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

$$D \rightarrow T \{ L.inh = T.type \} L$$
$$T \rightarrow \textbf{int} \{ T.type = int \}$$
$$T \rightarrow \textbf{real} \{ T.type = real \}$$
$$L \rightarrow \{ L_1.inh = L.inh \} L_1, \textbf{id}$$
$$\ldots$$

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) $L \rightarrow E$ 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) $F \rightarrow$ digit	$F.val =$ digit.$lexval$

SDT

(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) $F \rightarrow$ digit $\{ F.val =$ digit.$lexval \}$

S-属性定义的*SDT* 实现

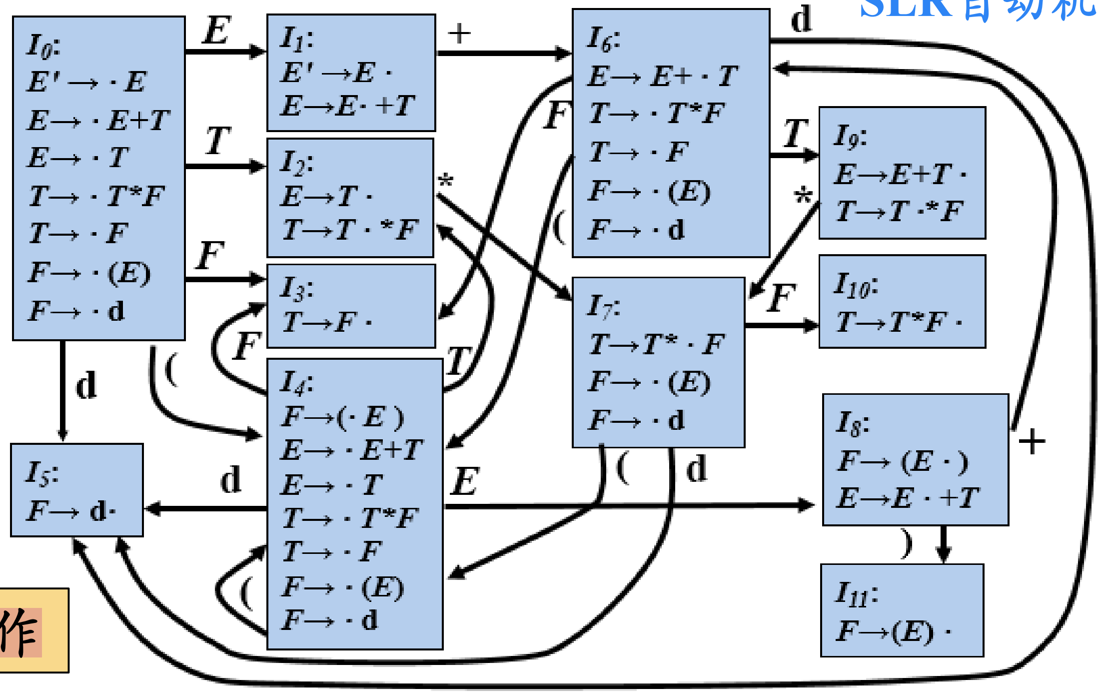
- 如果一个*S-SDD*的基本文法可以使用*LR*分析技术，那么它的*SDT*可以在*LR*语法分析过程中实现
  - 例

S-SDD

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

当归约发生时执行相应的语义动作

SLR自动机



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	$Y.y$
S_m	Z	$Z.z$
…	…	…

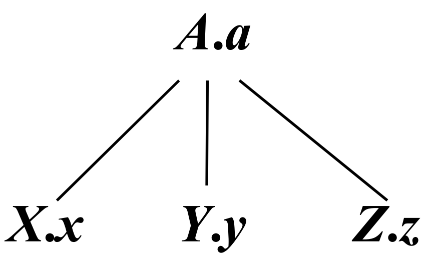
top →

- 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

Symbol	Value
…	…
X	$X.x$
Y	$Y.y$
Z	$Z.z$

$top \rightarrow$



$A \rightarrow XYZ \quad \{\ A.a = f\,(X.x,\ Y.y,\ Z.z)\ \}$

Symbol	Value
…	…
A	$A.a$

$top \rightarrow$

stack[*top*-2].*symb* = *A*

stack[*top*-2].*val* = *f*(*stack*[*top*-2].*val*, *stack*[*top*-1].*val*, *stack*[*top*].*val*)

top =*top*-2;

