编译原理

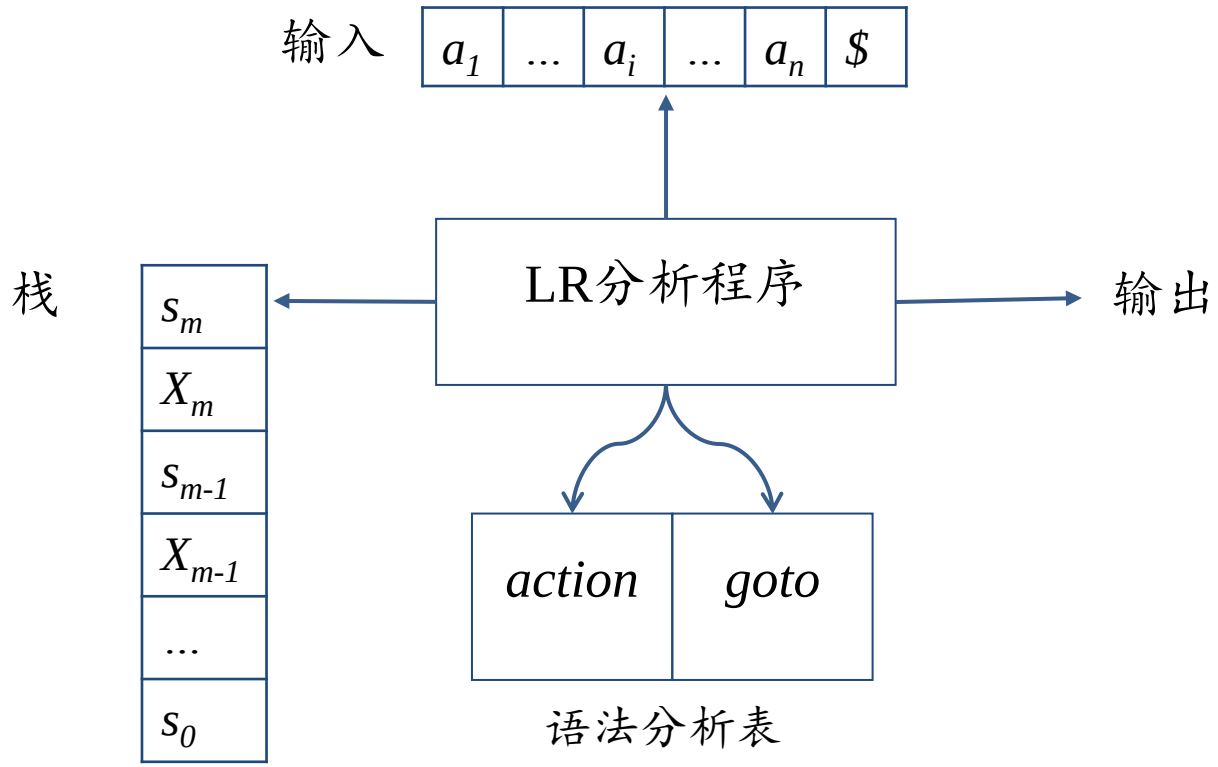
LR语法分析

计算机科学与技术学院 王中卿

目录

- LR语法分析器模型
- LR语法分析算法
- LR分析算法的特点
- LR分析方法和LL分析方法的比较

LR语法分析器模型



LR语法分析算法

- 输入：一个输入串w和一个LR语法分析表。
- 输出：如果w在$L(G)$中，输出w的自底向上语法分析过程中的归约步骤；否则给出错误提示。
- 方法：最初，语法分析器栈中的内容为初试状态S_0，输入缓冲区的内容为$w\,\$$。然后，执行语法分析程序。

LR语法分析算法实例

- 例，对于下列文法

(1) $E \rightarrow E + T$　　(2) $E \rightarrow T$

(3) $T \rightarrow T * F$　　(4) $T \rightarrow F$

(5) $F \rightarrow (E)$　　(6) $F \rightarrow$ **id**

语法分析表中ACTION的编码方法如下：

s_i：表示移入并将状态i压栈。

r_j：表示按照编号为j的产生式进行归约。

acc：表示接受。

空白：表示出错。

LR语法分析算法实例

状态	ACTION						GOTO		
	id	**+**	*****	**(**	**)**	**$**	*E*	*T*	*F*
0	*s*5			*s*4			1	2	3
1		*s*6				*acc*			
2		*r*2	*s*7		*r*2	*r*2			
3		*r*4	*r*4		*r*4	*r*4			
4	*s*5			*s*4			8	2	3
5		*r6*	*r6*		*r6*	*r6*			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9		r1	r7		r1	r1			
10		r3	r3		r3	r3			
11		r5	r5		r5	r5			

