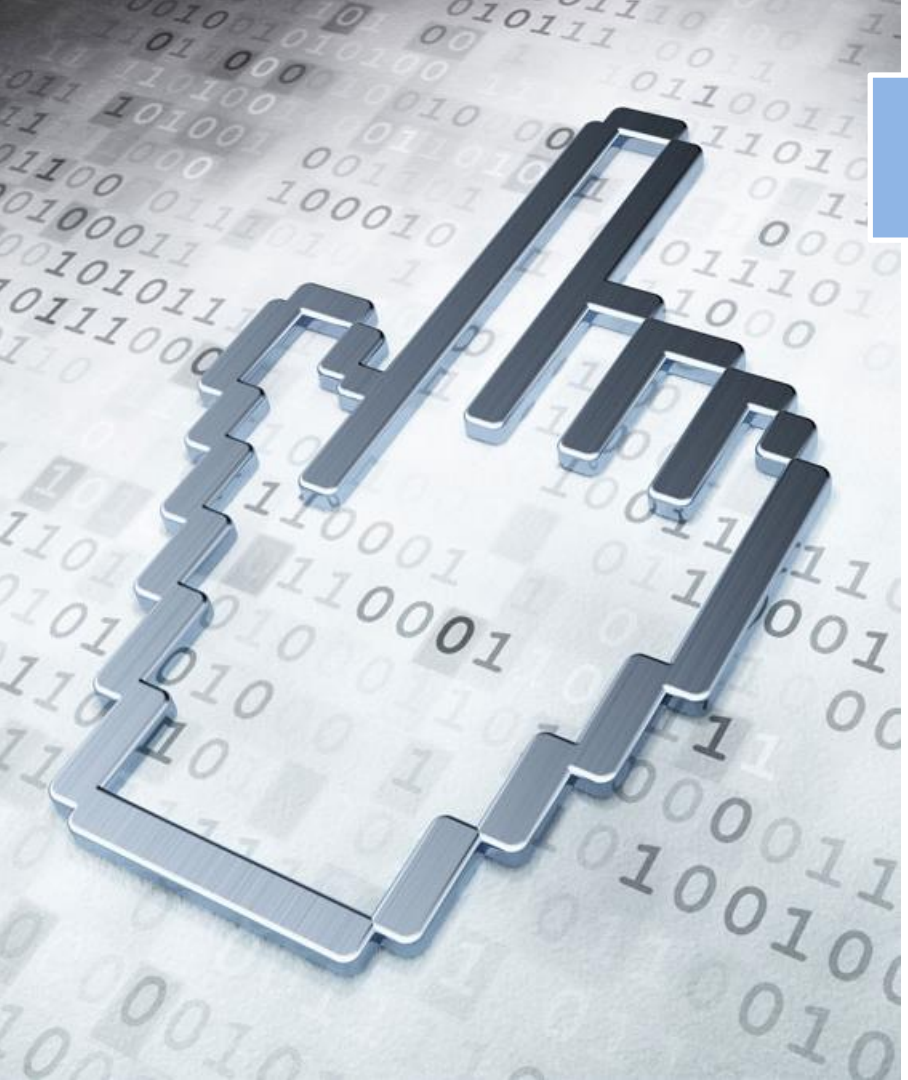


提纲

4.3 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（构造句子的最左推导）

  自底向上的语法分析采用最左归约方式（反向构造句子的最右推导）
- 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