例：在自底向上语法分析栈中实现桌面计算器

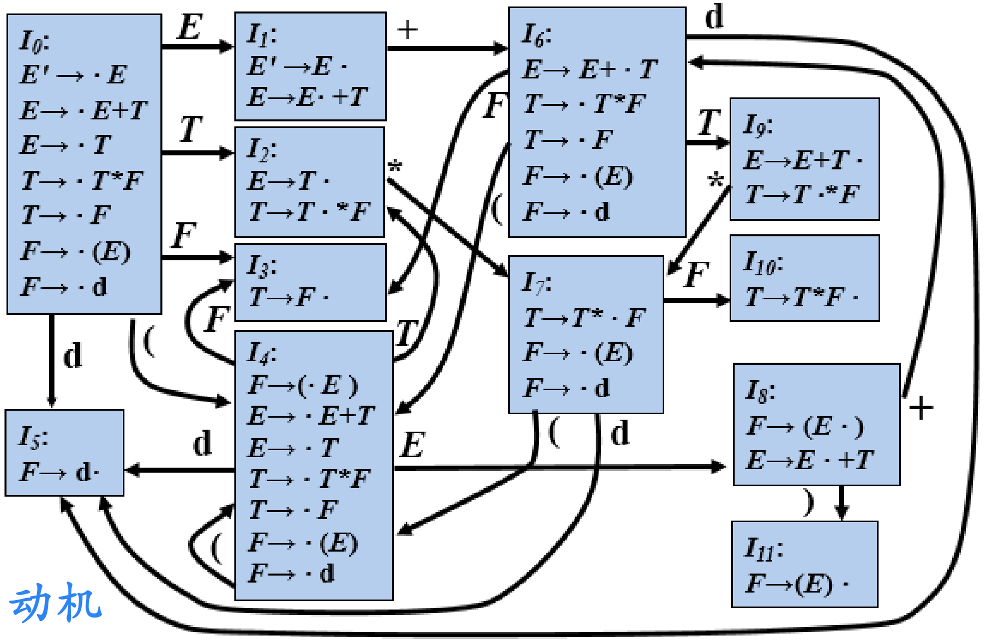
产生式	语义动作	
(1)$E' \to E$	print($E.val$)	{ print ($stack[top].val$);}
(2)$E \to E_1 + T$	$E.val = E_1.val + T.val$	{ $stack[top\text{-}2].val = stack[top\text{-}2].val + stack[top].val$; top=top-2; }
(3)$E \to T$	$E.val = T.val$	
(4)$T \to T_1 * F$	$T.val = T_1.val \times F.val$	{ $stack[top\text{-}2].val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

