



# 编译原理

## 第五章 语法分析——自下而上分析

# 第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

# 第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

# 语法分析的方法

## ■ 自下而上分析法 (Bottom-up)

### □ 基本思想

- 从输入串开始，逐步进行归约，直到文法的开始符号
- 从树末端开始，构造语法树
- 核心问题：确定可归约串

### □ 算符优先分析法

- 按照算符的优先关系和结合性质进行语法分析
- 适合分析表达式

### □ LR 分析法

- 规范归约：句柄作为可归约串

## 5.3 LR 分析法

- 计算思维的典型方法
  - 知识与控制的分离
  - 自动化

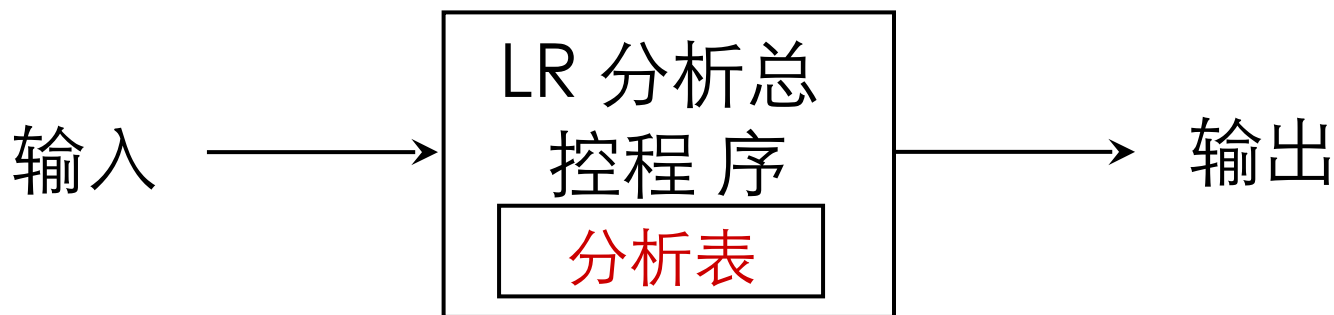
- LR 分析法：1965 年由 Knuth 提出



For his major contributions to the analysis of algorithms and the design of programming languages, and in particular for his contributions to "the art of computer programming" through his well-known books in a continuous series by this title.

□ LR 分析器工作

Donald Ervin Knuth



# 主要介绍

1. 总控程序 (LR 分析器) 的处理思想
2. LR 分析表的构造方法及原理

# 短语、直接短语和句柄

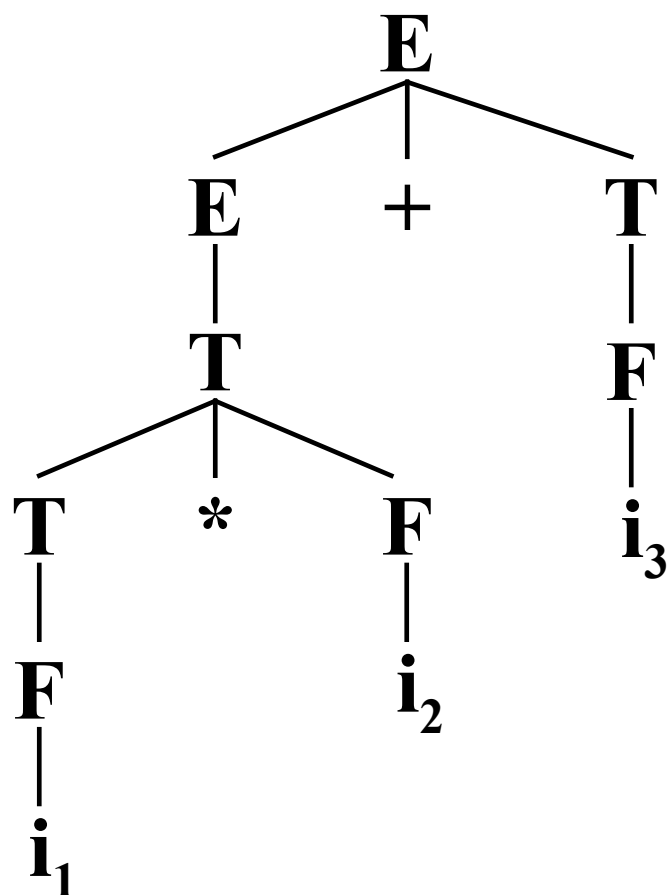
- 定义：令  $G$  是一个文法， $S$  是文法的开始符号，假定  $\alpha\beta\delta$  是文法  $G$  的一个句型，如果有
$$S \xRightarrow{*} \alpha A \delta \quad A \xRightarrow{+} \beta$$

