第四章 语法分析

自底向上分析概述

哈尔滨工业大学 陈鄞

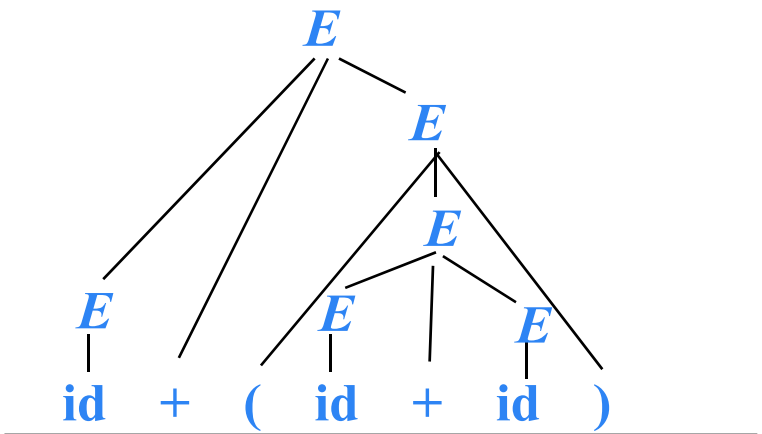
自底向上的语法分析

- 从分析树的底部(叶节点)向顶部(根节点)方向构造分析树
- 可以看成是将输入串w归约为文法开始符号S的过程
- 自顶向下的语法分析采用最左推导方式

  自底向上的语法分析采用最左归约方式（反向构造最右推导）
- 自底向上语法分析的通用框架
  - 移入-归约分析(*Shift-Reduce Parsing*)

例：移入-归约分析

文法
① $E \rightarrow E+E$
② $E \rightarrow E*E$
③ $E \rightarrow (E)$
④ $E \rightarrow id$



每次归约的符号串称为“句柄”

栈	剩余输入	动作
$	id+(id+id) $	
$ id	+(id+id) $	移入
$ E	+(id+id) $	归约: $E \rightarrow id$
$ E+	(id+id) $	移入
$ E+(	id+id) $	移入
$ E+(id	+id) $	移入
$ E+(E	+id) $	归约: $E \rightarrow id$
$ E+(E+	id) $	移入
$ E+(E+id	) $	移入
$ E+(E+E	) $	归约: $E \rightarrow id$
$ E+(E	) $	归约: $E \rightarrow E+E$
$ E+(E)	$	移入
$ E+E	$	归约: $E \rightarrow (E)$
$ E	$	归约: $E \rightarrow E+E$

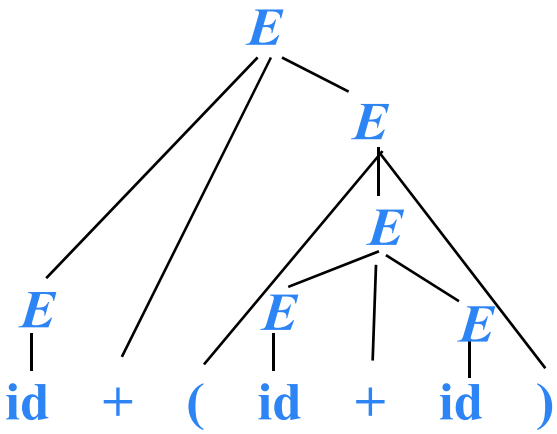
例：移入-归约分析

最右推导的每一步都是一个规范句型

栈内符号串 + 剩余输入 = “规范句型”

文法

① $E \rightarrow E+E$
② $E \rightarrow E*E$
③ $E \rightarrow (E)$
④ $E \rightarrow id$



栈	剩余输入	动作
$	id+(id+id) $	
$ id	+(id+id) $	移入
$ E	+(id+id) $	归约: E→id
$ E+	(id+id) $	移入
$ E+(	id+id) $	移入
$ E+(id	+id) $	移入
$ E+(E	+id) $	归约: E→id
$ E+(E+	id) $	移入
$ E+(E+id	) $	移入
$ E+(E+E	) $	归约: E→id
$ E+(E	) $	归约: E→E+E
$ E+(E)	$	移入
$ E+E	$	归约: E→(E)
$ E	$	归约: E→E+E

移入-归约分析的工作过程

- 在对输入串的一次从左到右扫描过程中，语法分析器将零个或多个输入符号移入到栈的顶端，直到它可以对栈顶的一个文法符号串β进行归约为止
- 然后，它将β归约为某个产生式的左部
- 语法分析器不断地重复这个循环，直到它检测到一个语法错误，或者栈中包含了开始符号且输入缓冲区为空(当进入这样的格局时，语法分析器停止运行，并宣称成功完成了语法分析)为止

移入-归约分析器可采取的4种动作

- 移入：将下一个输入符号移到栈的顶端
- 归约：被归约的符号串的右端必然处于栈顶。语法分析器在栈中确定这个串的左端，并决定用哪个非终结符来替换这个串
- 接收：宣布语法分析过程成功完成
- 报错：发现一个语法错误，并调用错误恢复子例程