输入： 3*5+4

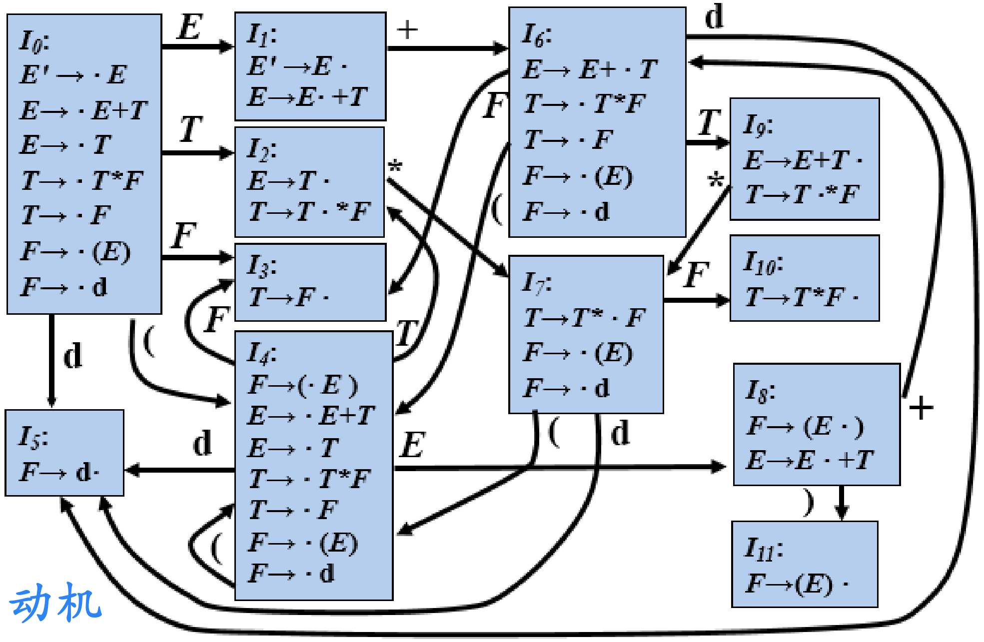
↑↑

状态	符号	属性
0	$	_
5	d	3



SLR自动机

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



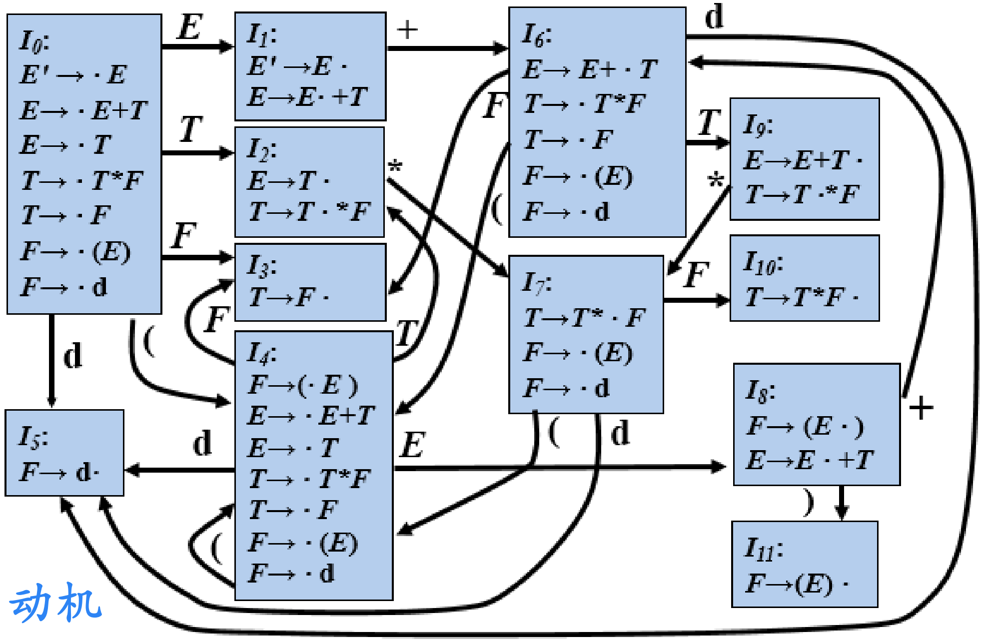
SLR自动机

输入： 3*5+4

↑↑

状态	符号	属性
0	$	_
3	*F*	3

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

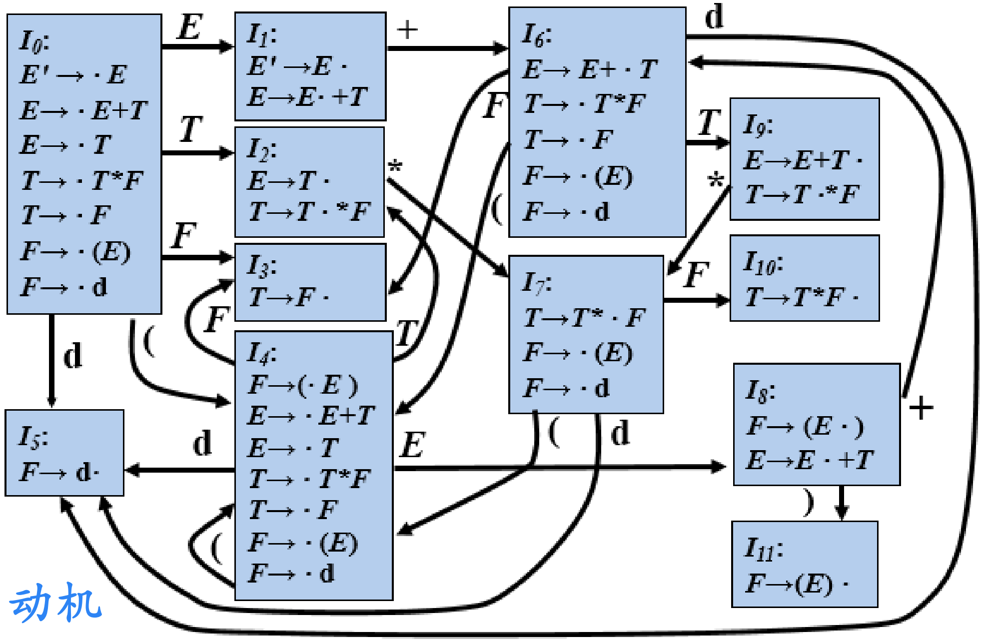


SLR自动机

输入： 3*5+4

状态	符号	属性
0	$	_
2	*T*	3
7	*	_
5	d	5

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



SLR自动机

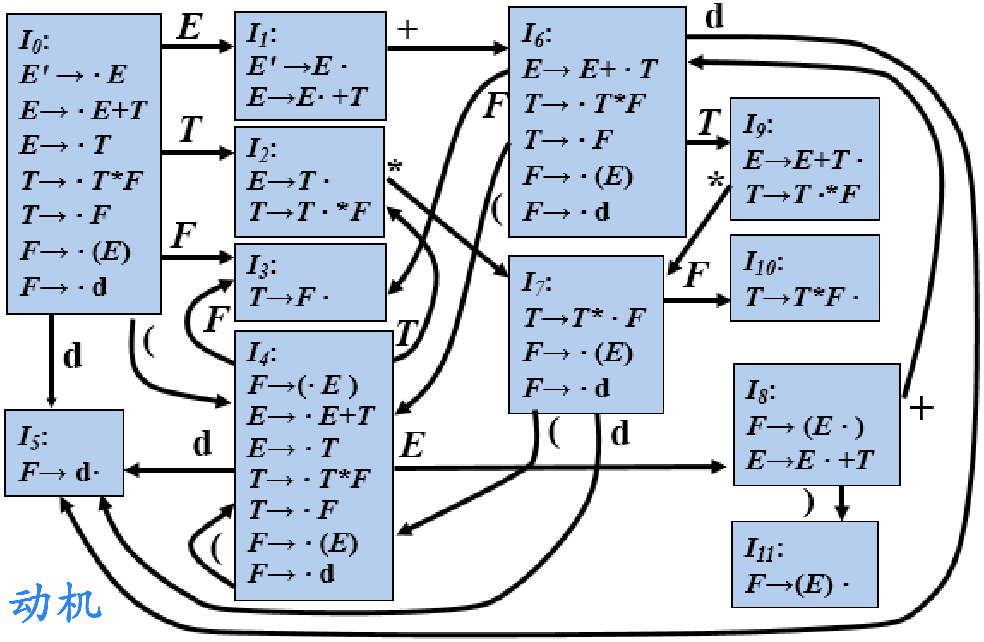
输入： 3*5+4

状态	符号	属性
0	$	_
2	*T*	15
7	*	_
10	*F*	5

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* × *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)$F \rightarrow$ digit	

