

第五章 语法制导翻译

语法制导翻译方案SDT

哈尔滨工业大学 陈鄞



语法制导翻译方案SDT

- 语法制导翻译方案(SDT)是在产生式右部中嵌入了程序片段(称为语义动作)的CFG

➤ 例

$$\begin{aligned} D &\rightarrow T \{ L.inh = T.type \} L \\ T &\rightarrow \text{int} \{ T.type = \text{int} \} \\ T &\rightarrow \text{real} \{ T.type = \text{real} \} \\ L &\rightarrow \{ L_1.inh = L.inh \} L_1, \text{id} \\ &\dots \end{aligned}$$

语法制导翻译方案SDT

- 语法制导翻译方案(SDT)是在产生式右部中嵌入了程序片段(称为语义动作)的CFG
- SDT可以看作是SDD的具体实施方案
- 本节主要关注如何使用SDT来实现两类重要的SDD, 因为在这两种情况下, SDT可在语法分析过程中实现
 - 基本文法可以使用LR分析技术, 且SDD是S属性的
 - 基本文法可以使用LL分析技术, 且SDD是L属性的

将S-SDD转换为SDT

➤ 将一个S-SDD转换为SDT的方法：将每个语义动作都放在产生式的最后

➤ 例

S-SDD

产生式	语义规则
(1) $L \rightarrow E \text{ n}$	$L.val = E.val$
(2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
(3) $E \rightarrow T$	$E.val = T.val$
(4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
(5) $T \rightarrow F$	$T.val = F.val$
(6) $F \rightarrow (E)$	$F.val = E.val$
(7) $F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

SDT

(1) $L \rightarrow E \text{ n } \{ L.val = E.val \}$
(2) $E \rightarrow E_1 + T \{ E.val = E_1.val + T.val \}$
(3) $E \rightarrow T \{ E.val = T.val \}$
(4) $T \rightarrow T_1 * F \{ T.val = T_1.val \times F.val \}$
(5) $T \rightarrow F \{ T.val = F.val \}$
(6) $F \rightarrow (E) \{ F.val = E.val \}$
(7) $F \rightarrow \text{digit} \{ F.val = \text{digit.lexval} \}$

S-属性定义的SDT实现

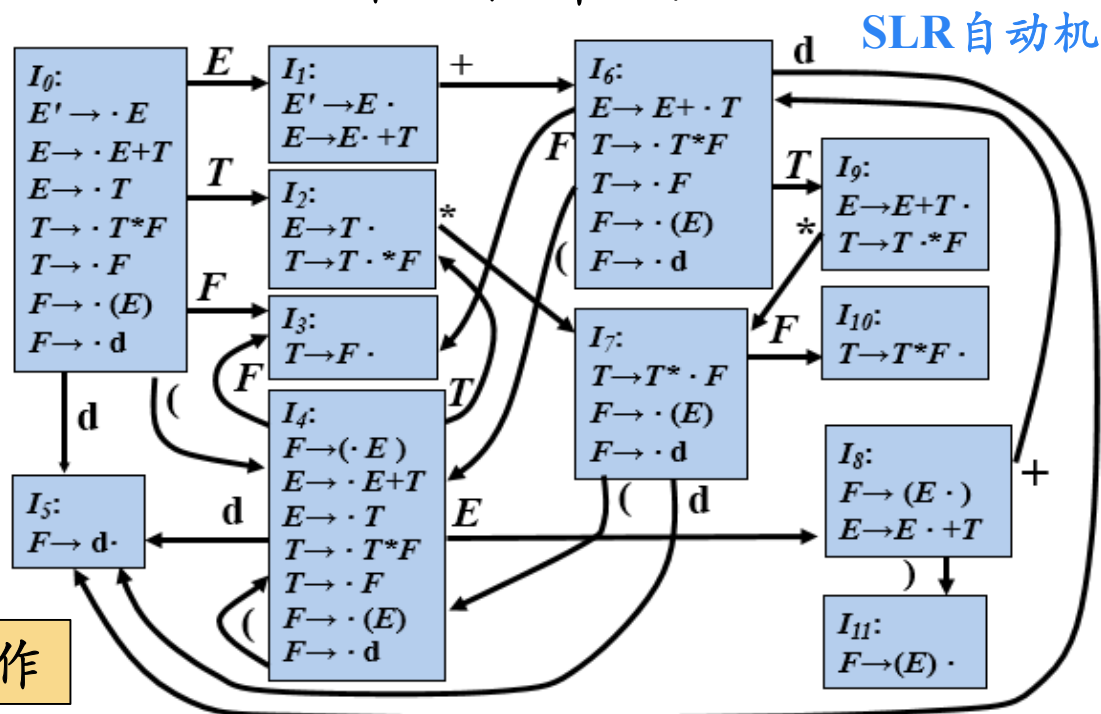
- 如果一个S-SDD的基本文法可以使用LR分析技术，那么它的SDT可以在LR语法分析过程中实现

