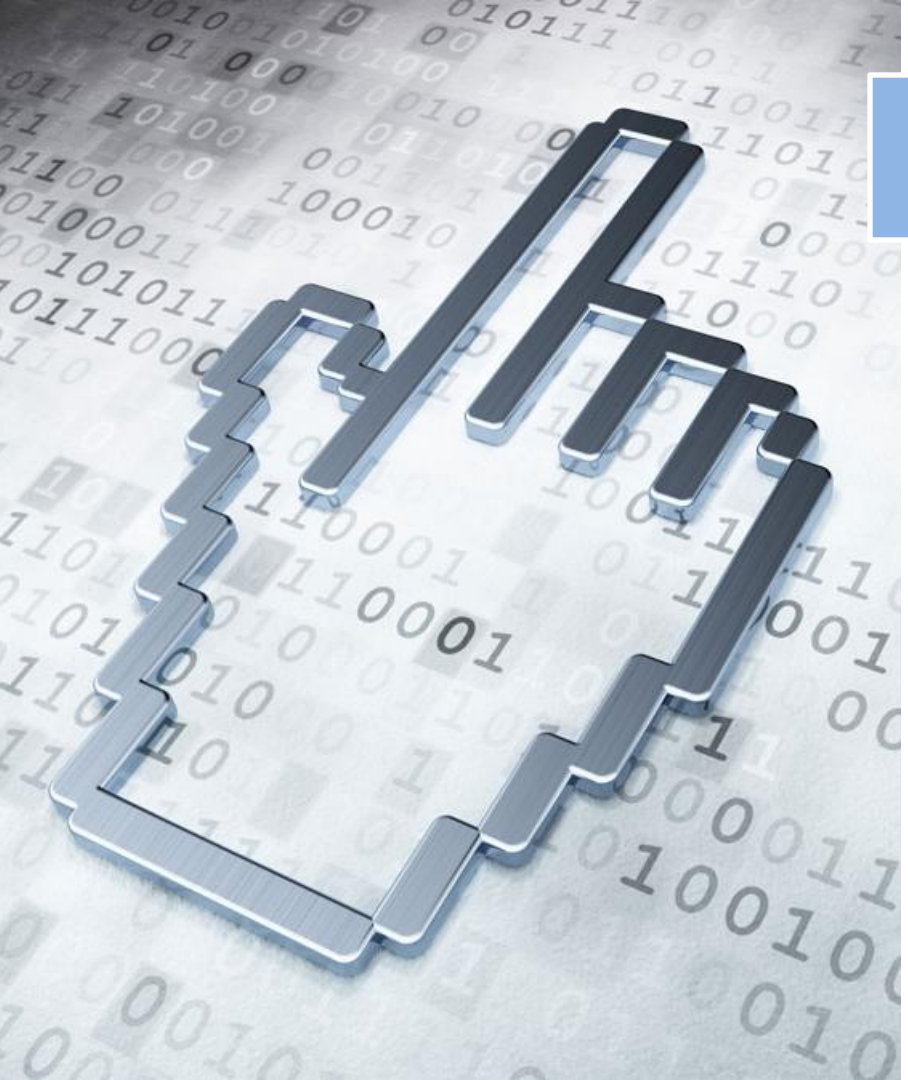


提纲

4.3 自底向上的语法分析

- 任务：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。

- 核心思想：从左到右扫描输入串，反复选择产生式进行最左归约（反向构建最右推导），如果最后能归约出文法的开始符号，则输入串是合法的句子，否则有语法错误。

目标：构建最左归约过程

➢设文法G为：$S \rightarrow aABe \quad A \rightarrow Ab|b \quad B \rightarrow d$

句子分析：　*abbde*

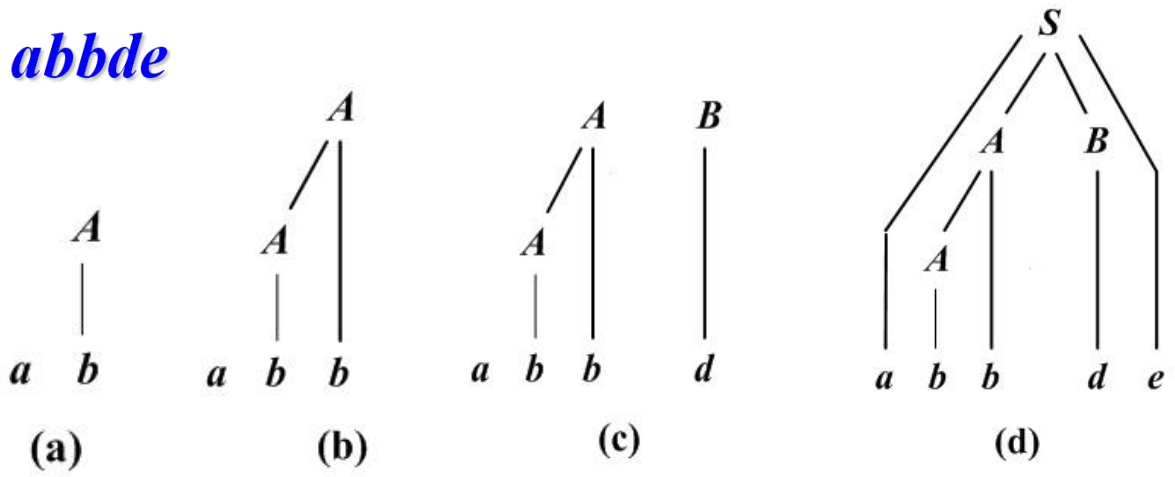
abbde

$\underset{rm}{\Leftarrow}$ *aAbde*

$\underset{rm}{\Leftarrow}$ *aAde*

$\underset{rm}{\Leftarrow}$ *aABe*

$\underset{rm}{\Leftarrow}$ *S*



- 对于非二义性文法，任意右句型的最左归约过程是唯一的
- 最左归约每一步得到的都是最右句型，称为规范句型

最左归约的核心问题：选择哪个子串进行归约？归约为哪个非终结符？

目标：构建最左归约过程

➢设文法G为：$S \rightarrow aABe \quad A \rightarrow Ab|b \quad B \rightarrow d$

句子分析：abbde

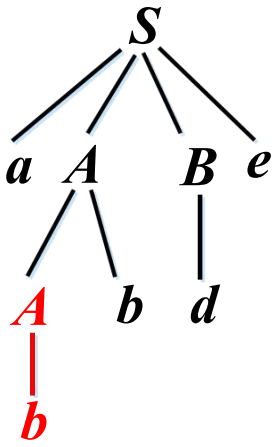
abbde

$\Leftarrow_{rm}$ aAbde

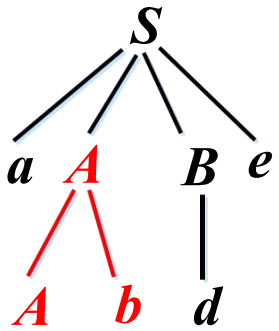
$\Leftarrow_{rm}$ aAde

$\Leftarrow_{rm}$ aABe

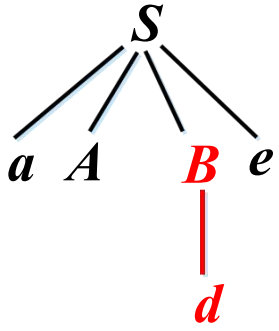
$\Leftarrow_{rm}$ S



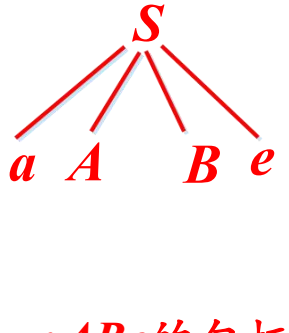
abbde的句柄



aAbde的句柄



aAde的句柄



aABe的句柄

➢ 规范句型每步最左归约选择归约的子串都是该规范句型对应分析树的最左直接短语(只有父子两代子树的边缘)，称为规范句型的句柄

句柄的定义

- 最左归约每步得到的都是规范句型（最右句型）

  $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha \beta w$，称$\beta$是规范句型$\alpha\beta w$的句柄。

