第六章：自底向上LR语法分析方法

提纲

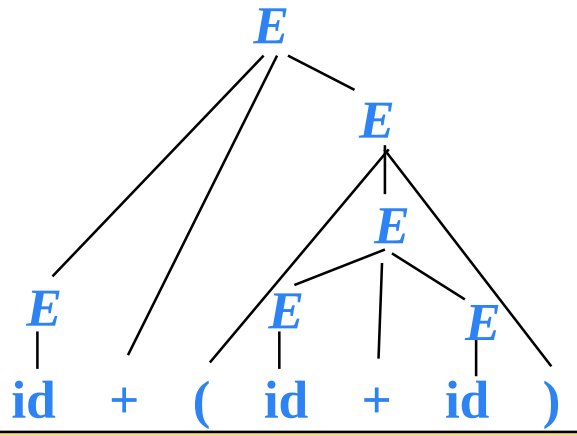
自底向上的语法分析

- 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

栈内符号串 + 剩余输入 = “规范句型”

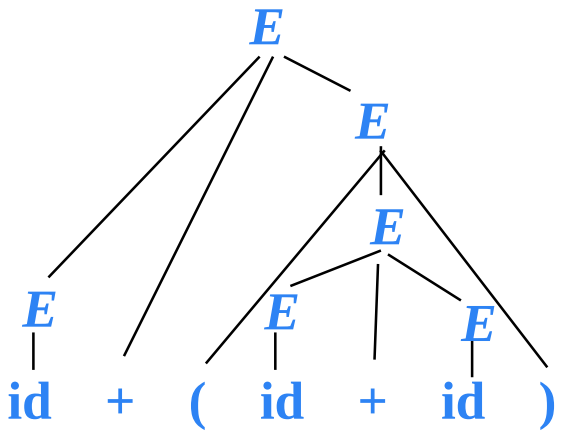
文法

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(当出现这种情况时，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：

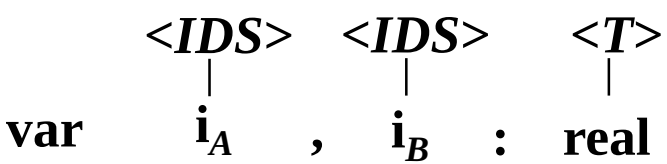
(1) <*S*>→var <*IDS*> : <*T*>

(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

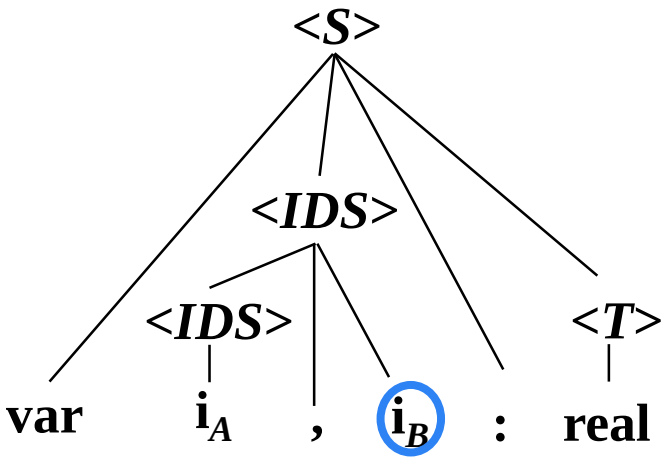
例：

(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

造成错误的原因：
错误地识别了句柄

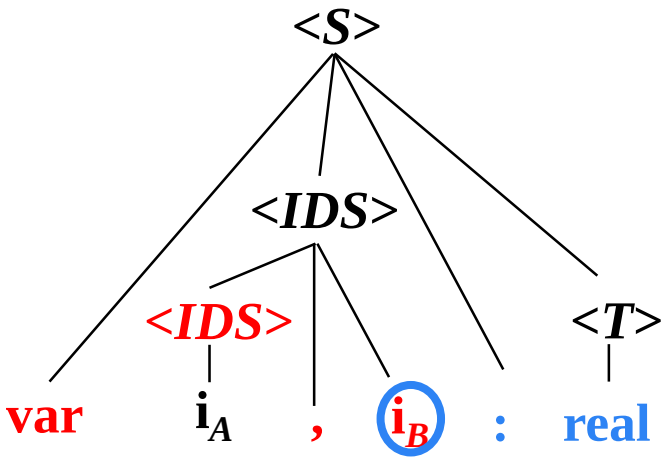
例：

(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*>		
$ 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

exercise

1在自顶向下的语法分析方法中，分析的关键是(D)。

A.寻找句柄

B.寻找句型

C.消除递归

D.选择候选式

exercise

2在自底向上的语法分析方法中，分析的关键是(A)。

A.寻找句柄

B.寻找句型

C.消除递归

D.选择候选式

exercise

3一个句型中的(A)称为该句型的句柄。

A.最左直接短语

B.最右直接短语

C.终结符

D.非终结符

exercise

4在规范归约中，用(B)来刻画可归约串。

A.直接短语

B.句柄

C.最左素短语

D.素短语

exercise

5下列动作中，不是自下而上分析动作的是(B)。

A.移进

B.展开

C.接受

D.报错

exercise

6下列动作中，不是自上而下分析动作的是(C)。

A.匹配

B.展开

C.移进

D.报错

exercise

7设有文法G[T]:

T→T*F|F

F→F↑P|P

P→(T)|a

该文法句型T*P↑(T*F)的句柄是下列符号串(C);

该文法句型T*F↑(T*F)的句柄是下列符号串(B)。

A.（T*F）

B.T*F

C.P

D.P↑(T*F)

提纲

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$k=1$

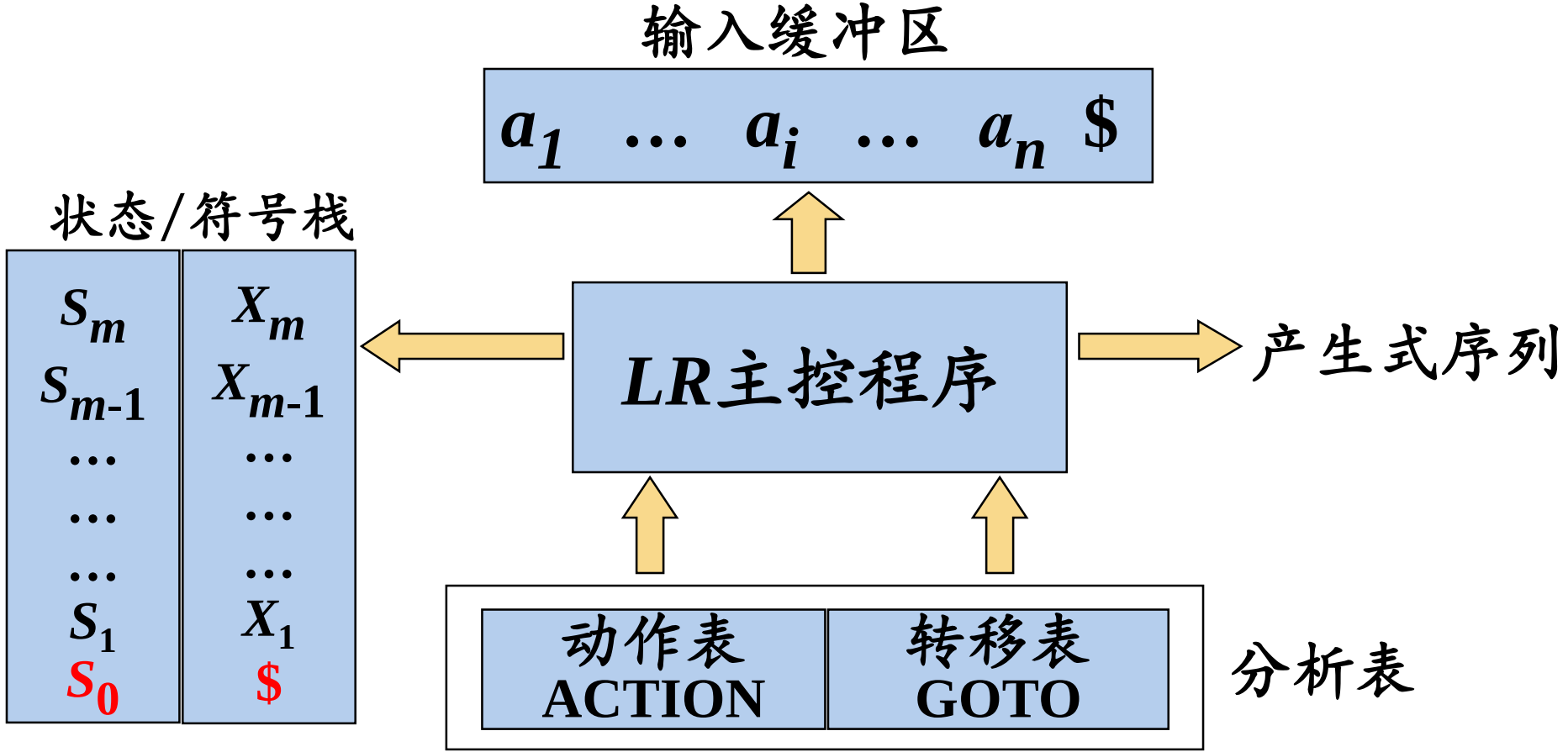
LR 分析法的基本原理

- 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



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* $

B

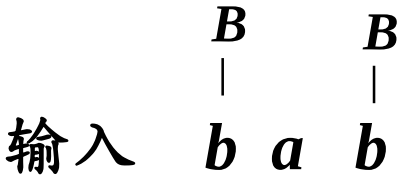
|

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		



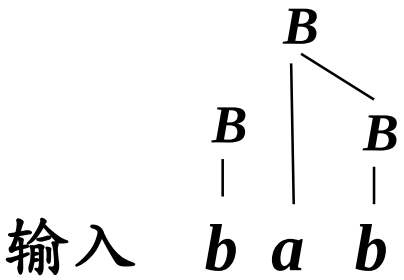
栈　　剩余输入

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 $B \rightarrow b$

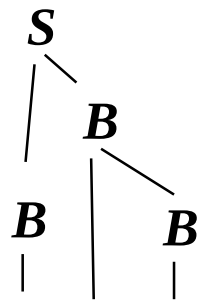


状态	ACTION			GOTO	
	a	*b*	$	*S*	*B*
0	s3	s4		1	2
1			acc		
2	s3	s4			5
3	s3	s4			6
4	r3	r3	r3		
5	r1	r1	r1		
6	r2	r2	r2		

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 $B \rightarrow b$



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*	$

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，那么当前形式变为：

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

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，那么形式变为：

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

如果GOTO[s_{m-k}, A]=y，那么形式变为：

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

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

exercise

1若a为终结符，则A→α•aβ为(B)项目。

A.归约

B.移进

C.接受

D.待约

exercise

2若B为非终结符，则 A→a•Bb 为(D)。

A.移进项目

B.归约项目

C.接受项目

D.待约项目

exercise

3若B为非终结符，则A→α• 为(A)项目。

A.归约

B.移进

C.接受

D.待约

exercise

4 LR分析器的核心部分是一张分析表，该表由(D)组成。

A.ACTION表

B.GOTO表

C.预测分析表

D.ACTION表和GOTO表

exercise

5 LR分析表中的动作表（action）是以（D）作为列标题的。

A.终结符

B.非终结符

C.终结符或非终结符

D.终结符和结束符$

exercise

6 LR分析表中的转移表（goto）是以（B）作为列标题的。

A.终结符

B.非终结符

C.终结符或非终结符

D.表示状态的整型数

提纲

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$A \rightarrow \cdot$

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) $F \rightarrow \mathbf{id}$

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) $F \rightarrow \mathbf{id}$

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

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 $T\rightarrow \mathrm{r}$

(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) $T\rightarrow \mathrm{r}\cdot$

- 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

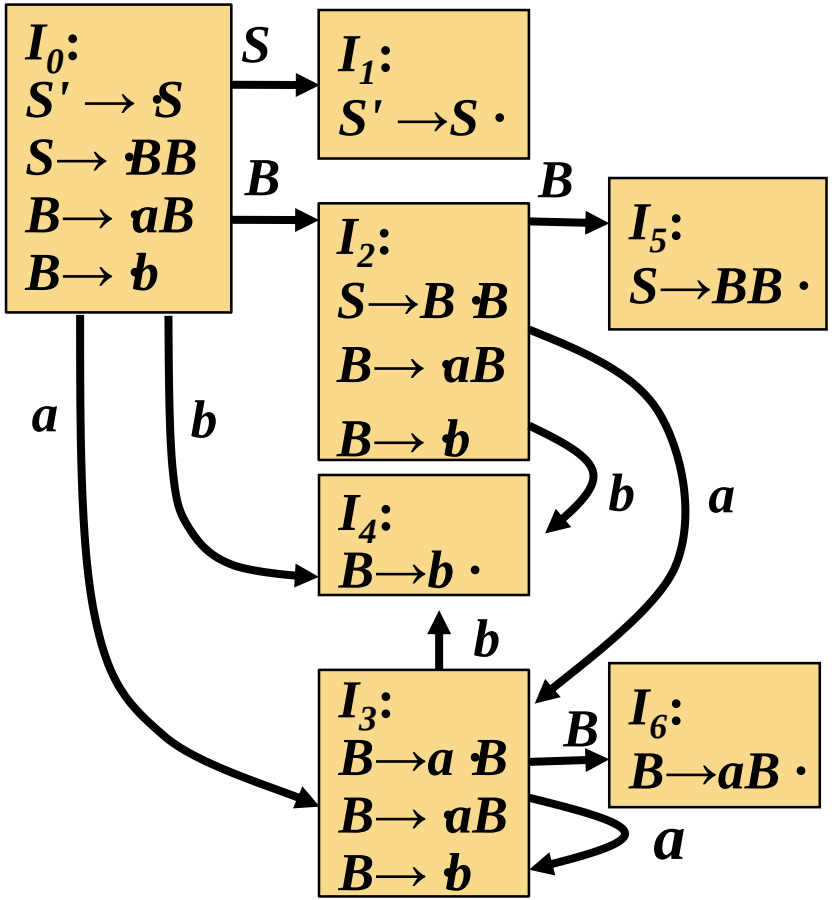
文法

(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

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

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;
}
```

例：LR(0)自动机

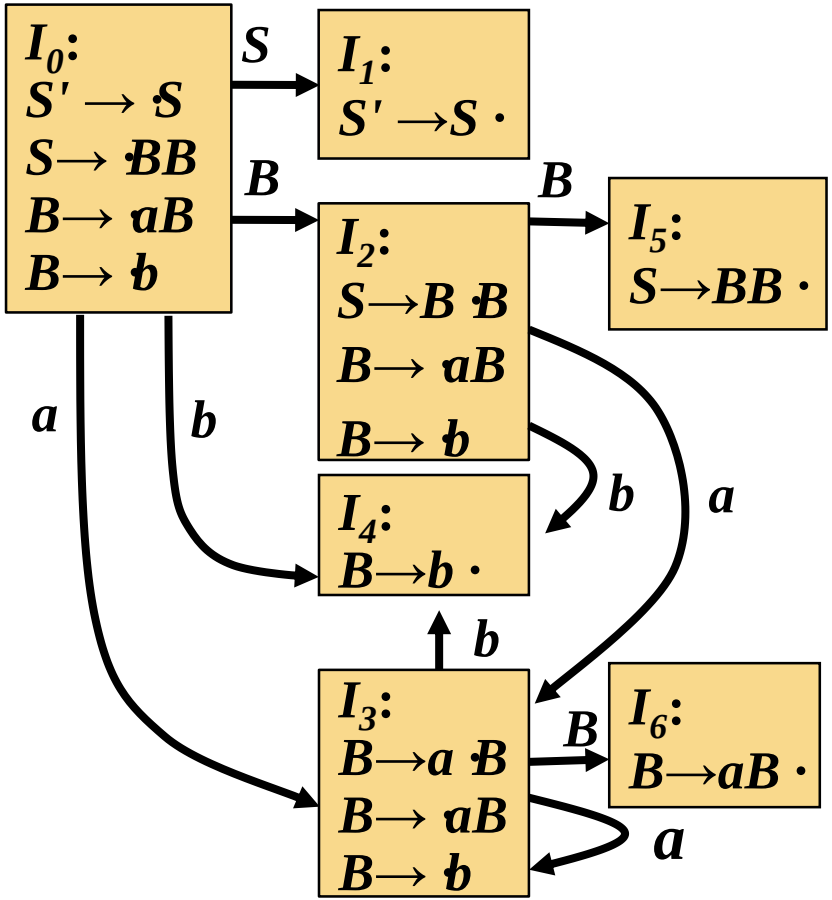
文法

(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		

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

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

```
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*)

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

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

初始项目

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

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

归约项目

接收项目

```
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;
```

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

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

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

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

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

(7) $I\to I\cdot,i$

(8) $I\to i\cdot$

(9) $I\to I,\cdot i$

(10) $I\to I,i\cdot$

(11) $T\to \cdot r$

(12) $T\to r\cdot$

(13) $T\to \cdot r$

(14) $T\to r\cdot$

LR(0)分析表构造算法

- 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

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

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) $F \to \text{id}$

- 构造G'的规范$LR(0)$项集族$C = \{ I_0, I_1, \dots, I_n \}$
- 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* $A \to \alpha\, a\beta \in I_i$ *and* $GOTO(I_i, a) = I_j$ *then* $ACTION[i, a] = sj$
  - *if* $A \to \alpha \cdot B\beta \in I_i$ *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”

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

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

初始项目　接收项目　归约项目

$LR(0)$ 自动机的形式化定义

- 文法

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

- $LR(0)$ 自动机

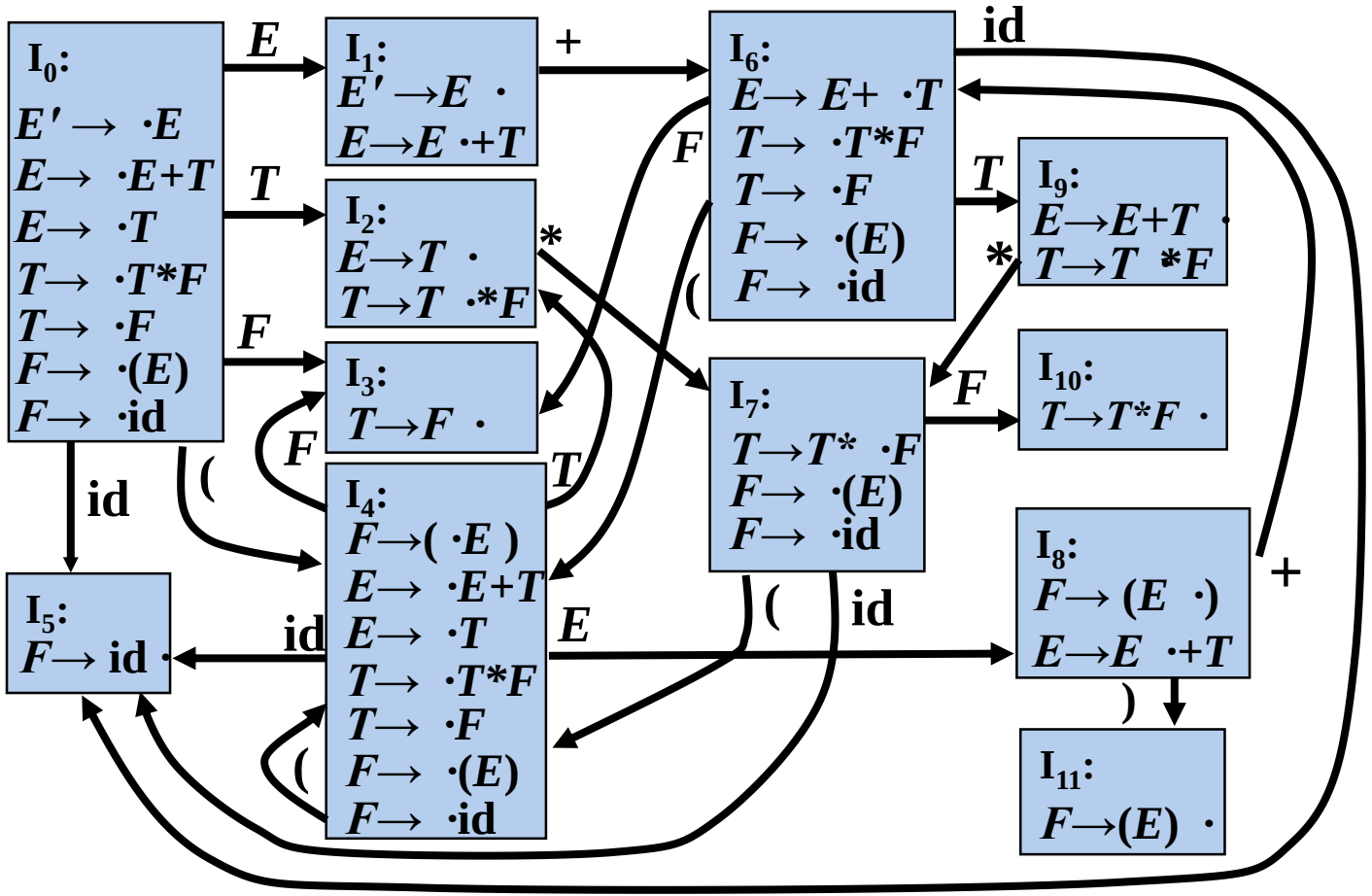
$$M = (C, V_N \cup V_T , GOTO, I_0 , F)$$