➤ 例

S-SDD

产生式	语义规则
(1) $L \rightarrow E$ n	$L.val = E.val$
(2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$
(3) $E \rightarrow T$	$E.val = T.val$
(4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$
(5) $T \rightarrow F$	$T.val = F.val$
(6) $F \rightarrow (E)$	$F.val = E.val$
(7) $F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

当归约发生时执行相应的语义动作



扩展的LR语法分析栈

在分析栈中使用一个附加的域来存放综合属性值

	状态	文法符号	综合属性
	S_0	\$	

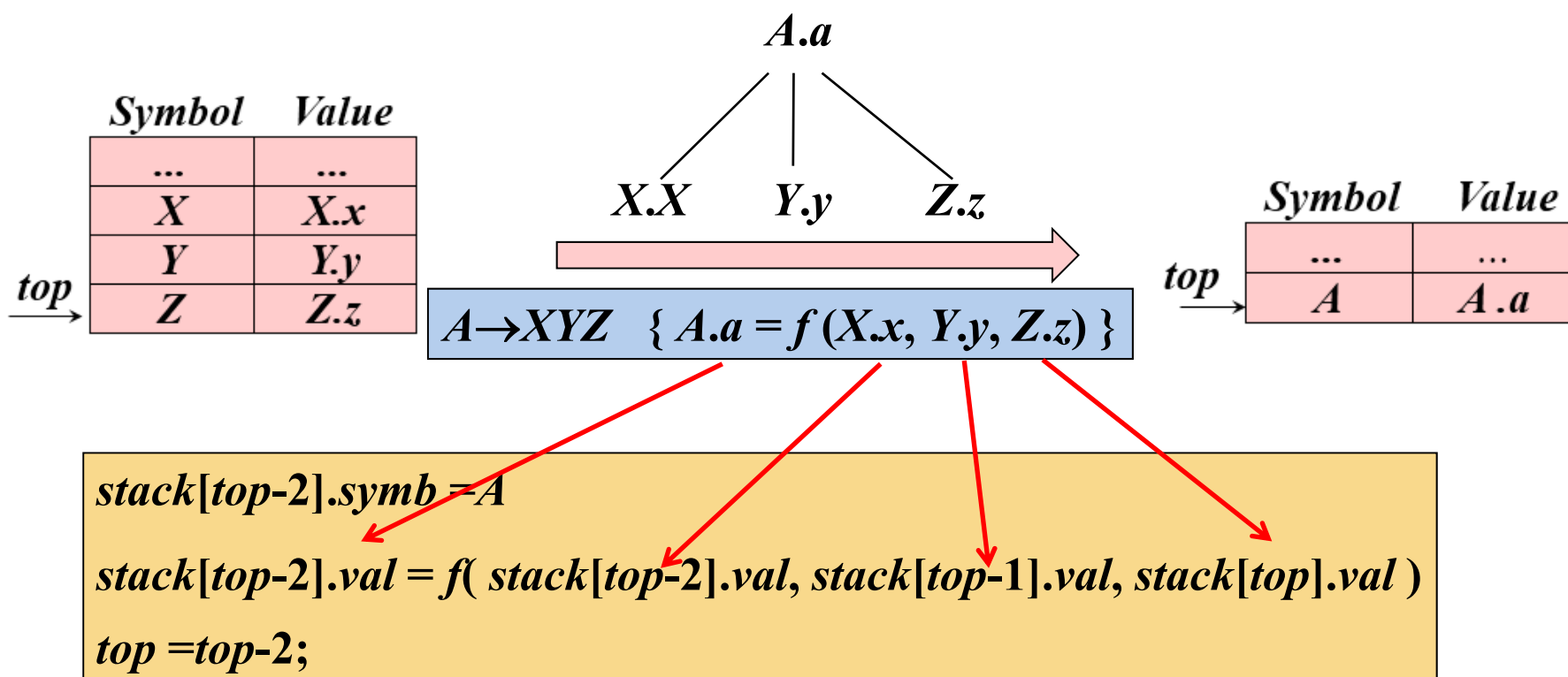
	S_{m-2}	X	$X.x$
	S_{m-1}	Y	$Y.y$
\xrightarrow{top}	S_m	Z	$Z.z$