- 从句子开始，每步都选择句柄进行归约得到的总是规范句型
- 规范句型句柄右侧的串w一定是只包含终结符号。

如何在规范句型中找到句柄呢？

自底向上分析的通用框架——移入-归约分析

$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha\beta w$

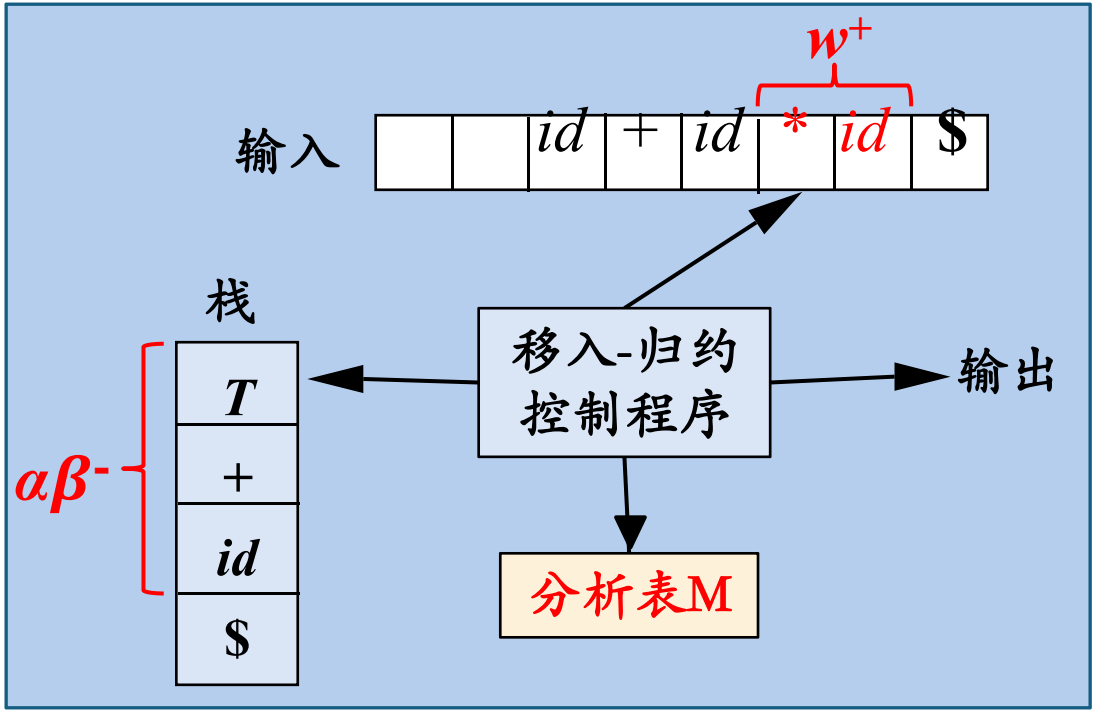
- 初始格局：

栈：$

输入缓冲区：w$

- 分析过程：

从左到右扫描输入串，根据分析表将输入符号压入栈，一旦句柄在栈顶形成，再根据分析表将其归约为对应的产生式头。重复这个过程，直到分析结束。



任何时刻：栈内容 + 剩余输入串 = 规范句型

例：移入-归约分析

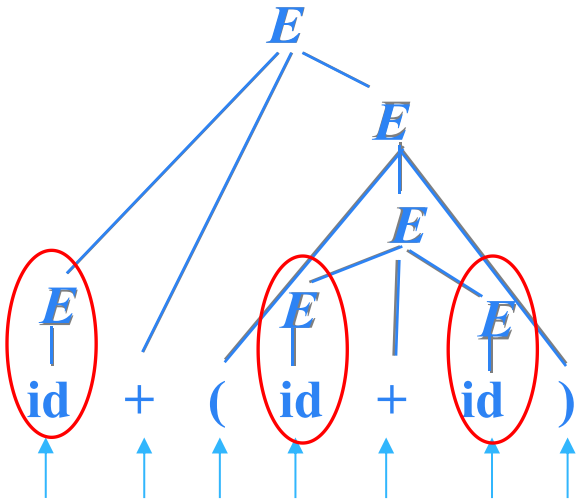
文法

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

自底向上分析的通用分析器——移入-归约分析

➢ 分析器的四种动作