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

例

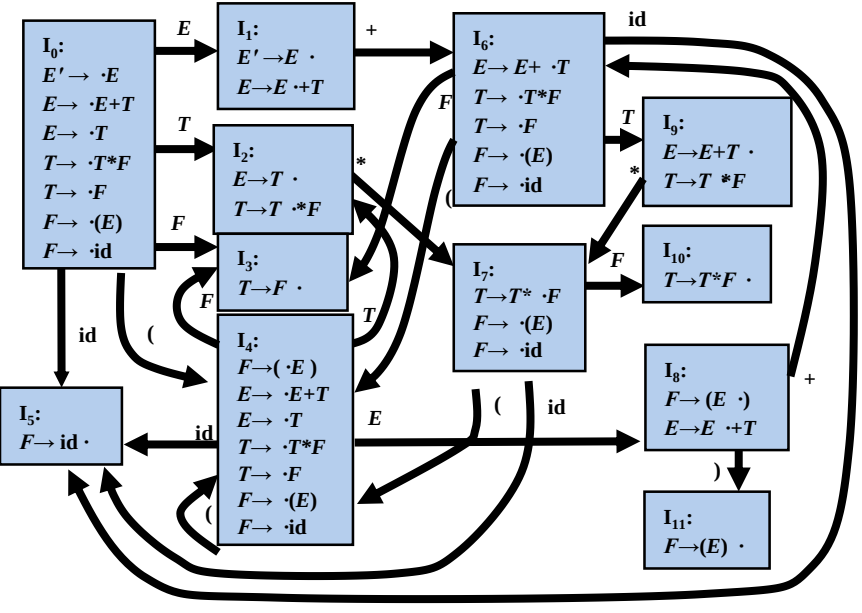
文法

(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) $F \rightarrow \text{id}$



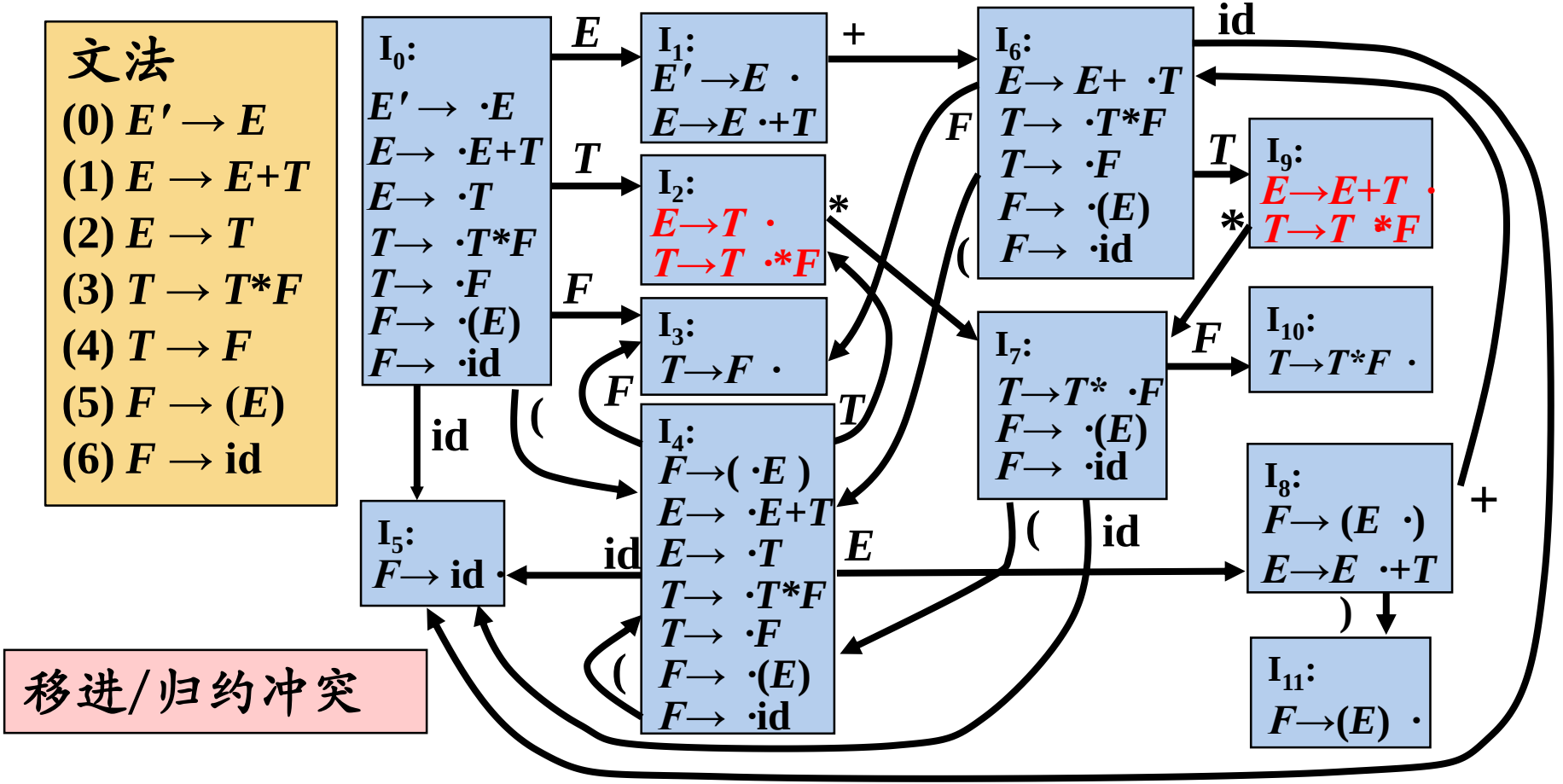
例

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) $F \rightarrow id$



状态	ACTION						GOTO		
	id	+	*	(	)	$	E	T	F
0	s5			s4			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：移进规约冲突


文法
(0) E′ → E
(1) E → E+T
(2) E → T
(3) T → T*F
(4) T → F
(5) F → (E)
(6) F → id
I0:
E′ → ·E
E→ ·E+T
E→ ·T
T→ ·T*F
T→ ·F
F→ ·(E)
F→ ·id
I1:
E′ →E ·
E→E ·+T
I2:
E→T ·
T→T ·*F
I3:
T→F ·
I4:
F→(·E)
E→ ·E+T
E→ ·T
T→ ·T*F
T→ ·F
F→ ·(E)
F→ ·id
I5:
F→ id ·
I6:
E→ E+ ·T
T→ ·T*F
T→ ·F
F→ ·(E)
F→ ·id
I7:
T→T* ·F
F→ ·(E)
F→ ·id
I8:
F→ (E ·)
E→E ·+T
I9:
E→E+T ·
T→T ·*F
I10:
T→T*F ·
I11:
F→(E) ·
E
T
F
(
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			

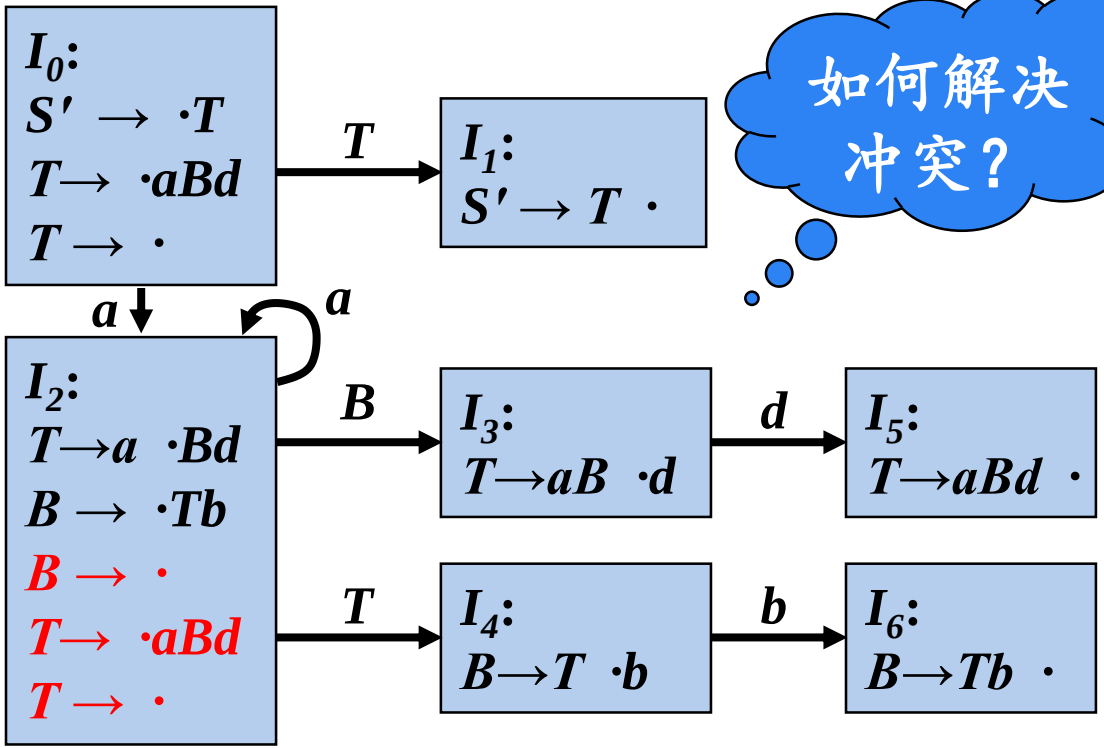
例

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

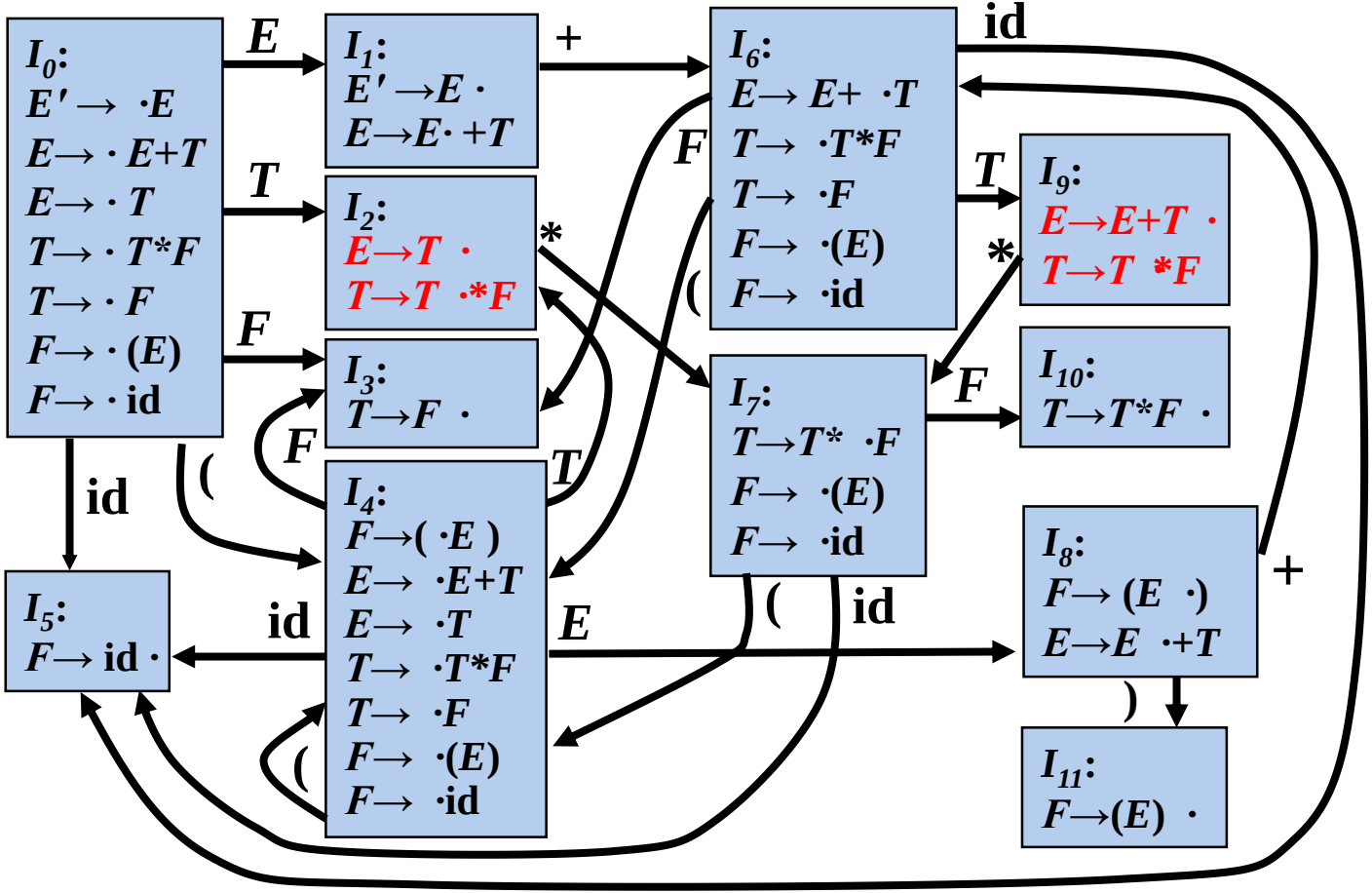
提纲

例：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) $F \rightarrow \text{id}$

X	$FOLLOW(X)$
E	), +, $
T	), +, $, *
F	), +, $, *



SLR (1)分析

➢SLR(1)分析法的基本思想

已知项目集I:

$A_1 \rightarrow \alpha_1 . a_1 \beta_1$

$A_2 \rightarrow \alpha_2 . a_2 \beta_2$

…

$A_m \rightarrow \alpha_m . a_m \beta_m$

（m个移进项目）

$B_1 \rightarrow \gamma_1 .$

$B_2 \rightarrow \gamma_2 .$

…

$B_n \rightarrow \gamma_n .$

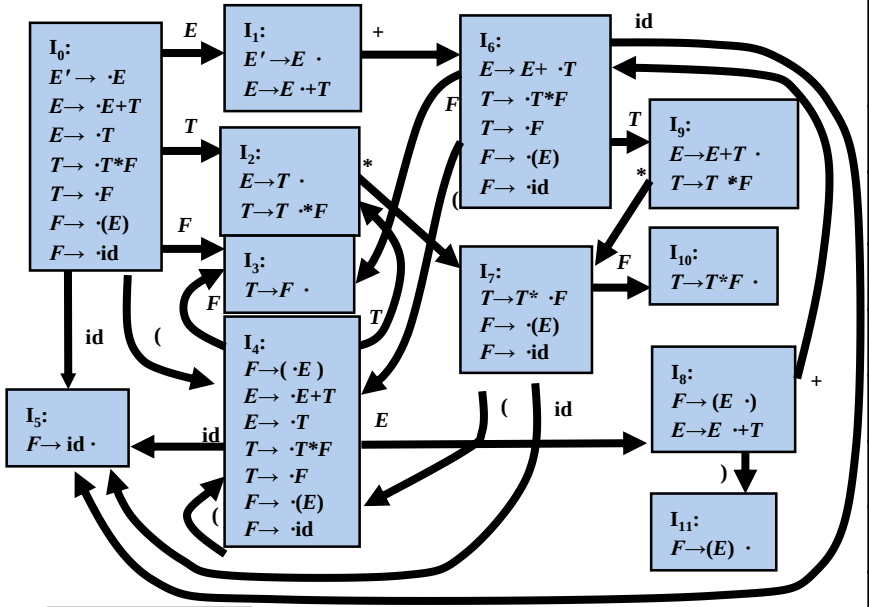
（n个归约项目）

如果集合$\{a_1, a_2, \ldots, a_m\}$和$FOLLOW(B_1)$，$FOLLOW(B_2)$，…，$FOLLOW(B_n)$两两不相交，则项目集$I$中的冲突可以按以下原则解决：

设a是下一个输入符号

- 若$a \in \{a_1, a_2, \ldots, a_m\}$，则移进a
- 若$a \in FOLLOW(B_i)$，则用产生式$B_i \rightarrow \gamma_i$归约
- 此外，报错

表达式文法的SLR(1)分析表



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) $F \rightarrow id$

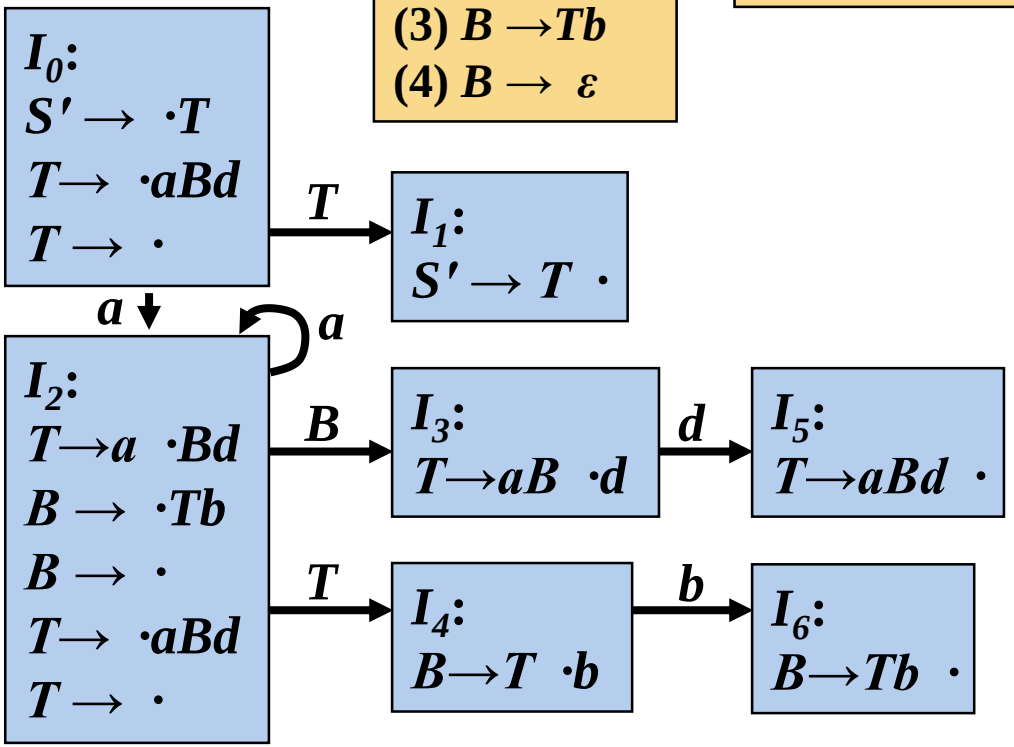
X	FOLLOW(X)
E	), +, $
T	), +, $, *
F	), +, $, *