➤ 若支持多个属性

➤ 使栈记录变得足够大

➤ 在栈记录中存放指针

将语义动作中的抽象定义式改写成具体可执行的栈操作



例：在自底向上语法分析栈中实现桌面计算器

产生式	语义动作	
(1) $E' \rightarrow E$	$\text{print}(E.val)$	{ $\text{print}(\text{stack}[\text{top}].val)$; }
(2) $E \rightarrow E_1 + T$	$E.val = E_1.val + T.val$	{ $\text{stack}[\text{top}-2].val = \text{stack}[\text{top}-2].val + \text{stack}[\text{top}].val$; $\text{top}=\text{top}-2$; }
(3) $E \rightarrow T$	$E.val = T.val$	
(4) $T \rightarrow T_1 * F$	$T.val = T_1.val \times F.val$	{ $\text{stack}[\text{top}-2].val = \text{stack}[\text{top}-2].val \times \text{stack}[\text{top}].val$; $\text{top}=\text{top}-2$; }
(5) $T \rightarrow F$	$T.val = F.val$	
(6) $F \rightarrow (E)$	$F.val = E.val$	{ $\text{stack}[\text{top}-2].val = \text{stack}[\text{top}-1].val$; $\text{top}=\text{top}-2$; }
(7) $F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$	

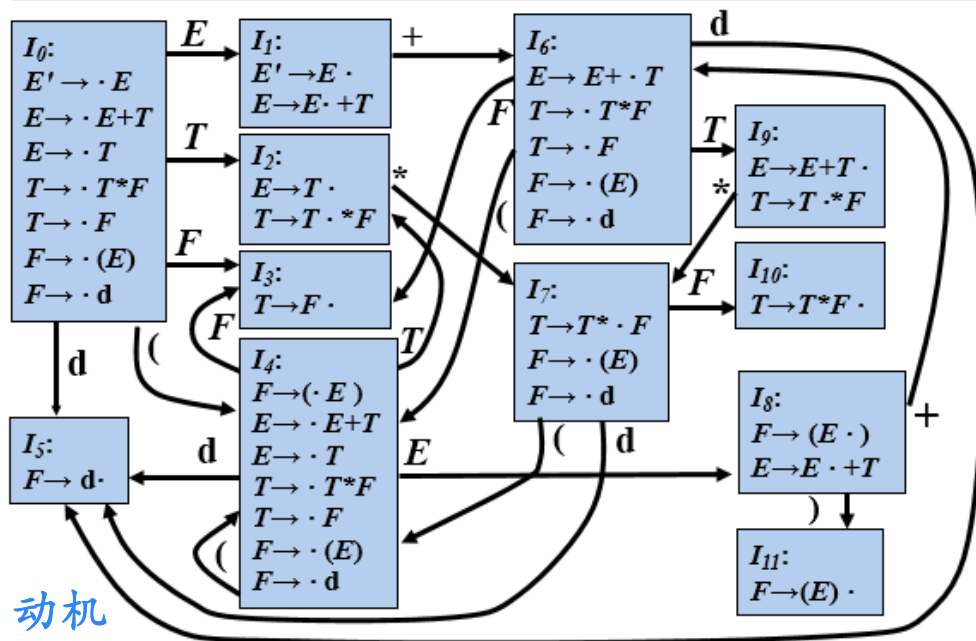


产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val ; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑