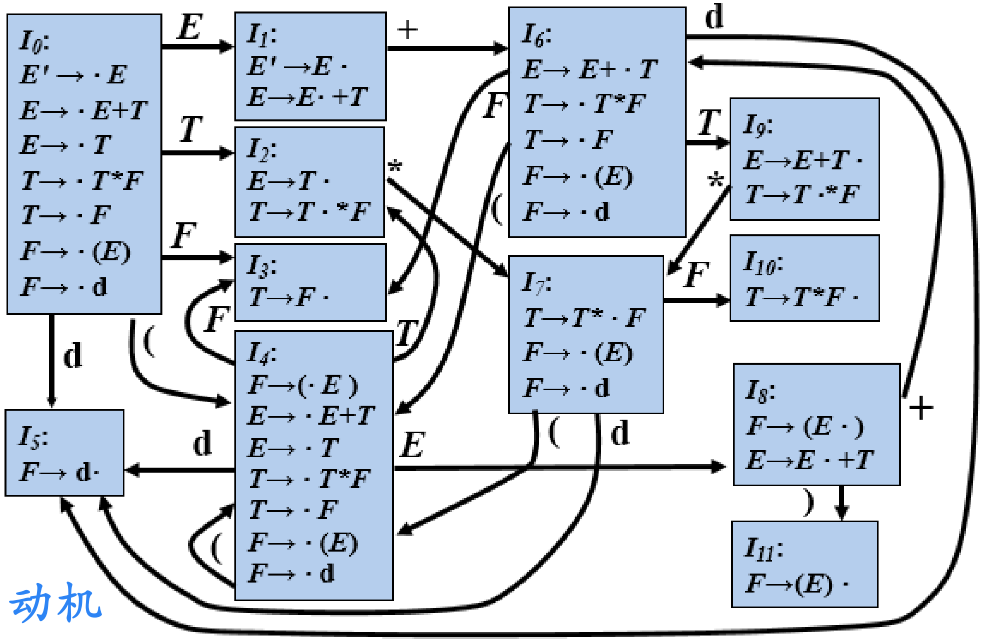
输入： 3*5+4

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* × *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)$F \rightarrow$ digit	

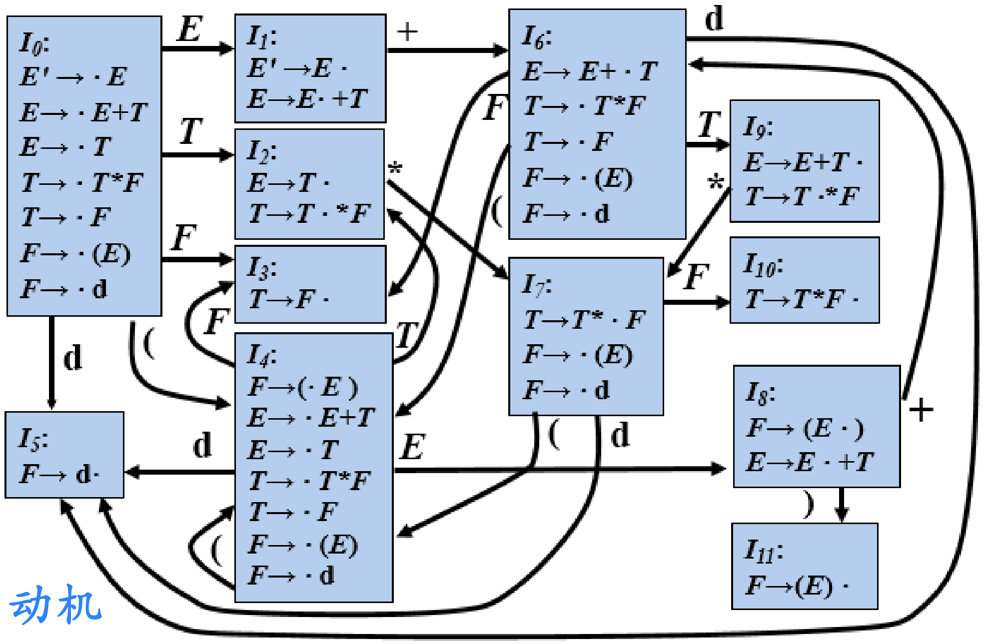


SLR自动机

输入： 3*5+4

状态	符号	属性
0	$	_
1	*E*	15
6	+	_
5	d	4

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* × *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)$F \rightarrow$ digit	