状态	ACTION						GOTO		
	id	+	*	(	)	$	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			

例

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

$FOLLOW(S')=\{ \$ \}$
$FOLLOW(T)=\{ \$, b \}$
$FOLLOW(B)=\{ d \}$



SLR(1)分析表

状态	ACTION				GOTO	
	a	b	d	$	T	B
0	s2	r2		r2	1	
1				acc		
2	s2	r2	r4	r2	4	3
3			s5			
4		s6				
5		r1		r1		
6			r3			

SLR(1) 分析表构造算法

- 构造G'的规范$LR(0)$项集族$C = \{ I_0, I_1, \dots, I_n \}$。
- 根据I_i构造得到状态i。状态i的语法分析动作按照下面的方法决定：
  - if $A \to \alpha\, a\beta \in I_i$ 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 $A \to \alpha \in I_i$且$A \neq S'$ then for $\forall a \in FOLLOW(A)$ do
    $ACTION[i, a] = rj$ （j是产生式$A \to \alpha$的编号）
  - if $S' \to S \in I_i$ then $ACTION[i, \$] = acc$;
- 没有定义的所有条目都设置为“error”。

如果给定文法的SLR(1)分析表中不存在有冲突的动作，那么该文法称为SLR(1)文法

表达式的*SLR*(1)分析

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

给出i+i*i的*SLR*(1)分析过程

状态	ACTION						GOTO		
	id	+	*	(	)	$	*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			

表达式的*SLR*(1)分析

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) $F \rightarrow id$

状态栈	符号栈	输入	action	go
0	$	id+id*id$	s_5	
05	$id	+id*id$	r_6 F→id	go(0,F)=3
03	$F	+id*id$	r_4 T→F	go(0,T)=2
02	$T	+id*id$	r_2 E→T	go(0,E)=1
01	$E	+id*id$	s_6	
016	$E+	id*id$	s_5	
0165	$E+id	*id$	r_6 F→id	go(6,F)=3
0163	$E+F	*id$	r_4 T→F	go(6,T)=9
0169	$E+T	*id$	s_7	
01697	$E+T *	id$	s_5	
0169 75	$E+T *id	$	r_6 F→id	go(7,F)=10
01697 10	$E+T*F	$	r_3 T→T*F	go(6,T)=9
0169	$E+T	$	r_1 E→E+T	go(0,E)=1
01	$E	$	acc	

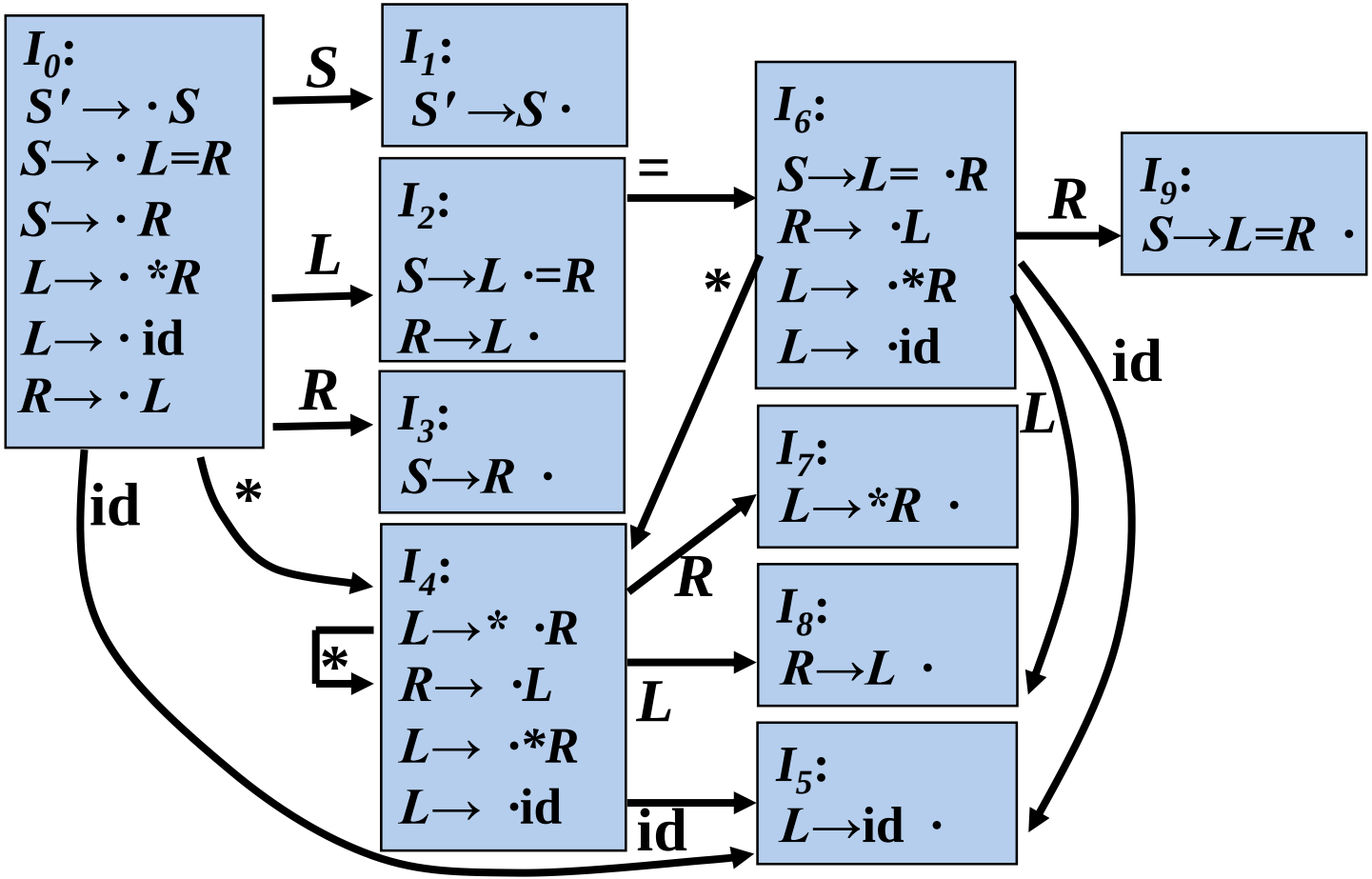
i+i*i的*SLR*(1)分析过程

状态	ACTION						GOTO		
	id	+	*	(	)	$	*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			

SLR(1) 分析举例

➢例

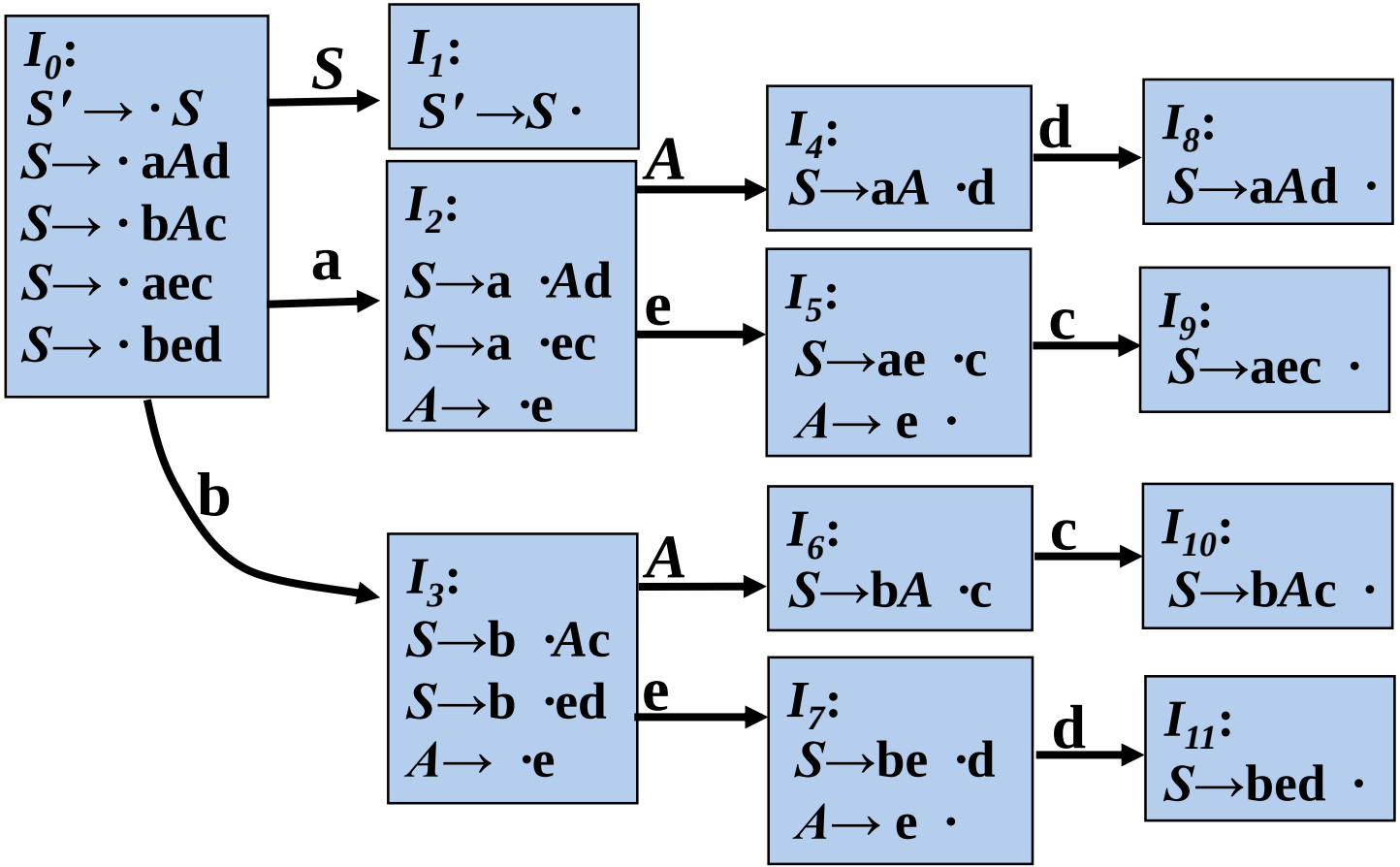
0) S′→S
1) S→L=R
2) S→R
3) L→*R
4) L→id
5) R→L



课堂练习

➢例

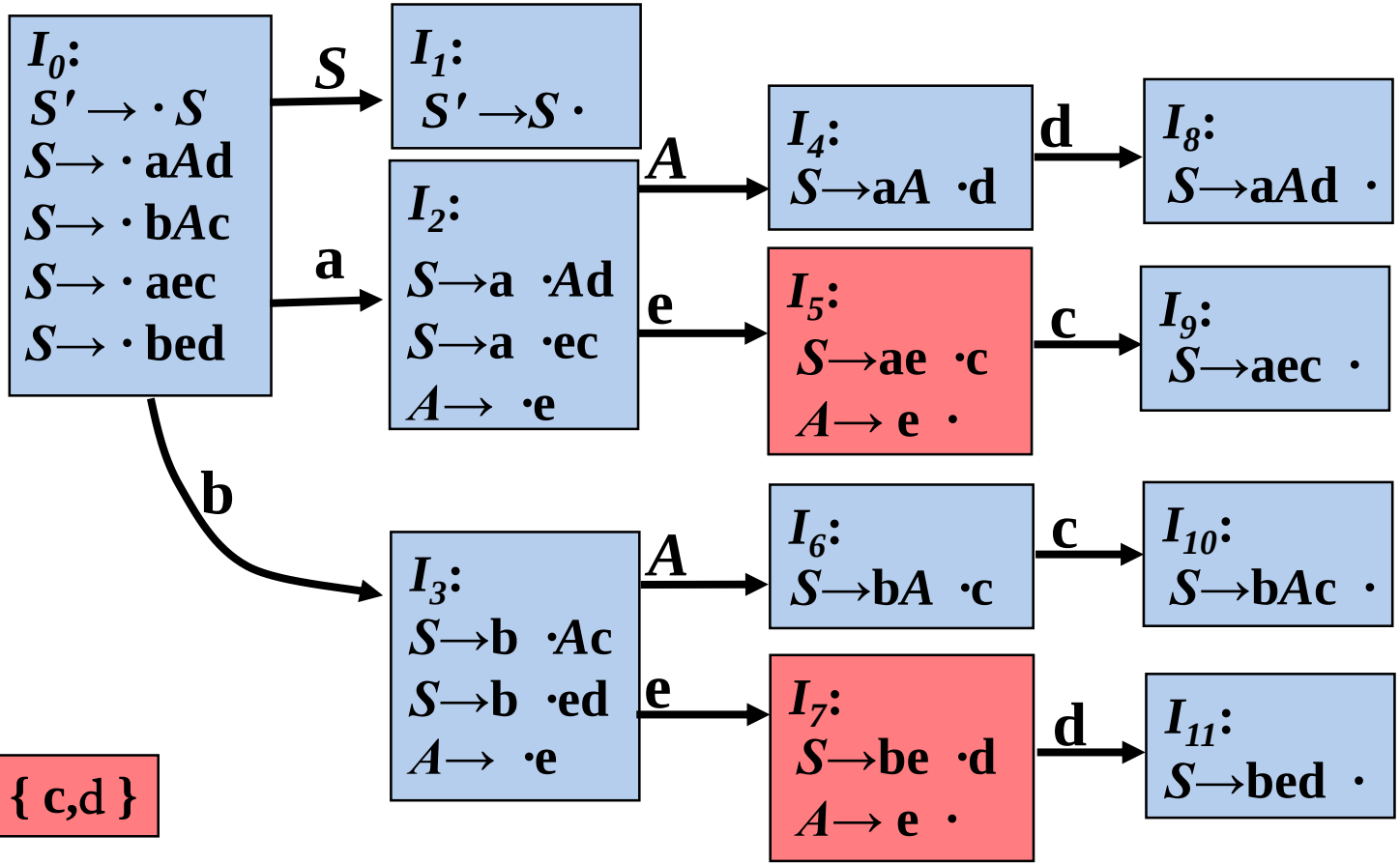
0) $S'\rightarrow S$
1) $S\rightarrow aAd$
2) $S\rightarrow bAc$
3) $S\rightarrow aec$
4) $S\rightarrow bed$
5) $A\rightarrow e$



SLR (1)分析中的冲突

➢例

0) $S'\rightarrow S$
1) $S\rightarrow aAd$
2) $S\rightarrow bAc$
3) $S\rightarrow aec$
4) $S\rightarrow bed$
5) $A\rightarrow e$

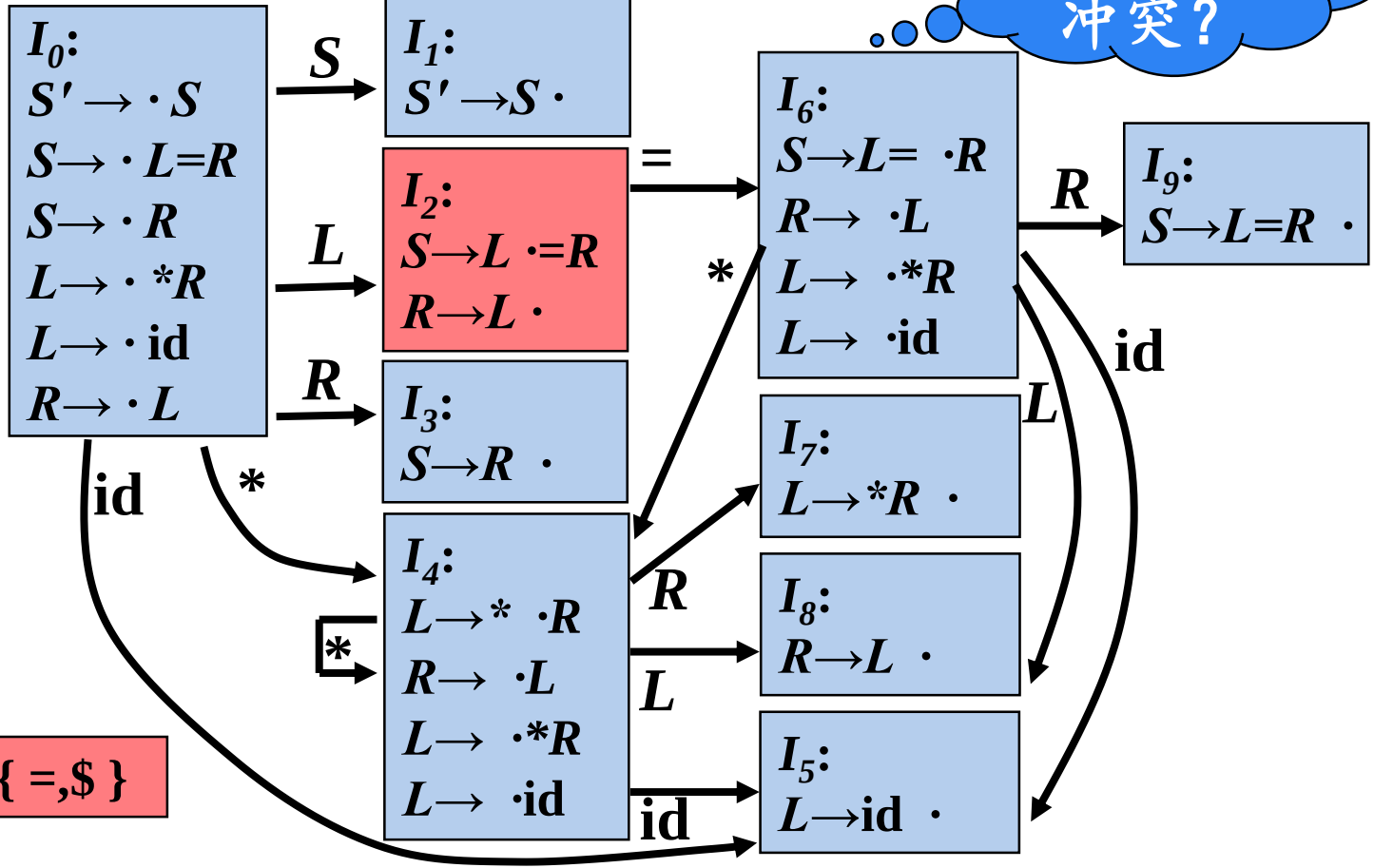


FOLLOW(*A*) = { c,d }

SLR (1)分析中的冲突

➢例

0) $S'\rightarrow S$
1) $S\rightarrow L=R$
2) $S\rightarrow R$
3) $L\rightarrow *R$
4) $L\rightarrow id$
5) $R\rightarrow L$