状态 符号 属性

0	\$	_
5	d	3



SLR自动机

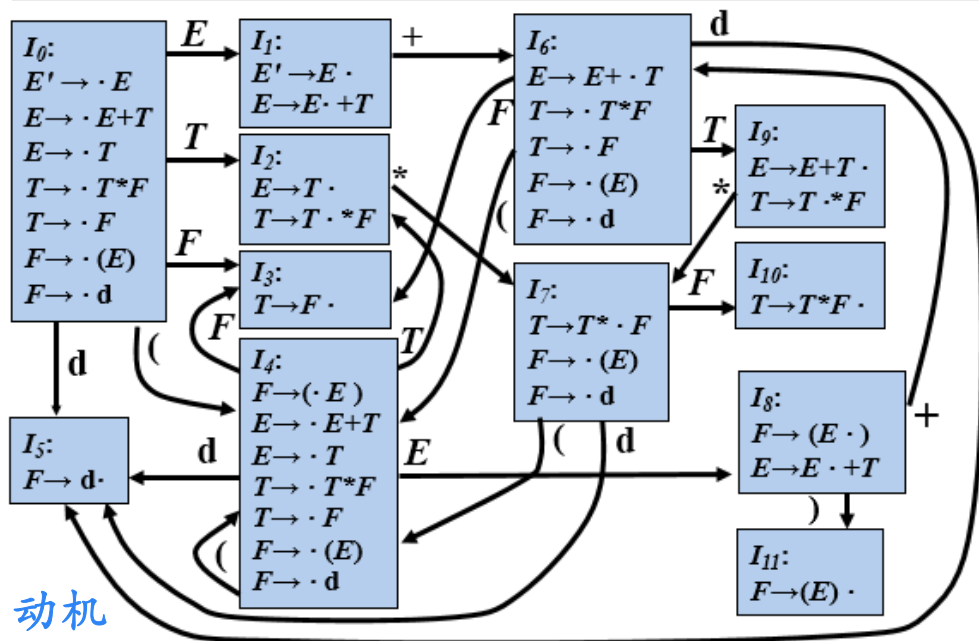


产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑

状态 符号 属性

0	\$	_
3	F	3



SLR自动机

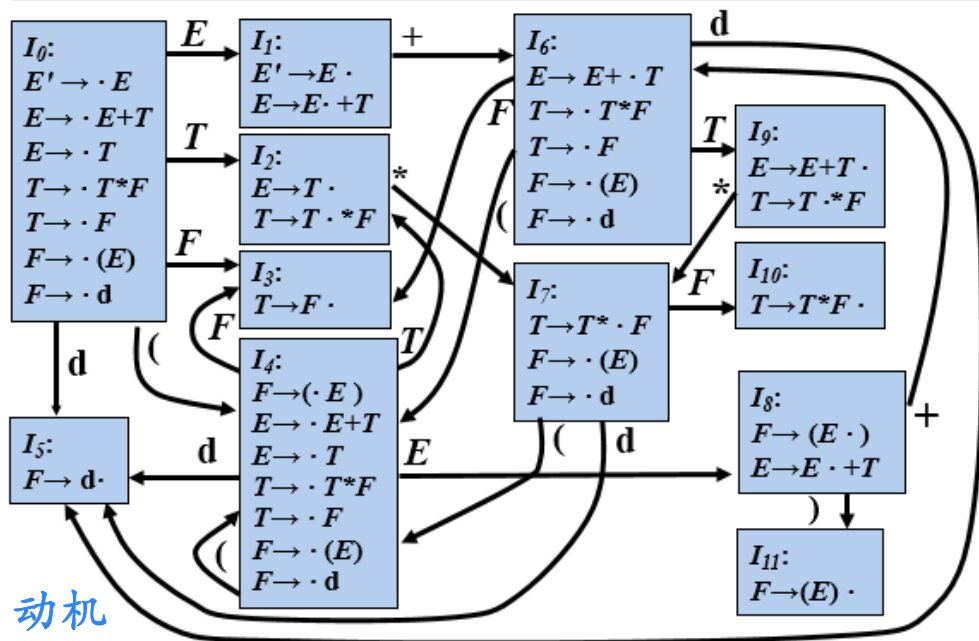


产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑↑↑

状态 符号 属性

0	\$	_
2	T	3
7	*	_
5	d	5



SLR自动机

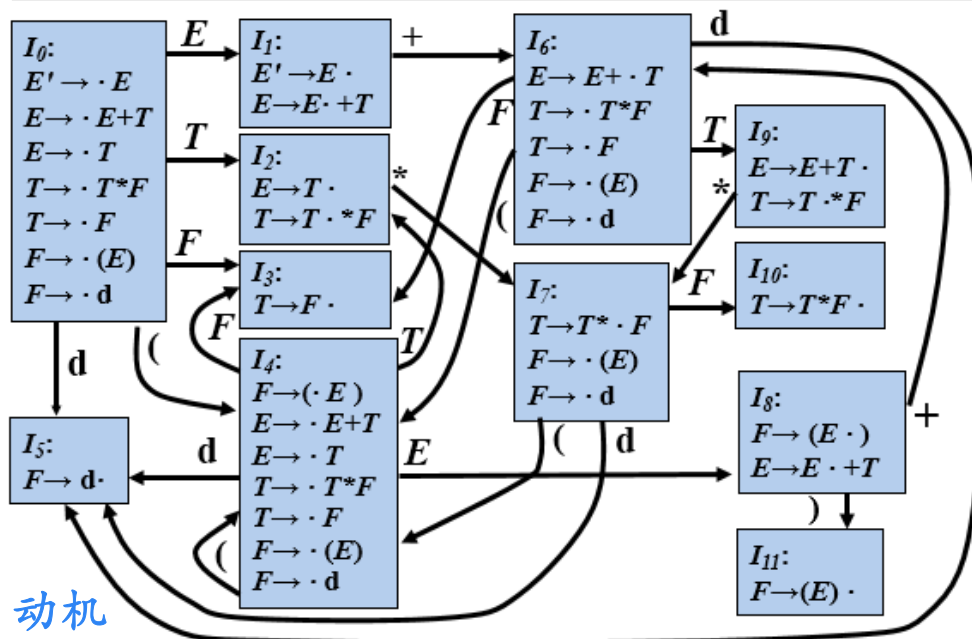