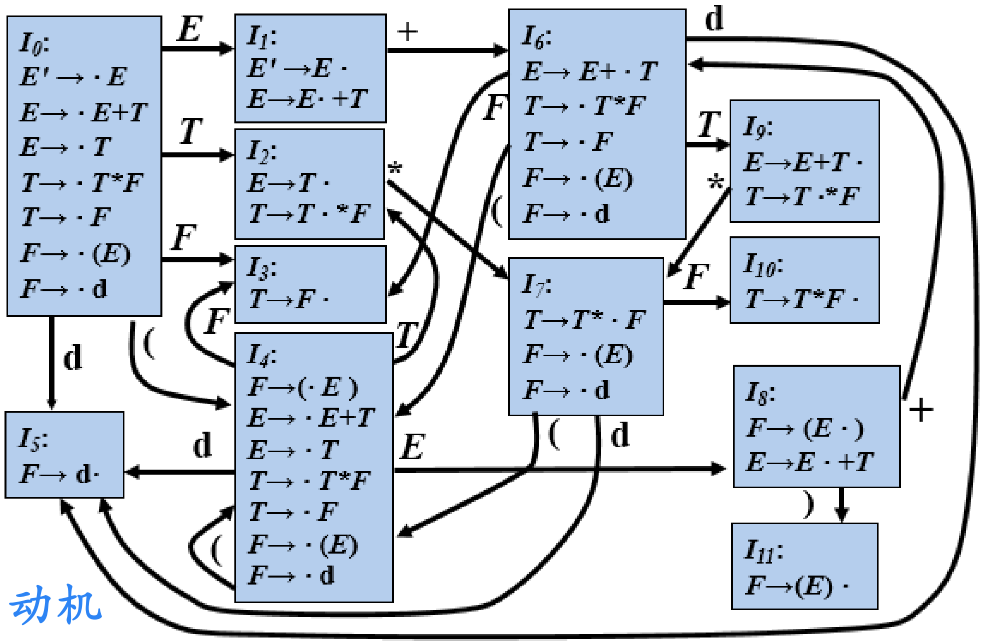


SLR自动机

输入： 3*5+4

状态	符号	属性
0	$	_
1	*E*	15
6	+	_
3	*F*	4

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



SLR自动机

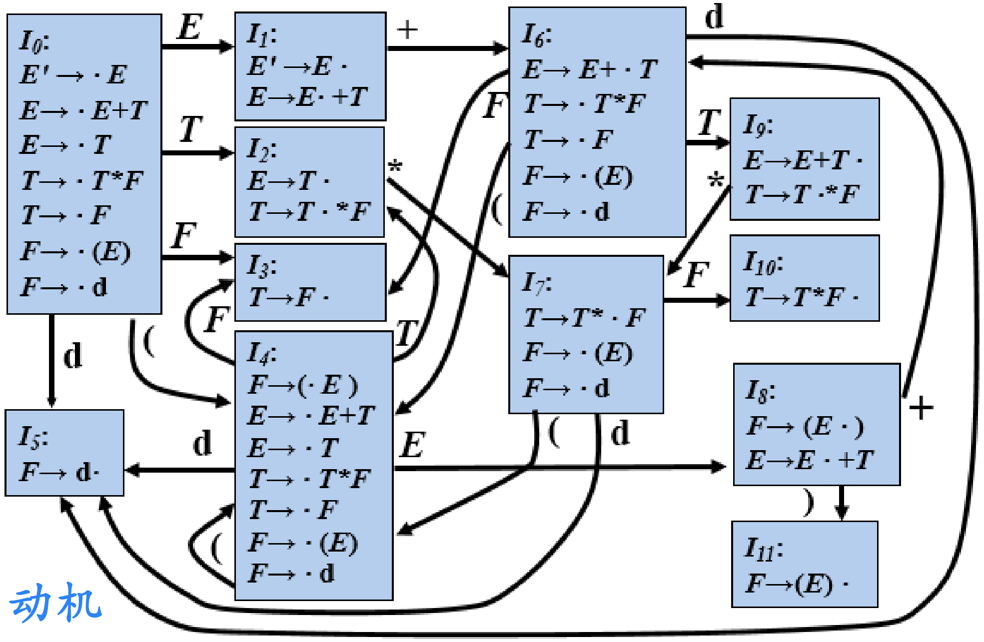
输入： 3*5+4

状态	符号	属性
0	$	_
1	*E*	19
6	+	_
9	*T*	4

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* × *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)$F \rightarrow$ digit	

输入： 3*5+4

状态	符号	属性
0	$	_
1	*E*	19



SLR自动机

将*L-SDD*转换为*SDT*

- 将*L-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

eg

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)	$F \rightarrow$ digit	$F.val =$ 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) $F \rightarrow$ digit $\{\ F.val =$ digit.$lexval\ \}$

L-属性定义的*SDT* 实现

相同左部的select集不相交

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

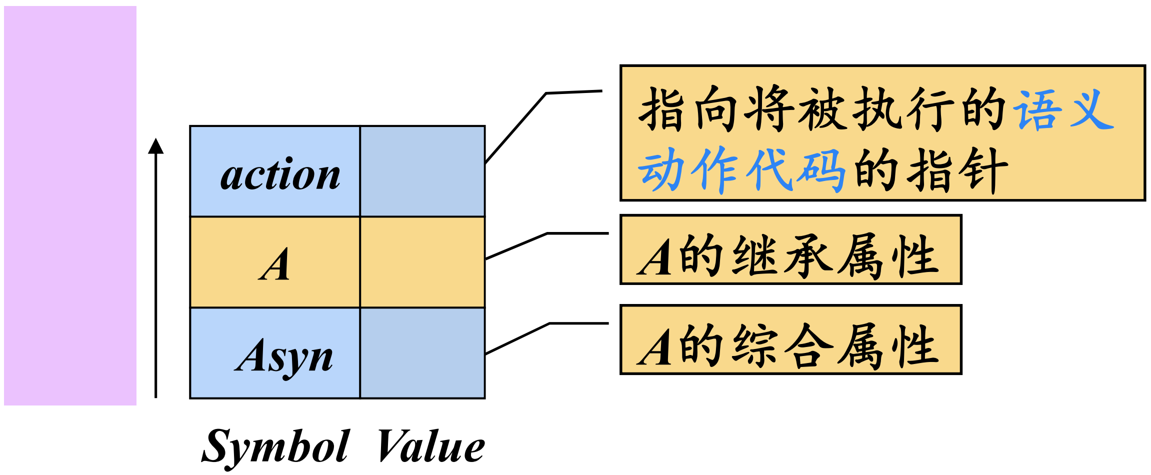
第五章 语法制导翻译

在非递归的预测分析过程中进行翻译

哈尔滨工业大学 陈鄞

在非递归的预测分析过程中进行翻译

➢ 扩展语法分析栈



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) $F \rightarrow$ digit { $F.val$ = digit.$lexval$ }