如何解决冲突？

$FOLLOW(R) = \{ =,\$ \}$

提纲

$LR(1)$分析法的提出

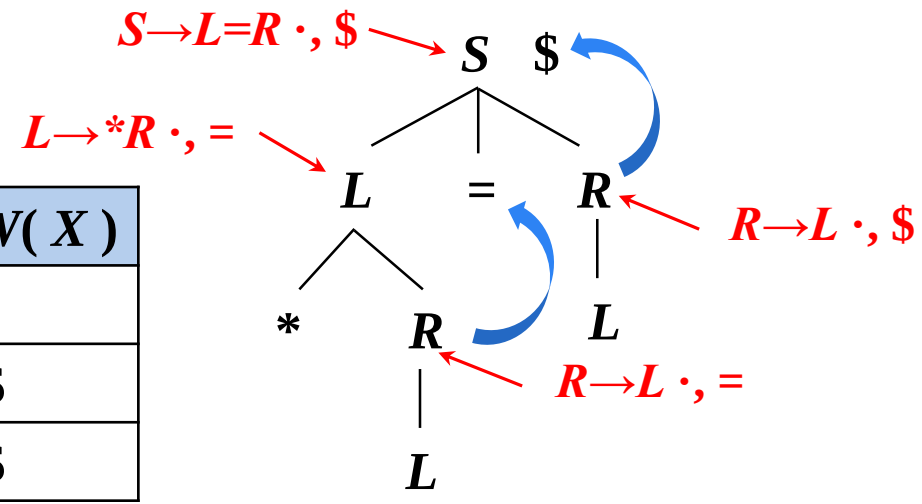
- $SLR(1)$分析存在的问题
  - $SLR(1)$只是简单地考察下一个输入符号b是否属于与归约项目$A\rightarrow\alpha$相关联的$FOLLOW(A)$，但$b\in FOLLOW(A)$只是归约α的一个必要条件，而非充分条件

LR(1)分析法的提出

- 对于产生式$A \to \alpha$的归约，在不同的使用位置，A会要求不同的后继符号

0) $S' \to S$
1) $S \to L=R$
2) $S \to R$
3) $L \to *R$
4) $L \to \text{id}$
5) $R \to L$

X	$FOLLOW(X)$
S	\$
L	=, \$
R	=, \$



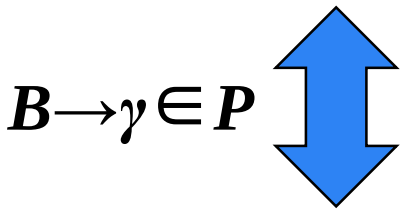
- 在特定位置，A的后继符集合是FOLLOW(A)的子集

规范*LR*(1)项目

- 将一般形式为 [*A*→*α* *β*, a]的项称为 ***LR*(1) 项**，其中*A*→*αβ* 是一个产生式，a 是一个终结符或$，它表示在当前状态下，*A* 后面跟随的符号，称为该项的展望符(*lookahead*)
  - *LR*(1) 中的1指的是项的第二个分量的长度
  - 在形如[*A*→*α* *β*, a]且*β* ≠ ε的项中，展望符a没有任何作用
  - 但是一个形如[*A*→*α* ; a]的项只有在下一个输入符号等于a时才可以按照*A*→*α* 进行归约
    - 这样的a的集合总是*FOLLOW*(*A*)的子集，而且它通常是一个真子集

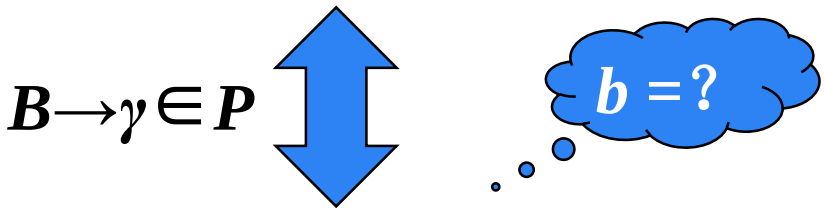
等价$LR(1)$项目

$[A \rightarrow \alpha B\beta, a]$



等价$LR(1)$项目

$[A \to \alpha B\beta, a]$



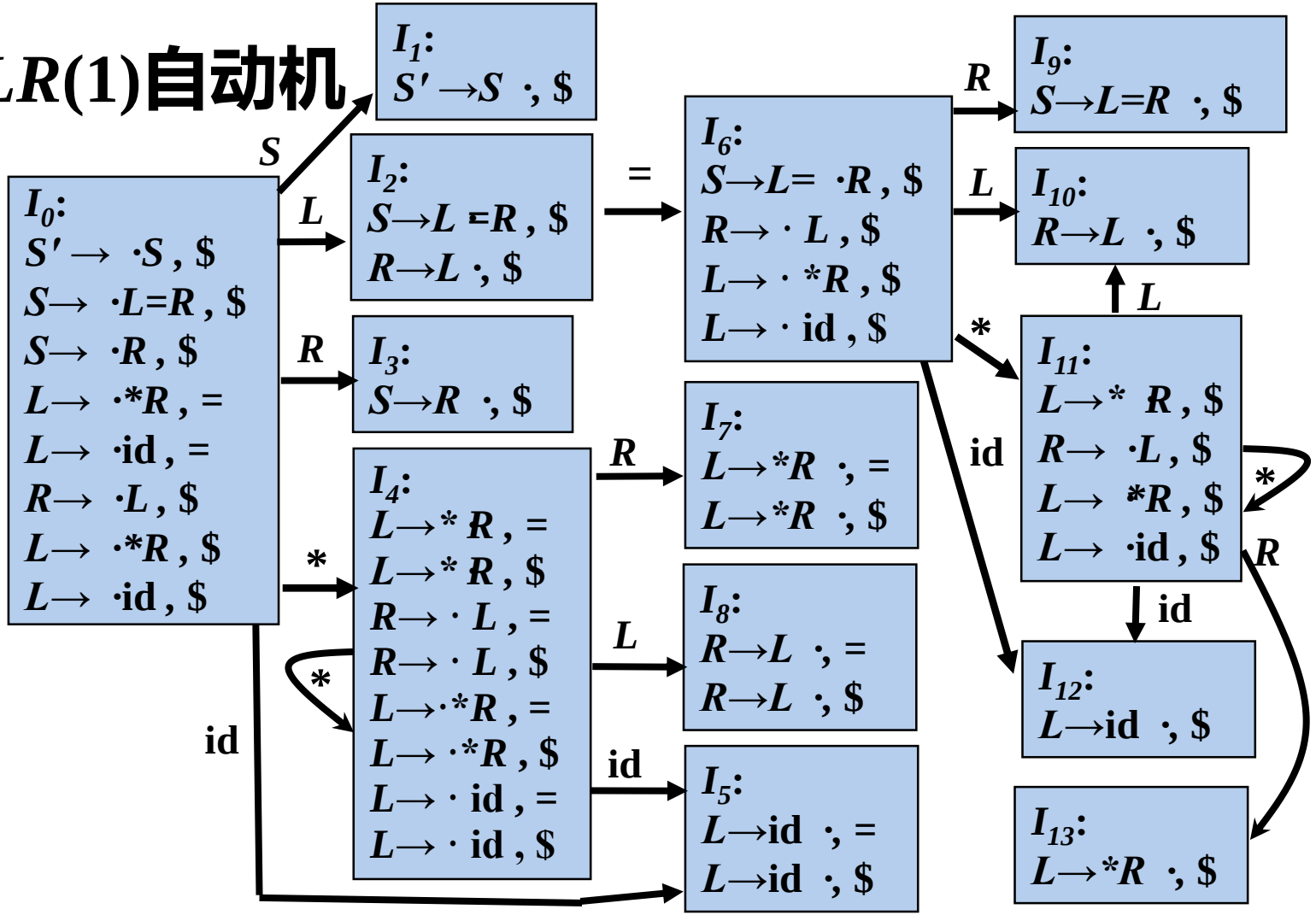
$[B \to \gamma, b]$

$b \in FIRST(\beta a)$

当$\beta \Rightarrow^{+} \varepsilon$时，此时$b$=a叫继承的展望符，否则叫自生的展望符

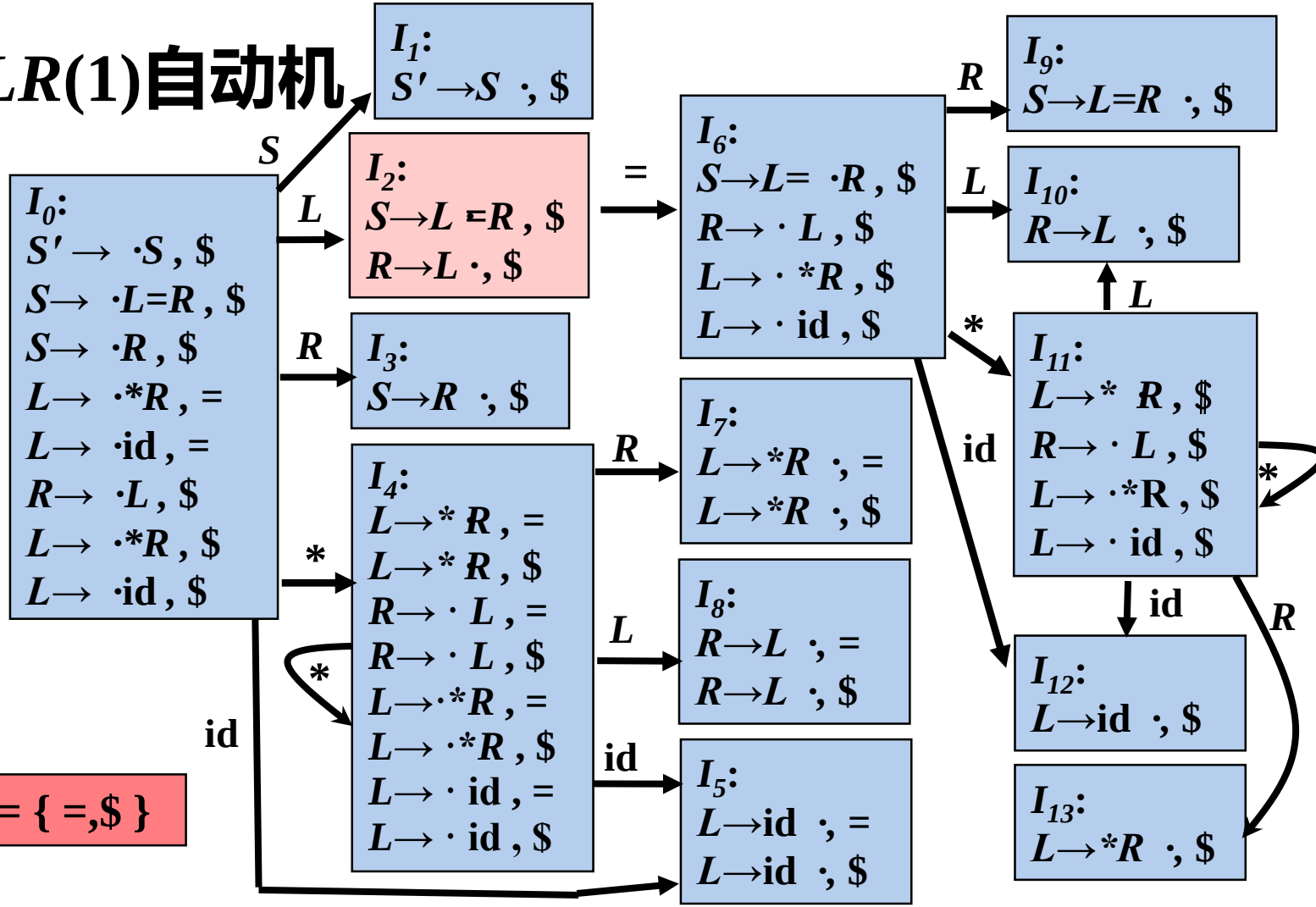
例：LR(1)自动机

0) $S' \rightarrow S$
1) $S \rightarrow L=R$
2) $S \rightarrow R$
3) $L \rightarrow *R$
4) $L \rightarrow id$
5) $R \rightarrow L$



例：LR(1)自动机

0) $S' \rightarrow S$
1) $S \rightarrow L=R$
2) $S \rightarrow R$
3) $L \rightarrow *R$
4) $L \rightarrow id$
5) $R \rightarrow L$



FOLLOW(R) = { =,$ }

LR(1)项目集闭包

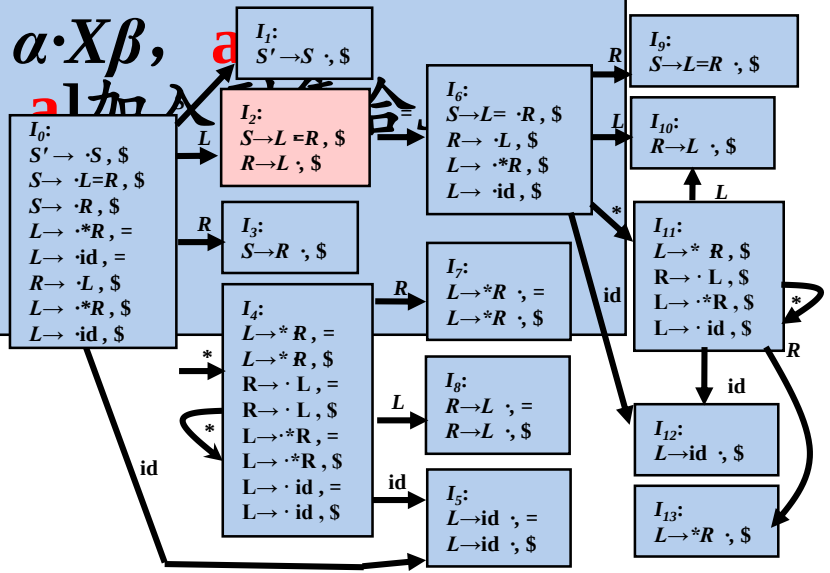
$CLOSURE(I) = I \cup \{ [B \rightarrow \gamma, b] \mid [A \rightarrow \alpha B\beta, a] \in CLOSURE(I), B \rightarrow \gamma \in P, b \in FIRST(\beta a) \}$

```
SetOfItems CLOSURE ( I ) {
        repeat
                for ( I中的每个项[A → α·Bβ，a ])
                        for (G'的每个产生式B → γ)
                                for ( FIRST (βa)中的每个符号b )
                                        将[ B → ·γ ，b]加入到集合I中；
        until 不能向I中加入更多的项；
        until I ；
}
```

GOTO 函数

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

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



为文法G'构造$LR(1)$项集族

```
void items (G' ) {
        将C初始化为{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 不再有新的项集加入到C中;
}
```

$LR(1)$自动机的形式化定义

- 文法

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

- $LR(1)$自动机

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

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

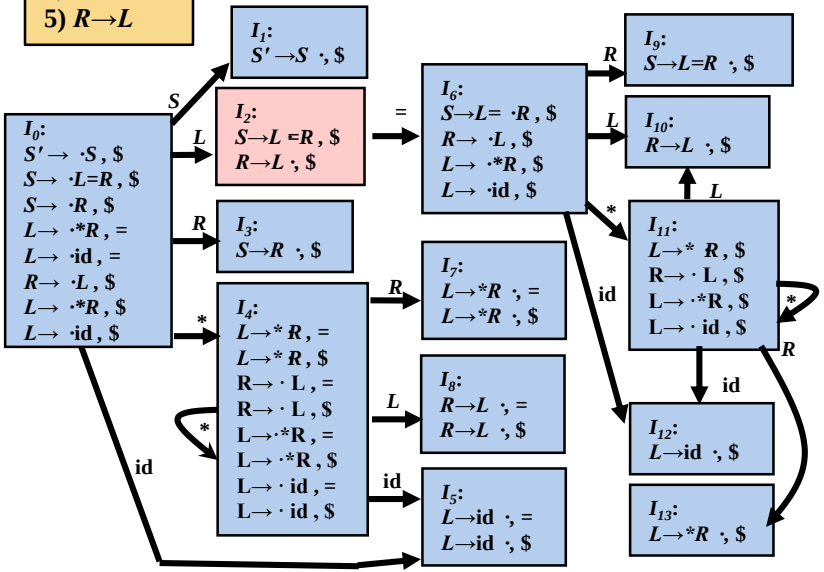
LR(1)分析表构造算法

- 构造G'的规范LR(1)项集族$C = \{ I_0, I_1, \dots, I_n \}$
- 根据I_i构造得到状态i。状态i 的语法分析动作按照下面的方法决定：
  - if $[A \to \alpha \, a\beta, b] \in I_i$ and $GOTO(I_i, a) = I_j$ then $ACTION[i, a] = sj$
  - if $[A \to \alpha \, B\beta, b] \in I_i$ and $GOTO(I_i, B) = I_j$ then $GOTO[i, B] = j$
  - if $[A \to \alpha \, ; a] \in I_i$且$A \neq S'$ then $ACTION[i, a] = rj$
    （j是产生式$A \to \alpha$的编号）
  - if $[S' \to S \, ; \$] \in I_i$ then $ACTION[i, \$] = acc$;
- 没有定义的所有条目都设置为“error”

如果LR(1)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(1)文法

赋值语句文法的 *LR*(1)分析表

0) $S' \rightarrow S$
1) $S \rightarrow L=R$
2) $S \rightarrow R$
3) $L \rightarrow *R$
4) $L \rightarrow id$
5) $R \rightarrow L$

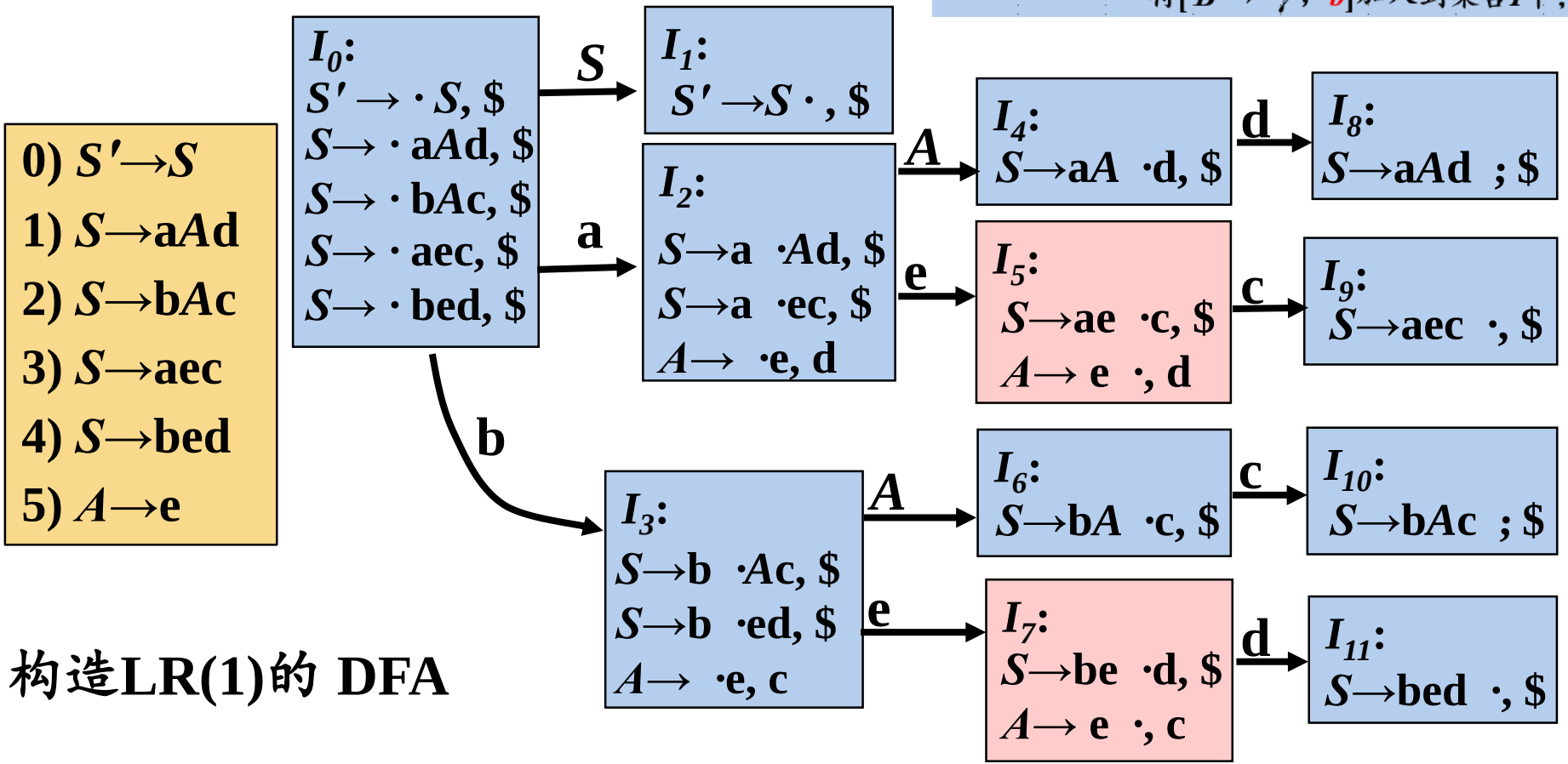


状态	ACTION				GOTO		
	*	id	=	$	S	L	R
0	s4	s5			1	2	3
1				acc			
2			s6	r5			
3				r2			
4	s4	s5				8	7
5			r4	r4			
6	s11	s12				10	9
7			r3	r3			
8			r5	r5			
9				r1			
10				r5			
11	s11	s12				10	13
12				r4			
13				r3			