LR语法分析器处理*id*id+id*各个步骤

栈	输　入	动　作
0	id ∗ id + id $	移进
0 id 5	∗ id + id $	按$F \rightarrow$ id归约
0 F 3	∗ id + id $	按$T \rightarrow F$归约
0 T 2	∗ id + id $	移进
0 T 2 ∗ 7	id + id $	移进
0 T 2 ∗ 7 id 5	+ id $	按$F \rightarrow$ id归约
0 T 2 ∗ 7 F 10	+ id $	按$T \rightarrow T*F$归约
. . .	. . .	. . .
0 E 1	$	接受

LR语法分析特点

- 概念

  可行前缀：右句型的前缀，该前缀不超过最右句柄的右端

  $S \Rightarrow^*_{rm} \gamma A w \Rightarrow_{rm} \gamma \beta w$

  $\gamma\beta$的任何前缀（包括ε和$\gamma\beta$本身）都是可行前缀

  例，假设

  $$E \Rightarrow^*_{rm} F * \mathbf{id} \Rightarrow_{rm} (E) * \mathbf{id}$$

  可行前缀可以是(，(E，(E)，但不会是(E)*，因为(E)是句柄($F \rightarrow (E)$)，语法分析器必须在移入*之前将(E)归约成F。

LR语法分析特点

- 概念

  可行前缀：右句型的前缀，该前缀不超过最右句柄的右端

- 定义

  LR文法：我们能为之构造出所有条目都唯一的LR分析表。

  直观上说，只要存在这样一个从左到右扫描的移入-归约语法分析器，它总是能够在某文法的最右句型的句柄出现在栈顶时识别出这个句柄，那么这个文法就是LR的。

LR语法分析特点

- 栈中的文法符号总是形成一个可行前缀

栈	输　入	动　作
0	id * id + id $	移进
0 **id** 5	* id + id $	按$F \rightarrow$ id归约
0 F 3	* id + id $	按$T \rightarrow F$归约
0 T 2	* id + id $	移进
0 T 2 * 7	id + id $	移进
0 T 2 * 7 **id** 5	+ id $	按$F \rightarrow$ id归约
0 T 2 * 7 F 10	+ id $	按$T \rightarrow T*F$归约
. . .	. . .	. . .
0 E 1	$	接受

LR语法分析特点

- 栈中的文法符号总是形成一个可行前缀
- 分析表的转移函数本质上是识别可行前缀的DFA

LR语法分析特点

- 例，对于下列文法

(1) $E \to E + T$　　(2) $E \to T$

(3) $T \to T * F$　　(4) $T \to F$

(5) $F \to (E)$　　(6) $F \to \mathbf{id}$

下表蓝色部分构成识别可行前缀DFA的状态转换表

状态	动　作						转　移		
	id	+	*	(	)	**$**	*E*	*T*	*F*
0	*s*5			*s*4			1	2	3
1		*s*6				*acc*			
2		*r*2	*s*7		*r*2	*r*2			
3		*r*4	*r*4		*r*4	*r*4			
4	*s*5			*s*4			8	2	3

LR语法分析特点

- 栈中的文法符号总是形成一个可行前缀
- 分析表的转移函数本质上是识别可行前缀的DFA
- 栈顶的状态符号包含了确定句柄所需要的一切信息

LR语法分析特点

栈	输　入	动　作
0	id * id + id $	移进
0 **id** 5	* id + id $	按$F \rightarrow$ id归约
0 F 3	* id + id $	按$T \rightarrow F$归约
0 T 2	* id + id $	移进
0 T 2 * 7	id + id $	移进
0 T 2 * 7 **id** 5	+ id $	按$F \rightarrow$ id归约
0 T 2 * 7 F 10	+ id $	按$T \rightarrow T*F$归约
. . .	. . .	. . .
0 E 1	$	接受

LR语法分析特点

- 栈中的文法符号总是形成一个可行前缀
- 分析表的转移函数本质上是识别可行前缀的DFA
- 栈顶的状态符号包含了确定句柄所需要的一切信息
- 是已知的最一般的无回溯的移进–归约方法
- 能分析的文法类是预测分析法能分析的文法类的真超集
- 能及时发现语法错误
- 手工构造分析表的工作量太大