1) $T \rightarrow F$ { a_1} T' { a_2 }

2) $T' \rightarrow {*}F$ { a_3 } T_1' { a_4 }

3) $T' \rightarrow \varepsilon$ { a_5 }

4) $F \rightarrow$ digit { a_6 }

a_1 : $T'.inh = F.val$

a_2: $T.val = T'.syn$

a_3: $T_1'.inh = T'.inh \times F.val$

a_4 : $T'.syn = T_1'.syn$

a_5 : $T'.syn = T'.inh$

a_6 : $F.val$ = digit.$lexval$

动作符号

例

SDT

1) $T \rightarrow F\ \{a_1\}\ T'\ \{a_2\}$

2) $T' \rightarrow *F\ \{a_3\}\ T_1'\ \{a_4\}$

3) $T' \rightarrow \varepsilon\ \{a_5\}$

4) $F \rightarrow \text{digit}\ \{a_6\}$

a_1: $T'.inh = F.val$

a_2: $T.val = T'.syn$

a_3: $T_1'.inh = T'.inh \times F.val$

a_4: $T'.syn = T_1'.syn$

a_5: $T'.syn = T'.inh$

a_6: $F.val = \text{digit}.lexval$

输入: 3 * 5

T	*Tsyn*	$
	val	

例

SDT	a_1 : $T'.inh = F.val$
1) $T \rightarrow F\ \{a_1\}\ T'\ \{a_2\}$	a_2: $T.val = T'.syn$
2) $T' \rightarrow *F\ \{a_3\}\ T_1'\ \{a_4\}$	a_3: $T_1'.inh = T'.inh \times F.val$
3) $T' \rightarrow \varepsilon\ \{a_5\}$	a_4 : $T'.syn = T_1'.syn$
4) $F \rightarrow \text{digit}\ \{a_6\}$	a_5 : $T'.syn = T'.inh$
	a_6 : $F.val = \text{digit}.lexval$

输入: 3 * 5

每个非终结符后面都会跟一个综合属性
归约的时候非终结符没了，但是综合属性留着

F	*Fsyn*	$\{a_1\}$	*T '*	*T ' syn*	$\{a_2\}$	*Tsyn*	$
	val		*inh*	*syn*		*val*	

例

SDT

1) $T \rightarrow F$ { a_1} T' {a_2 }

2) $T' \rightarrow *F$ {a_3 } T_1' {a_4 }

3) $T' \rightarrow \varepsilon$ {a_5 }

4) $F \rightarrow$ digit {a_6 }

a_1 : $T'.inh = F.val$

a_2: $T.val = T'.syn$

a_3: $T_1'.inh = T'.inh \times F.val$

a_4 : $T'.syn = T_1'.syn$

a_5 : $T'.syn = T'.inh$

a_6 : $F.val = \text{digit}.lexval$

输入: 3 * 5

stack[*top*-1].*val*=*stack*[*top*].*digit_lexval*

digit	{a_6}	*Fsyn*	{a_1}	*T '*	*T ' syn*	{a_2}	*Tsyn*	$
lexval=3	*digit_lexval*=3	*val*=3	*Fval*=3	*inh*	*syn*		*val*	

例

SDT

1) $T \rightarrow F\ \{a_1\}\ T'\ \{a_2\}$

2) $T' \rightarrow *F\ \{a_3\}\ T_1'\ \{a_4\}$

3) $T' \rightarrow \varepsilon\ \{a_5\}$

4) $F \rightarrow \text{digit}\ \{a_6\}$

a_1 : $T'.inh = F.val$

a_2: $T.val = T'.syn$

a_3: $T_1'.inh = T'.inh \times F.val$

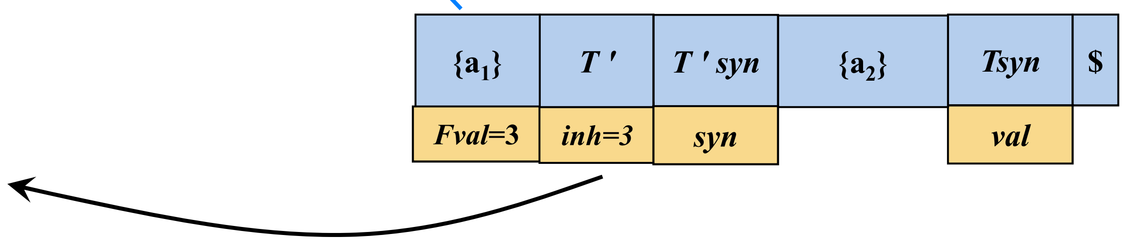
a_4 : $T'.syn = T_1'.syn$

a_5 : $T'.syn = T'.inh$

a_6 : $F.val = \text{digit}.lexval$

输入: 3 * 5

stack[*top*-1].*inh*=*stack*[*top*].*Fval*



例

SDT

1) $T \rightarrow F\ \{a_1\}\ T'\ \{a_2\}$
2) $T' \rightarrow {*}F\ \{a_3\}\ T_1'\ \{a_4\}$
3) $T' \rightarrow \varepsilon\ \{a_5\}$
4) $F \rightarrow \text{digit}\ \{a_6\}$

a_1 : $T'.inh = F.val$
a_2 : $T.val = T'.syn$
a_3 : $T_1'.inh = T'.inh \times F.val$
a_4 : $T'.syn = T_1'.syn$
a_5 : $T'.syn = T'.inh$
a_6 : $F.val = \text{digit}.lexval$

