表指明动作, GOTO 表仅用于归约时的状态转换

状态	ACTION					GOTO			
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

<i>sn</i>	移入输入符号, 并进入状态 <i>n</i>
<i>rk</i>	使用 <i>k</i> 号产生式进行归约
<i>gn</i>	转换到状态 <i>n</i>
<i>acc</i>	成功接受, 结束
空白	错误

再次板书演示“栈”上操作: 移人与归约

(1) $E \rightarrow E + T$

(2) $E \rightarrow T$

(3) $T \rightarrow T * F$

(4) $T \rightarrow F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow \text{id}$

状态	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

$w = \text{id} * \text{id}\$$

栈中存储语法分析器的状态(编号), “编码”了语法分析树的上边缘

```

1: procedure LR()
2:   PUSH( $S, s_0$ )                                ▷ 或 PUSH( $S, \$_{s_0}$ )
3:   token  $\leftarrow$  NEXT-TOKEN()
4:   while (1) do
5:      $s \leftarrow$  TOP( $S$ )
6:     if ACTION[ $s, \text{token}$ ] =  $s_i$  then                ▷ 移入
7:       PUSH( $S, i$ )                                ▷ 或 PUSH( $S, \text{token}_{s_i}$ )
8:       token  $\leftarrow$  NEXT-TOKEN()
9:     else if ACTION[ $s, \text{token}$ ] =  $r_j$  then            ▷ 归约;  $j : A \rightarrow \alpha$ 
10:       $|\alpha|$  次 POP( $S$ )
11:       $s \leftarrow$  TOP( $S$ )
12:      PUSH( $S, \text{GOTO}[s, A]$ ) ▷ 转换状态; 或 PUSH( $S, A_{\text{GOTO}[s, A]}$ )
13:    else if ACTION[ $s, \text{token}$ ] =  $acc$  then            ▷ 接受
14:      break
15:    else
16:      ERROR(...)

```

行号	栈	符号	输入	动作
(1)	0	\$	id * id \$	移入到 5
(2)	0 5	\$ id	* id \$	按照 $F \rightarrow id$ 归约
(3)	0 3	\$ F	* id \$	按照 $T \rightarrow F$ 归约
(4)	0 2	\$ T	* id \$	移入到 7
(5)	0 2 7	\$ T *	id \$	移入到 5
(6)	0 2 7 5	\$ T * id	\$	按照 $F \rightarrow id$ 归约
(7)	0 2 7 10	\$ T * F	\$	按照 $T \rightarrow T * F$ 归约
(8)	0 2	\$ T	\$	按照 $E \rightarrow T$ 归约
(9)	0 1	\$ E	\$	接受

$w = \text{id} * \text{id}$ 的分析过程

如何构造 LR 分析表?

状态	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

在当前状态 (编号)下, 面对当前文法符号时, 该采取什么动作

状态是什么？如何跟踪状态？

状态	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

状态是语法分析树的上边缘, 存储在栈中

可以用**自动机**跟踪状态变化 (自动机中的路径 \Leftrightarrow 栈中符号/状态编号)

何时归约？使用哪条产生式进行归约？

状态	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	s7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

必要条件: 当前状态中, 已观察到某个产生式的完整右部

何时归约？使用哪条产生式进行归约？

Definition (句柄 (Handle))

在输入串的 (唯一) 反向最右推导中, **如果** 下一步是逆用产生式 $A \rightarrow \alpha$ 将 α 归约为 A , 则称 α 是**当前句型的句柄**。

最右句型	句柄	归约用的产生式
$\boxed{id_1} * id_2$	id_1	$F \rightarrow id$
$\boxed{F} * id_2$	F	$T \rightarrow F$
$T * \boxed{id_2}$	id_2	$F \rightarrow id$
$\boxed{T * F}$	$T * F$	$T \rightarrow T * F$
\boxed{T}	T	$E \rightarrow T$