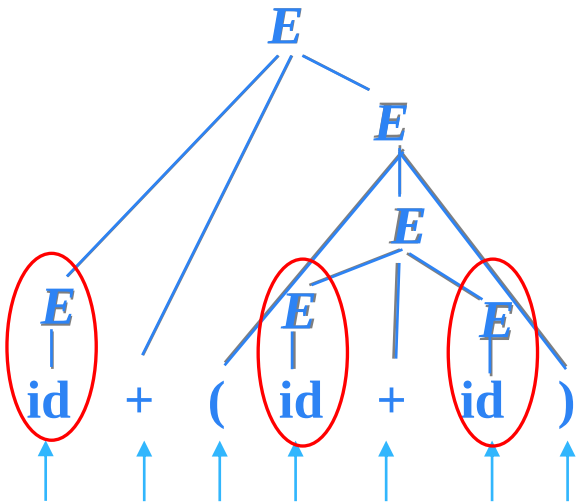
文法

① $E \rightarrow E+E$

② $E \rightarrow E*E$

③ $E \rightarrow (E)$

④ $E \rightarrow id$



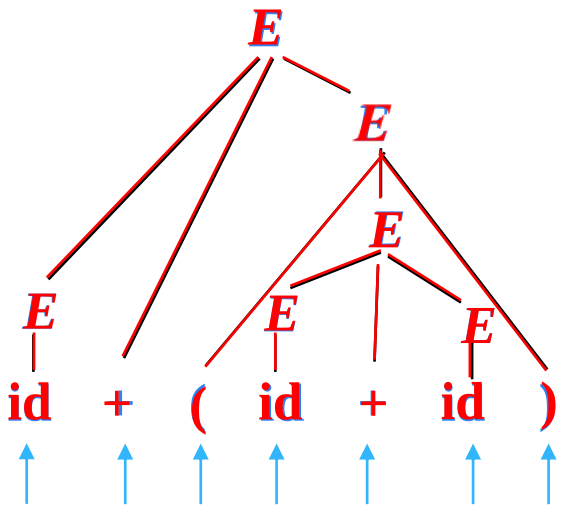
最左归约

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

例：移入-归约分析

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 $E \rightarrow id$



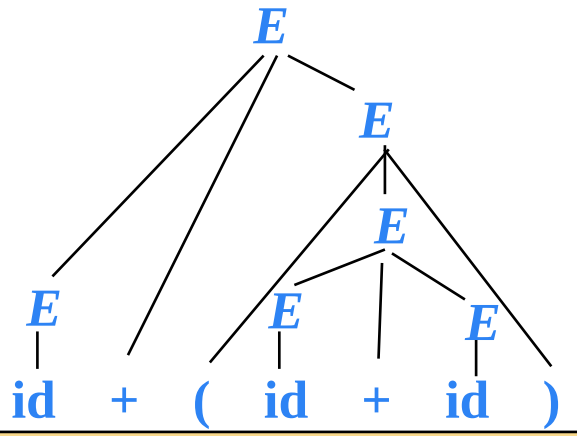
最左归约

最右推导

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

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→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

- 归约-归约冲突
- 移入-归约冲突

归约-归约冲突

例：

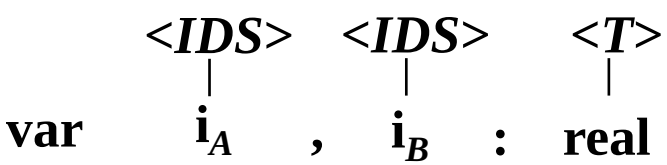
(1) <*S*>→var <*IDS*> : <*T*>

(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



归约-归约冲突

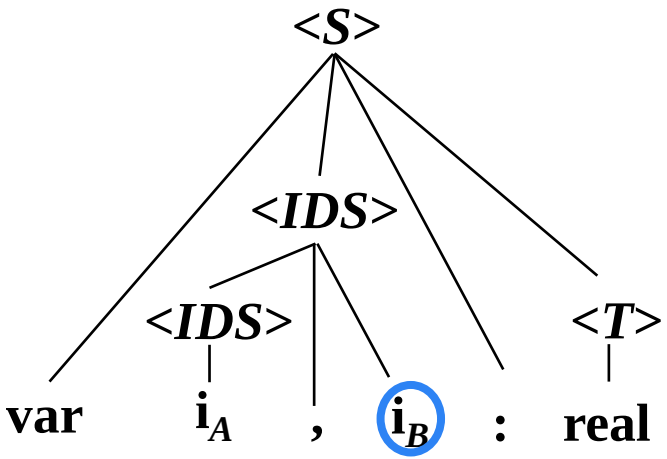
例：

(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	$	移入
$ var <*IDS*> :<*T*>	$	归约
$ <*S*>	$	归约