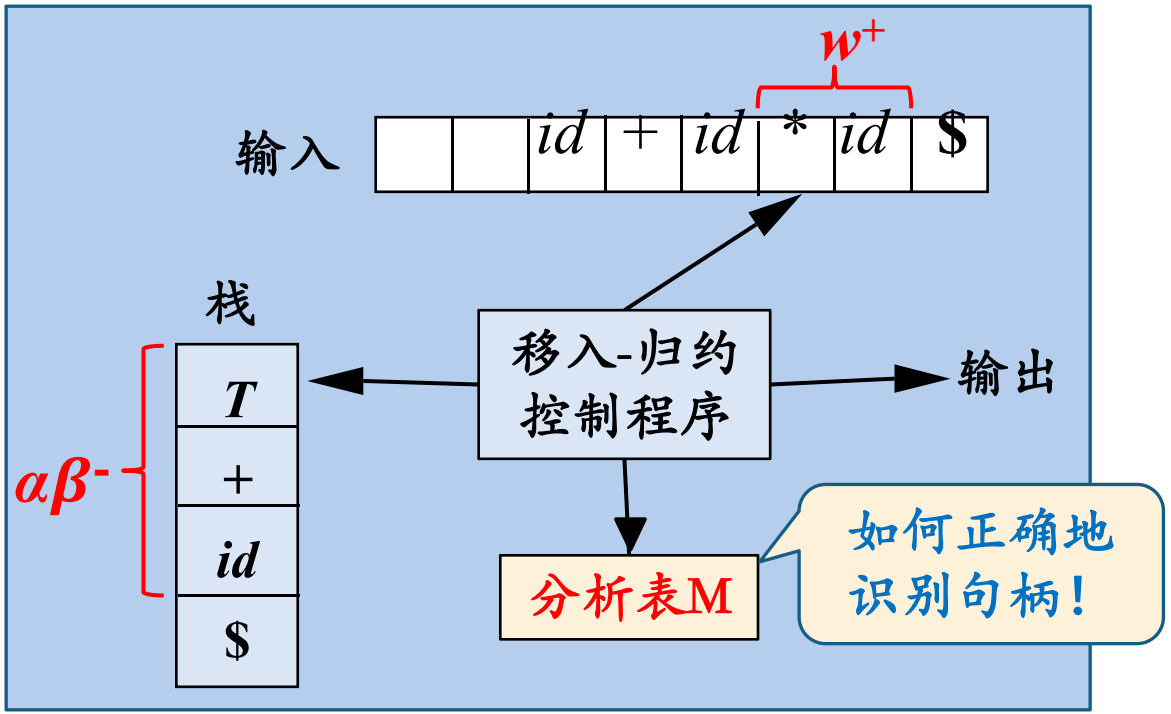
1) 移入：将当前输入符号移入栈

2) 归约：将栈顶的句柄（某产生式右部）替换为对应产生式的头部。

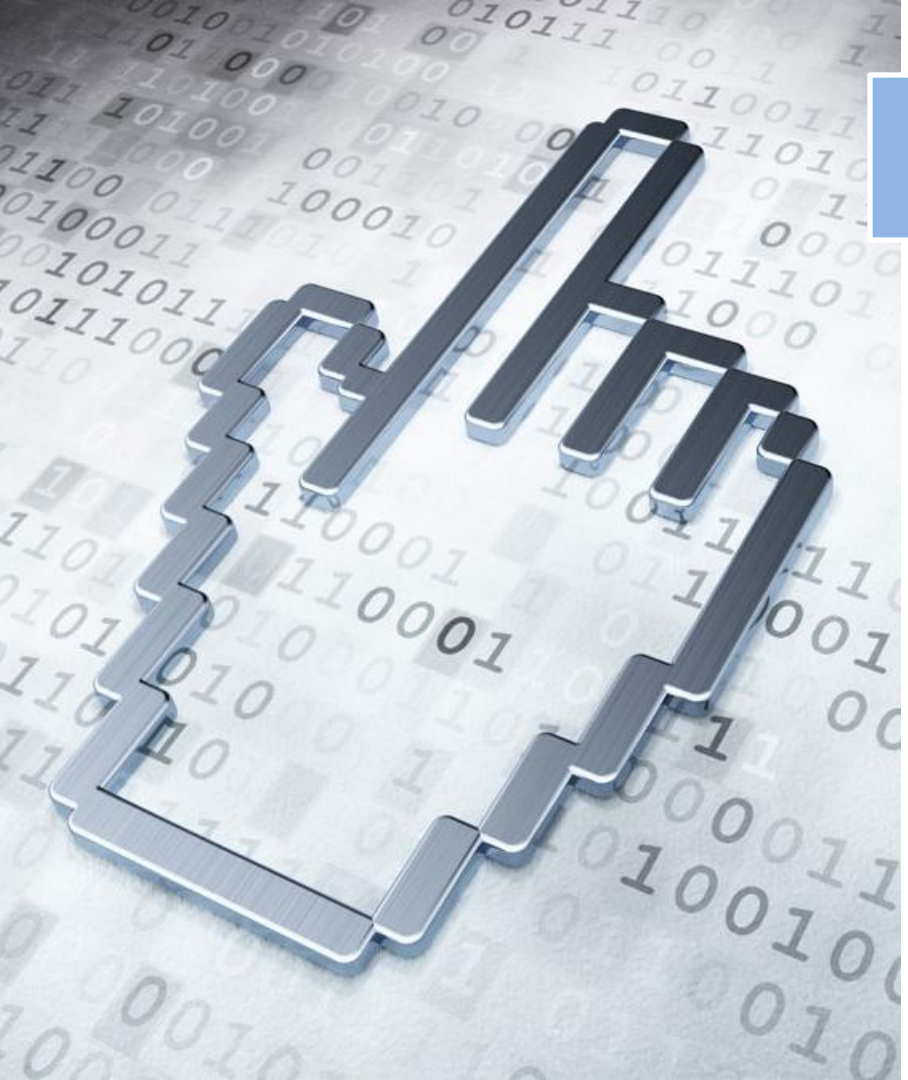
3) 接受：分析成功。

格局：S

4) 出错：出错处理。表项无定义。



分析表：指导分析器何时进行移入，何时进行归约，用哪个产生式进行归约。



提纲

4.4 *LR* 分析法（状态法）

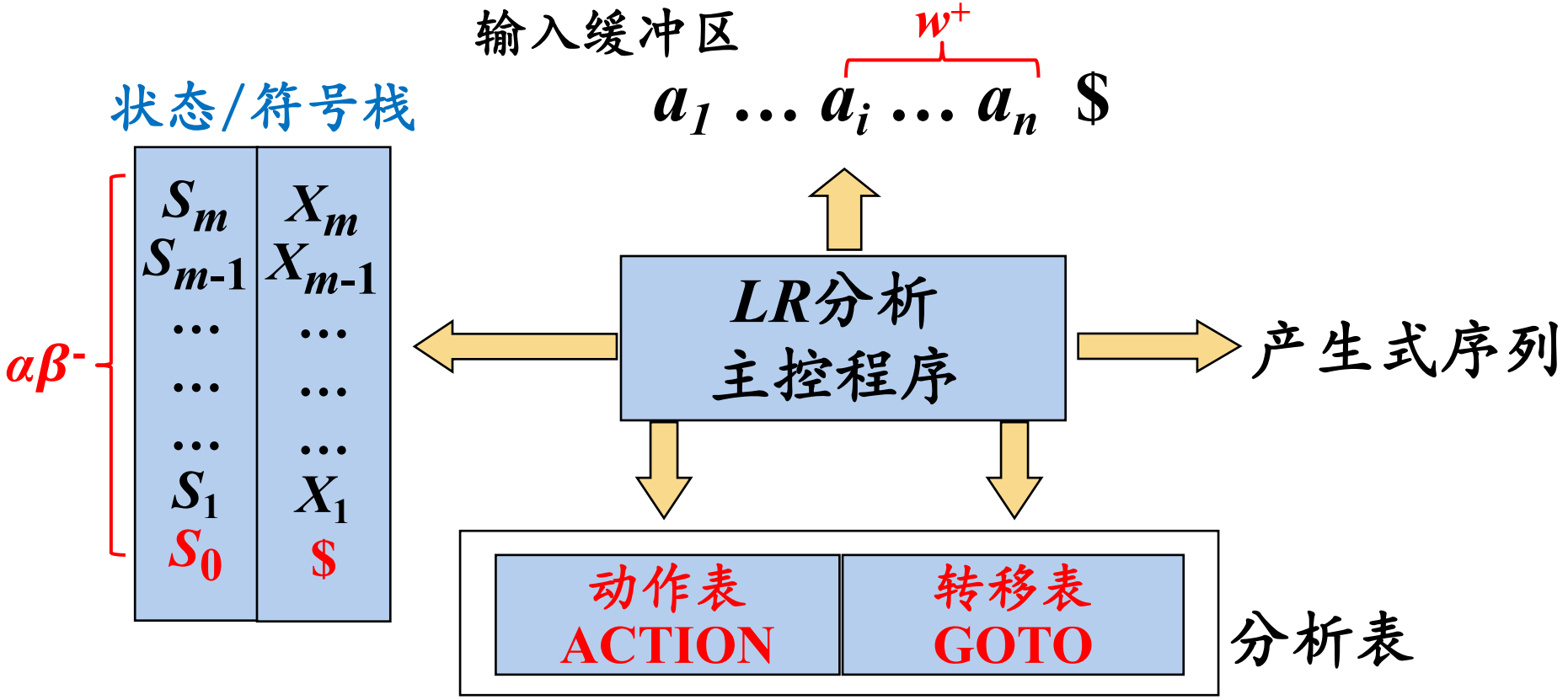
- *LR*分析技术是美国计算机科学家高德纳(Donald Knuth)于1965年首先提出来的。
- 优点：适用范围广；分析速度快；报错准确。
  - *LR*分析方法是已知的最通用的无回溯的移进—归约分析方法；
  - 几乎所有描述程序设计语言的上下文无关文法，都能够构造*LR*语法分析器；
  - 可以使用*LR*方法进行分析的文法类是可以使用*LL*方法进行语法分析的文法类的真超集。