移入-归约分析中存在的问题

例：

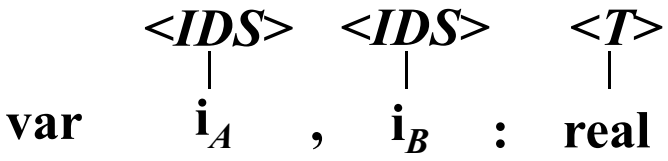
(1) <*S*>→var <*IDS*> : <*T*>

(2) <*IDS*>→i

(3) <*IDS*>→<*IDS*> , i

(4) <*T*>→real | int

栈	剩余输入	动作
$	var i_A , i_B : real $	
$ var	i_A , i_B : real $	移入
$ var i_A	, i_B : real $	移入
$ var <*IDS*>	, i_B : real $	归约
$ var <*IDS*> ,	i_B : real $	移入
$ var <*IDS*> , i_B	: real $	移入
$ var <*IDS*> , <*IDS*>	: real $	归约
$ var <*IDS*> , <*IDS*> :	real $	移入
$ var <*IDS*> , <*IDS*> : real	$	移入
$ var <*IDS*> , <*IDS*> :<*T*>	$	归约



移入-归约分析中存在的问题

造成错误的原因：
错误地识别了句柄

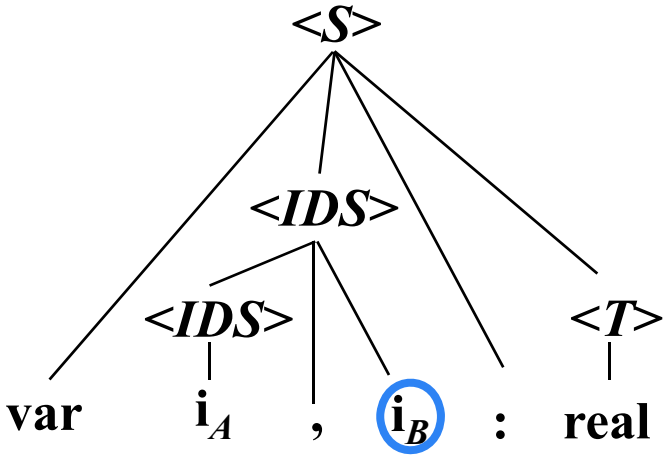
例：

(1) <*S*>→var <*IDS*> : <*T*>

(2) <*IDS*>→i

(3) <*IDS*>→<*IDS*> , i

(4) <*T*>→real | int



栈	剩余输入	动作
$	var i_A , i_B : real $	
$ var	i_A , i_B : real $	移入
$ var i_A	, i_B : real $	移入
$ var <*IDS*>	, i_B : real $	归约
$ var <*IDS*> ,	i_B : real $	移入
$ var <*IDS*> , i_B	: real $	移入
$ var <*IDS*>	: real $	归约
$ var <*IDS*> :	real $	移入
$ var <*IDS*> : real	$	移入
$ var <*IDS*> :<*T*>	$	归约
$ <*S*>	$	归约

句柄：句型的最左直接短语

移入-归约分析中存在的问题

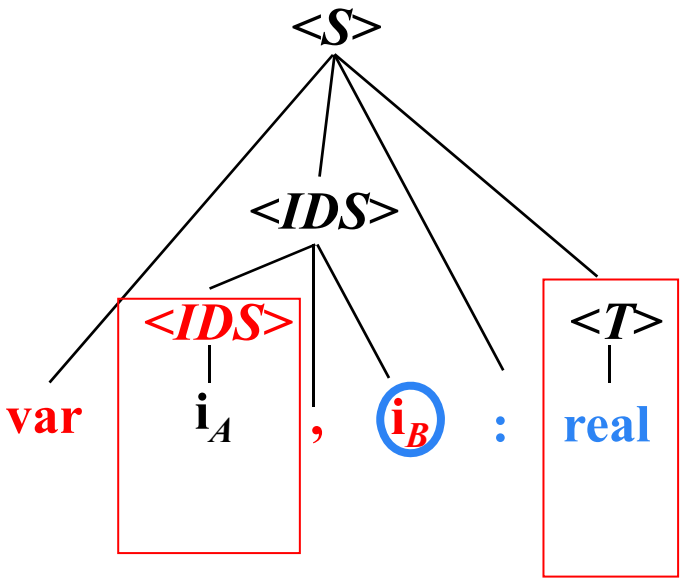
例：

(1) <*S*>→var <*IDS*> : <*T*>

(2) <*IDS*>→i

(3) <*IDS*>→<*IDS*> , i

(4) <*T*>→real | int



造成错误的原因：
错误地识别了句柄

栈	剩余输入	动作
$	var i_A , i_B : real $	
$ var	i_A , i_B : real $	移入
$ var i_A	, i_B : real $	移入
$ var <*IDS*>	, i_B : real $	归约
$ var <*IDS*> ,	i_B : real $	移入
$ var <*IDS*> , i_B	: real $	移入
$ var <*IDS*>	: real $	归约
$ var <*IDS*> :	real $	移入
$ var <*IDS*> :real	$	归约
$ var <*IDS*> :<*T*>	$	归约
$ <*S*>	$	归约

如何正确地识别句柄？

句柄：句型的最左直接短语

第四章 语法分析

自底向上分析概述

哈尔滨工业大学 陈鄞

第四章 语法分析

LR分析法概述

哈尔滨工业大学 陈鄞

如何正确地识别句柄？

如何正确地识别句柄？

LR 分析法

- *LR* 文法(*Knuth*, 1963) 是最大的、可以构造出相应移入-归约语法分析器的文法类
  - *L*: 对输入进行从左到右的扫描
  - *R*: 反向构造出一个最右推导序列
- *LR*(*k*)分析
  - 需要向前查看*k*个输入符号的*LR*分析