句柄：句型的最左直接短语

归约-归约冲突

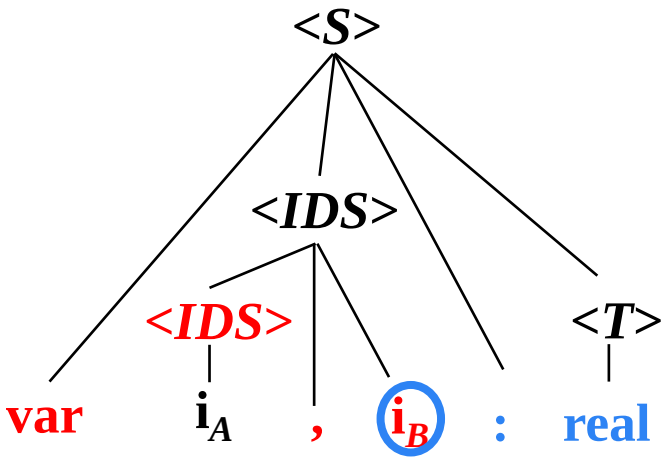
例：

(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？

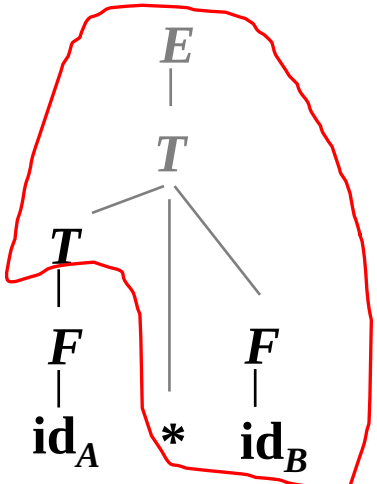
句柄：句型的最左直接短语

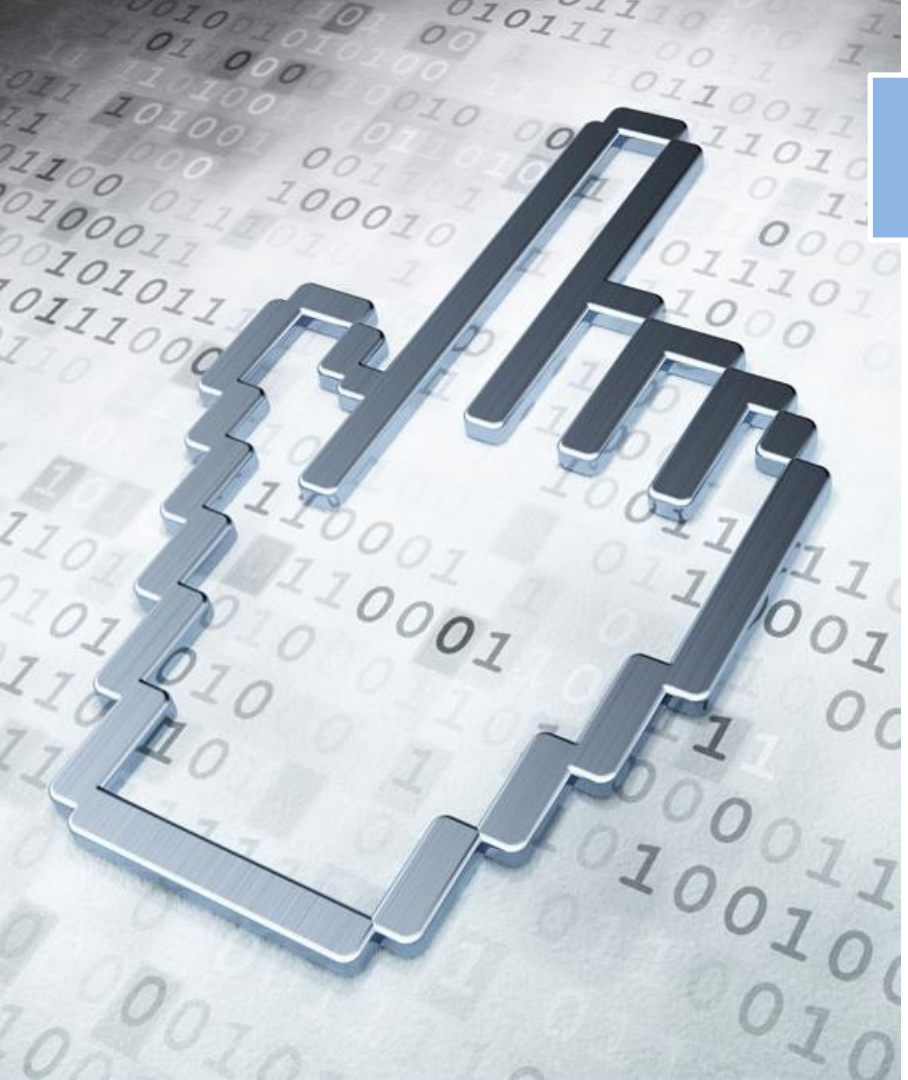
移入-归约冲突

➢例

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) $F \rightarrow \mathrm{id}$

栈	剩余输入	动作
$	$\mathrm{id}_A * \mathrm{id}_B$ $	
$ id_A	$* \mathrm{id}_B$ $	移入
$ F	$* \mathrm{id}_B$ $	归约
$ T	$* \mathrm{id}_B$ $	归约