则  $\beta$  称是句型  $\alpha\beta\delta$  相对于非终结符  $A$  的**短语**。

特别是，如果有  $A \Rightarrow \beta$ ，则称  $\beta$  是句型  $\alpha\beta\delta$  相对于规则  $A \rightarrow \beta$  的**直接短语**。

一个句型的最左直接短语称为该句型的**句柄**。

# 短语、直接短语和句柄



■ 在一个句型对应的语法树中

- 以某非终结符为根的两代以上的子树的所有末端结点从左到右排列就是相对于该非终结符的一个短语
- 如果子树只有两代，则该短语就是直接短语



# 规范归约

- 定义：假定 $\alpha$ 是文法  $G$  的一个句子，我们称序列

$$\alpha_n, \alpha_{n-1}, \dots, \alpha_0$$

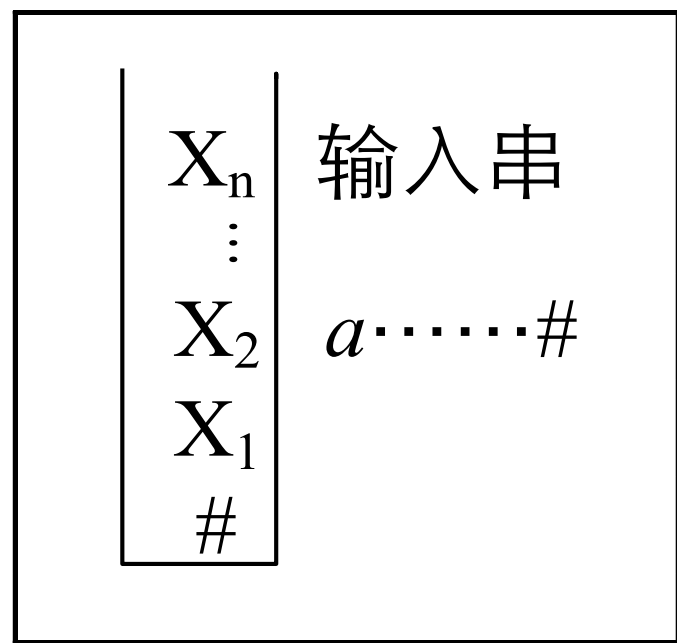
是 $\alpha$ 的一个规范归约，如果此序列满足：

1.  $\alpha_n = \alpha$
2.  $\alpha_0$  为文法的开始符号，即 $\alpha_0 = S$
3. 对任何  $i$ ， $0 \leq i \leq n$ ， $\alpha_{i-1}$  是从 $\alpha_i$  经把句柄替换成为相应产生式左部符号而得到的