LR 语法分析器的关键就是高效**寻找每个归约步骤所使用的句柄**。

句柄可能在哪里？

Theorem

存在一种 LR 语法分析方法, 保证句柄总是出现在栈顶。

句柄可能在哪里？

Theorem

存在一种 LR 语法分析方法，保证句柄总是出现在栈顶。

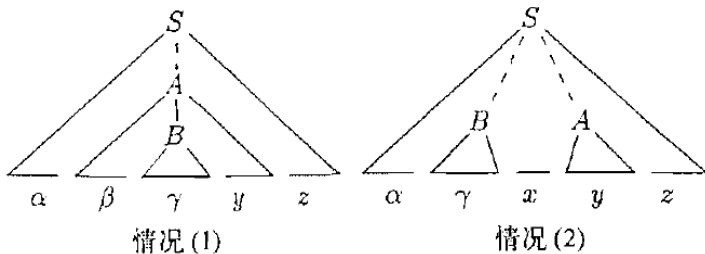


图 4-29 一个最右推导中两个连续步骤的两种情况

$$S \xrightarrow{*}_{\text{rm}} \alpha Az \xrightarrow{*}_{\text{rm}} \alpha \beta B y z \xrightarrow{*}_{\text{rm}} \alpha \beta \gamma y z \quad S \xrightarrow{*}_{\text{rm}} \alpha B x A z \xrightarrow{*}_{\text{rm}} \alpha B x y z \xrightarrow{*}_{\text{rm}} \alpha \gamma x y z$$

可以用**自动机**跟踪**栈**的状态变化
(自动机中的路径 \Leftrightarrow 栈中符号/状态编号)

Theorem

存在一种 LR 语法分析方法, 保证**句柄总是出现在栈顶**。

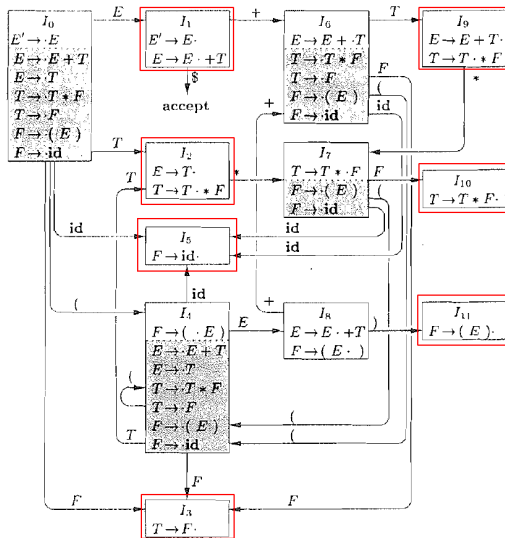
可以用**自动机**跟踪**栈**的状态变化
(自动机中的路径 \Leftrightarrow 栈中符号/状态编号)

Theorem

存在一种 LR 语法分析方法, 保证**句柄总是出现在栈顶**。

希望能够在自动机的当前状态识别可能的句柄

LR(0) 句柄识别有穷状态自动机 (Handle-Finding Automaton)



状态是什么？

状态刻画了“当前观察到的**针对所有产生式的右部的前缀**”

Definition ($LR(0)$ 项 (Item))

文法 G 的一个 $LR(0)$ **项**是 G 的某个产生式加上一个位于体部的**点**。

项指明了语法分析器已经观察到了某个产生式的某个前缀

状态刻画了“当前观察到的**针对所有产生式的右部的前缀**”

Definition ($LR(0)$ 项 (Item))

文法 G 的一个 $LR(0)$ **项**是 G 的某个产生式加上一个位于体部的**点**。

项指明了语法分析器已经观察到了某个产生式的某个前缀

$$A \rightarrow XYZ$$

$$[A \rightarrow \cdot XYZ]$$

$$[A \rightarrow X \cdot YZ]$$

$$[A \rightarrow XY \cdot Z]$$

$$[A \rightarrow XYZ \cdot]$$

(产生式 $A \rightarrow \epsilon$ 只有一个项 $[A \rightarrow \cdot]$)

状态刻画了“当前观察到的**针对所有产生式的右部的前缀**”

Definition (项集)

项集就是若干**项**构成的集合。

因此, 句柄识别自动机的一个**状态**可以表示为一个**项集**

状态刻画了“当前观察到的**针对所有产生式的右部的前缀**”

Definition (项集)

项集就是若干**项**构成的集合。

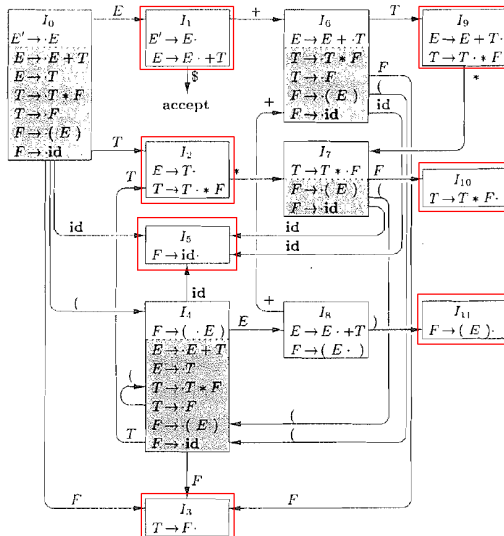
因此, 句柄识别自动机的一个**状态**可以表示为一个**项集**

Definition (项集族)