提纲

4.4 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

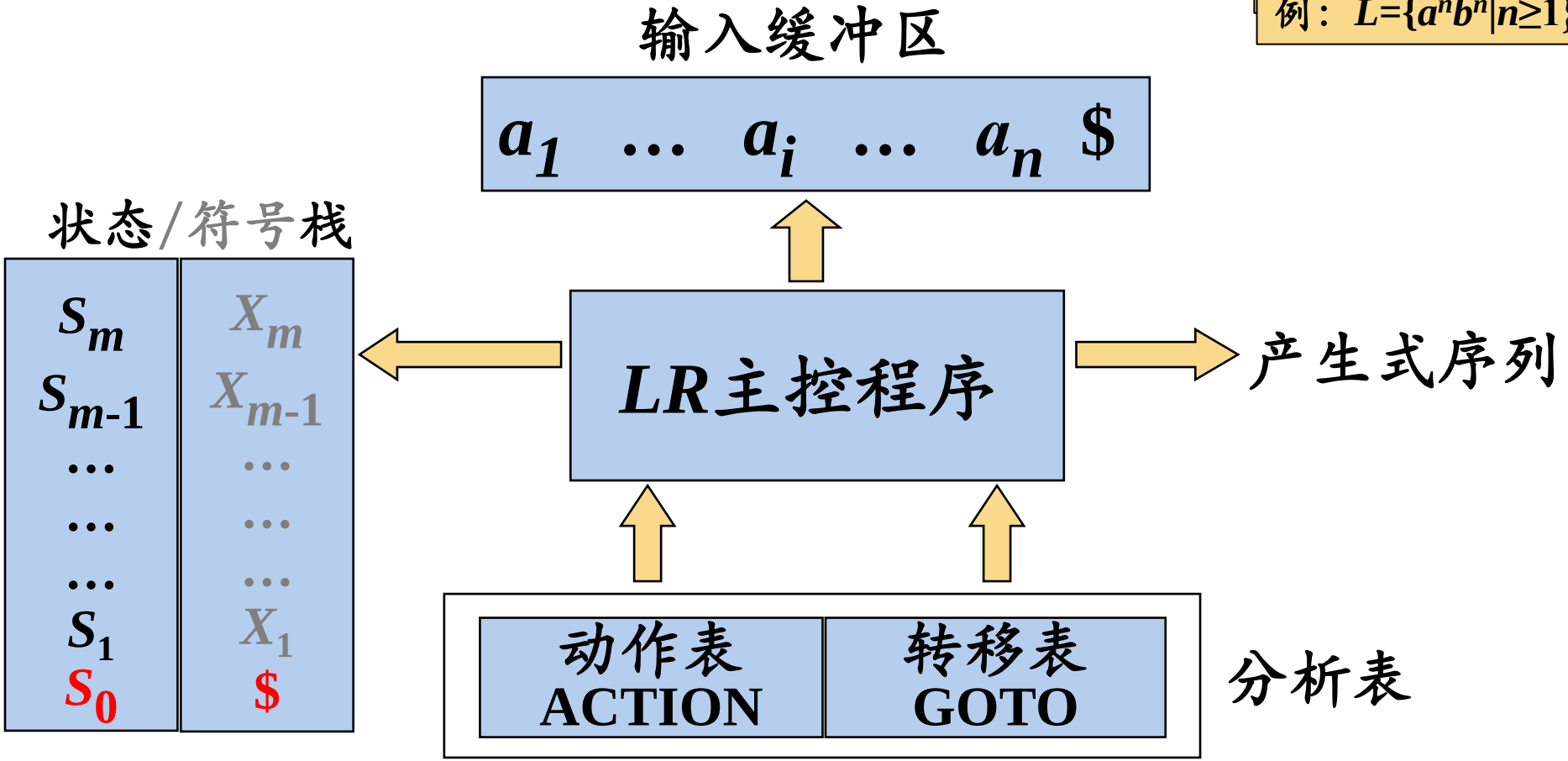
- 自底向上分析的关键问题是什么？
  - 如何正确地识别句柄
- 句柄是逐步形成的，用“状态”表示句柄识别的进展程度
  - 例：$S \rightarrow bBB$
    - $S \rightarrow \cdot bBB$ ⟵ 移进状态
    - $S \rightarrow b \cdot BB$
    - $S \rightarrow bB \cdot B$