4.4 *LR* 分析法

- *LR*(k)分析法可分析*LR*(k)文法产生的语言
  - *L*：从左到右扫描输入符号
  - *R*：反向构造最右推导（最左归约）
  - *k*：超前读入k个符号，以便确定分析器的动作。
    - $k=0$ 和 $k=1$ 这两种情况具有实践意义
    - 当省略(*k*)时，表示$k=1$

LR 分析器的总体结构

$S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A w \underset{rm}{\Rightarrow} \alpha \beta w$



LR 分析表的结构

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

- 不同的LR分析法，表项的构建方法不同，但结构相同，分析过程相同

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：移入(shift)，即将当前输入符号和状态*n* 压入栈

rn：归约(reduce)，即用第*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		

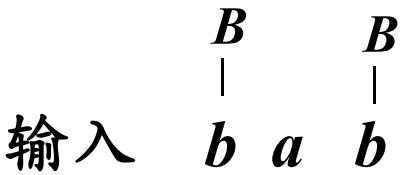
栈	剩余输入
0 4	
$ *B*	*b a b* $

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		



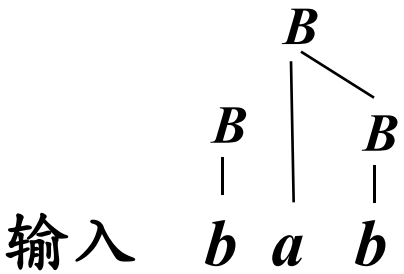
栈　　剩余输入

0 2 3 4　　*a b* $

$ *B*

LR 分析表的结构

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

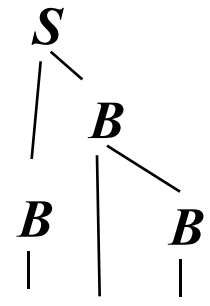


状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

栈	剩余输入
0 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 $B \rightarrow b$

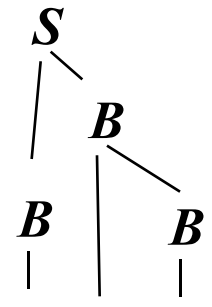


状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

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 $B \rightarrow b$



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 1		
$ *S*	$	分析成功!