项集族就是若干**项集**构成的集合。

因此, 句柄识别自动机的**状态集**可以表示为一个**项集族**

LR(0) 句柄识别自动机



项、项集、项集族

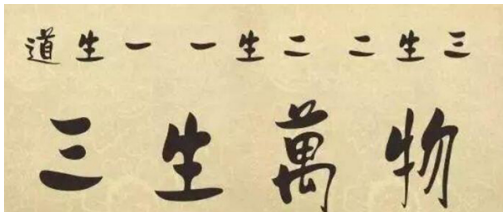
Definition (增广文法 (Augmented Grammar))

文法 G 的**增广文法** G' 是在 G 中加入产生式 $S' \rightarrow S$ 得到的文法。

目的: 告诉语法分析器何时停止分析并接受输入符号串

语法分析器**当前栈中仅有 S 且面对 $\$$** ,
要使用 $S' \rightarrow S$ 进行归约时, 输入符号串被接受

$LR(0)$ 句柄识别自动机



初始状态是什么？

点指示了栈顶, 左边 (与路径) 是栈中内容, 右边是期望看到的文法符号串

$$(0) E' \rightarrow E$$

$$(1) E \rightarrow E + T$$

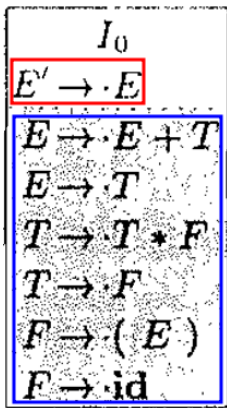
$$(2) E \rightarrow T$$

$$(3) T \rightarrow T * F$$

$$(4) T \rightarrow F$$

$$(5) F \rightarrow (E)$$

$$(6) F \rightarrow \text{id}$$



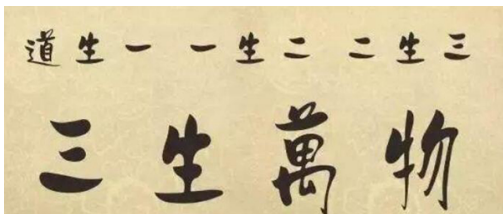
$\text{CLOSURE}(\{[E' \rightarrow \cdot E]\})$

$$\text{CLOSURE}(\{[E' \rightarrow \cdot E]\})$$

一开始, 栈为空, 期望输入是 E 可以展开得到的一个句子并以 $\$$ 结束。

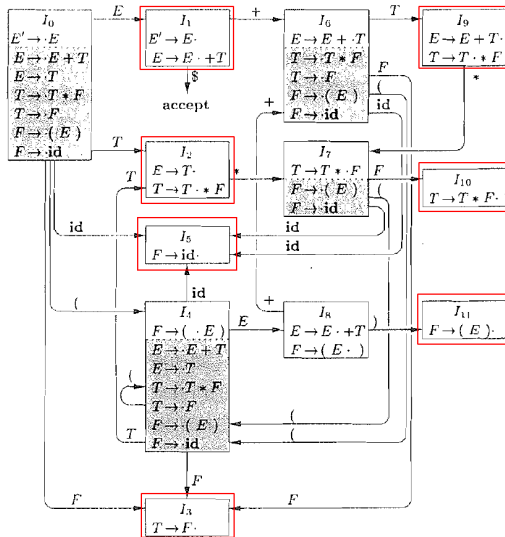
输入以 E 开始, 意味着它可能以 E 的任何一个右部开始。

$LR(0)$ 句柄识别自动机



状态之间如何转移？

板书演示 LR(0) 句柄识别自动机的构造过程



状态编号约定

```

SetOfItems CLOSURE( $I$ ) {
     $J = I$ ;
    repeat
        for ( $J$ 中的每个项  $A \rightarrow \alpha \cdot B \beta$ )
            for ( $G$  的每个产生式  $B \rightarrow \gamma$ )
                if (项  $B \rightarrow \cdot \gamma$  不在  $J$  中)
                    将  $B \rightarrow \cdot \gamma$  加入  $J$  中;
    until 在某一轮中没有新的项被加入到  $J$  中;
    return  $J$ ;
}