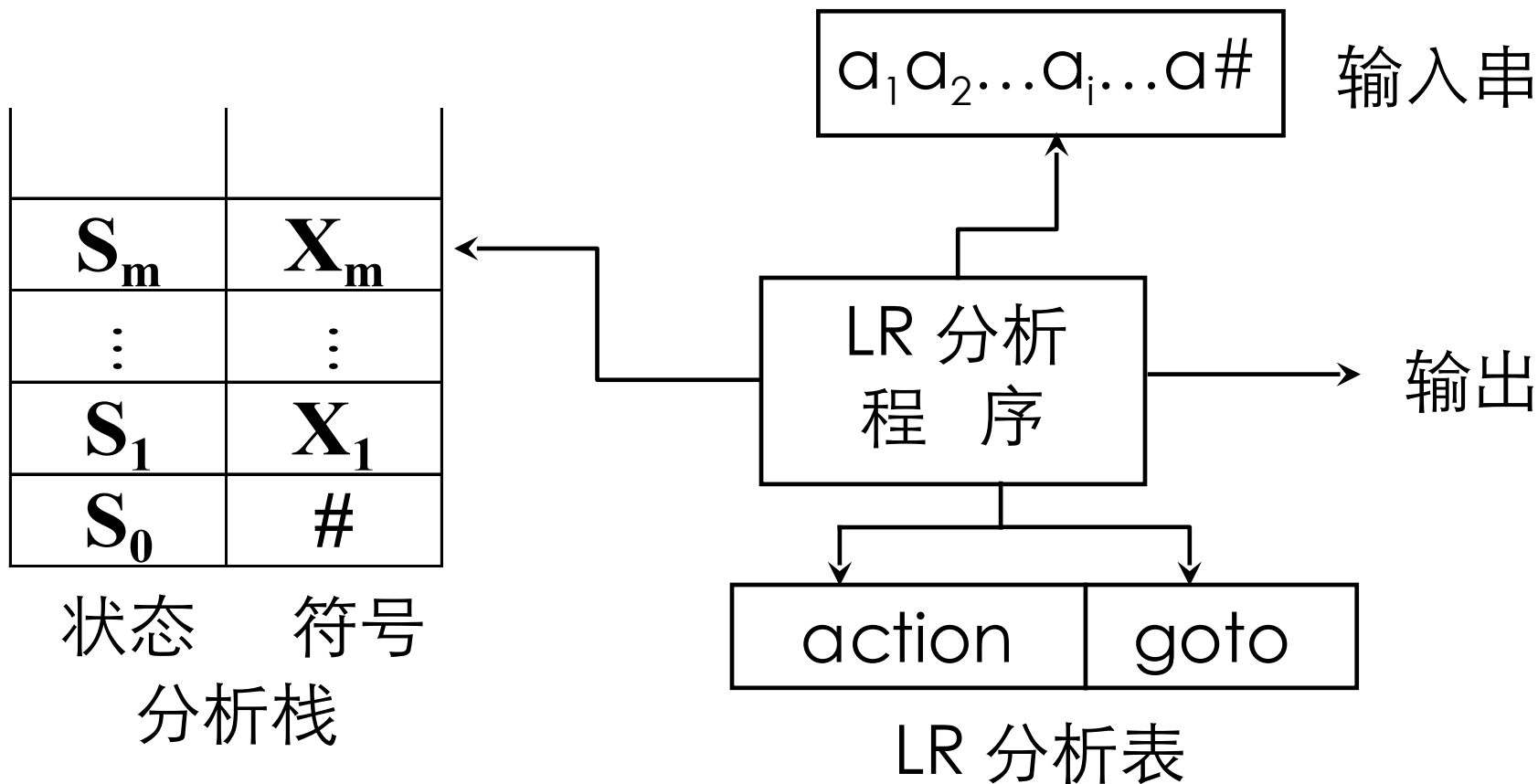
## 5.3.1 LR 分析器

■ **规范归约**的关键问题是寻找句柄。

- **历史**：已移入符号栈的内容
- **展望**：根据产生式推测未来可能遇到的输入符号
- **现实**：当前的输入符号



- LR 分析方法：把“**LR 分析**”抽象成**状态**；由栈顶的**状态**和**输入符号**唯一确定每一步工作
- 计算思维的典型方法
  - 知识与控制的分离
  - 自动化



# LR 分析器

## ■ LR 分析器的核心是一张分析表

- ACTION[s , a] : 当状态 s 面临输入符号 a 时，应采取什么动作。
- GOTO[s , X] : 状态 s 面对文法符号 X 时，下一状态是什么