    } 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

下推自动机
(Push Down Automata,
例： $L=\{a^n b^n | n \geq 1\}$



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		

sn：将符号*a*、状态*n*压入栈
rn：用第*n*个产生式进行归约

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		

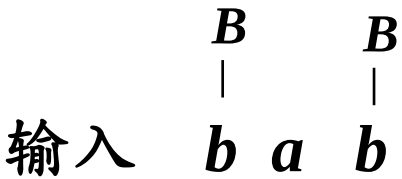
栈	剩余输入
0 4	
$ *B*	*bab* $

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		



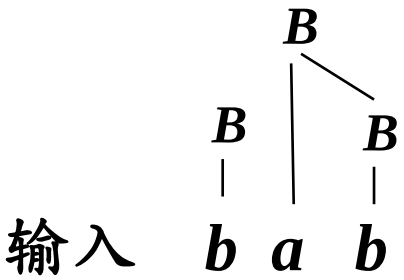
栈　　剩余输入

0 2 3 4

$ *B*　　*ab* $

LR 分析表的结构

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

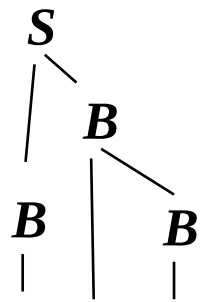


状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

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 $B \rightarrow b$



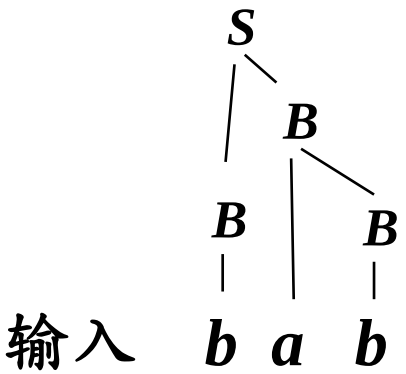
输入 *b a b*

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		

$$S_i \xrightarrow[A\,(\text{归约})]{a\,(\text{移入})} S_j$$

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

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

①如果ACTION $[s_m, a_i]$= sx，那么格局变为：

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

LR 分析器的工作过程

- 初始化

s_0	
\$	$a_1a_2\ldots a_n$ \$

- 一般情况下

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

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：

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

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;
   }
```

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

4.4.1 LR(0) 分析

- 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

- $S \rightarrow bBB \cdot$ ⟵ 归约项目

项目描述了句柄识别的状态

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

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$S' \rightarrow S$而得到的文法
- 例

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

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

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) $S'\rightarrow \cdot S$　(2) $S\rightarrow \cdot vI{:}T$　(3) $S\rightarrow v\cdot I{:}T$　(4) $S\rightarrow vI\cdot{:}T$　(5) $S\rightarrow vI{:}\cdot T$　(7) $I\rightarrow \cdot I,i$　(8) $I\rightarrow I\cdot,i$　(9) $I\rightarrow I,\cdot i$　(11) $I\rightarrow \cdot i$　(13) $T\rightarrow \cdot r$