> *k* = 0 和 *k* = 1 这两种情况具有实践意义
> 当省略(*k*)时，表示*k* =1

LR 分析法的基本原理

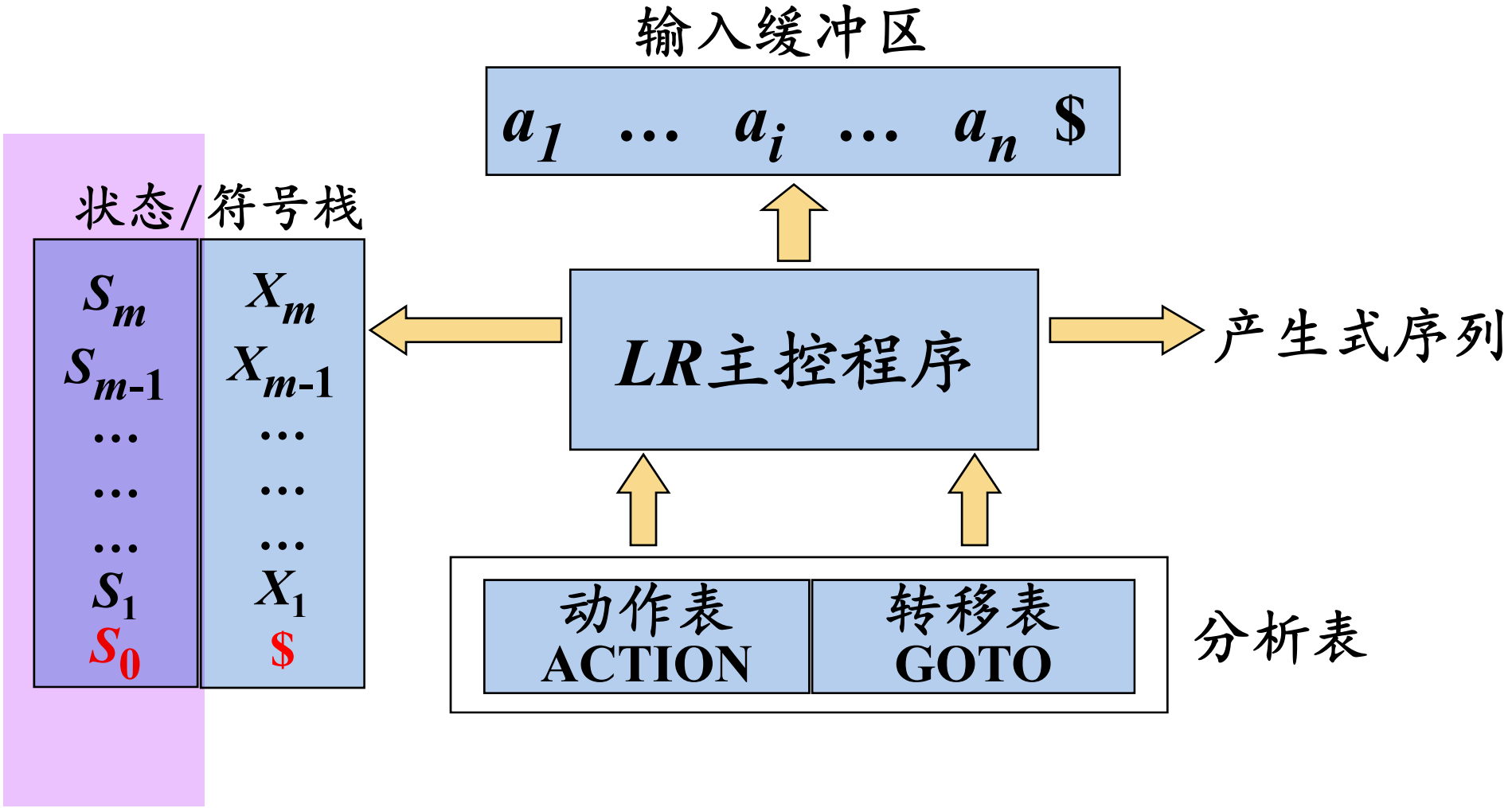
- 自底向上分析的关键问题是什么？
  - 如何正确地识别句柄
- 句柄是逐步形成的，用“状态”表示句柄识别的进展程度
  - 例：$S \rightarrow bBB$
    - $S \rightarrow \cdot bBB$ ← 移进状态
    - $S \rightarrow b \cdot BB$ （期待b）
    - $S \rightarrow bB \cdot B$ （期待B）

  待约状态

  - $S \rightarrow bBB \cdot$ ← 归约状态

*LR*分析器基于这样一些状态来构造自动机进行句柄的识别

LR 分析器（自动机）的总体结构



LR 分析表的结构

- 例
  - 文法
    - ① $S \rightarrow BB$
    - ② $B \rightarrow aB$
    - ③ $B \rightarrow b$

状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

非终结符

终结符

后继状态

sn：将符号*a*、状态*n* 压入栈
rn：用第*n*个产生式进行归约

reduce

LR 分析表的结构

- 例
  - 文法
    - ① $S \rightarrow BB$
    - ② $B \rightarrow aB$
    - ③ $B \rightarrow b$

状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

栈　　剩余输入

状态栈 0 4

符号栈 $ *B*　　*b a b* $

p27 5：47----可以直接看视频

LR 分析表的结构

- 例
  - 文法
    - ① $S \rightarrow BB$
    - ② $B \rightarrow aB$
    - ③ $B \rightarrow b$