输入: 3 * 5

*	F	$Fsyn$	$\{a_3\}$	T_1'	$T_1'syn$	$\{a_4\}$	$T'syn$	$\{a_2\}$	$Tsyn$	$
		val	$T'inh$=3	inh	syn		syn		val	

例

SDT

1) $T \rightarrow F$ $\{a_1\}$ T' $\{a_2\}$

2) $T' \rightarrow *F$ $\{a_3\}$ T_1' $\{a_4\}$

3) $T' \rightarrow \varepsilon$ $\{a_5\}$

4) $F \rightarrow \text{digit}$ $\{a_6\}$

a_1 : $T'.inh = F.val$

a_2 : $T.val = T'.syn$

a_3 : $T_1'.inh = T'.inh \times F.val$

a_4 : $T'.syn = T_1'.syn$

a_5 : $T'.syn = T'.inh$

a_6 : $F.val = \text{digit}.lexval$

输入: 3 * 5

`stack[top-1].val=stack[top].digit_lexval`

digit	$\{a_6\}$	*Fsyn*	$\{a_3\}$	T_1'	$T_1'syn$	$\{a_4\}$	*T ' syn*	$\{a_2\}$	*Tsyn*	\$
lexval=5	*digit_lexval*=5	*val*=5	*T ' inh*=3 *Fval*=5	*inh*	*syn*		*syn*		*val*	

例

SDT

1) $T \rightarrow F\ \{a_1\}\ T'\ \{a_2\}$

2) $T' \rightarrow *F\ \{a_3\}\ T_1'\ \{a_4\}$

3) $T' \rightarrow \varepsilon\ \{a_5\}$

4) $F \rightarrow \text{digit}\ \{a_6\}$

a_1 : $T'.inh = F.val$

a_2 : $T.val = T'.syn$

a_3 : $T_1'.inh = T'.inh \times F.val$

a_4 : $T'.syn = T_1'.syn$

a_5 : $T'.syn = T'.inh$

a_6 : $F.val = \text{digit}.lexval$

输入: 3 * 5

stack[*top*-1].*inh*=*stack*[*top*].*T'inh* × *stack*[*top*].*Fval*

$\{a_3\}$	T_1'	$T_1'syn$	$\{a_4\}$	$T'\ syn$	$\{a_2\}$	$Tsyn$	$
$T'\ inh=3$	$inh=15$	syn		syn		val	
$Fval=5$							

例

SDT

1) $T \rightarrow F\ \{a_1\}\ T'\ \{a_2\}$

2) $T' \rightarrow *F\ \{a_3\}\ T_1'\ \{a_4\}$

3) $T' \rightarrow \varepsilon\ \{a_5\}$

4) $F \rightarrow \text{digit}\ \{a_6\}$

a_1 : $T'.inh = F.val$

a_2: $T.val = T'.syn$

a_3: $T_1'.inh = T'.inh \times F.val$

a_4 : $T'.syn = T_1'.syn$

a_5 : $T'.syn = T'.inh$

a_6 : $F.val = \text{digit}.lexval$

输入: 3 * 5

$stack[top\text{-}1].syn = stack[top].T_1'inh$

$\{a_5\}$	$T_1'syn$	$\{a_4\}$	$T'\ syn$	$\{a_2\}$	$Tsyn$	$
$T_1'\ inh$=15	syn=15	$T_1'syn$=15	syn		val	

例

SDT

1) $T \rightarrow F$ { a_1} T' {a_2 }

2) $T' \rightarrow *F$ {a_3 } T_1' {a_4 }

3) $T' \rightarrow \varepsilon$ {a_5 }

4) $F \rightarrow$ digit {a_6 }

a_1 : $T'.inh = F.val$

a_2: $T.val = T'.syn$

a_3: $T_1'.inh = T'.inh \times F.val$

a_4 : $T'.syn = T_1'.syn$

a_5 : $T'.syn = T'.inh$

a_6 : $F.val$ = digit.*lexval*

输入: 3 * 5

stack[*top*-1].*syn*=*stack*[*top*].T_1'*syn*

{a_4}	*T ' syn*	{a_2}	*Tsyn*	$
T_1'*syn*=15	*syn*=15	*T' syn*=15	*val*	

例

SDT

1) $T \rightarrow F$ { a_1} T' {a_2 }

2) $T' \rightarrow *F$ {a_3 } T_1' {a_4 }

3) $T' \rightarrow \varepsilon$ {a_5 }

4) $F \rightarrow$ digit {a_6 }

a_1 : $T'.inh = F.val$

a_2: $T.val = T'.syn$

a_3: $T_1'.inh = T'.inh \times F.val$

a_4 : $T'.syn = T_1'.syn$

a_5 : $T'.syn = T'.inh$

a_6 : $F.val =$ digit.*lexval*

输入: 3 * 5

stack[*top*-1].*val*=*stack*[*top*].*T'syn*

{a_2}	*Tsyn*	$
T' syn=15	*val*=15	

分析栈中的每一个记录都对应着一段执行代码

- 综合记录出栈时，要将综合属性值复制给后面特定的语义动作
- 变量展开时（即变量本身的记录出栈时），如果其含有继承属性，则要将继承属性值复制给后面特定的语义动作