(1) $S'\rightarrow S\cdot$　(6) $S\rightarrow vI{:}T\cdot$　(10) $I\rightarrow I,i\cdot$　(12) $I\rightarrow i\cdot$　(14) $T\rightarrow r\cdot$

归约项目

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 $T\rightarrow \mathrm{r}$

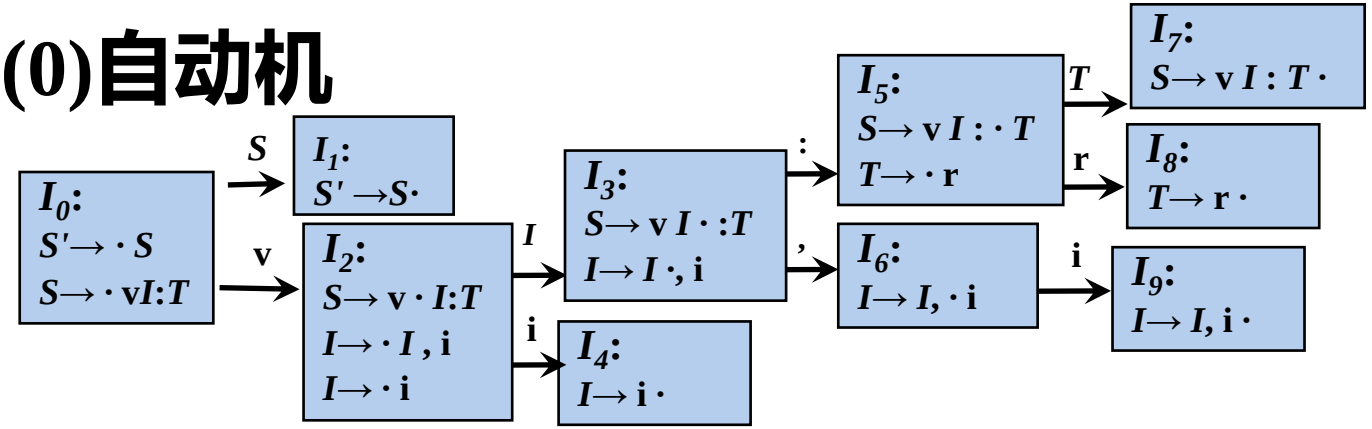
(0) $S'\rightarrow \cdot S$
(1) $S'\rightarrow S\cdot$
(2) $S\rightarrow \cdot \mathrm{v}I:T$
(3) $S\rightarrow \mathrm{v}\cdot I:T$
(4) $S\rightarrow \mathrm{v}I\cdot :T$
(5) $S\rightarrow \mathrm{v}I:\cdot T$
(6) $S\rightarrow \mathrm{v}I:T\cdot$
(7) $I\rightarrow \cdot I,\mathrm{i}$
(8) $I\rightarrow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

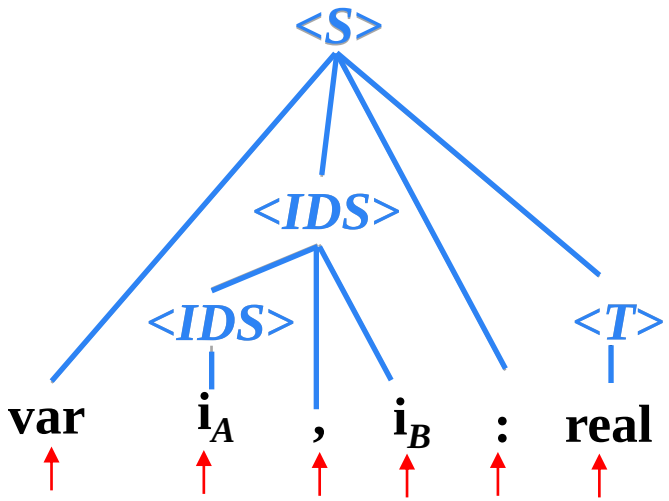
这15个项目中是否会
有某些项目是等价的？

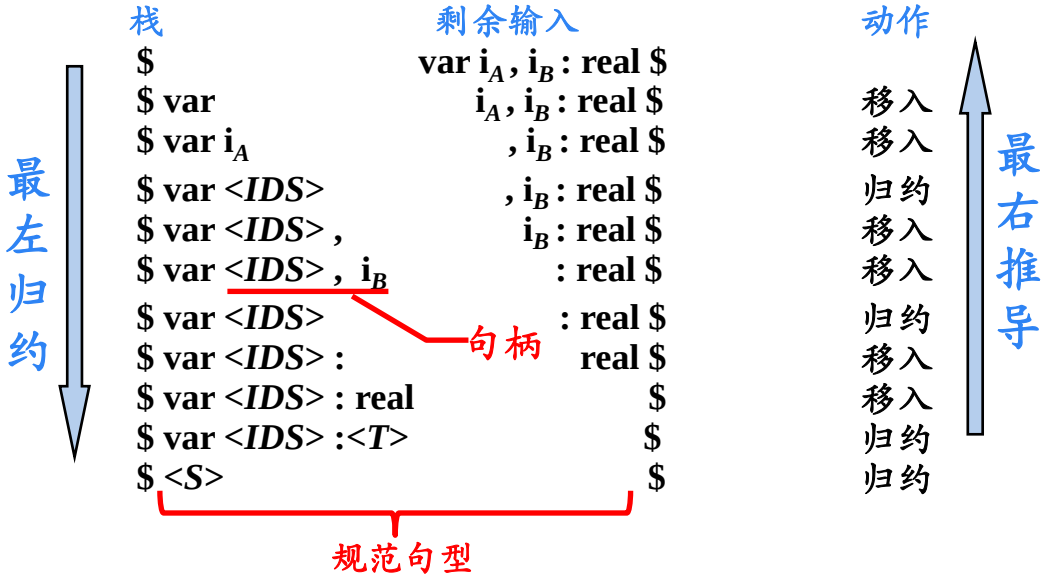
可以把等价的项目组成一个项目集(I)，称为项目集闭包
(*Closure of Item Sets*)，每个项目集闭包对应着自动机的一个状态

例：LR(0)自动机

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

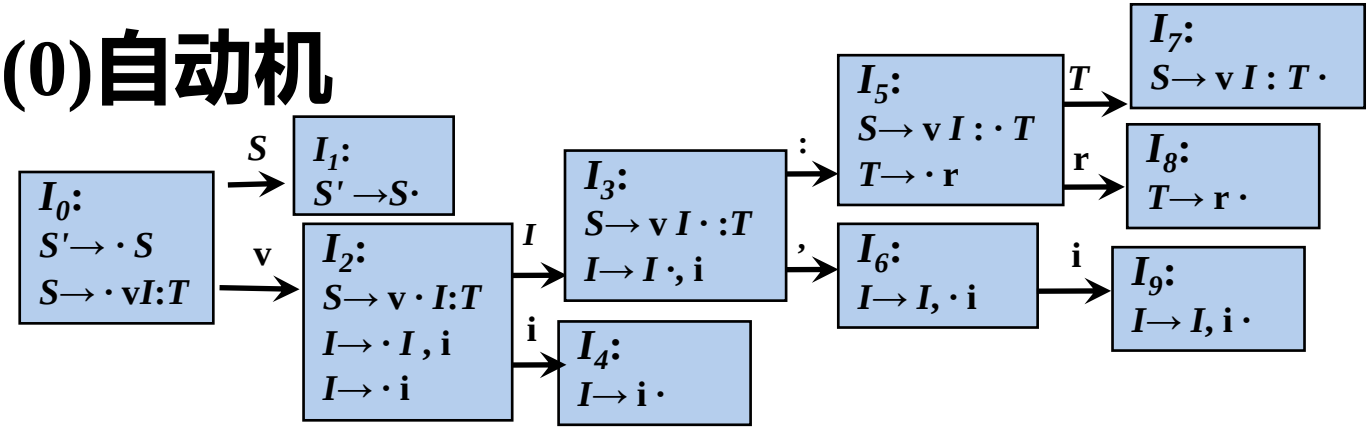






例：LR(0)自动机

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



- 分析栈中的内容有什么特点？
  - 是某一规范句型的前缀

分析栈中内容 + 剩余输入符号 = 规范句型

  - 不能越过规范句型的句柄

规范句型 var <IDS>, i : real的前缀

- var
- var <IDS>
- var <IDS>,
- var <IDS>, i
- var <IDS>, i :
- var <IDS>, i : real

} 不会出现在分析栈中

最左归约

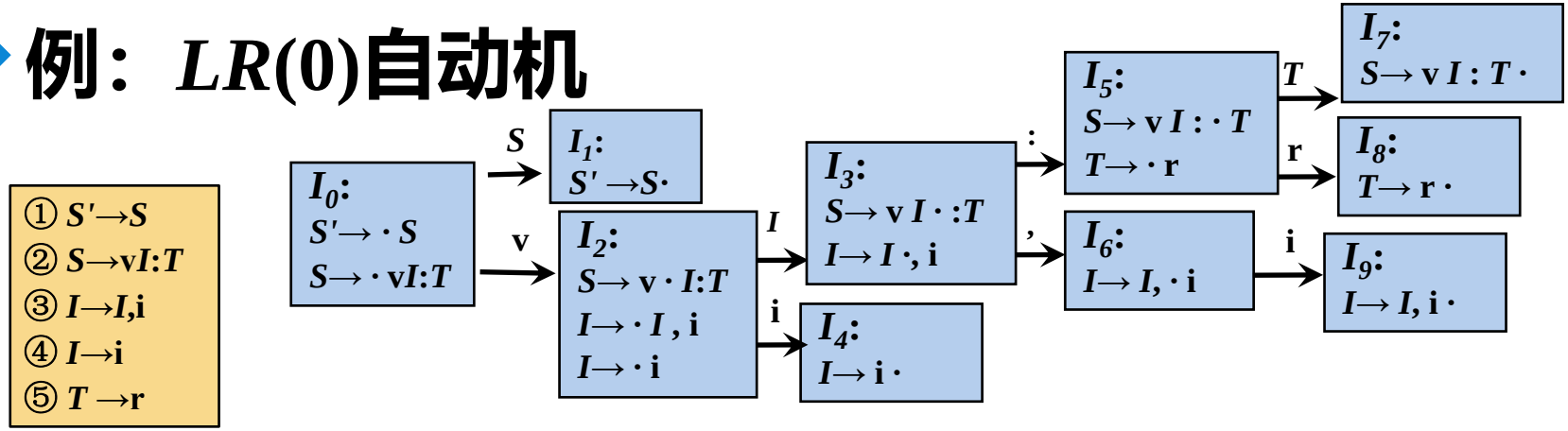
栈	剩余输入	动作
$	var i_A, i_B : real $	
$ var	i_A, i_B : real $	移入
$ var i_A	, i_B : real $	移入
$ var <IDS>	, i_B : real $	归约
$ var <IDS> ,	i_B : real $	移入
$ var <IDS> , i_B	: real $	移入
$ var <IDS>	: real $	归约
$ var <IDS> :	real $	移入
$ var <IDS> : real	$	移入
$ var <IDS> :<T>	$	归约
$ <S>	$	归约

句柄

规范句型

最右推导

例：LR(0)自动机



- 分析栈中的内容有什么特点?
  - 是某一规范句型的前缀

    分析栈中内容 + 剩余输入符号 = 规范句型
  - 不能越过规范句型的句柄

规范句型 var <IDS>, i : real的前缀

- var
- var <IDS>
- var <IDS>,
- var <IDS>, i
- var <IDS>, i :
- var <IDS>, i : real

} 不会出现在分析栈中

规范句型的活前缀（Active Prefix）：

不含句柄右侧任意符号的规范句型的前缀

LR自动机中从初始状态开始的每一条路径对应一个规范句型活前缀

CLOSURE( )函数

- 计算给定项目集I的闭包

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

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;
}
```