产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑↑↑

状态 符号 属性

0	\$	_
2	T	15
7	*	_
10	F	5



SLR自动机

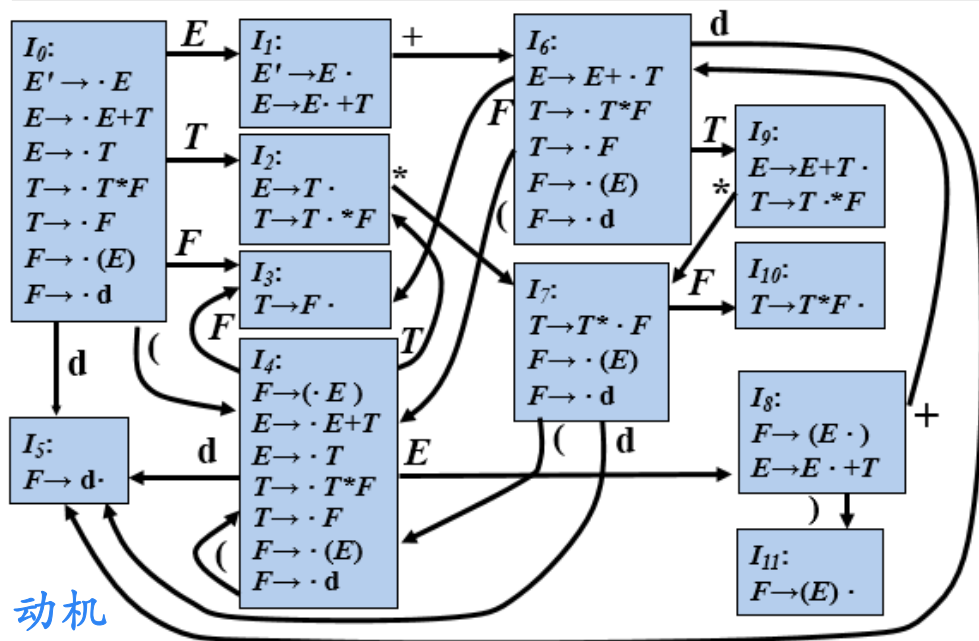


产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑↑↑

状态 符号 属性

0	\$	_
2	T	15



SLR自动机

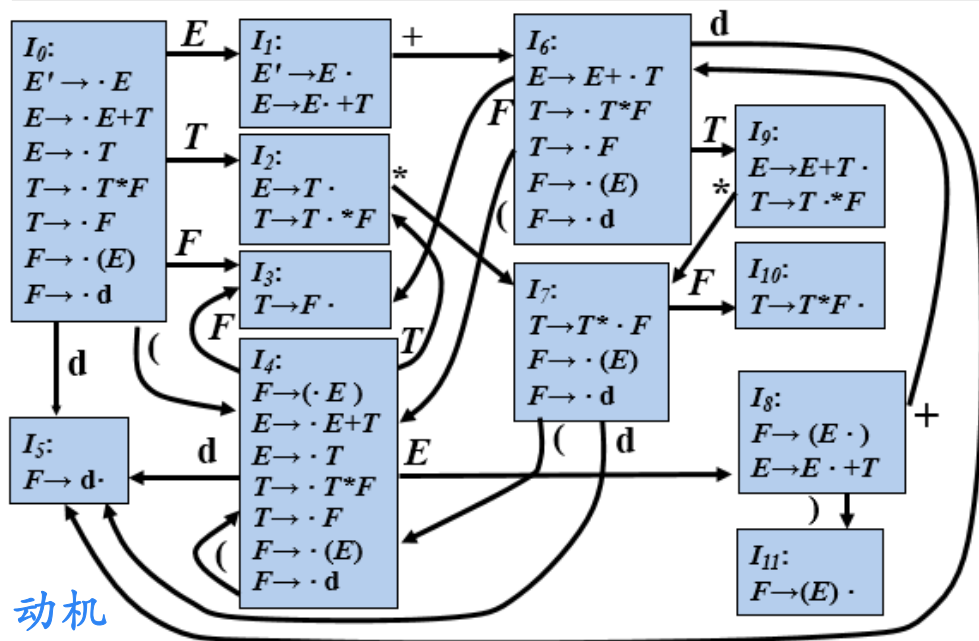


产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑↑↑↑↑↑

状态 符号 属性

0	\$	_
1	E	15
6	+	_
5	d	4