课堂练习

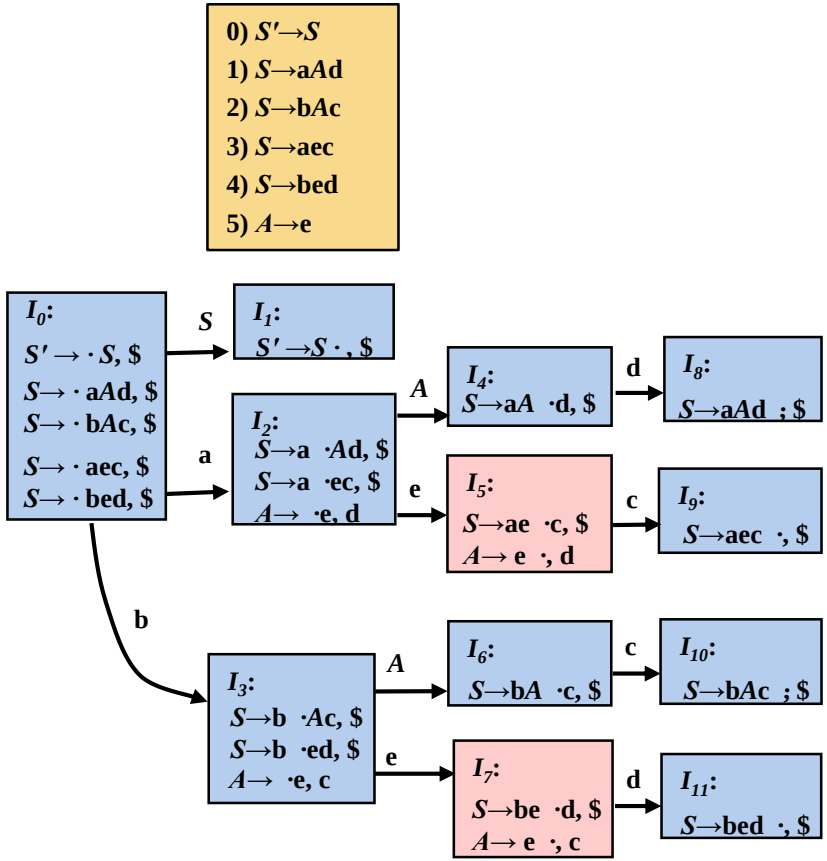
```
for ( I中的每个项[A → α·Bβ，a ])
    for (G'的每个产生式B → γ)
        for ( FIRST (βa)中的每个符号b )
            将[B → ·γ，b]加入到集合I中；
```

0) $S'\rightarrow S$
1) $S\rightarrow aAd$
2) $S\rightarrow bAc$
3) $S\rightarrow aec$
4) $S\rightarrow bed$
5) $A\rightarrow e$



构造LR(1)的 DFA

课堂练习



构造LR(1)分析表

状态	ACTION						GOTO	
	a	b	c	d	e	$	S	A
0	s2	s3					1	
1						acc		
2					s5			4
3					s7			6
4				s8				
5			s9	r5				
6			s10					
7			r5	s11				
8						r1		
9						r3		
10						r2		
11						r4		

课堂练习

分析bec和beec是否为该文法的句子？

0) $S'\rightarrow S$
1) $S\rightarrow aAd$
2) $S\rightarrow bAc$
3) $S\rightarrow aec$
4) $S\rightarrow bed$
5) $A\rightarrow e$

状态	ACTION						GOTO	
	a	b	c	d	e	$	S	A
0	s2	s3					1	
1						acc		
2					s5			4
3					s7			6
4				s8				
5			s9	r5				
6			s10					
7			r5	s11				
8						r1		
9						r3		
10						r2		
11						r4		

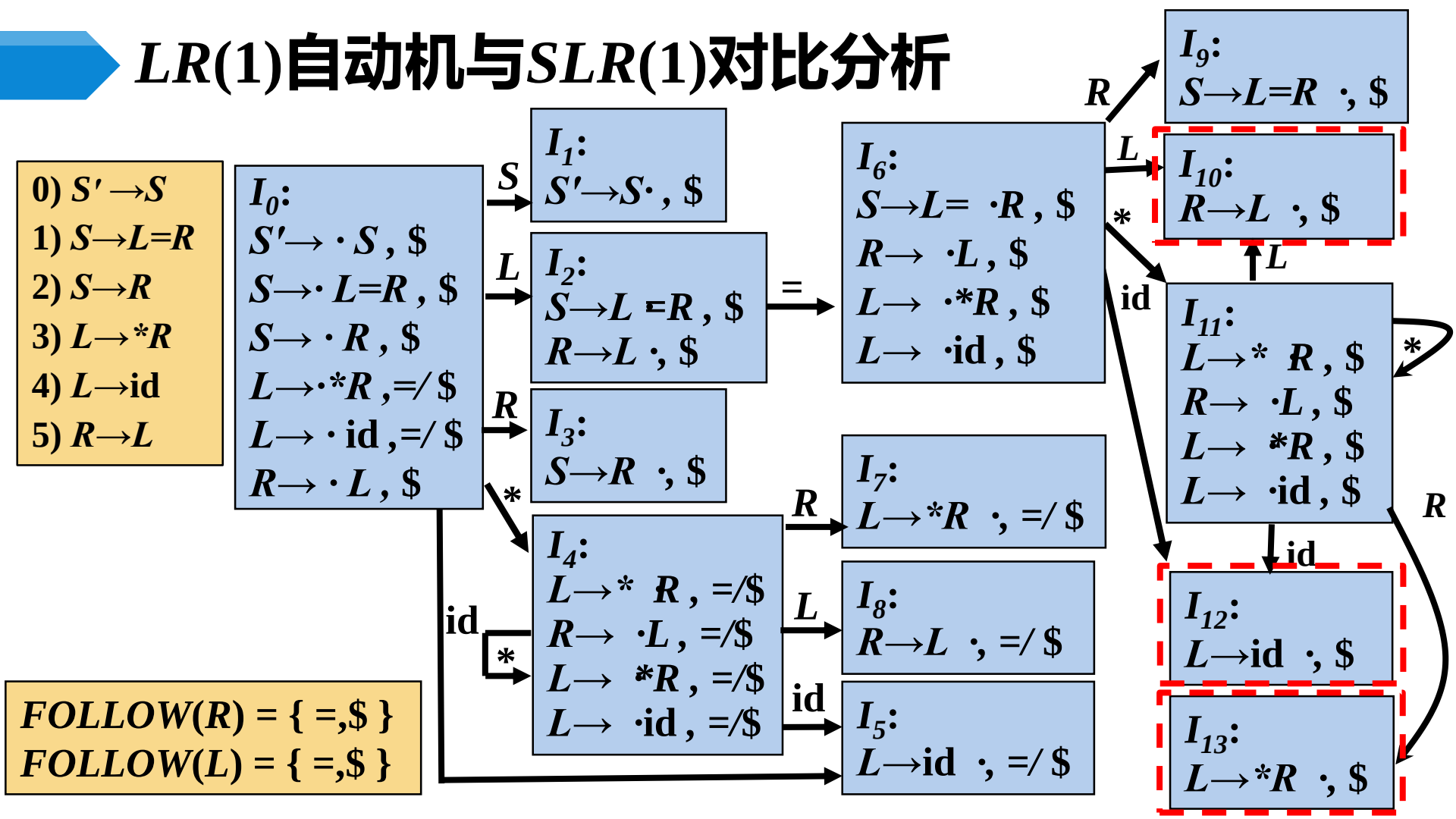
bec

状态栈	符号栈	输入	ACTION	GOTO
0	$	bec$	s3	
03	$b	ec$	s7	
037	$be	c$	r5	goto6
036	$bA	c$	s10	
03610	$bAc	$	r2	goto1
01	$S	$	acc	

beec

状态栈	符号栈	输入	ACTION	GOTO
0	$	beec$	s3	
03	$b	eec$	s7	
037	$be	ec$	err	

LR(1)自动机与SLR(1)对比分析



LR(1)自动机与SLR(1)对比分析

0) $S' \rightarrow S$
1) $S \rightarrow L=R$
2) $S \rightarrow R$
3) $L \rightarrow *R$
4) $L \rightarrow id$
5) $R \rightarrow L$

I_0:
$S' \rightarrow \cdot S$, \$
$S \rightarrow \cdot L=R$, \$
$S \rightarrow \cdot R$, \$
$L \rightarrow \cdot *R$, =/ \$
$L \rightarrow \cdot id$, =/ \$
$R \rightarrow \cdot L$, \$

S → I_1: $S' \rightarrow S \cdot$, \$

L → I_2: $S \rightarrow L \cdot =R$, \$; $R \rightarrow L \cdot$, \$

= → I_6: $S \rightarrow L= \cdot R$, \$; $R \rightarrow \cdot L$, \$; $L \rightarrow \cdot *R$, \$; $L \rightarrow \cdot id$, \$

R → I_3: $S \rightarrow R \cdot$, \$

* → I_4: $L \rightarrow * \cdot R$, =/\$; $R \rightarrow \cdot L$, =/\$; $L \rightarrow \cdot *R$, =/\$; $L \rightarrow \cdot id$, =/\$

id → I_5: $L \rightarrow id \cdot$, =/ \$

R → I_7: $L \rightarrow *R \cdot$, =/ \$

L → I_8: $R \rightarrow L \cdot$, =/ \$

R → I_9: $S \rightarrow L=R \cdot$, \$

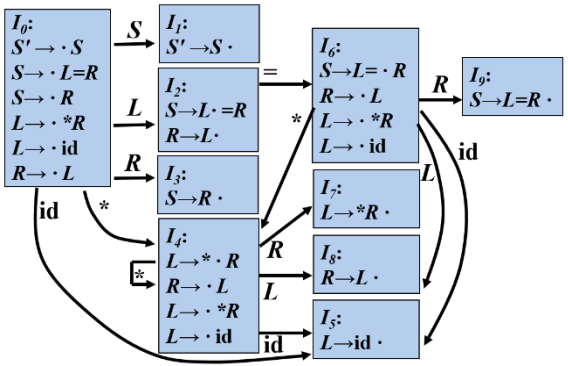
L → I_{10}: $R \rightarrow L \cdot$, \$

* → I_{11}: $L \rightarrow * \cdot R$, \$; $R \rightarrow \cdot L$, \$; $L \rightarrow \cdot *R$, \$; $L \rightarrow \cdot id$, \$

id → I_{12}: $L \rightarrow id \cdot$, \$

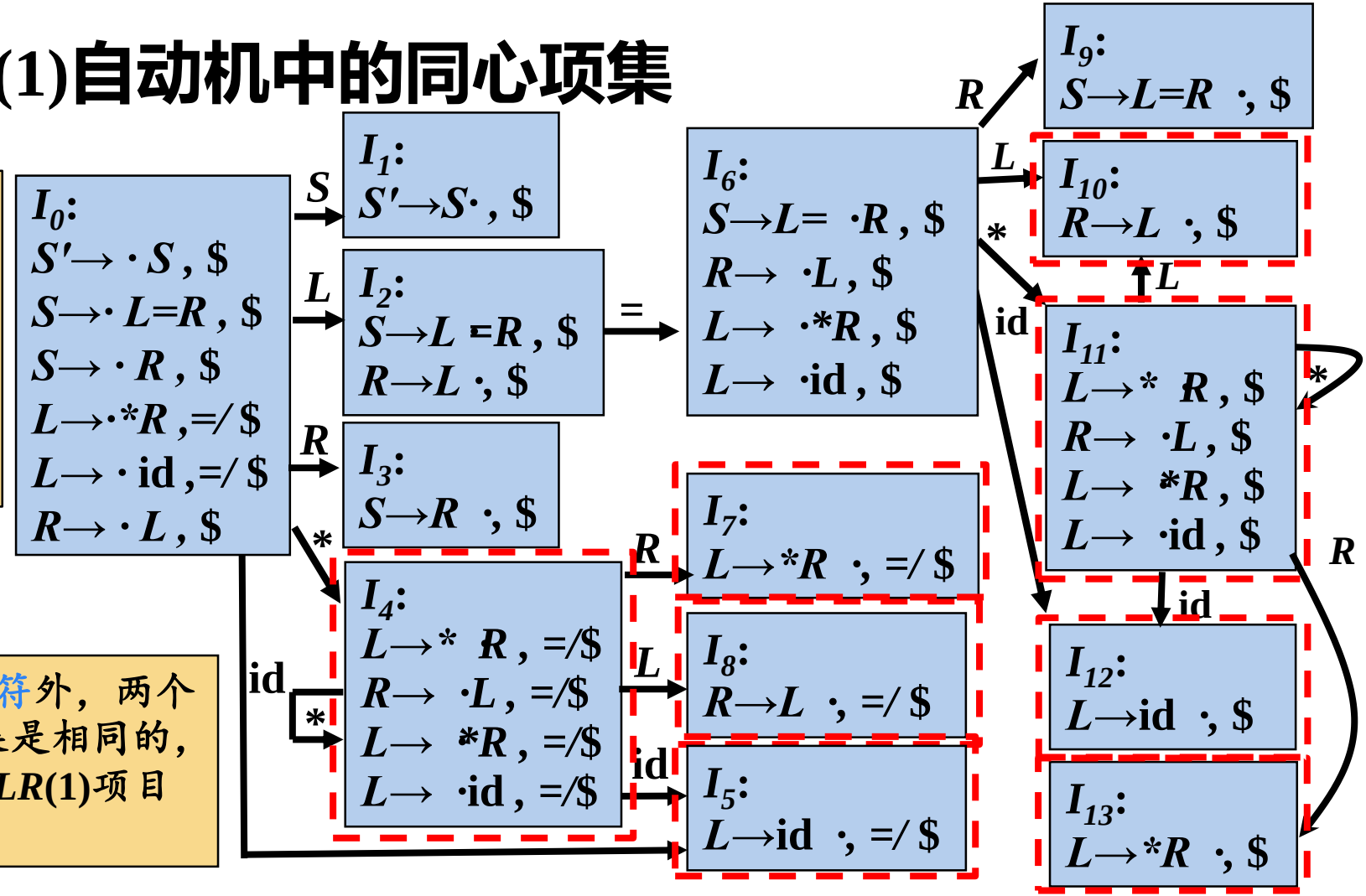
R → I_{13}: $L \rightarrow *R \cdot$, \$

SLR



LR(1)自动机中的同心项集

0) S′→S
1) S→L=R
2) S→R
3) L→*R
4) L→id
5) R→L



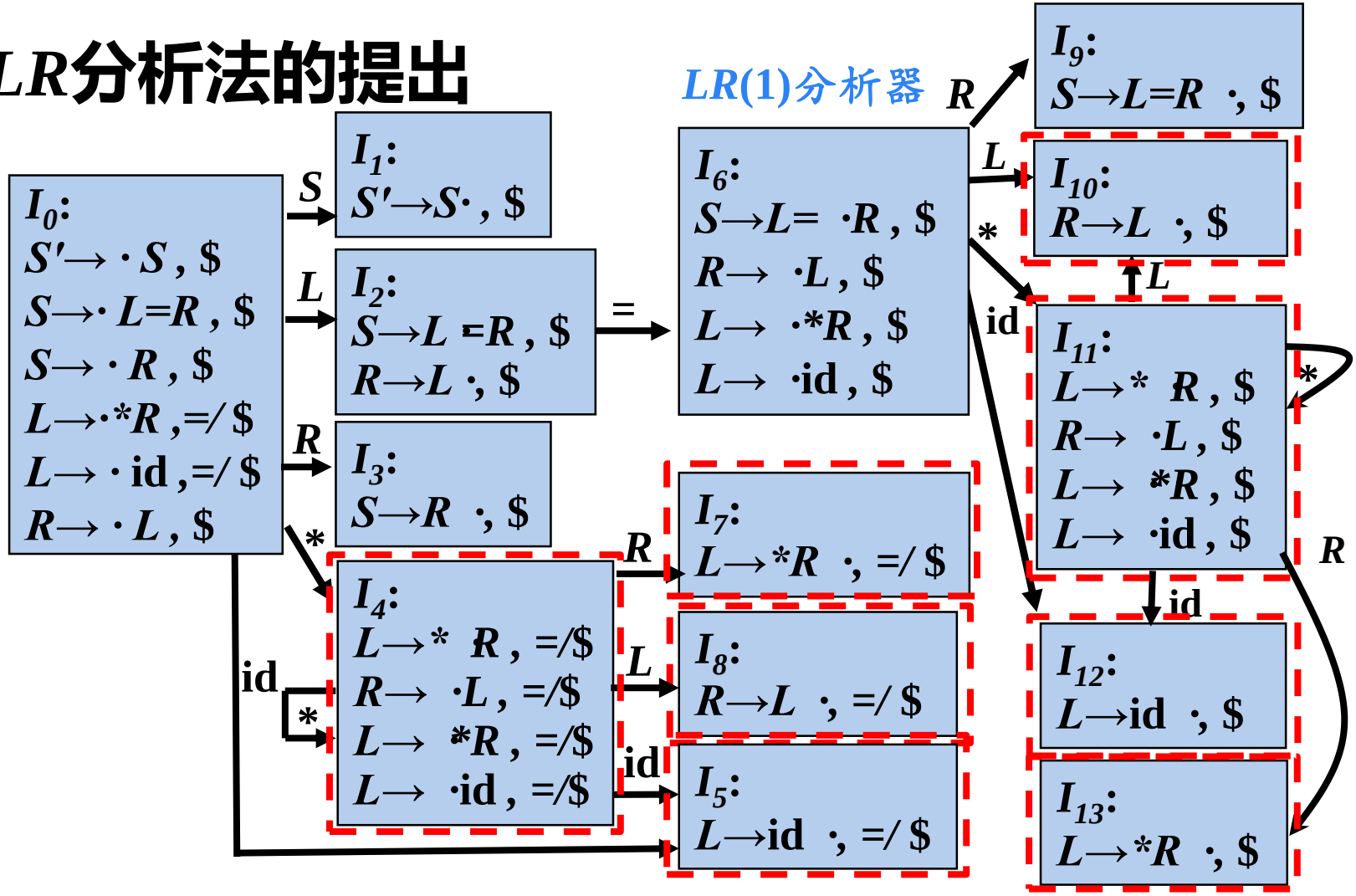
如果除展望符外，两个LR(1)项目集是相同的，则称这两个LR(1)项目集是同心的