$LR(0)$分析表构造方法

- **CLOSURE ({[$S'\rightarrow \cdot S$] })$\rightarrow C$**　　C：自动机状态集合
- **for each $I_i \in C$**
  - *if* $A\rightarrow \alpha\cdot a\beta \in I_i$: 令$I_j=GOTO(I_i, a)$; $ACTION[i, a]=s_j$
  - *if* $A\rightarrow \alpha\cdot B\beta \in I_i$:令$I_j=GOTO(I_i, B)$; $GOTO[i, B]=j$
  - *if* $A\rightarrow \alpha\cdot \in I_i$: { *if* $A=S'$ $(S'\rightarrow S\cdot \in I_i)$ $ACTION[i, \$]=acc$
    if $A\neq S'$　*for* $\forall a \in V_T \cup \{\$\}$ *do* $ACTION[i, a]=r_j$ (j是产生式$A\rightarrow\alpha$的编号)
- 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

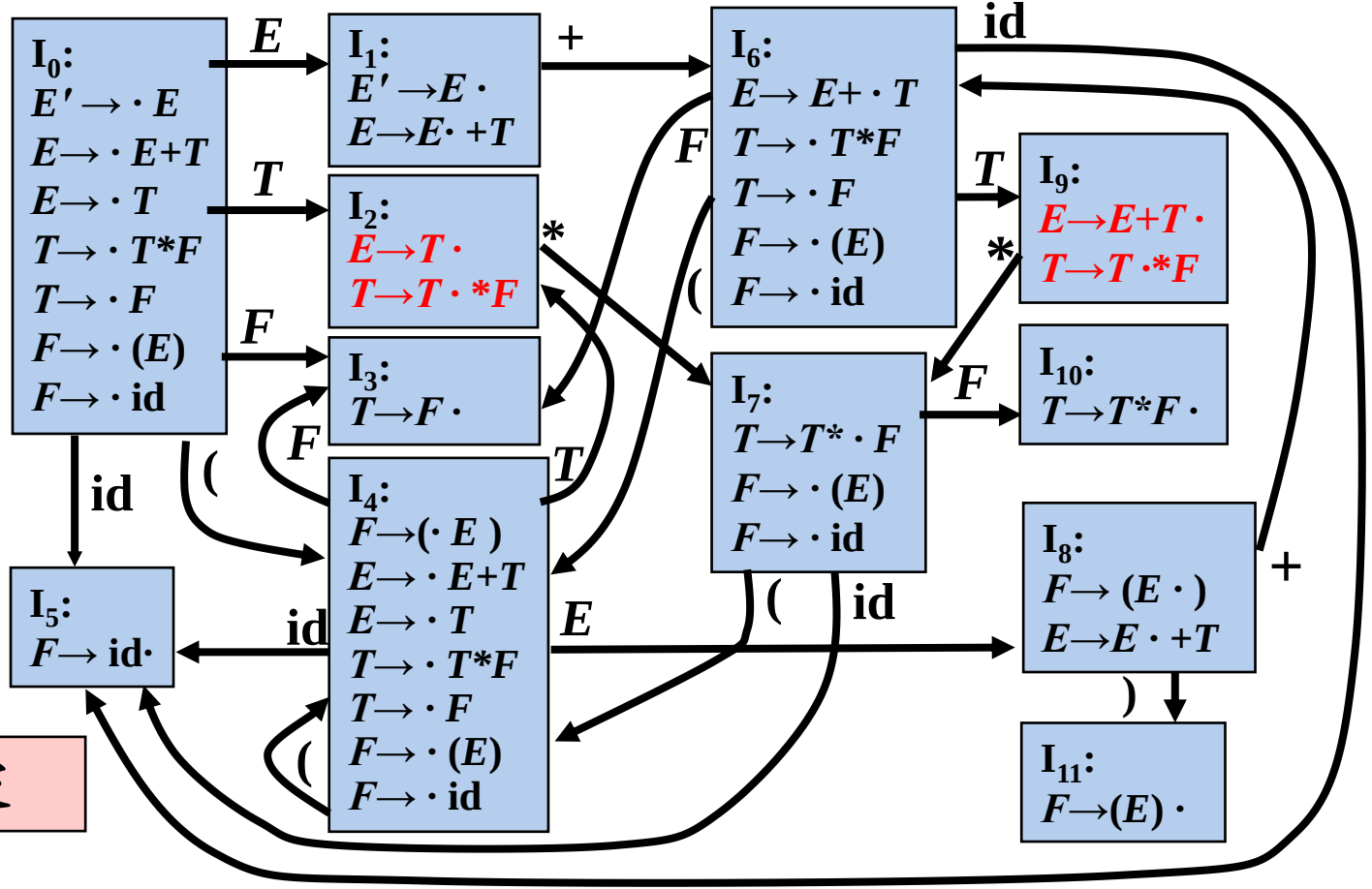
$$M = (C, V_N \cup V_T , GOTO, I_0 , F)$$