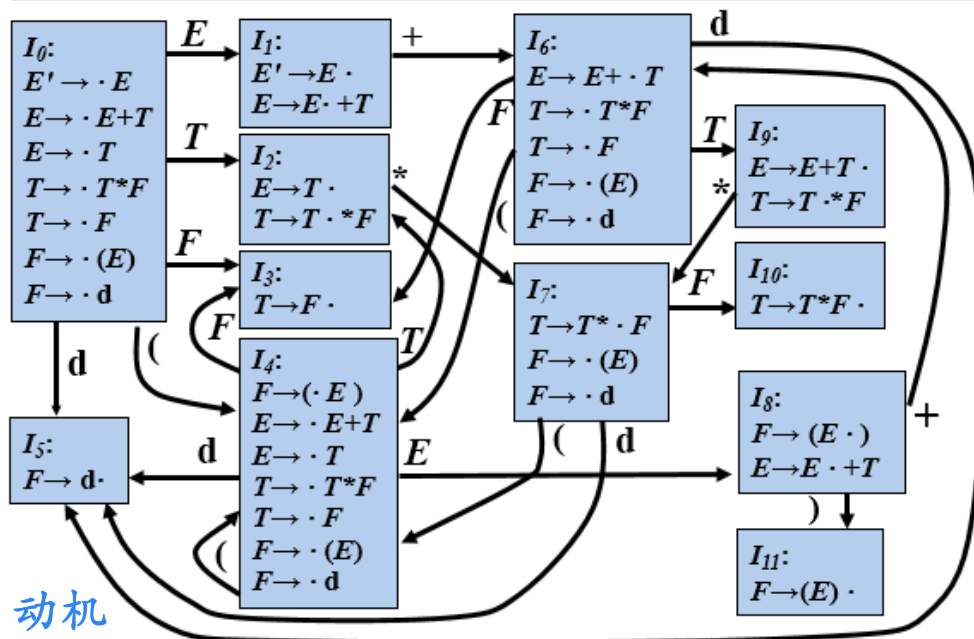
SLR自动机



产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑↑↑↑↑↑

状态	符号	属性
0	\$	-
1	E	15
6	+	-
3	F	4



SLR自动机

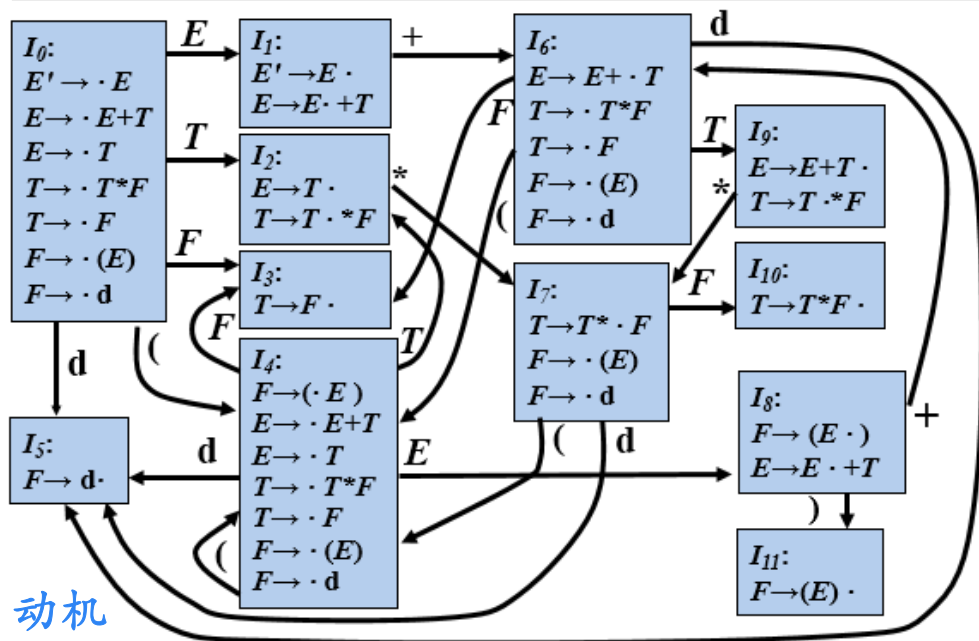


产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑↑↑↑↑↑

状态 符号 属性

0	\$	-
1	E	19
6	+	-
9	T	4



SLR自动机

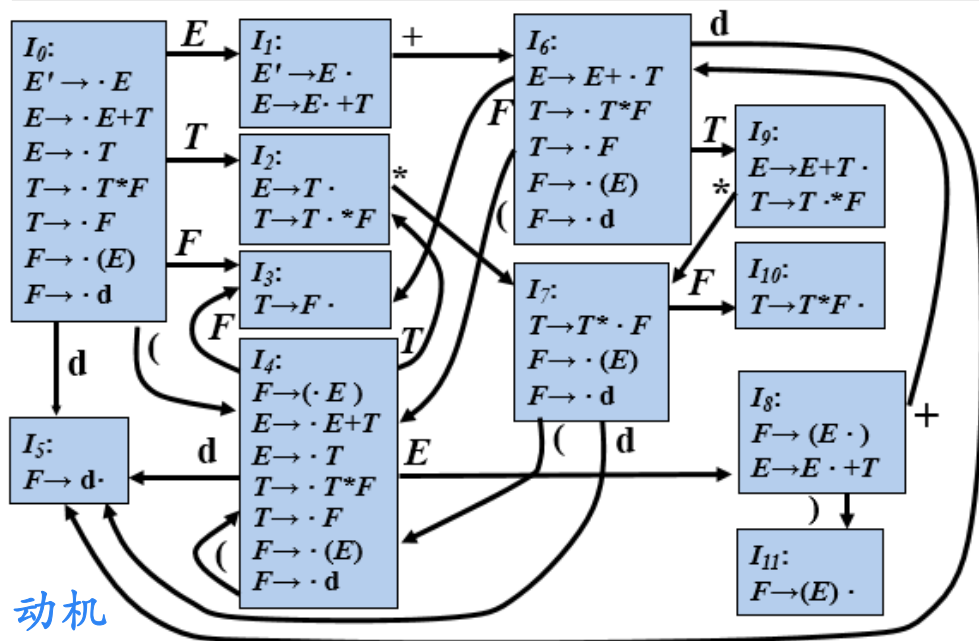


产生式	语义动作
(1) $E' \rightarrow E$	{ print (stack[top].val); }
(2) $E \rightarrow E_1 + T$	{ stack[top-2].val = stack[top-2].val + stack[top].val ; top=top-2; }
(3) $E \rightarrow T$	
(4) $T \rightarrow T_1 * F$	{ stack[top-2].val = stack[top-2].val \times stack[top].val ; top=top-2; }
(5) $T \rightarrow F$	
(6) $F \rightarrow (E)$	{ stack[top-2].val = stack[top-1].val; top=top-2; }
(7) $F \rightarrow \text{digit}$	