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	
\$	$a_1a_2\ldots a_n$ \$

➢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，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 ) {
        将状态t压入栈中；// 不需要压入符号
        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;
}
```

- 归约过程中不关心这个符号是什么。
- 每个状态与唯一的一个终结符号对应。因此，可以省略符号栈。

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

- 句柄是逐步形成的，用“项目”表示句柄识别的进展程度
- 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：$S \rightarrow bBB$

- $S \rightarrow \cdot bBB$ ⟵ 移入项目
- $S \rightarrow b \cdot BB$
- $S \rightarrow bB \cdot B$ } 待约项目
- $S \rightarrow bBB \cdot$ ⟵ 归约项目

项目描述了句柄识别的进展状态

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LR(0)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 $T\rightarrow \mathrm{r}$

初始项目

(0) $S'\rightarrow \cdot S$　(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) $S\rightarrow \mathrm{v}I:\cdot T$　(7) $I\rightarrow \cdot I,\mathrm{i}$　(8) $I\rightarrow I\cdot ,\mathrm{i}$　(9) $I\rightarrow I,\cdot \mathrm{i}$　(11) $I\rightarrow \cdot \mathrm{i}$　(13) $T\rightarrow \cdot \mathrm{r}$

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

接收项目　归约项目

- 移进项目：形如$A\rightarrow \alpha\cdot a\beta$，其中$a\in V_T$分析时，需要把$a$移进栈
- 待约项目：形如$A\rightarrow \alpha\cdot B\beta$，其中$B\in V_N$，等待归约出$B$
- 归约项目：形如$A\rightarrow \alpha\cdot$句柄已形成，可以归约

文法中的LR(0)项目

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

(2) $S\to \cdot vI{:}T$

(3) $S\to v\cdot I{:}T$ (7) $I\to \cdot I,i$

(4) $S\to vI\cdot{:}T$ (8) $I\to I\cdot,i$

(0) $S'\to \cdot S$ (5) $S\to vI{:}\cdot T$ (9) $I\to I,\cdot i$ (11) $I\to \cdot i$ (13) $T\to \cdot r$

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

- 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
  - $A\to\alpha\cdot X\beta$的后继项目是$A\to\alpha X\cdot\beta$

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

使用LR(0)项目可以构造LR(0)自动机，该自动机可以识别可能出现在栈中的所有串，可用于做出语法分析决定。

例：LR(0)自动机

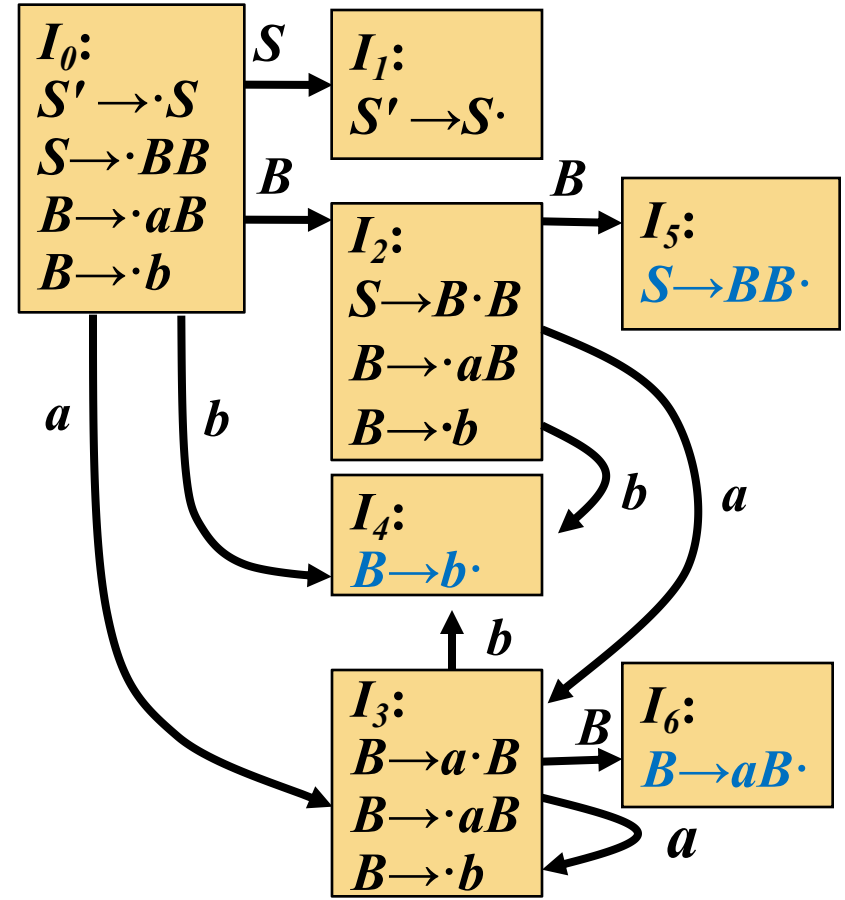
文法

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$



核心项目：S‘ →·S和所有点不在最左端的项目。

非核心项目：除了S‘ →·S之外的点在最左端的项目。

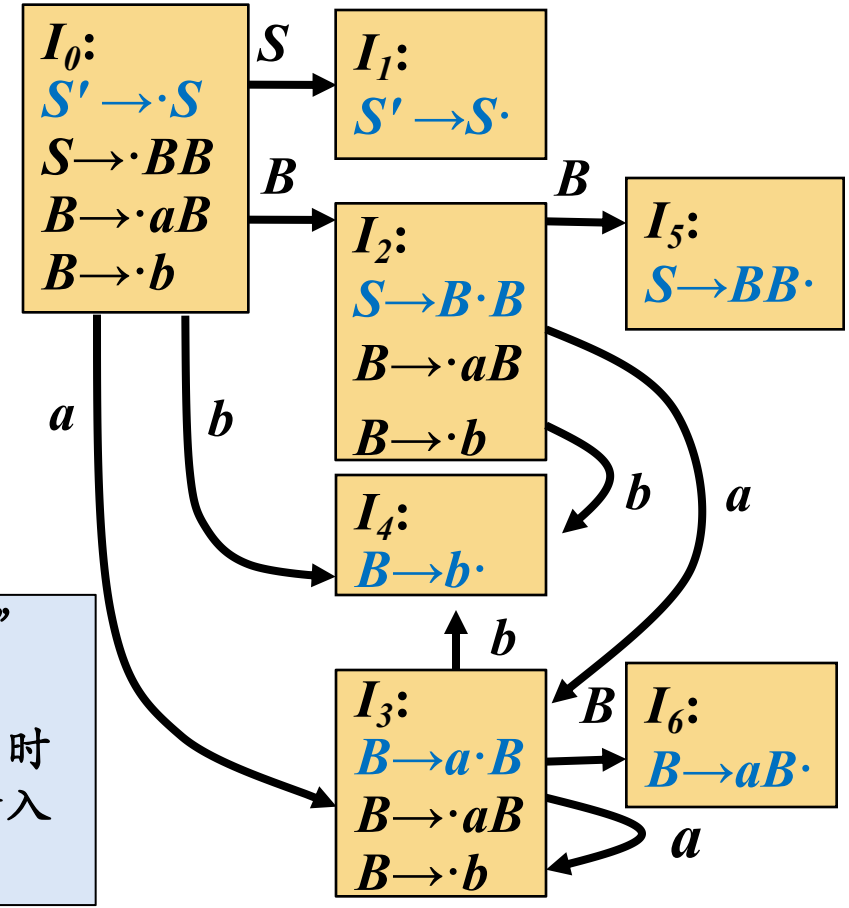
项集的闭包：LR(0)自动机每个状态都是核心项目集合的闭包。

LR(0)项集规范族：

所有规范LR(0)项集的集合

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)中的“0”表示：
执行归约动作时，不需查看输入符号。

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		

例：LR(0)自动机

该自动机识别所有可能出现在分析栈中的符号串，称为规范句型的活前缀。

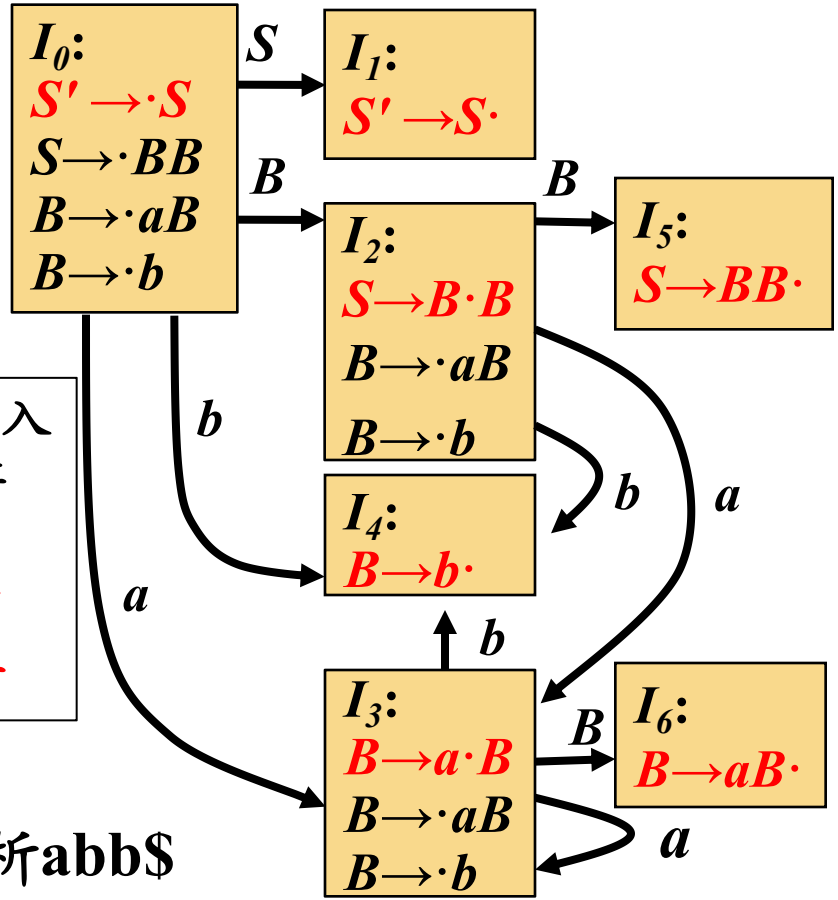
文法

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$



- 某状态所有入边上标记的符号是相同的。即核心项目圆点左侧的符号

例如：分析abb$

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		

分析栈中的内容有什么特点？*

- 是某一规范句型的前缀
- 且不能越过句柄的右端
  - 例： 句型 var <IDS>, i : real