~~puzzled~~

例

1) $T \rightarrow F\ \{a_1\}\ T'\ \{a_2\}$
2) $T' \rightarrow *F\ \{a_3\}\ T_1'\ \{a_4\}$
3) $T' \rightarrow \varepsilon\ \{a_5\}$
4) $F \rightarrow \text{digit}\ \{a_6\}$

a_1 : $T'.inh = F.val$
a_2: $T.val = T'.syn$
a_3: $T_1'.inh = T'.inh \times F.val$
a_4 : $T'.syn = T_1'.syn$
a_5 : $T'.syn = T'.inh$
a_6 : $F.val = \text{digit}.lexval$

1) $T \rightarrow F\ \{a_1{:}T'.inh=F.val\}\ T'\ \{a_2{:}T.val=T'.syn\}$

符号	属性	执行代码
F		
Fsyn	*val*	*stack*[*top*-1].*Fval* = *stack*[*top*].*val*； *top*=*top*-1;
a_1	*Fval*	*stack*[*top*-1].*inh* = *stack*[*top*].*Fval*； *top*=*top*-1;
T'	*inh*	根据当前输入符号选择产生式进行推导 若选 2): *stack*[*top*+3].*T'inh* =*stack*[*top*].*inh*; *top*=*top*+6; 若选 3): *stack*[*top*].*T'inh* =*stack*[*top*].*inh*;
T'syn	*syn*	*stack*[*top*-1].*T'syn* = *stack*[*top*].*syn*； *top*=*top*-1;
a_2	*T'syn*	*stack*[*top*-1].*val* = *stack*[*top*].*T'syn*； *top*=*top*-1;

例

1) $T \to F\ \{a_1\}\ T'\ \{a_2\}$
2) $T' \to *F\ \{a_3\}\ T_1'\ \{a_4\}$
3) $T' \to \varepsilon\ \{a_5\}$
4) $F \to \text{digit}\ \{a_6\}$

a_1: $T'.inh = F.val$
a_2: $T.val = T'.syn$
a_3: $T_1'.inh = T'.inh \times F.val$
a_4: $T'.syn = T_1'.syn$
a_5: $T'.syn = T'.inh$
a_6: $F.val = \text{digit}.lexval$

2) $T' \to *F\{a_3: T_1'.inh = T'.inh \times F.val\}T_1'\{a_4: T'.syn = T_1'.syn\}$

符号	属性	执行代码
*		
F		
Fsyn	*val*	*stack*[*top*-1].*Fval* = *stack*[*top*].*val*； *top*=*top*-1;
a_3	*T′inh; Fval*	*stack*[*top*-1].*inh* = *stack*[*top*].*T′inh*× *stack*[*top*].*Fval*； *top*=*top*-1;
T_1'	*inh*	根据当前输入符号选择产生式进行推导 若选2): *stack*[*top*+3].*T′inh* = *stack*[*top*].*inh*； *top*=*top*+6; 若选3): *stack*[*top*].*T′inh* = *stack*[*top*].*inh*;
$T_1'syn$	*syn*	*stack*[*top*-1].$T_1'syn$ = *stack*[*top*].*syn*； *top*=*top*-1;
a_4	$T_1'syn$	*stack*[*top*-1].*syn* = *stack*[*top*].$T_1'syn$； *top*=*top*-1;

例

1) $T \to F\ \{a_1\}\ T'\ \{a_2\}$
2) $T' \to *F\ \{a_3\}\ T_1'\ \{a_4\}$
3) $T' \to \varepsilon\ \{a_5\}$
4) $F \to \text{digit}\ \{a_6\}$

a_1: $T'.inh = F.val$
a_2: $T.val = T'.syn$
a_3: $T_1'.inh = T'.inh \times F.val$
a_4: $T'.syn = T_1'.syn$
a_5: $T'.syn = T'.inh$
a_6: $F.val = \text{digit}.lexval$

3) $T' \to \varepsilon\ \{a_5{:}T'.syn = T'.inh\}$

符号	属性	执行代码
a_5	$T'inh$	$stack[top-1].syn = stack[top].T'inh$; $top=top-1$;

例

1) $T \rightarrow F$ { a_1} T' {a_2 }
2) $T' \rightarrow *F$ {a_3 } T_1' {a_4 }
3) $T' \rightarrow \varepsilon$ {a_5 }
4) $F \rightarrow$ digit {a_6 }

a_1 : $T'.inh = F.val$
a_2: $T.val = T'.syn$
a_3: $T_1'.inh = T'.inh \times F.val$
a_4 : $T'.syn = T_1'.syn$
a_5 : $T'.syn = T'.inh$
a_6 : $F.val$ = digit.*lexval*

4) $F \rightarrow$ digit {a_6:$F.val = digit.lexval$}

符号	属性	执行代码
digit	*lexval*	*stack*[*top*-1].*digitlexval* = *stack*[*top*].*lexval*; *top*=*top*-1;
a_6	*digitlexval*	*stack*[*top*-1].*val* = *stack*[*top*].*digitlexval*; *top*=*top*-1;

第五章 语法制导翻译

在非递归的预测分析过程中进行翻译

哈尔滨工业大学 陈鄞