```

$$J = \text{GOTO}(I, X) = \text{CLOSURE}\left(\left\{[A \rightarrow \alpha X \cdot \beta] \mid [A \rightarrow \alpha \cdot X \beta] \in I\right\}\right)$$
$$(X \in N \cup T)$$

```
void items( $G'$ ) {
     $C = \{ \text{CLOSURE}(\{[S' \rightarrow \cdot S]\}) \}$ ;  初始状态
    repeat
        for ( $C$  中的每个项集  $I$ )
            for (每个文法符号  $X$ )
                if (GOTO( $I, X$ ) 非空且不在  $C$  中)
                    下一个状态 将 GOTO( $I, X$ ) 加入  $C$  中;
    until 在某一轮中没有新的项集被加入到  $C$  中;
}
```

图 4-33 规范 LR(0)项集族的计算

```

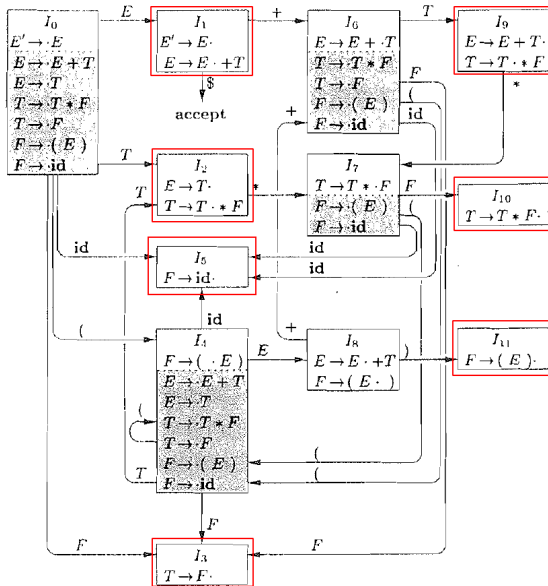
void items( $G'$ ) {
     $C = \{\text{CLOSURE}(\{[S' \rightarrow \cdot S]\})\}$ ; 初始状态
    repeat
        for (  $C$  中的每个项集  $I$  )
            for ( 每个文法符号  $X$  )
                if (  $\text{GOTO}(I, X)$  非空且不在  $C$  中 )
                    下一个状态 将  $\text{GOTO}(I, X)$  加入  $C$  中;
    until 在某一轮中没有新的项集被加入到  $C$  中;
}

```

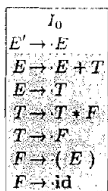
图 4-33 **规范 LR(0) 项集族** 的计算

接受状态: $F = \{I \in C \mid \exists k. [k : A \rightarrow \alpha \cdot] \in I\}$

(此时, 栈顶对应着 k 号产生式的完整的右部, **有望**进行归约)

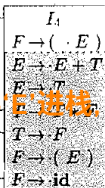


红色框中的状态为 **接受状态**

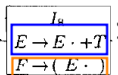


‘(’与‘E’进栈, 转而追踪“E+T”

(



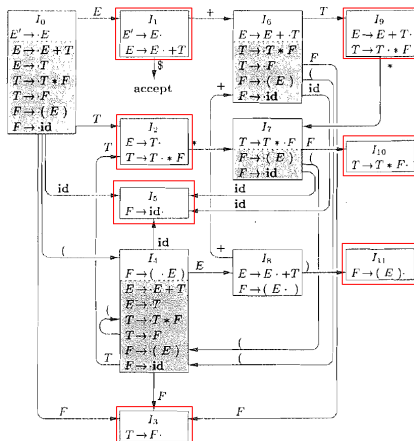
E



‘(’与‘E’进栈, 继续追踪“(E)”

点指示了栈顶, 左边 (与路径) 是栈中内容, 右边是期望看到的文法符号串

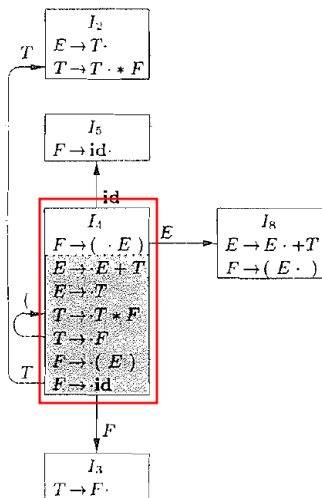
LR(0) 分析表



	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			g1	g2	g3
1		s6				acc			
2	r2	r2	s7, r2	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			g8	g2	g3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4				g9	g3
7	s5			s4				g10	
8		s6				s11			
9	r1	r1	s7, r1	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