提纲

LALR分析法的提出

➢例

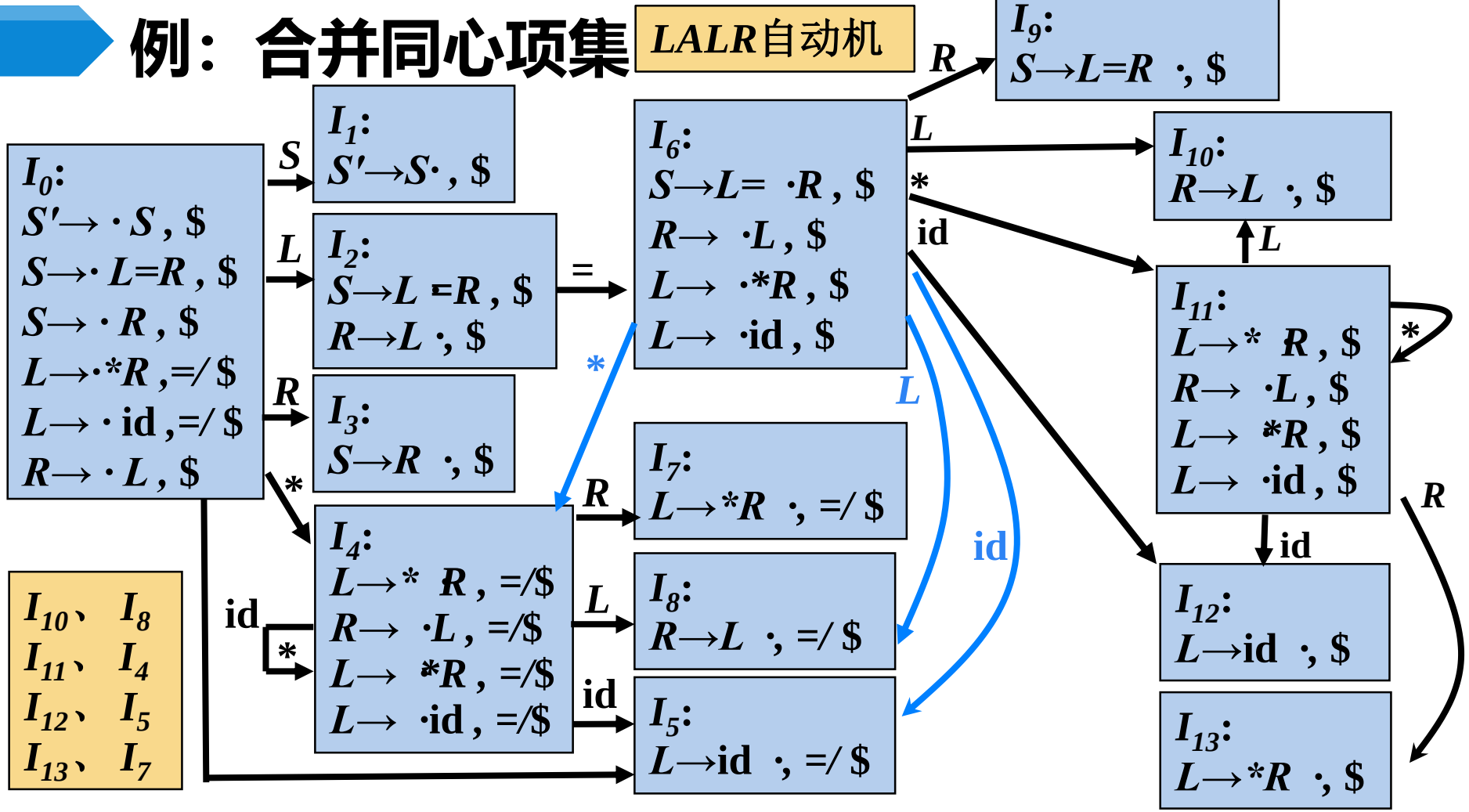
0) $S' \rightarrow S$
1) $S \rightarrow L=R$
2) $S \rightarrow R$
3) $L \rightarrow *R$
4) $L \rightarrow id$
5) $R \rightarrow L$



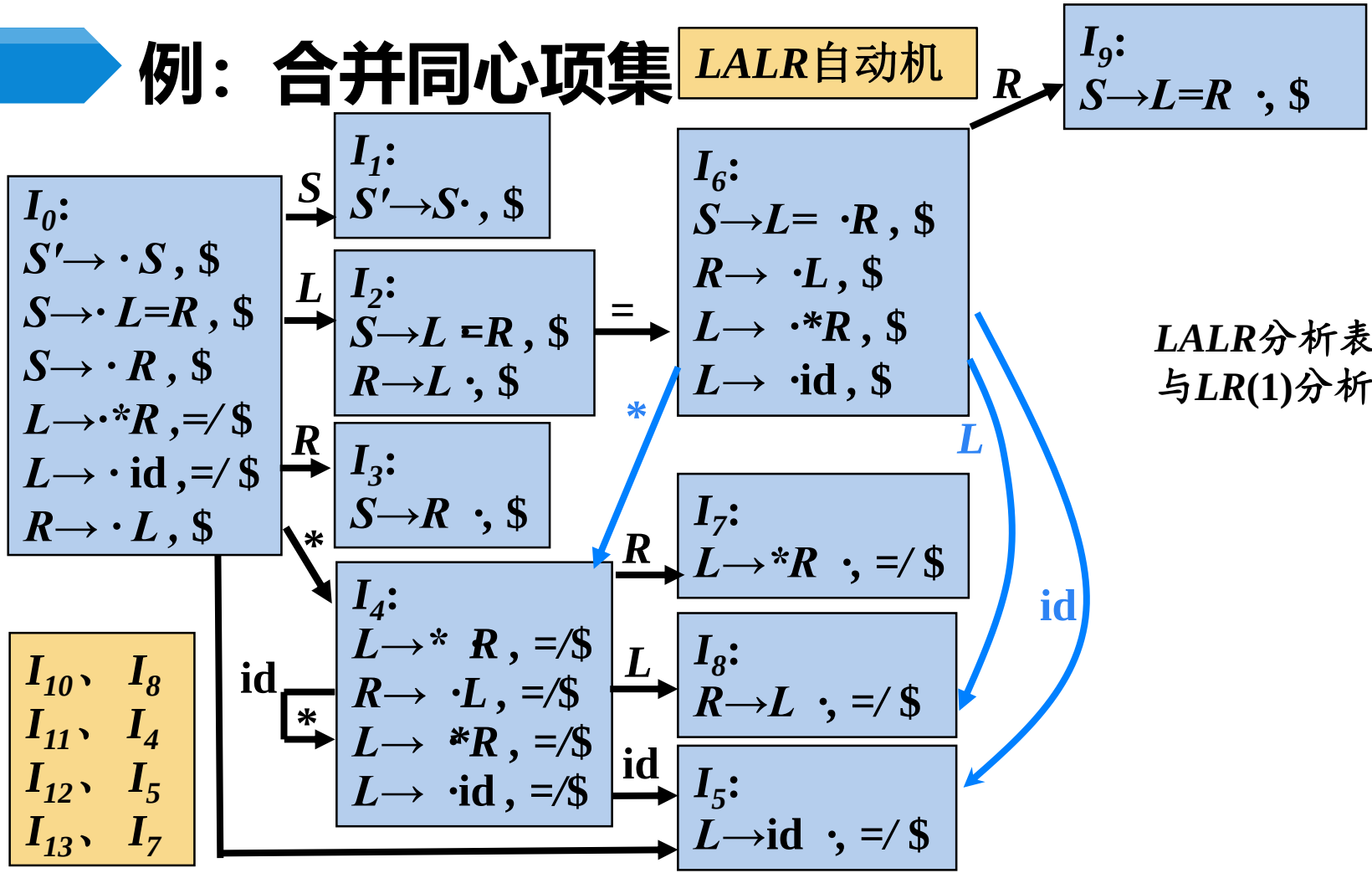
*LALR (lookahead-LR)*分析的基本思想

- 寻找具有相同核心的*LR* (1) 项集，并将这些项集合并为一个项集。 所谓项集的核心就是其第一分量的集合
- 然后根据合并后得到的项集族构造语法分析表
- 如果分析表中没有语法分析动作冲突，给定的文法就称为*LALR* (1) 文法，就可以根据该分析表进行语法分析

例：合并同心项集

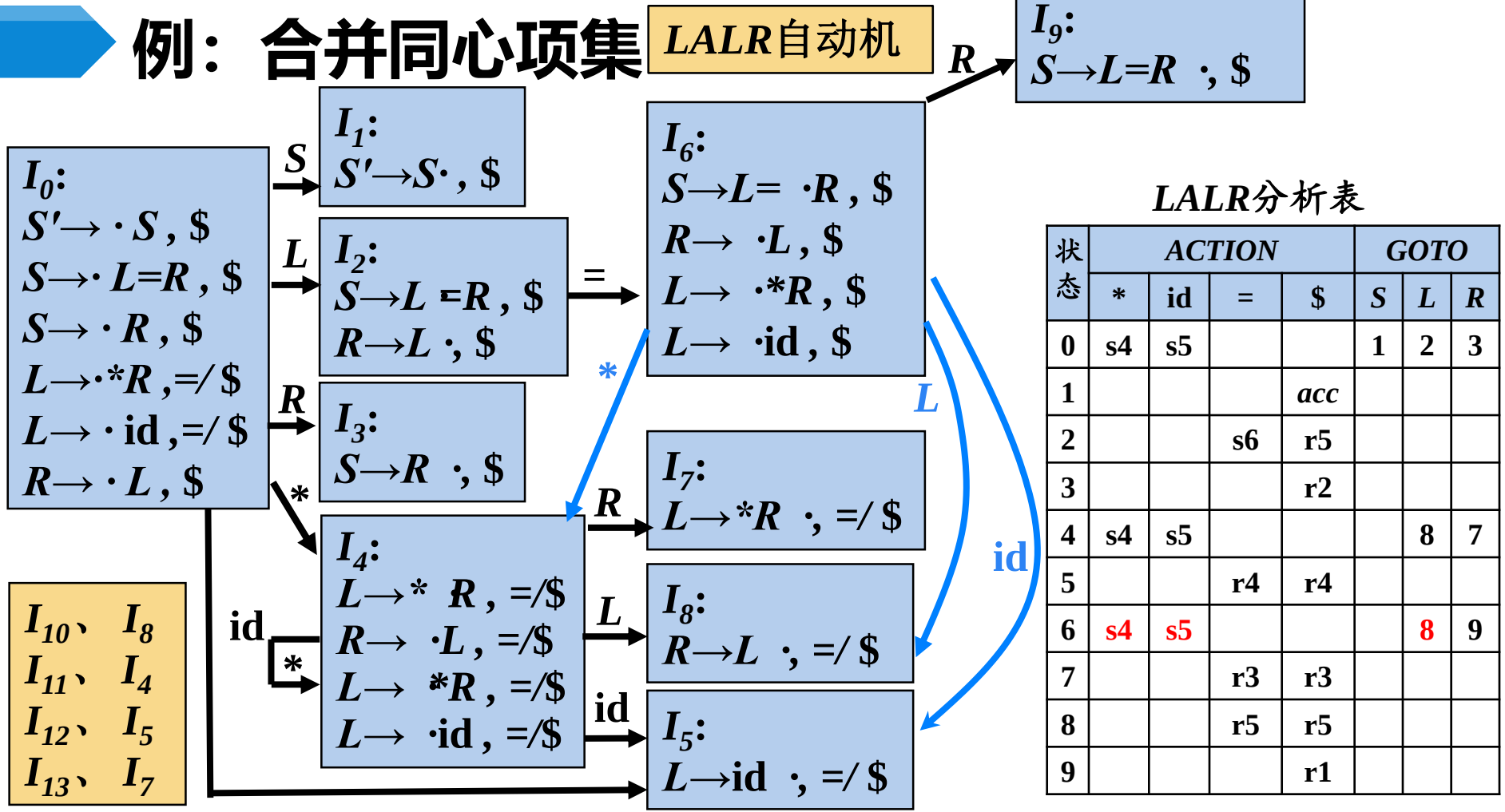

LALR自动机
I_0:
S′→·S , $
S→·L=R , $
S→·R , $
L→·*R ,=/ $
L→·id ,=/ $
R→·L , $
I_1:
S′→S· , $
I_2:
S→L·=R , $
R→L· , $
I_3:
S→R· , $
I_4:
L→*·R , =/$
R→·L , =/$
L→·*R , =/$
L→·id , =/$
I_5:
L→id· , =/ $
I_6:
S→L=·R , $
R→·L , $
L→·*R , $
L→·id , $
I_7:
L→*R· , =/ $
I_8:
R→L· , =/ $
I_9:
S→L=R· , $
I_{10}:
R→L· , $
I_{11}:
L→*·R , $
R→·L , $
L→·*R , $
L→·id , $
I_{12}:
L→id· , $
I_{13}:
L→*R· , $
S
L
R
*
id
=
I_{10}、I_8
I_{11}、I_4
I_{12}、I_5
I_{13}、I_7

例：合并同心项集



LALR分析表的构造
与LR(1)分析表一样

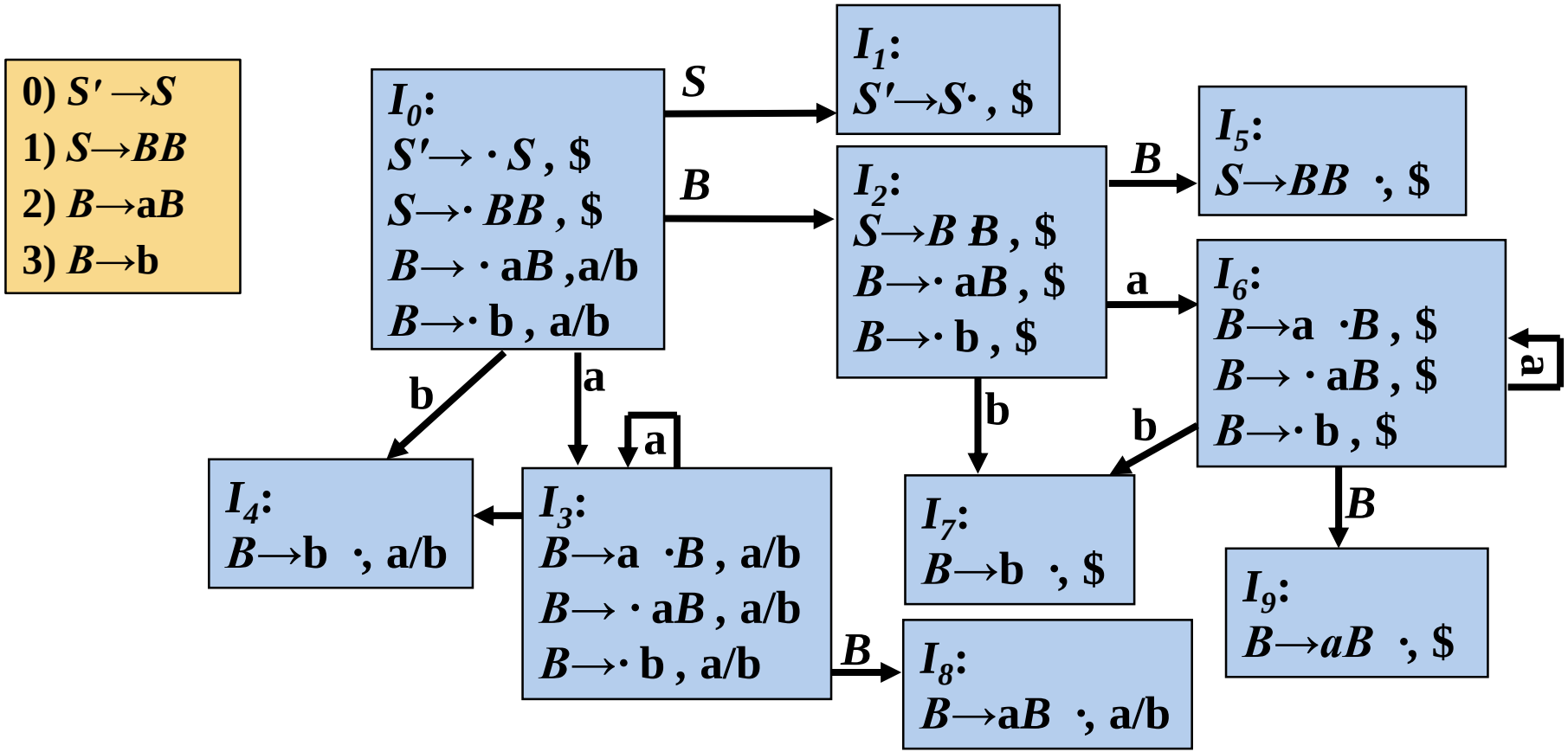
例：合并同心项集



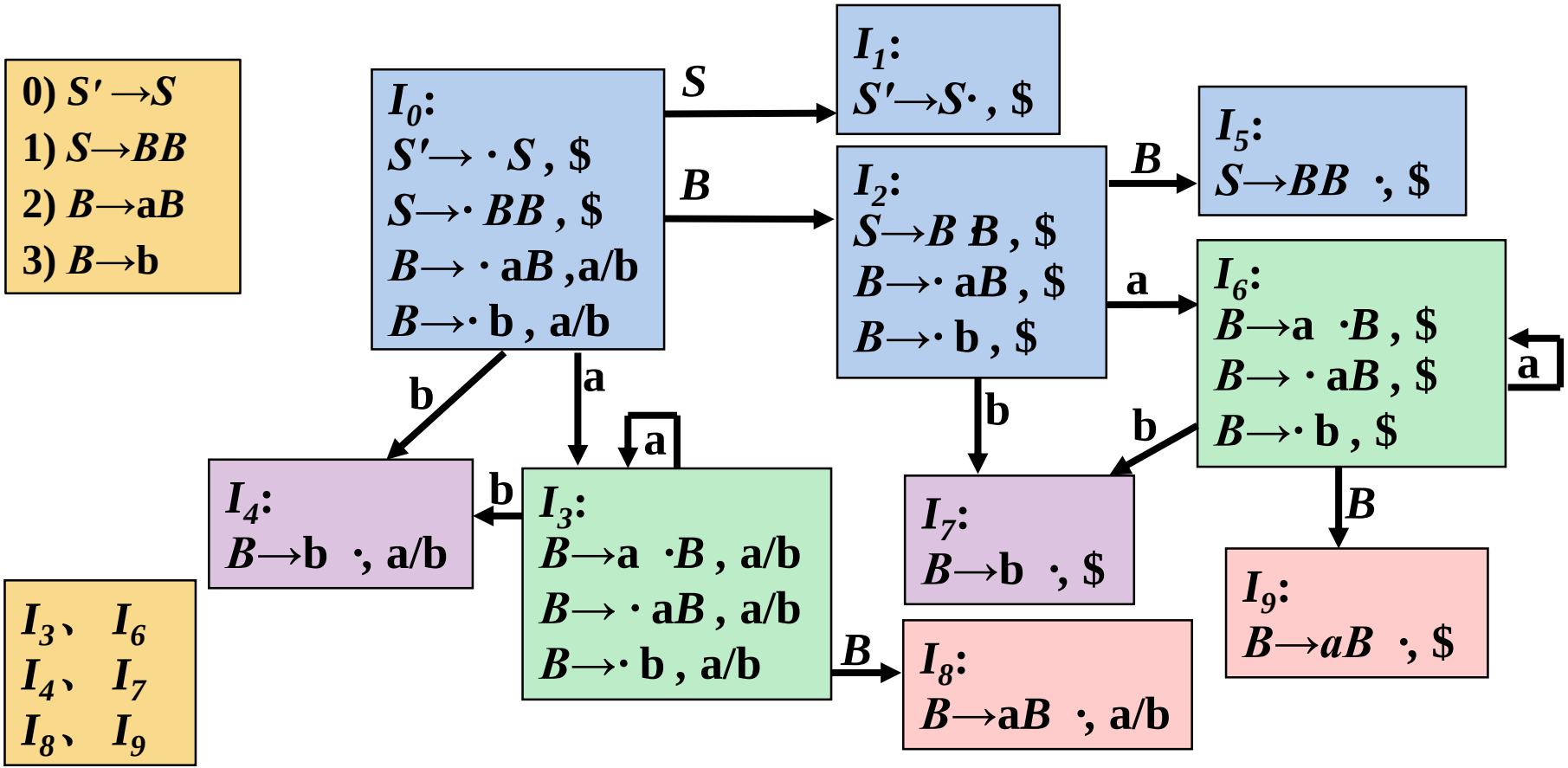
LALR分析表

状态	ACTION				GOTO		
	*	id	=	$	S	L	R
0	s4	s5			1	2	3
1				acc			
2			s6	r5			
3				r2			
4	s4	s5				8	7
5			r4	r4			
6	s4	s5				8	9
7			r3	r3			
8			r5	r5			
9				r1			