输入: 3*5+4
↑↑↑↑↑↑↑

状态 符号 属性

0	\$	_
1	E	19



SLR自动机

将 $L\text{-}SDD$ 转换为 SDT

➤ 将 $L\text{-}SDD$ 转换为 SDT 的规则

- 将计算某个非终结符号 A 的继承属性的动作插入到产生式右部中紧靠在 A 的本次出现之前的位置上
- 将计算一个产生式左部符号的综合属性的动作放置在这个产生式右部的最右端

消除直接左递归

$$\begin{array}{l}
 A \rightarrow A\alpha \mid \beta (\alpha \neq \varepsilon, \beta \text{ 不以 } A \text{ 开头}) \quad r = \beta\alpha^* \\
 \downarrow \\
 A \rightarrow \beta A' \\
 A' \rightarrow \alpha A' \mid \varepsilon
 \end{array}$$

事实上，这种消除过程就是把左递归转换成了右递归

$$\begin{array}{l}
 A \Rightarrow A\alpha \\
 \Rightarrow A\alpha\alpha \\
 \Rightarrow A\alpha\alpha\alpha \\
 \dots \\
 \Rightarrow A\alpha \dots \alpha \\
 \Rightarrow \beta \alpha \dots \alpha
 \end{array}$$

➤ 例

$$\begin{array}{l}
 E \rightarrow E + T \mid T \\
 T \rightarrow T * F \mid F \\
 F \rightarrow \text{digit}
 \end{array}$$

α (under T in $E \rightarrow E + T$)
 β (under T in $E \rightarrow E + T$)
 α (under F in $T \rightarrow T * F$)
 β (under F in $T \rightarrow T * F$)

$$\begin{array}{l}
 E \rightarrow T E' \\
 E' \rightarrow + T E' \mid \varepsilon \\
 T \rightarrow F T' \\
 T' \rightarrow * F T' \mid \varepsilon \\
 F \rightarrow \text{digit}
 \end{array}$$

$$\begin{array}{l}
 A' \Rightarrow \alpha A' \\
 \Rightarrow \alpha\alpha A' \\
 \Rightarrow \alpha\alpha\alpha A' \\
 \dots \\
 \Rightarrow \alpha \dots \alpha A' \\
 \Rightarrow \alpha \dots \alpha
 \end{array}$$

例

➤ L-SDD

	产生式	语义规则
(1)	$T \rightarrow F T'$	$T'.inh = F.val$ $T.val = T'.syn$
(2)	$T' \rightarrow * F T_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
(3)	$T' \rightarrow \varepsilon$	$T'.syn = T'.inh$
(4)	$F \rightarrow \text{digit}$	$F.val = \text{digit.lexval}$

➤ SDT

- 1) $T \rightarrow F \{ T'.inh = F.val \} T' \{ T.val = T'.syn \}$
- 2) $T' \rightarrow * F \{ T_1'.inh = T'.inh \times F.val \} T_1' \{ T'.syn = T_1'.syn \}$
- 3) $T' \rightarrow \varepsilon \{ T'.syn = T'.inh \}$
- 4) $F \rightarrow \text{digit} \{ F.val = \text{digit.lexval} \}$

L -属性定义的SDT实现

➤ 如果一个 L -SDD的基本文法可以使用 LL 分析技术，那么它的SDT可以在 LL 或 LR 语法分析过程中实现


➤ 例

- 1) $T \rightarrow F \{ T'.inh = F.val \} T' \{ T.val = T'.syn \}$
- 2) $T' \rightarrow *F \{ T_1'.inh = T'.inh \times F.val \} T_1' \{ T'.syn = T_1'.syn \}$
- 3) $T' \rightarrow \varepsilon \{ T'.syn = T'.inh \}$
- 4) $F \rightarrow \text{digit} \{ F.val = \text{digit.lexval} \}$

$SELECT(1) = \{ \text{digit} \}$
 $SELECT(2) = \{ * \}$
 $SELECT(3) = \{ \$ \}$
 $SELECT(4) = \{ \text{digit} \}$

L -属性定义的 SDT 实现

- 如果一个 L - SDD 的基本文法可以使用 LL 分析技术, 那么它的 SDT 可以在 LL 或 LR 语法分析过程中实现
 - 在非递归的预测分析过程中进行语义翻译
 - 在递归的预测分析过程中进行语义翻译
 - 在 LR 分析过程中进行语义翻译



第五章 语法制导翻译

语法制导翻译方案SDT

哈尔滨工业大学 陈鄞