状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

栈　　剩余输入

0 2 3 4

$ *B*　　*a b* $

B

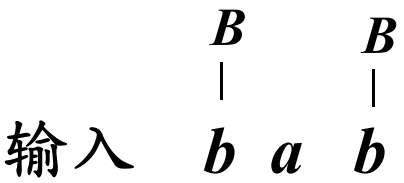
|

输入　*b a b*

LR 分析表的结构

- 例
  - 文法
    - ① $S \rightarrow BB$
    - ② $B \rightarrow aB$
    - ③ $B \rightarrow b$

状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

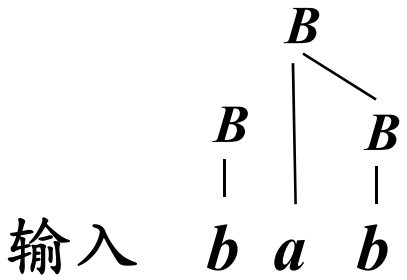


栈　　剩余输入

0 2 3 6　　$B a B　　$

LR 分析表的结构

- 例
  - 文法
    - ① $S \rightarrow BB$
    - ② $B \rightarrow aB$
    - ③ $B \rightarrow b$



状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

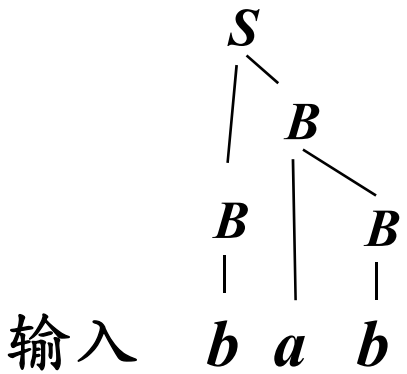
栈　　剩余输入

0 2 5

$ *B B*　　$

LR 分析表的结构

- 例
  - 文法
    - ① $S \rightarrow BB$
    - ② $B \rightarrow aB$
    - ③ $B \rightarrow b$



状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

栈	剩余输入
0 1	
$ *S*	$

LR 分析器的工作过程

- 初始化

s_0	
\$	$a_1a_2\ldots a_n$ \$

- 一般情况下

$s_0s_1\ldots s_m$	
$\$X_1\ldots X_m$	$a_ia_{i+1}\ldots a_n$ \$

①如果ACTION $[s_m, a_i]$= sx，那么格局变为：

$s_0s_1\ldots s_m x$	
$\$X_1\ldots X_m a_i$	$a_{i+1}\ldots a_n$ \$

LR 分析器的工作过程

➢初始化

s_0	
\$	$a_1 a_2 \ldots a_n$ \$

➢一般情况下

$s_0 s_1 \ldots s_m$	
$\$X_1 \ldots X_m$	$a_i a_{i+1} \ldots a_n$ \$

②如果ACTION[s_m, a_i]= rx 表示用第x个产生式$A \rightarrow X_{m-(k-1)} \ldots X_m$

进行归约，那么格局变为：

$s_0 s_1 \ldots s_{m-k}$	
$\$ X_1 \ldots X_{m-k} A$	$a_i a_{i+1} \ldots a_n$ \$

如果GOTO[s_{m-k}, A]=y，那么格局变为：

剩余符号栈没有变化

$s_0 s_1 \ldots s_{m-k}\ y$	
$\$ X_1 \ldots X_{m-k} A$	$a_i a_{i+1} \ldots a_n$ \$

LR 分析器的工作过程

- 初始化

s_0	
\$	$a_1a_2\ldots a_n$ \$

- 一般情况下

$s_0s_1\ldots s_m$	
$\$X_1\ldots X_m$	$a_ia_{i+1}\ldots a_n$ \$

③如果ACTION[s_m, a_i]=*acc*，那么分析成功

④如果ACTION[s_m, a_i]=*err*，那么出现语法错误

LR 分析算法

- 输入：串*w*和*LR*语法分析表，该表描述了文法*G*的ACTION函数和GOTO函数。
- 输出：如果*w*在 *L*(*G*)中，则输出*w*的自底向上语法分析过程中的归约步骤；否则给出一个错误指示。
- 方法：初始时，语法分析器栈中的内容为初始状态s_0，输入缓冲区中的内容为*w*$。然后，语法分析器执行下面的程序：

```
令a为w$的第一个符号;
while(1) { /* 永远重复*/
      令s是栈顶的状态;
      if ( ACTION [s，a] = st ) {
            将t压入栈中;
            令a为下一个输入符号;
      } else  if (ACTION [s，a] =归约A → β ) {
            从栈中弹出 | β | 个符号;
            将GOTO [t，A]压入栈中;
            输出产生式 A → β ;
      } else  if (ACTION [s，a] =接受) break;  /* 语法分析完成*/
      else调用错误恢复例程;
   }
```

关键在于如何构造分析表，存在不同的方法

如何构造给定文法的*LR*分析表？

- ***LR*(0)分析**
- ***SLR*分析**
- ***LR*(1)分析**
- ***LALR*分析**

第四章 语法分析

LR分析法概述

哈尔滨工业大学 陈鄞

第四章 语法分析

LR(0)分析

哈尔滨工业大学 陈鄞

LR(0) 项目

- 右部某位置标有圆点的产生式称为相应文法的一个*LR*(0)项目（简称为项目）

$$A \rightarrow \alpha_1 \cdot \alpha_2$$

例：$S \rightarrow bBB$

- $S \rightarrow \cdot bBB$ ⟵ 移进项目
- $S \rightarrow b \cdot BB$
- $S \rightarrow bB \cdot B$