状态	ACTION						GOTO		
	i	+	*	(	)	#	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			12
6	s5			s4				0	2

# LR 分析器

**移进 (shift)** 把  $(s, a)$  的下一状态  $s'$  和输入符号  $a$  推进栈，下一输入符号变成现行输入符号

状态	ACTION						GOTO		
	i	+	*	(	)	#	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			

# LR 分析器

**归约 (reduce)**：指用某产生式  $A \rightarrow \beta$  进行归约。假若  $\beta$  的长度为  $r$ ，归约动作是，去除栈顶  $r$  个项，使状态  $s_{m-r}$  变成栈顶状态，然后把  $(s_{m-r}, A)$  的下一状态  $s' = \text{GOTO}[s_{m-r}, A]$  和文法符号  $A$  推进栈

状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5						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			

# LR 分析器

**接受** 宣布分析成功，停止分析器工作

状态	ACTION						GOTO		
	i	+	*	(	)	#	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			

# LR 分析器

报错

状态	ACTION						GOTO		
	i	+	*	(	)	#	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			



# LR 分析器

- 分析开始时：

状态	已归约串	输入串
$(s_0 ,$	$\# ,$	$a_1 a_2 \dots a_n \#)$

- 以后每步的结果可以表示为：

$$(s_0 s_1 \dots s_m , \# X_1 \dots X_m , a_i a_{i+1} \dots a_n \#)$$

$(s_0 s_1 \dots s_m, \# X_1 \dots X_m, a_i a_{i+1} \dots a_n \#)$

$(s_0 s_1 \dots s_{m-r} s_{m-r+1} \dots s_m, \# X_1 \dots X_{m-r} X_{m-r+1} \dots X_m, a_i a_{i+1} \dots a_n \#)$

$(s_0 s_1 \dots s_{m-r}, \# X_1 \dots X_{m-r}, a_i a_{i+1} \dots a_n \#)$

$(s_0 s_1 \dots s_{m-r}, \# X_1 \dots X_{m-r} A, a_i a_{i+1} \dots a_n \#)$

$(s_0 s_1 \dots s_{m-r} s, \# X_1 \dots X_{m-r} A, a_i a_{i+1} \dots a_n \#)$

此处,  $s = \text{GOTO}(s_{m-r}, A)$ ,  $r$  为  $\beta$  的长度,  $\beta = X_{m-r+1} \dots X_m$

3. 若  $\text{ACTION}(s_m, a_i)$  为 "接受", 则三元式不再变化, 变化过程终止, 宣布分析成功.

4. 若  $\text{ACTION}(s_m, a_i)$  为 "报错", 则三元式变化过程终止, 报告错误.

# LR 分析器示例

文法  $G(E)$  :

(1)  $E \rightarrow E + T$

(2)  $E \rightarrow T$

(3)  $T \rightarrow T * F$

(4)  $T \rightarrow F$

(5)  $F \rightarrow (E)$

(6)  $F \rightarrow i$

其 LR 分析表为：

状态	ACTION						GOTO		
	i	+	*	(	)	#	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			

步骤	状态	符号	输入串
(1)	0	#	$i*i+i\#$
(2)	05	#i	$*i+i\#$
(3)	03	#F	$*i+i\#$
(4)	02	#T	$*i+i\#$
(5)	027	#T*	$i+i\#$