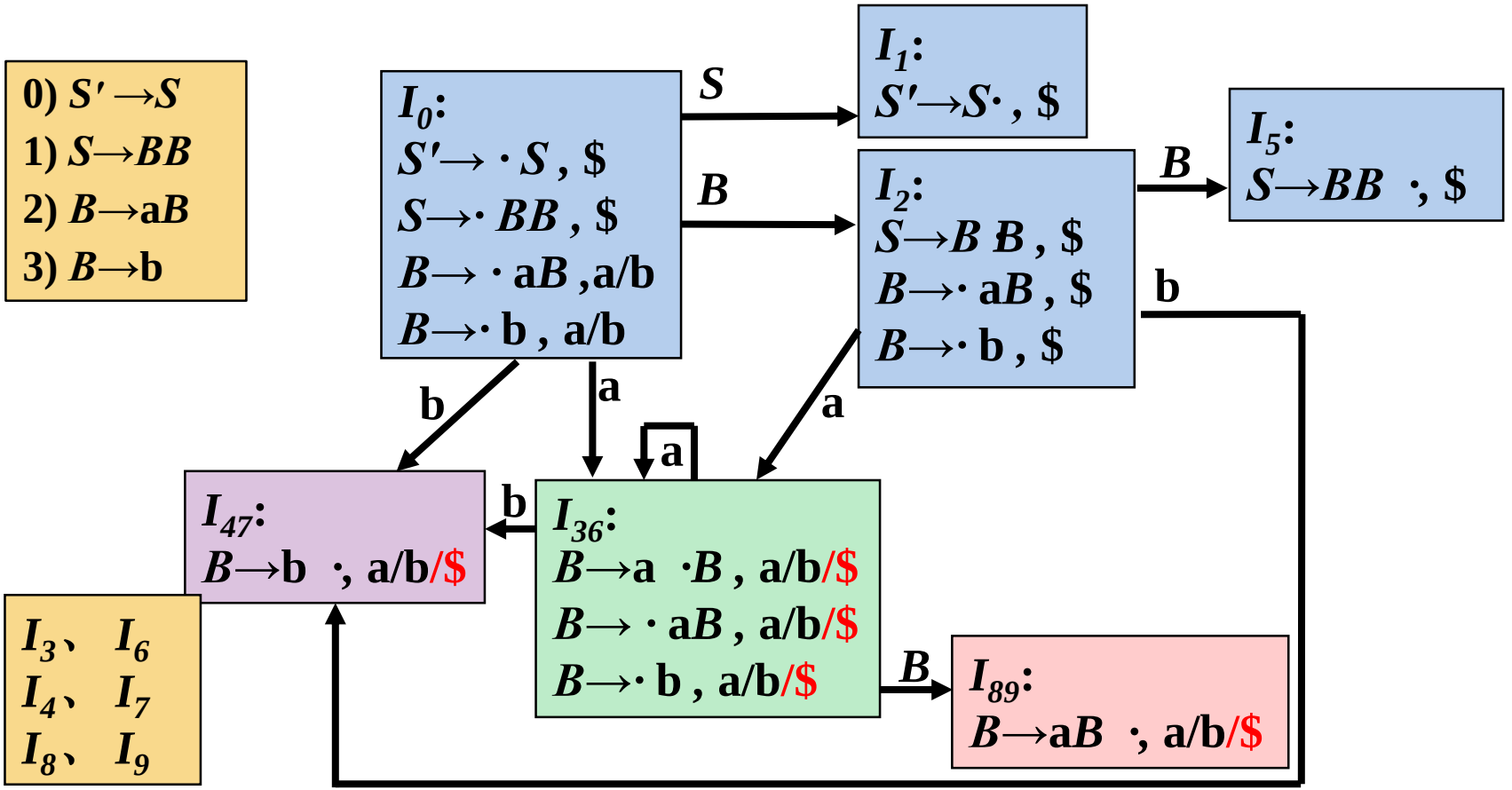
课堂练习-构造*LR*(1)项目集DFA



课堂练习-合并$LR(1)$同心项目集

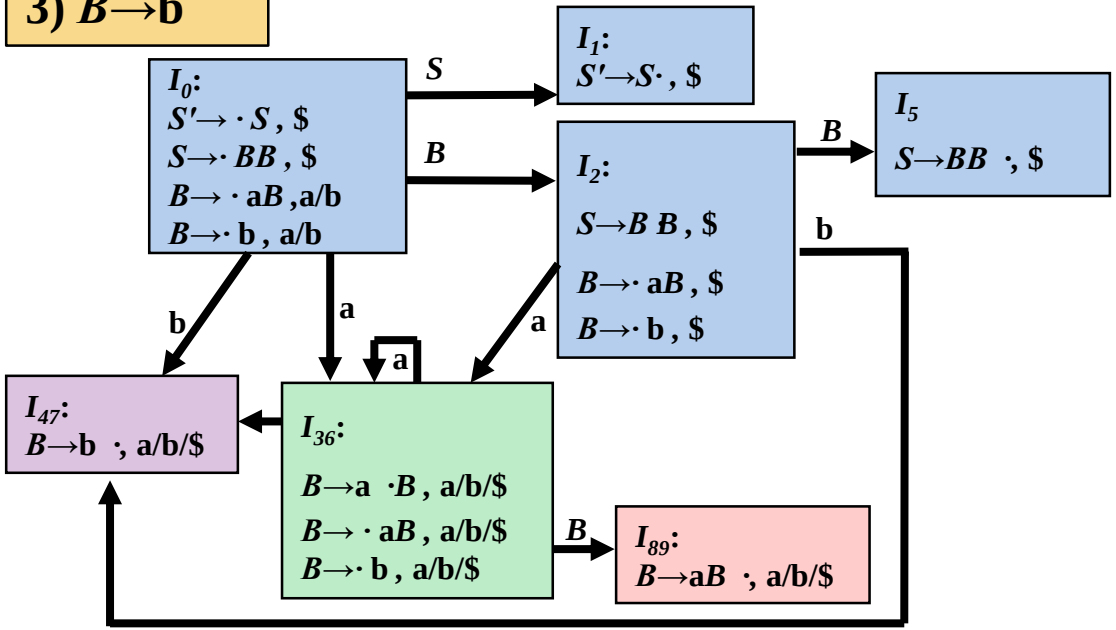


课堂练习-合并$LR(1)$同心项目集



课堂练习-构造*LALR*分析表

0) $S' \rightarrow S$
1) $S \rightarrow BB$
2) $B \rightarrow aB$
3) $B \rightarrow b$



*LALR*分析表

状态	*ACTION*			*GOTO*	
	a	b	$	*S*	*B*
0	s36	s47		1	2
1			*acc*		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

课堂练习-对输入串aabaab用*LALR进行*分析

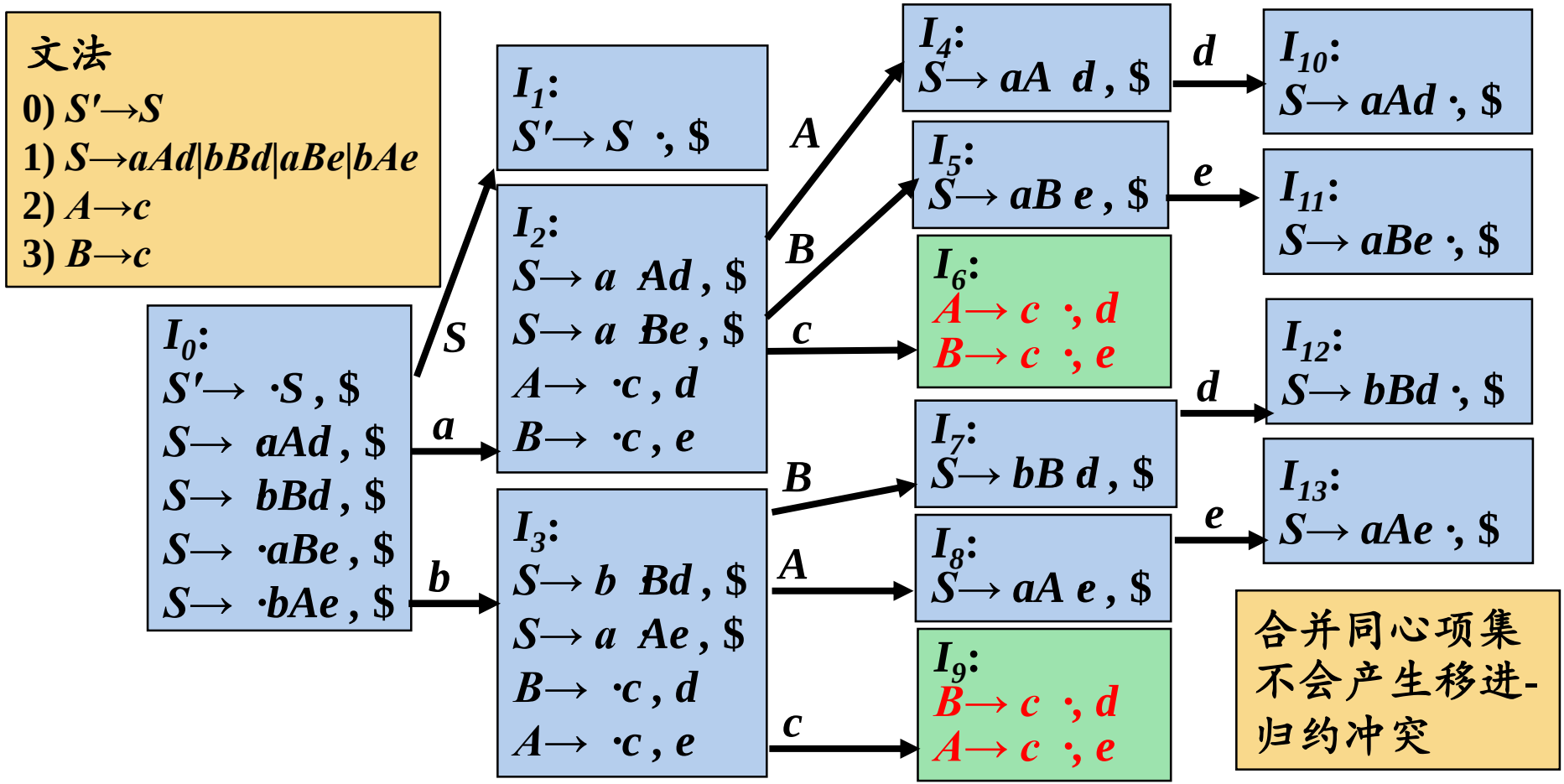
0) $S' \rightarrow S$
1) $S \rightarrow BB$
2) $B \rightarrow aB$
3) $B \rightarrow b$

*LALR*分析表

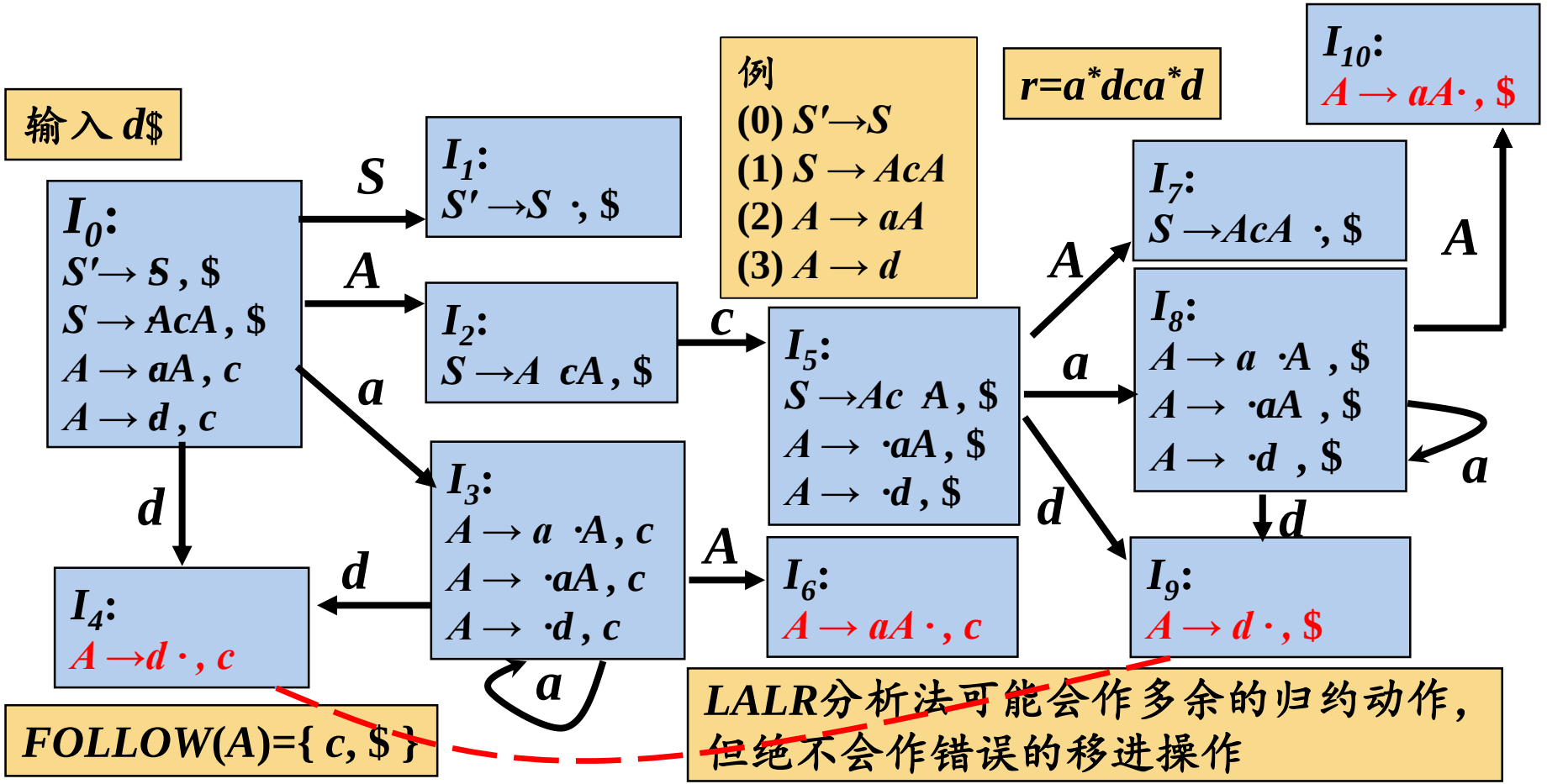
状态	*ACTION*			*GOTO*	
	a	b	$	*S*	*B*
0	s36	s47		1	2
1			*acc*		
2	s36	s47			5
36	s36	s47			89
47	r3	r3	r3		
5			r1		
89	r2	r2	r2		

状态栈	符号栈	输入串	action	goto
0	$	aabaab$	s36	
036	a$	abaab$	s36	
036 36	aa$	baab$	s47	
036 36 47	baa$	aab$	r3	89
036 36 89	Baa$	aab$	r2	89
036 89	Ba$	aab$	r2	2
02	B$	aab$	s36	
0236	aB$	ab$	s36	
0236 36	aaB$	b$	s47	
0236 36 47	baaB$	$	r3	89
0236 36 89	BaaB$	$	r2	89
02 36 89	BaB$	$	r2	5
025	BB$	$	r1	1
01	S$	$	acc	