} 待约项目

项目描述了句柄识别的状态

- $S \rightarrow bBB \cdot$ ⟵ 归约项目

产生式$A \rightarrow \varepsilon$只生成一个项目$A \rightarrow \cdot$

k个符号就有k+1个项目

增广文法 (*Augmented Grammar*)

- 如果G是一个以S为开始符号的文法，则G的增广文法G'就是在G中加上新开始符号S'和产生式$S' \to S$而得到的文法
- 例

1) $E \to E + T$
2) $E \to T$
3) $T \to T * F$
4) $T \to F$
5) $F \to (E)$
6) $F \to \mathrm{id}$

0) $E' \to E$
1) $E \to E + T$
2) $E \to T$
3) $T \to T * F$
4) $T \to F$
5) $F \to (E)$
6) $F \to \mathrm{id}$

引入这个新的开始产生式的目的是使得文法开始符号仅出现在一个产生式的左边，从而使得分析器只有一个接受状态

文法中的项目

① $S'\rightarrow S$　② $S\rightarrow vI:T$　③ $I\rightarrow I,i$　④ $I\rightarrow i$　⑤ $T\rightarrow r$

初始项目

(0) $S'\rightarrow \cdot S$

(1) $S'\rightarrow S\cdot$

(2) $S\rightarrow \cdot vI:T$
(3) $S\rightarrow v\cdot I:T$
(4) $S\rightarrow vI\cdot :T$
(5) $S\rightarrow vI:\cdot T$
(6) $S\rightarrow vI:T\cdot$

(7) $I\rightarrow \cdot I,i$
(8) $I\rightarrow I\cdot ,i$
(9) $I\rightarrow I,\cdot i$
(10) $I\rightarrow I,i\cdot$

(11) $I\rightarrow \cdot i$
(12) $I\rightarrow i\cdot$

(13) $T\rightarrow \cdot r$
(14) $T\rightarrow r\cdot$

归约项目　圆点位于末尾

接收项目

文法中的项目

① $S'\rightarrow S$	② $S\rightarrow \mathrm{v}I:T$	③ $I\rightarrow I,\mathrm{i}$	④ $I\rightarrow \mathrm{i}$	⑤ $T\rightarrow \mathrm{r}$

(2) $S\rightarrow \cdot \mathrm{v}I:T$
(3) $S\rightarrow \mathrm{v}\cdot I:T$
(4) $S\rightarrow \mathrm{v}I\cdot :T$
(5) $S\rightarrow \mathrm{v}I:\cdot T$
(6) $S\rightarrow \mathrm{v}I:T\cdot$
(7) $I\rightarrow \cdot I,\mathrm{i}$
(8) $I\rightarrow I\cdot ,\mathrm{i}$
(9) $I\rightarrow I,\cdot \mathrm{i}$
(10) $I\rightarrow I,\mathrm{i}\cdot$
(0) $S'\rightarrow \cdot S$
(1) $S'\rightarrow S\cdot$
(11) $I\rightarrow \cdot \mathrm{i}$
(12) $I\rightarrow \mathrm{i}\cdot$
(13) $T\rightarrow \cdot \mathrm{r}$
(14) $T\rightarrow \mathrm{r}\cdot$

- 后继项目（*Successive Item*）
  - 同属于一个产生式的项目，但圆点的位置只相差一个符号，则称后者是前者的后继项目
  - $A\rightarrow \alpha\cdot X\beta$的后继项目是$A\rightarrow \alpha X\cdot \beta$

文法中的项目

① $S'\rightarrow S$　② $S\rightarrow \mathrm{v}I:T$　③ $I\rightarrow I,\mathrm{i}$　④ $I\rightarrow \mathrm{i}$　⑤ $T\rightarrow \mathrm{r}$

(0) $S'\rightarrow \cdot S$
(1) $S'\rightarrow S\cdot$
(2) $S\rightarrow \cdot \mathrm{v}I:T$
(3) $S\rightarrow \mathrm{v}\cdot I:T$
(4) $S\rightarrow \mathrm{v}I\cdot :T$
(5) $S\rightarrow \mathrm{v}I:\cdot T$
(6) $S\rightarrow \mathrm{v}I:T\cdot$
(7) $I\rightarrow \cdot I,\mathrm{i}$
(8) $I\rightarrow I\cdot ,\mathrm{i}$
(9) $I\rightarrow I,\cdot \mathrm{i}$
(10) $I\rightarrow I,\mathrm{i}\cdot$
(11) $I\rightarrow \cdot \mathrm{i}$
(12) $I\rightarrow \mathrm{i}\cdot$
(13) $T\rightarrow \cdot \mathrm{r}$
(14) $T\rightarrow \mathrm{r}\cdot$