    - 前缀

句柄

- ε
- var ⟺ <IDS>→ . <IDS> , i
- var <IDS> ⟺ <IDS>→ <IDS> . , i
- var <IDS>, ⟺ <IDS>→ <IDS> , . i
- var <IDS>, i ⟺ <IDS>→ <IDS> , i .
- var <IDS>, i :
- var <IDS>, i : real

活前缀

不会出现在分析栈中

语法：

(1) <S>→var <IDS> : <TYPE>

(2) <IDS>→<IDS> , i

(3) <IDS>→i

(4) <TYPE>→real|int|char|bool

对句柄的识别就相当于规范句型活前缀的识别

若移入当前符号不再构成活前缀时，就是形成了句柄，应该归约了，否则出错。

活前缀(*Active Prefix*)*

- 规范句型的活前缀（可行前缀）
  - 不含句柄右侧任意符号的规范句型的前缀

若$S' \underset{rm*}{\Longrightarrow} \alpha Aw \underset{rm}{\Rightarrow} \alpha\beta w$，是文法$G$的拓广文法$G'$的一个规范推导，$\beta$是规范句型$\alpha\beta w$的句柄；符号串$\gamma$是$\alpha\beta$的前缀，则称$\gamma$是文法$G$的一个活前缀。

活前缀是可以出现在分析栈中的符号串

例：LR(0)自动机

任一时刻分析栈中的符号串，一定是从初始状态到栈顶状态的路径上对应的符号串。

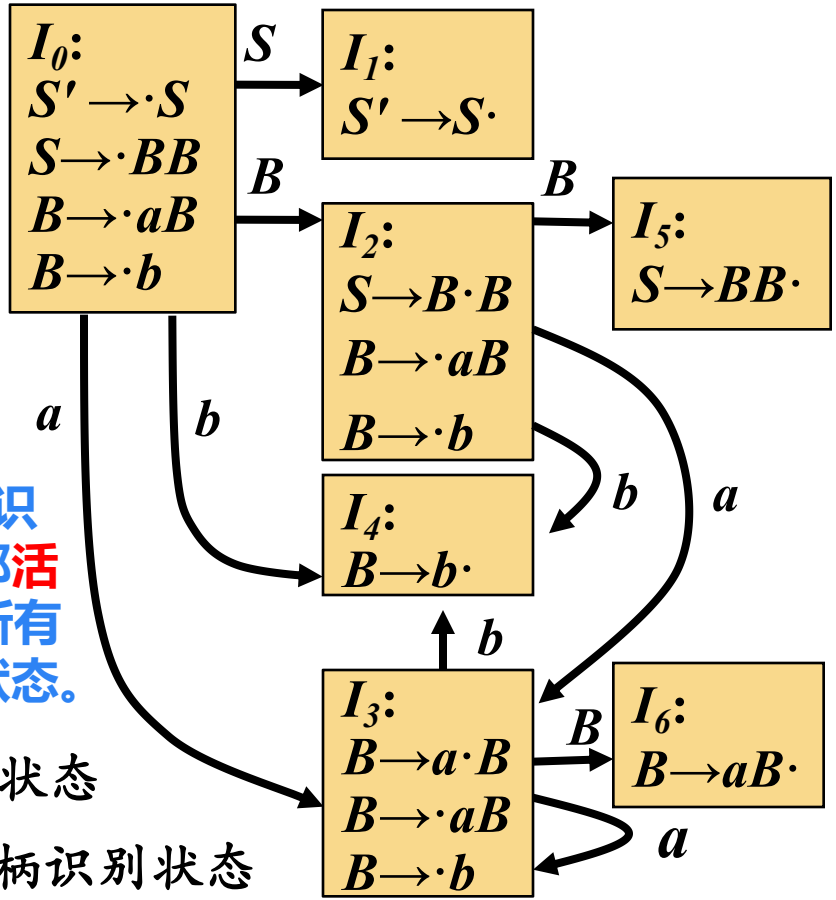
增广文法

(0) $S' \to S$

(1) $S \to BB$

(2) $B \to aB$

(3) $B \to b$



基于LR(0)项目识别增广文法全部活前缀的DFA。所有状态都是接收状态。

I_1是句子识别状态

I_1 I_4 I_5 I_6是句柄识别状态

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		

活前缀与句柄的关系*

- 设$A \rightarrow \beta$是句柄，栈中的规范句型活前缀与句柄的关系：
  - 不含句柄的任何符号：$A \rightarrow .\beta$
  - 包含句柄的部分符号：$A \rightarrow \beta_1 . \beta_2$
  - 包含句柄的全部符号：$A \rightarrow \beta.$

活前缀与句柄的关系*

- 同一活前缀可能对应不同规范句型的不同句柄的识别进度

例：对于活前缀 $E+T*$ 下面三个最右推导

$$E' \underset{rm}{\Rightarrow} E \underset{rm}{\Rightarrow} E+T \underset{rm}{\Rightarrow} E+T*F$$

$T \to T*.F$

$$E' \underset{rm}{\Rightarrow} E \underset{rm}{\Rightarrow} E+T \underset{rm}{\Rightarrow} E+T*F \underset{rm}{\Rightarrow} E+T*(E)$$

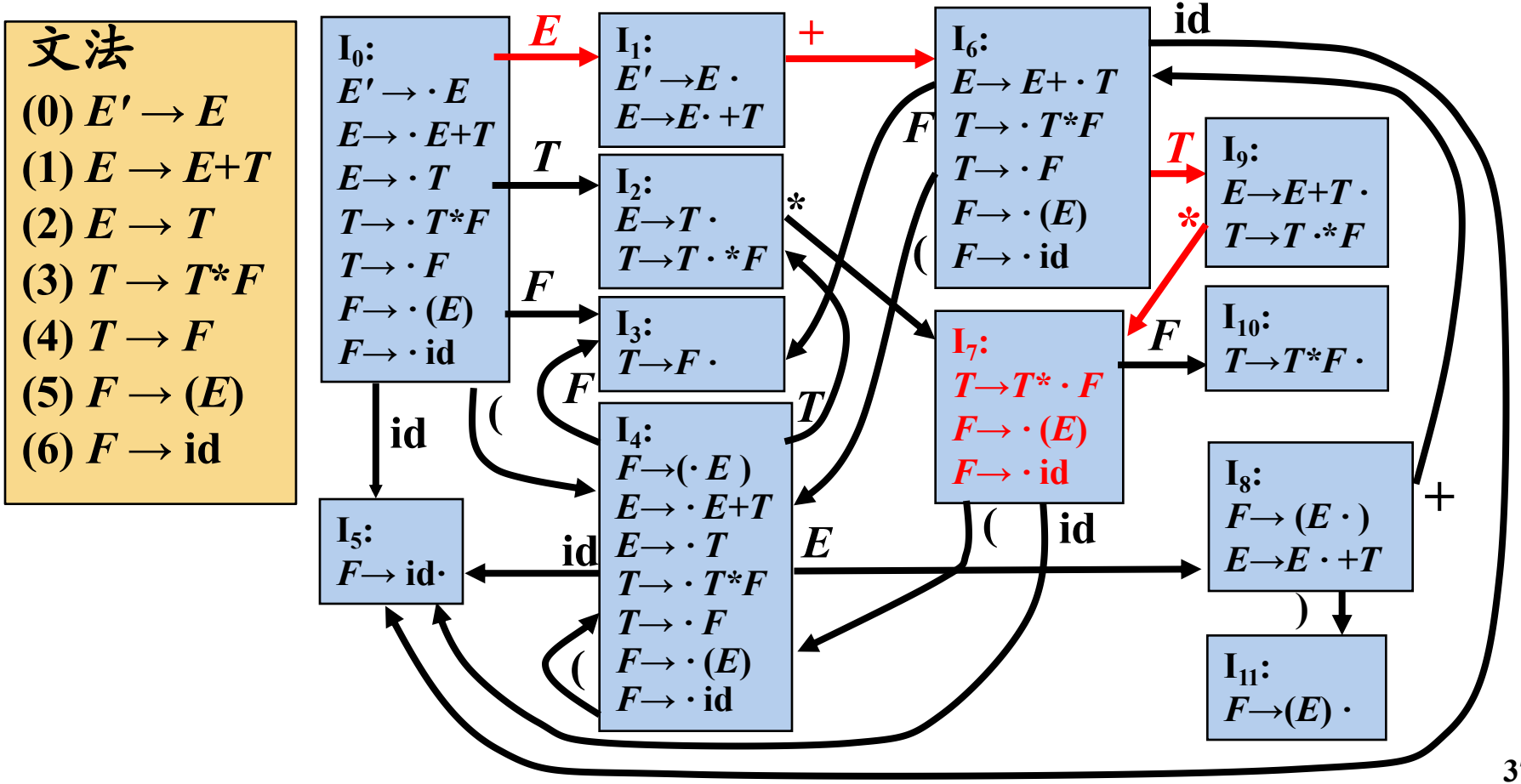
$F \to .(E)$

$$E' \underset{rm}{\Rightarrow} E \underset{rm}{\Rightarrow} E+T \underset{rm}{\Rightarrow} E+T*F \underset{rm}{\Rightarrow} E+T*id$$

$F \to .id$

称为活前缀的有效项

LR(0)自动机的状态是活前缀的有效项集*



规范句型活前缀的有效项*

➢ 有效项：

$$S \underset{rm*}{\Longrightarrow} \alpha A w \underset{rm}{\Rightarrow} \alpha \beta_1 \beta_2 w$$

则称，$LR(0)$ 项 $A \to \beta_1 \cdot \beta_2$ 对于活前缀 $\alpha\beta_1$ 是有效的。

- 若 $A \to \beta_1 \cdot$ 对活前缀 $\alpha\beta_1$ 是有效的，则它告诉我们，句柄已经形成，应把符号串 β_1 归约为A。
- 若 $A \to \beta_1 \cdot \beta_2$ 且 $\beta_2 \neq \varepsilon$，则它告诉我们句柄尚未形成，应该是移进。

找到当前活前缀的有效项，就能知道识别句柄的进度，就能知道是不是可以归约了!

38

LR(0)自动机的状态是活前缀的有效项集*

LR(0)自动机的状态是已识别的规范句型活前缀的有效项的集合

- LR语法分析理论的核心定理：

  从LR(0)自动机的初始状态沿某条路径到达的状态是这条路径识别的规范句型活前缀的有效项集。

39

构造LR分析表的算法

CLOSURE闭包函数和GOTO函数

- 计算项集 I 闭包：CLOSURE闭包函数

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

- 状态转移：GOTO函数

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

  - $A \rightarrow \alpha X.\beta$ 称为 $A \rightarrow \alpha.X\beta$ 的后继项目
  - GOTO(I, X)称为项集I关于X的后继项目集。

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({$A \rightarrow \alpha X \cdot \beta \mid \forall A \rightarrow \alpha \cdot X\beta \in I$ })