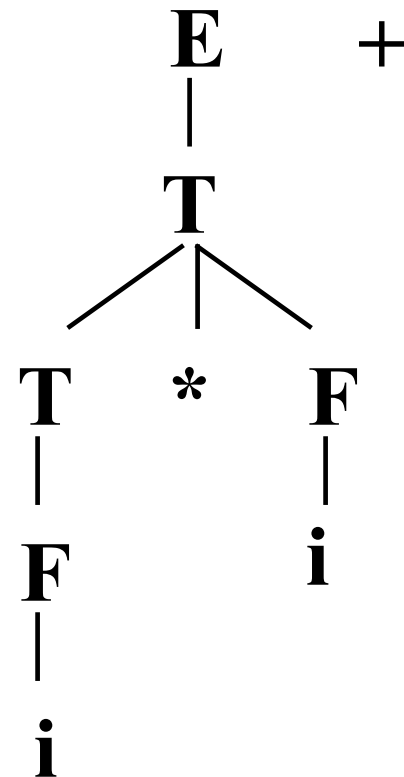
状态	ACTION						GOTO		
	i	+	*	(	)	#	E	T	F
0	s5			s4			1	2	3
1		s6				acc			
2		r2	s7		r2	r2			
3		r4	r4		r4	r4			
4	s5			s4			8	2	3
5		r6	r6		r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				

T  
|  
F  
|  
i

\*

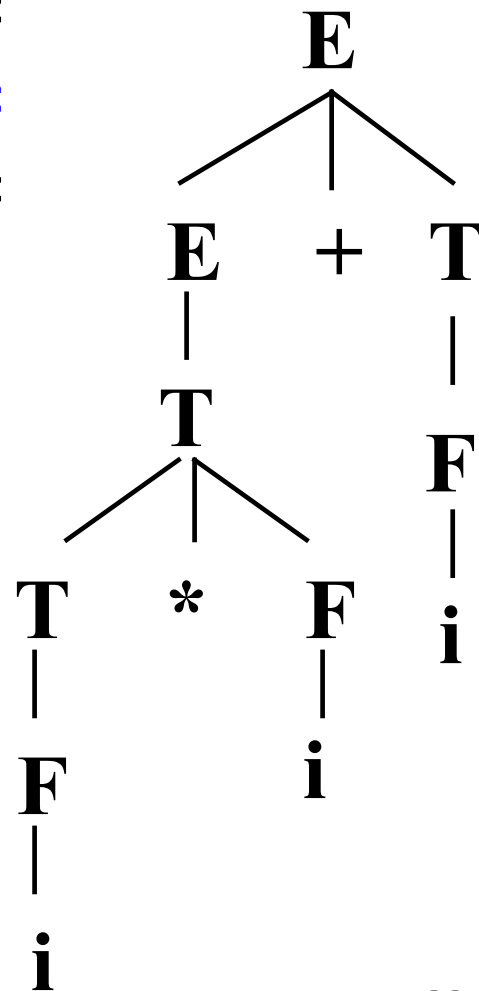
步骤	状态	符号	输入串
(5)	027	#T*	i+i#
(6)	0275	#T*i	+i#
(7)	027 <u>10</u>	#T*F	+i#
(8)	02	#T	+i#
(9)	01	#E	+i#
(10)	016	#E+	i#

状态	ACTION						GOTO		
	i	+	*	(	)	#	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			



步骤	状态	符号	输入串
(10)	016	#E+	i#
(11)	0165	#E+i	#
(12)	0163	#E+F	#
(13)	0169	#E+T	#
(14)	01	#E	#
(15)	接受		

状态	ACTION						GOTO		
	i	+	*	(	)	#	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			



# LR 文法

- 定义：对于一个文法，如果能够构造一张分析表，使得它的每个入口均是唯一确定的，则这个文法就称为 **LR 文法**。
- 定义：一个文法，如果能用一个每步顶多向前检查  $k$  个输入符号的 **LR 分析器** 进行分析，则这个文法就称为 **LR( $k$ ) 文法**。



# LR 文法与二义文法

- LR 文法不是二义的，二义文法肯定不会是 LR 的

- LR 文法  $\subset$  无二义文法

- 非 LR 结构

$$S \rightarrow iCtS \mid iCtSeS$$

栈

#...iCtS

输入

e...#

# 小结

- LR 分析器的工作原理
- LR 分析器的性质
  - 栈内的符号串和扫描剩下的输入符号串构成了一个规范句型
  - 一旦栈的顶部出现可归约串（句柄），则进行归约