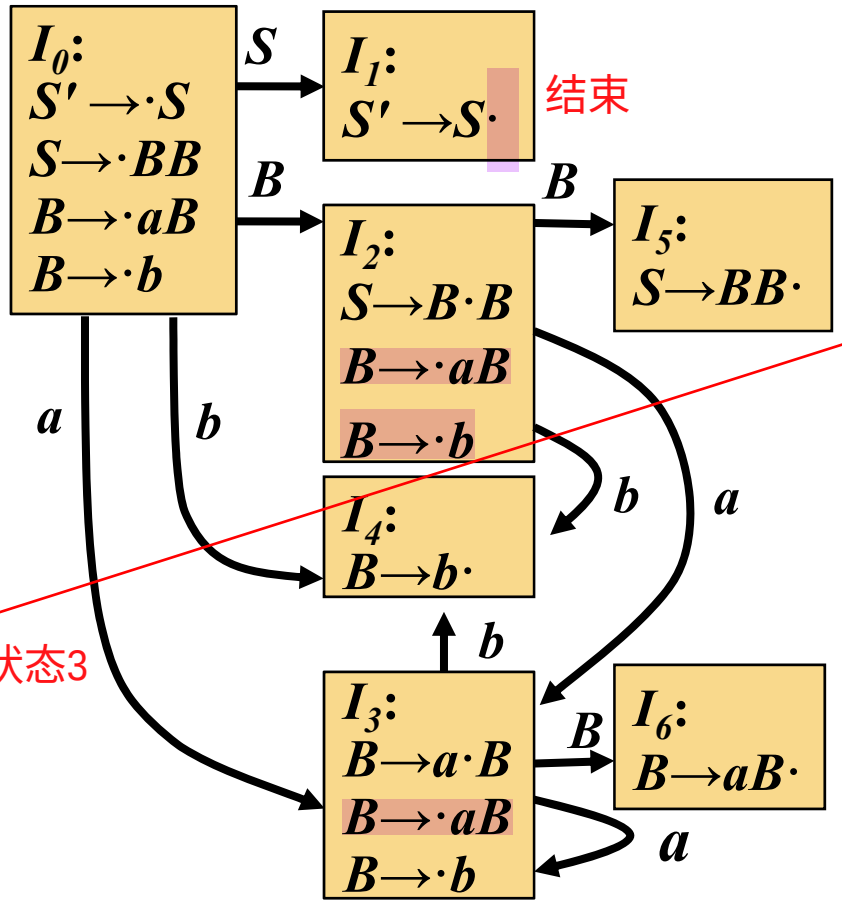
这15个项目中是否会
有某些项目是等价的？

可以把等价的项目组成一个项目集(I)，称为项目集闭包
(*Closure of Item Sets*)，每个项目集闭包对应着自动机的一个状态

原点后面是非终结符的时候就存在等价项目

例：LR(0)自动机

文法
(0) $S' \to S$
(1) $S \to BB$
(2) $B \to aB$
(3) $B \to b$



将终结符移入，并转到状态3

$LR(0)$分析表

状态	ACTION			GOTO	
	a	b	$	S	B
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

用第三个产生式归约

第四章 语法分析

LR(0)分析

哈尔滨工业大学 陈鄞

第四章 语法分析

LR(0)分析表构造算法

哈尔滨工业大学 陈鄞

例：LR(0)自动机

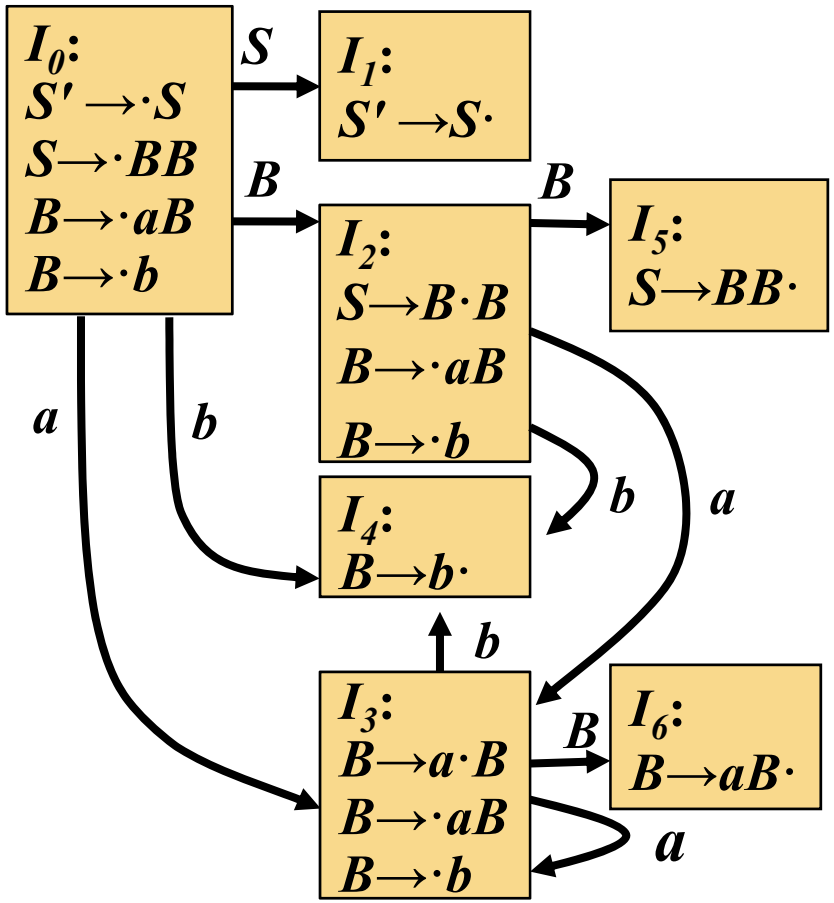
文法

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$



LR(0)分析表

状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

CLOSURE( )函数

- 计算给定项目集I的闭包

$$\mathbf{CLOSURE}(I) = I \cup \{B \to \cdot\gamma \mid A \to \alpha \cdot B\beta \in \mathbf{CLOSURE}(I), B \to \gamma \in P\}$$