```
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 $A\rightarrow\alpha\cdot a\beta\in I_i$ and $GOTO(I_i, a)=I_j$ then $ACTION[i, a]=sj$
  - if $A\rightarrow\alpha.B\beta\in I_i$ and $GOTO(I_i, B)=I_j$ then $GOTO[i, B]=j$
  - if $A\rightarrow\alpha\cdot\in I_i$ 且$A\neq S'$ then for $\forall a\in V_T\cup\{\$\}$ do $ACTION[i, a]=rj$
    （j是产生式$A\rightarrow\alpha$的编号）
  - if $S'\rightarrow S\cdot\in I_i$ 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

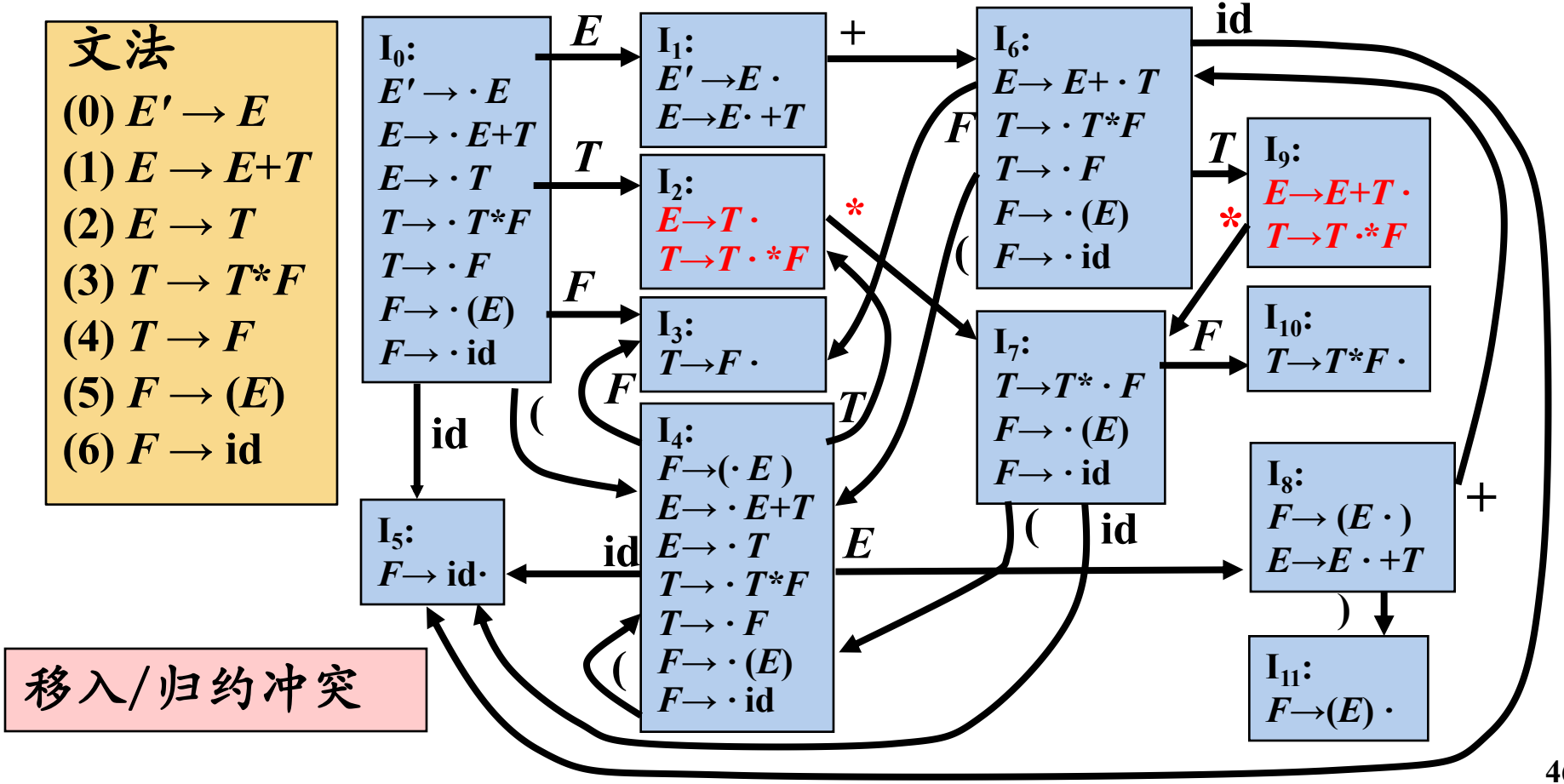
$$M=(C, V_N \cup V_T, GOTO, I_0, F)$$

- $C=\{I_0\} \cup \{I \mid \exists J \in C, X \in V_N \cup V_T, I=GOTO(J, X)\}$
- $I_0=CLOSURE(\{S' \rightarrow .S\})$
- $F=C$ (活前缀的识别状态)

语法分析句子的识别状态：
F ={包含项S′ →S·的状态}

思考：哪些状态是句柄的识别状态?