LR语法分析特点

- LR文法　*vs*　LL文法
  - *LR(K)*文法：向前看k个输入符号能够知道一个产生式的右部所能推导出的所有符号串，进而识别出这个产生式右部的出现。
  - *LL(K)*文法：看到了产生式右部推出的前k个符号后能够识别出用于归约的产生式。
  - *LR*文法比*LL*文法描述的语言更多。

LR分析方法和LL分析方法的比较

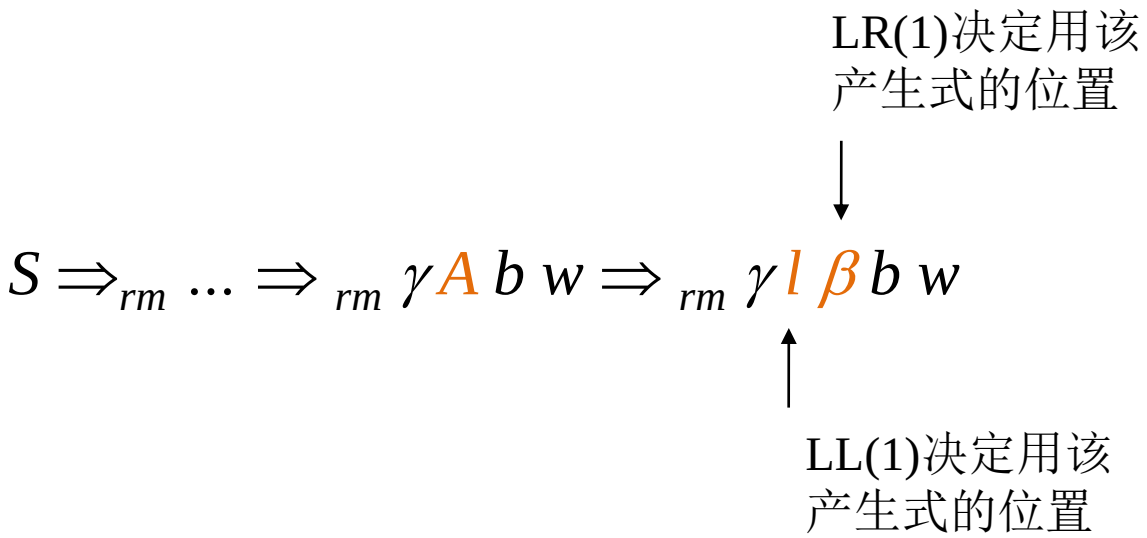
	LR(1)方 法	LL(1)方 法
建立分析树的方式	自 下 而 上	自 上 而 下

LR分析方法和LL分析方法的比较

	LR(1)方 法	LL(1)方 法
建立分析树的方式	自 下 而 上	自 上 而 下
归约还是推导	规 范 归 约	最 左 推 导

LR分析方法和LL分析方法的比较

- 在下面的推导中，最后一步用的是$A \rightarrow l\beta$



LR分析方法和LL分析方法的比较

	LR(1)方 法	LL(1)方 法
建立分析树的方式	自 下 而 上	自 上 而 下
归约还是推导	规 范 归 约	最 左 推 导
决定使用产生式的时机	看见产生式右部推出的整个终结符串后，才确定用哪个产生式进行归约	看见产生式右部推出的第一个终结符后，便要确定用哪个产生式推导

LR分析方法和LL分析方法的比较

	LR(1)方 法	LL(1)方 法
对文法的显式限制	对文法没有限制	无左递归、无公共左因子

LR分析方法和LL分析方法的比较

	LR(1)方 法	LL(1)方 法
对文法的显式限制	对文法没有限制	无左递归、无公共左因子
分析表比较	状态×文法符号 分析表大	非终结符×终结符 分析表小

LR分析方法和LL分析方法的比较

	LR(1)方 法	LL(1)方 法
对文法的显式限制	对文法没有限制	无左递归、无公共左因子
分析表比较	状态×文法符号 分析表大	非终结符×终结符 分析表小
分析栈比较	状态栈，通常状态比文法符号包含更多信息	文法符号栈

LR分析方法和LL分析方法的比较

	LR(1)方 法	LL(1)方 法
确定句柄	根据栈顶状态和下一个符号便可以确定句柄和归约所用产生式	无句柄概念

LR分析方法和LL分析方法的比较

	LR(1)方 法	LL(1)方 法
确定句柄	根据栈顶状态和下一个符号便可以确定句柄和归约所用产生式	无句柄概念
语法错误	决不会将出错点后的符号移入分析栈	和LR一样，决不会读过出错点而不报错

谢谢！

Thanks!