CLOSURE( )函数

```
SetOfItems CLOSURE ( I ) {
        J = I;
        repeat
                for ( J中的每个项A → α·Bβ )
                        for ( G的每个产生式B → γ )
                                if ( 项B → ·γ不在J中 )
                                        将B → ·γ加入J中;
        until 在某一轮中没有新的项被加入到J中;
        return J;
}
```

GOTO ( )函数

- 返回项目集I对应于文法符号X的后继项目集闭包

$$GOTO(I, X)=CLOSURE(\{A\rightarrow\alpha X\cdot\beta \mid A\rightarrow\alpha\cdot X\beta\in I\})$$

```
SetOfItems GOTO (I, X) {
        将J初始化为空集;
        for (I中的每个项A → α·Xβ)
                将项 A → αX·β 加入到集合J中;
        return CLOSURE (J);
}
```

构造*LR*(0)自动机的状态集

➢ 规范*LR*(0) 项集族(*Canonical LR*(0) *Collection*)

$$C=\{I_0\}\cup\{I \mid \exists J\in C, X\in V_N\cup V_T, I=GOTO(J, X)\}$$

```
void items( G' ) {
        C={ CLOSURE ({[ S'→ ·S ]} ) };
        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)
                                        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;
}
```

$LR(0)$分析表构造算法

- 构造G'的规范$LR(0)$项集族$C=\{I_0, I_1, \cdots, I_n\}$
- 令I_i对应状态i。状态i的语法分析动作按照下面的方法决定：
  - *if* $A\to\alpha\cdot a\beta\in I_i$ *and* $GOTO(I_i, a)=I_j$ *then* $ACTION[i, a]=sj$
  - *if* $A\to\alpha.B\beta\in I_i$ *and* $GOTO(I_i, B)=I_j$ *then* $GOTO[i, B]=j$
  - *if* $A\to\alpha\cdot\in I_i$ 且$A\neq S'$ *then for* $\forall a\in V_T\cup\{\$\}$ *do* $ACTION[i, a]=rj$
    （j是产生式$A\to\alpha$的编号）
  - *if* $S'\to S\cdot\in I_i$ *then* $ACTION[i, \$]=acc$
- 没有定义的所有条目都设置为“*error*”