GOTO 函数被拆分成 ACTION 表 (针对终结符) 与 GOTO 表 (针对非终结符)

$$(1) \text{GOTO}(I_i, a) = I_j \wedge a \in T \implies \text{ACTION}[i, a] \leftarrow sj$$



	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			g1	g2	g3
1	s6					acc			
2	r2	r2	s7, r2	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			g8	g2	g3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4				g9	g3
7	s5			s4					g10
8	s6					s11			
9	r1	r1	s7, r1	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

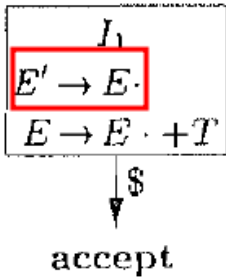
$$(2) \text{GOTO}(I_i, A) = I_j \wedge A \in N \implies \text{GOTO}[i, A] \leftarrow gj$$

I_9
$E \rightarrow T \cdot$
$T \rightarrow T \cdot * F$

I_{10}
$T \rightarrow T * F \cdot$

	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			g1	g2	g3
1		s6				acc			
2	r2	r2	s7, r2	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			g8	g2	g3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4				g9	g3
7	s5			s4					g10
8		s6			s11				
9	r1	r1	s7, r1	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

$$(3) [k : A \rightarrow \alpha \cdot] \in I_i \wedge A \neq S' \implies \forall t \in T \cup \{\$, \}. \text{ACTION}[i, t] = rk$$



	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			g1	g2	g3
1		s6				acc			
2	r2	r2	s7, r2	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			g8	g2	g3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4				g9	g3
7	s5			s4					g10
8		s6				s11			
9	r1	r1	s7, r1	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

$$(4) [S' \rightarrow S \cdot] \in I_i \implies \text{ACTION}[i, \$] \leftarrow acc$$

LR(0) 分析表构造规则总结

- (1) $\text{GOTO}(I_i, a) = I_j \wedge a \in T \implies \text{ACTION}[i, a] \leftarrow sj$
- (2) $\text{GOTO}(I_i, A) = I_j \wedge A \in N \implies \text{GOTO}[i, A] \leftarrow gj$
- (3) $[k : A \rightarrow \alpha \cdot] \in I_i \wedge A \neq S' \implies \forall t \in T \cup \{\$ \}. \text{ACTION}[i, t] = rk$
- (4) $[S' \rightarrow S \cdot] \in I_i \implies \text{ACTION}[i, \$] \leftarrow acc$

Definition ($LR(0)$ 文法)

如果文法 G 的 $LR(0)$ 分析表是无冲突的, 则 G 是 $LR(0)$ 文法。

	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			g1	g2	g3
1		s6				acc			
2	r2	r2	s7, r2	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			g8	g2	g3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4			g9	g3	
7	s5			s4				g10	
8		s6			s11				
9	r1	r1	s7, r1	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

非 $LR(0)$ 分析表/文法

$LR(0)$ 分析表每一行 (状态) 所选用的归约产生式是相同的

	ACTION						GOTO		
	id	+	*	()	\$	E	T	F
0	s5			s4			g1	g2	g3
1		s6				acc			
2	r2	r2	s7, r2	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			g8	g2	g3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4				g9	g3
7	s5			s4					g10
8		s6			s11				
9	r1	r1	s7, r1	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

归约时不需要向前看, 这就是“0”的含义

$LR(0)$ 语法分析器

L : 从左向右 (Left-to-right) 扫描输入

R : 构建反向 (Reverse) 最右推导

0 : 归约时无需向前看

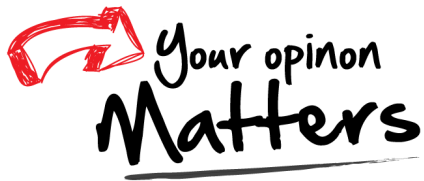
$LR(0)$ 自动机与栈之间的互动关系

向前走 \Leftrightarrow 移入

回溯 \Leftrightarrow 归约

自动机才是本质，栈是实现方式
(用栈记住“来时的路”，以便回溯)

Thank
You!



Office 926

hfwei@nju.edu.cn