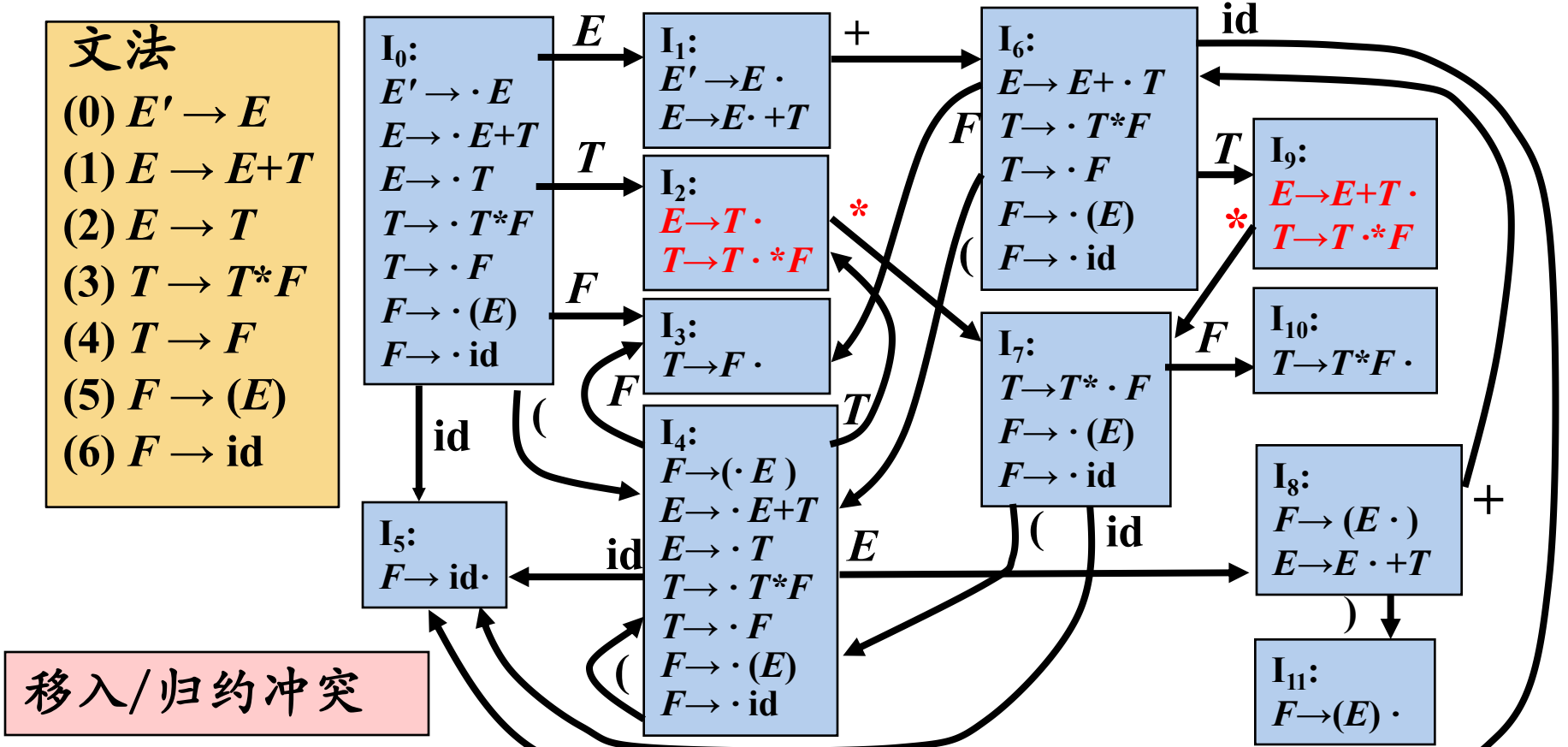
LR(0) 分析过程中的冲突



表达式文法的$LR(0)$分析表含有移入/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			

LR(0) 分析过程中的冲突

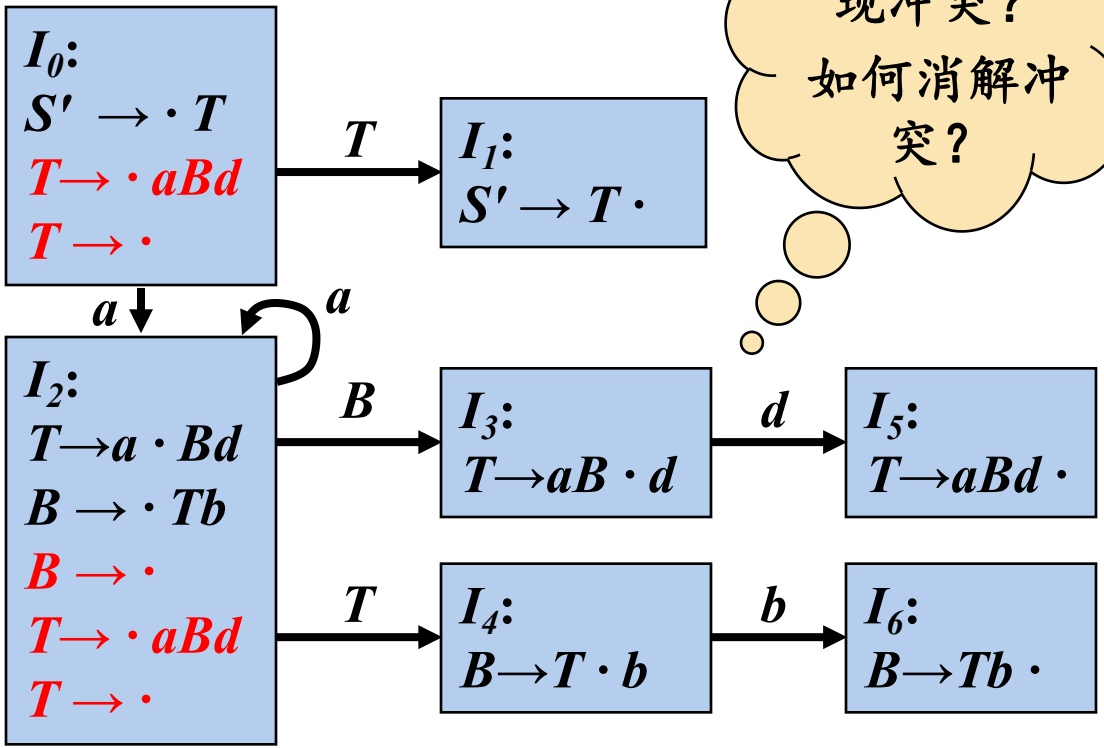


例：移入/归约冲突和归约/归约冲突

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



为什么会出现冲突？

如何消解冲突？

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

课后练习

- 画出文法的LR(0)自动机，并画出LR(0)分析表，说明是不是LR(0)文法，为什么？

$$S \rightarrow a \mid S + S \mid S\ S \mid S * \mid (S)$$

50