$LR(0)$ 自动机的形式化定义

- 文法

$$G=(V_N, V_T, P, S)$$

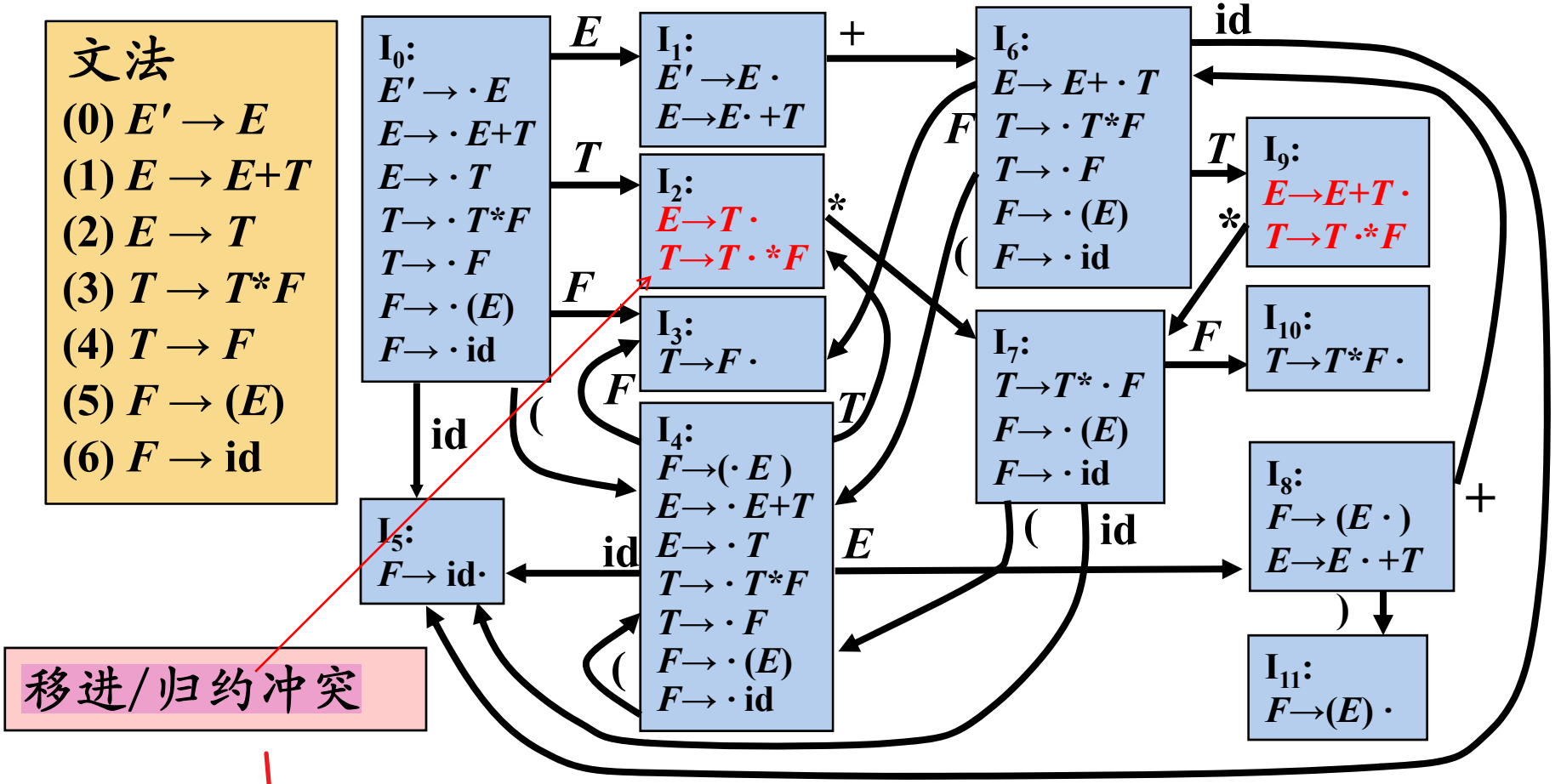
- $LR(0)$ 自动机

转化函数

$$M=(C, V_N \cup V_T, GOTO, I_0, F)$$

- $C=\{I_0\} \cup \{I \mid \exists J \in C, X \in V_N \cup V_T, I=GOTO(J,X)\}$
- $I_0=CLOSURE(\{S' \rightarrow .S\})$ 开始状态
- $F=\{CLOSURE(\{S' \rightarrow S.\})\}$ 终止状态

LR(0) 分析过程中的冲突

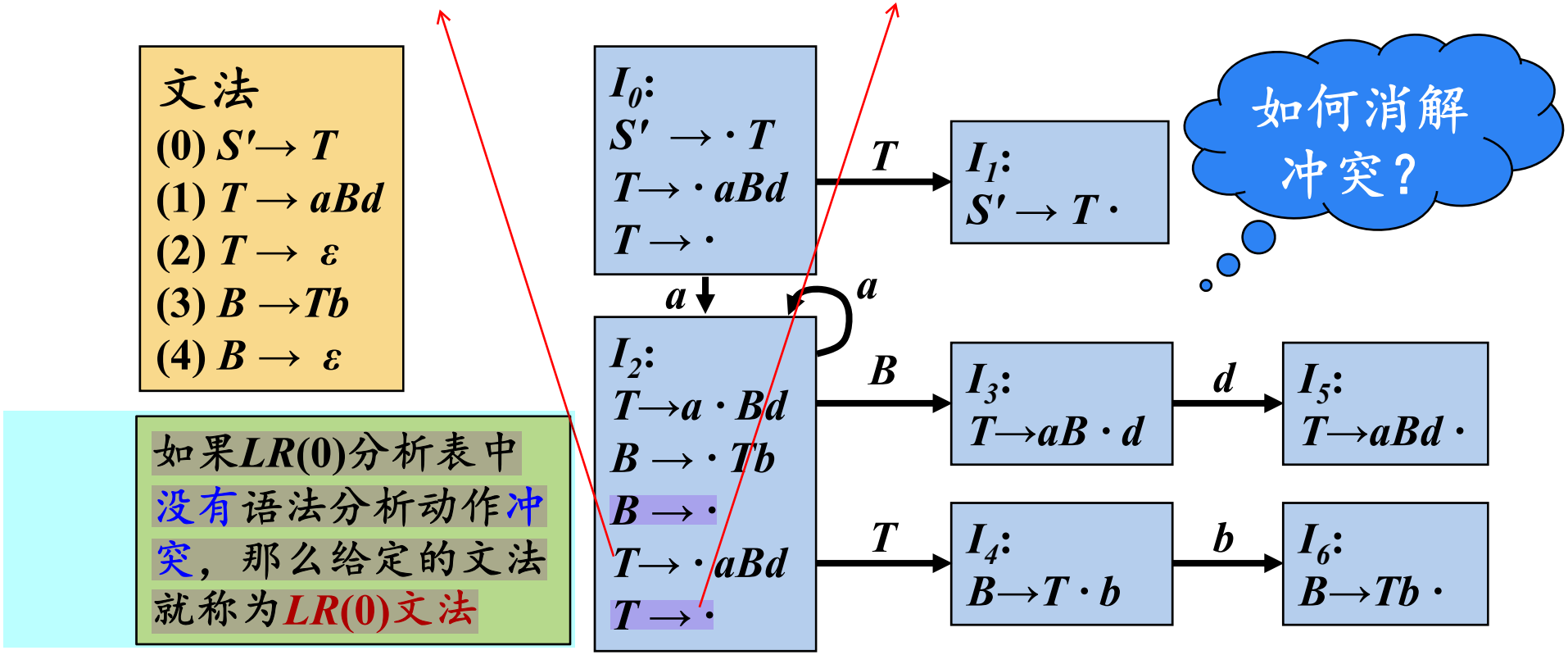


表达式文法的*LR*(0)分析表含有移进/归约冲突

状态	*ACTION*						*GOTO*		
	id	+	*	(	)	$	*E*	*T*	*F*
0	s5			s_4			1	2	3
1		s6				acc			
2	r2	r2	r2/s7	r2	r2	r2			
3	r4	r4	r4	r4	r4	r4			
4	s5			s4			8	2	3
5	r6	r6	r6	r6	r6	r6			
6	s5			s4				9	3
7	s5			s4					10
8		s6			s11				
9	r1	r1	r1/s7	r1	r1	r1			
10	r3	r3	r3	r3	r3	r3			
11	r5	r5	r5	r5	r5	r5			

例：移进/归约冲突和归约/归约冲突

需要用其他的分析法来解决



第四章 语法分析

LR(0)分析表构造算法

哈尔滨工业大学 陈鄞