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

LR(0) 分析过程中的冲突

文法

(0) $E' \rightarrow E$
(1) $E \rightarrow E+T$
(2) $E \rightarrow T$
(3) $T \rightarrow T*F$
(4) $T \rightarrow F$
(5) $F \rightarrow (E)$
(6) $F \rightarrow$ id



移进/归约冲突

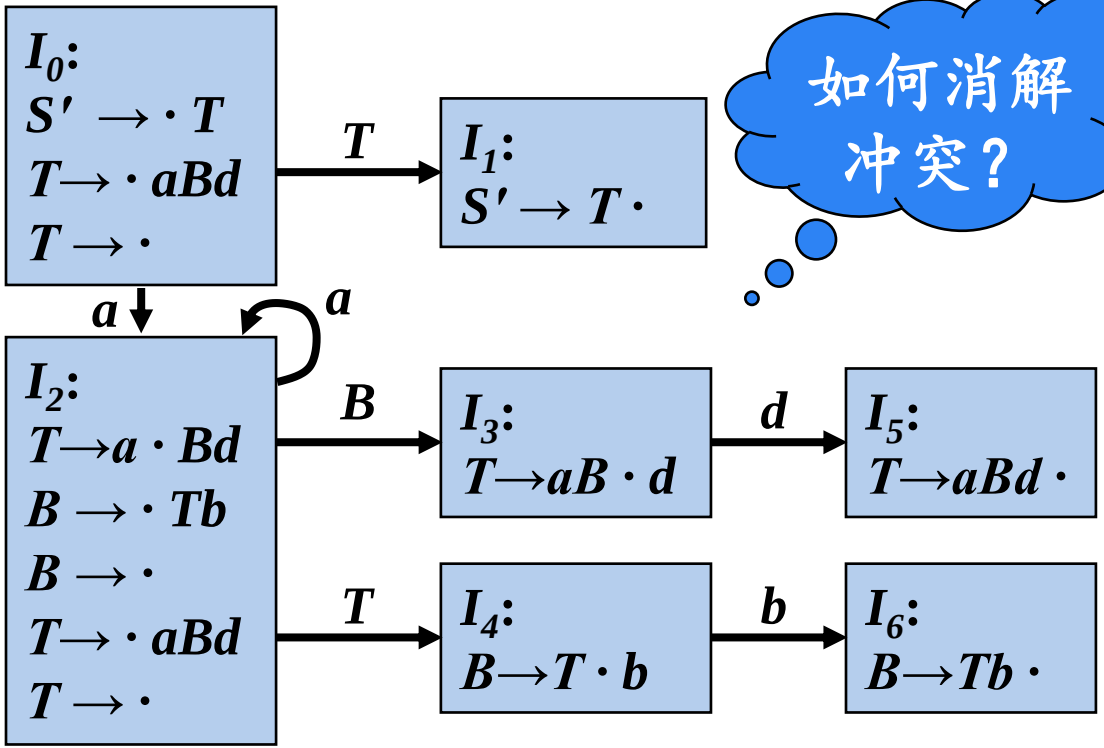
表达式文法的*LR*(0)分析表含有移进/归约冲突

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

文法
(0) $S' \rightarrow T$
(1) $T \rightarrow aBd$
(2) $T \rightarrow \varepsilon$
(3) $B \rightarrow Tb$
(4) $B \rightarrow \varepsilon$

如果LR(0)分析表中没有语法分析动作冲突，那么给定的文法就称为LR(0)文法



不是所有CFG都能用LR(0)方法进行分析，也就是说，CFG不总是LR(0)文法