合并同心项集时产生归约-归约冲突的例子

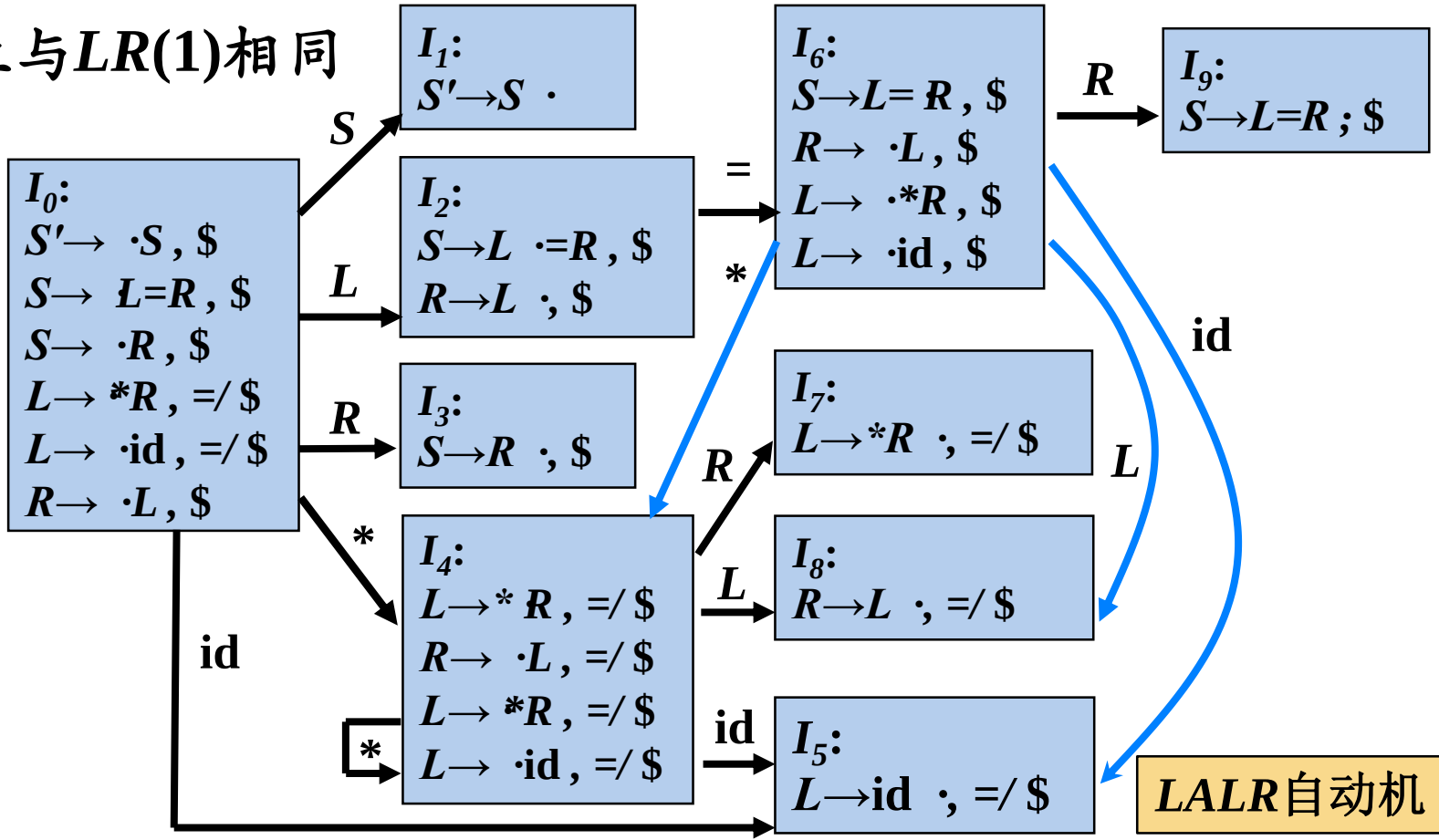


合并同心项集，节省空间，但可能推迟错误的发现



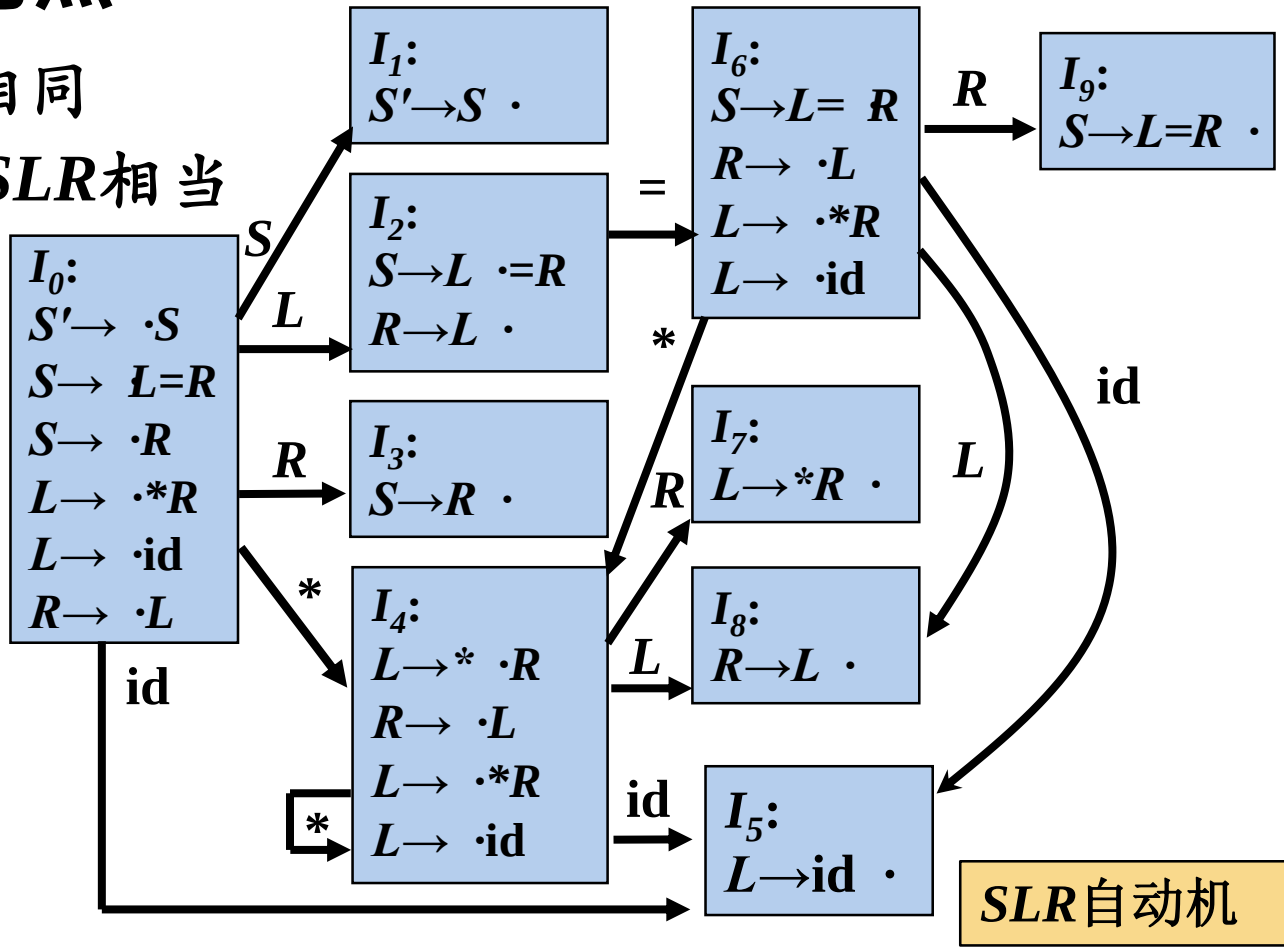
LALR(1)的特点

- 形式上与LR(1)相同



LALR(1)的特点

- 形式上与LR(1)相同
- 大小上与LR(0)/SLR相当



LALR(1)的特点

- 形式上与*LR*(1)相同
- 大小上与*LR*(0)/*SLR*相当
- 分析能力介于*SLR*和*LR*(1)二者之间

$$SLR<LALR(1)<LR(1)$$

- 合并后的展望符集合仍为*FOLLOW*集的子集

Exercise

1一个*LR*(1)文法合并同心集后若不是*LALR*(1)文法(　)。

A.则可能存在移进/归约冲突

B.则可能存在归约/归约冲突

C.则可能存在移进/归约冲突和归约/归约冲突

D.以上说法都不对

Exercise

2若状态k含有项目“A→α·”，且仅当输入符号a∈FOLLOW(A)时，才用规则“A→α”归约的语法分析方法是(　)。

A. LALR分析法

B. R(0)分析法

C. LR(1)分析法

D. SLR(1)分析法

Exercise

3同心集合并可能会产生新的(　　)冲突。

A.二义

B.移进/移进

C.移进/归约

D.归约/归约

Exercise

4就文法的描述能力来说，有(　　)。

A.SLR(1) ⊂ LR(0)

B.LR(1) ⊂ LR(0)

C.SLR(1) ⊂ LR(1)

D.无二义文法 ⊂ LR(1)

Exercise

5在LR(0)的Action表中，如果某行中存在标记为“rj”的栏，则(　　)。

A.该行必定填满“rj”

B.该行未必填满“rj”

C.其他行也有“rj”

D.goto表中也有“rj”

Exercise

6若状态k含有项目“A→α·”，对任意非终结符a，都用规则“A →α”归约的语法分析方法是(　　)。

A.LALR分析法

B.LR(0)分析法

C.LR(1)分析法

D.SLR(1)分析法

Exercise

7在SLR(1)的Action表中，如果某行中存在标记为“rj”的栏，则(　　)。

A.该行必定填满“rj”

B.该行未必填满“rj”

C.其他行也有“rj”

D.goto表中也有“rj”

提纲

二义性文法的特点

- 每个二义性文法都不是*LR*的
- 某些类型的二义性文法在语言的描述和实现中很有用
  - 更简短、更直观
    - 例

二义性文法

① $E \rightarrow E+E$

② $E \rightarrow E*E$

③ $E \rightarrow (E)$

④ $E \rightarrow \mathrm{id}$

非二义性文法

① $E \rightarrow E+T$

② $E \rightarrow T$

③ $T \rightarrow T*F$

④ $T \rightarrow F$

⑤ $F \rightarrow (E)$

⑥ $F \rightarrow \mathrm{id}$

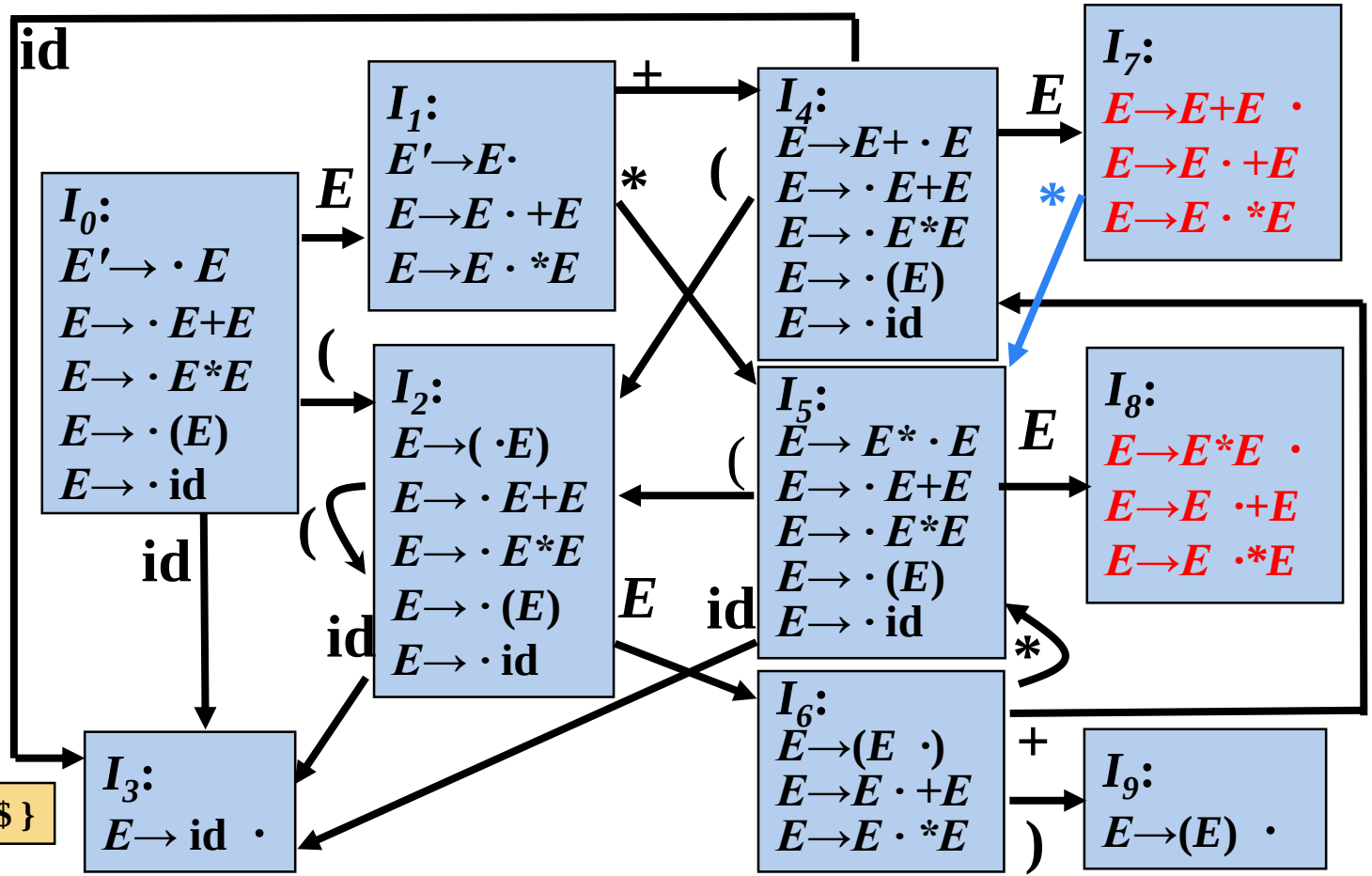
二义性算术表达式文法的$LR(0)$分析器

➢例

文法
$E \to E+E$
$E \to E*E$
$E \to (E)$
$E \to id$

用优先级和结合性解决冲突

$FOLLOW(E) = \{ +, *,), \$ \}$



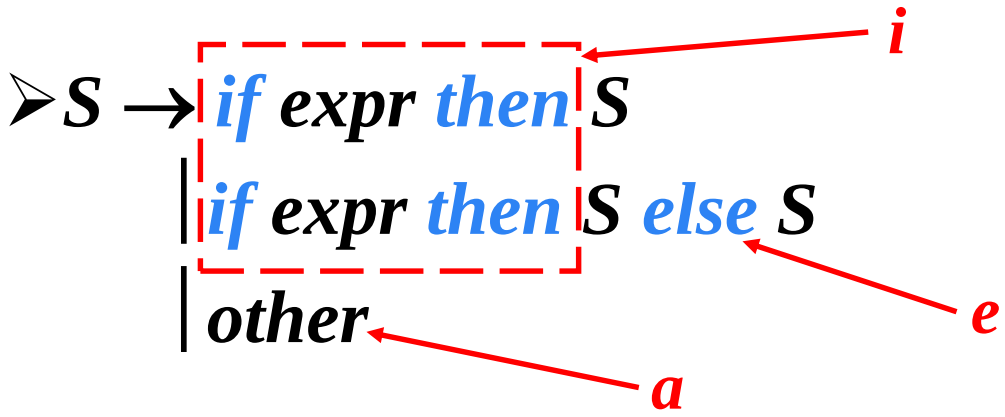
二义性算术表达式文法的*SLR*分析表

文法
$E \rightarrow E+E$
$E \rightarrow E*E$
$E \rightarrow (E)$
$E \rightarrow id$

$FOLLOW(E)=\{ +, *,), \$ \}$

状态	ACTION						GOTO
	id	+	*	(	)	$	E
0	s3			s2			1
1		s4	s5			acc	
2	s3			s2			6
3		r4	r4		r4	r4	
4	s3			s2			7
5	s3			s2			8
6		s4	s5		s9		
7		r1	s5		r1	r1	
8		r2	r2		r2	r2	
9		r3	r3		r3	r3	

例：二义性*if* 语句文法的*LR*分析

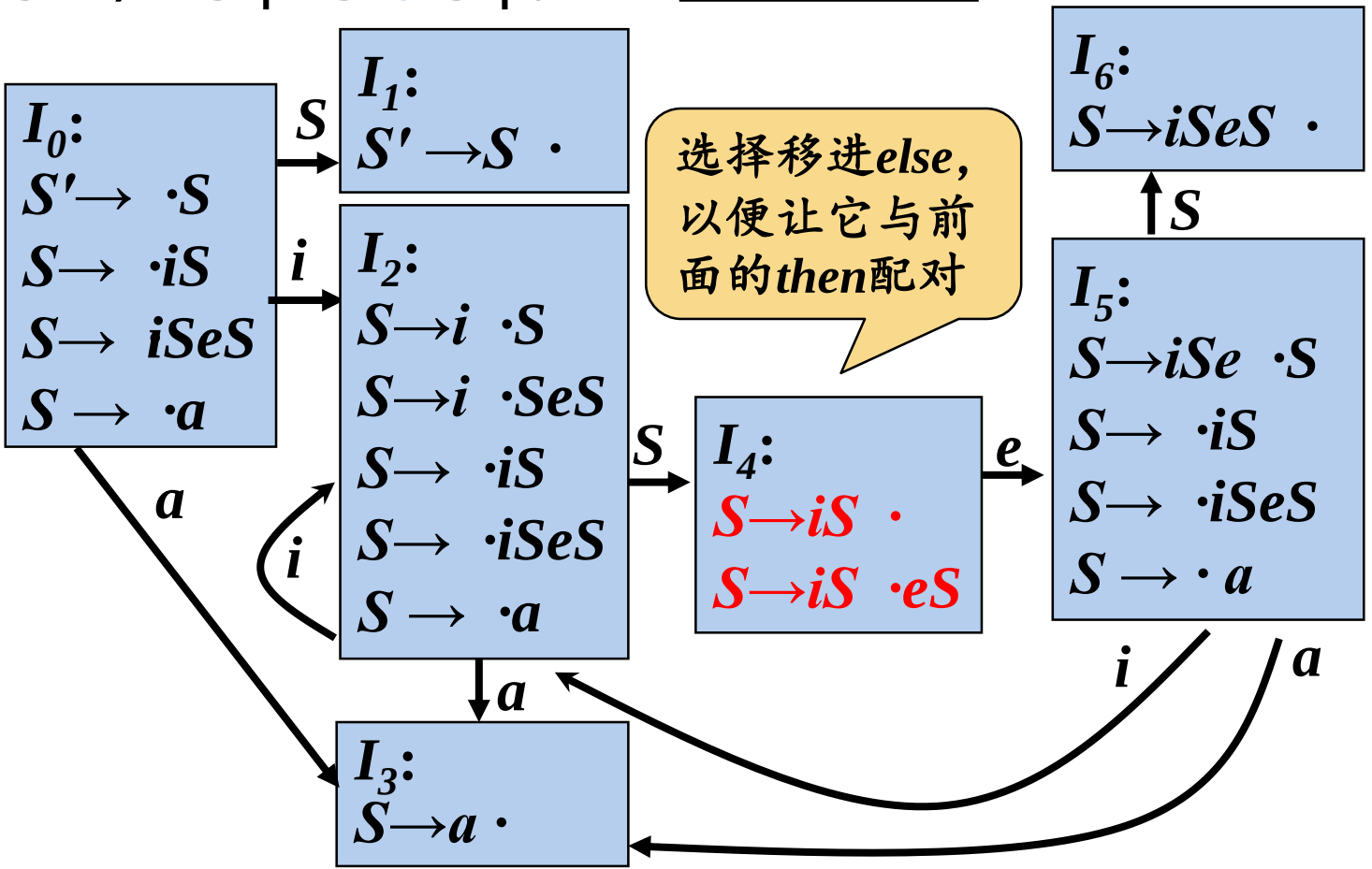


- $S \rightarrow$ *if expr then* S
  | *if expr then* S *else* S
  | *other*

- $S \rightarrow i\ S \mid i\ S\ e\ S \mid a$

$S \rightarrow iS \mid iSeS \mid a$

LR(0) DFA



二义性*if*语句文法的*SLR*分析表

状态	*ACTION*				*GOTO*
	i	e	a	$	*S*
0	s2		s3		1
1				acc	
2	s2		s3		4
3		r3		r3	
4		s5		r1	
5	s2		s3		6
6		r2		r2	

二义性文法的使用

- 应该保守地使用二义性文法，并且必须在严格控制之下使用，因为稍有不慎就会导致语法分析器所识别的语言出现偏差

Exercise

1LR(1)文法都是(　　)。

A.无二义性且无左递归

B.可能有二义性但无左递归

C.无二义性但可能是左递归

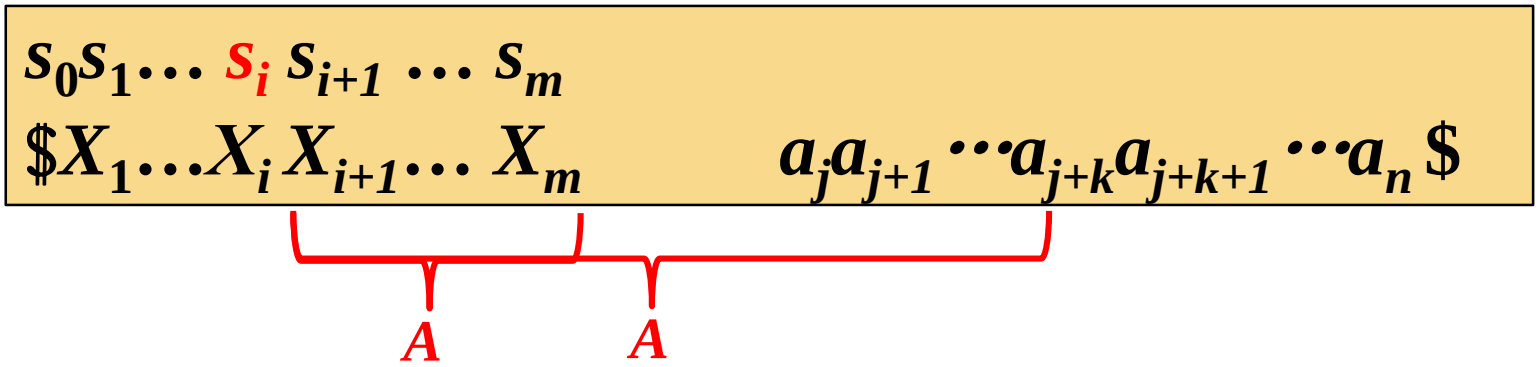
D.可以既有二义性又有左递归

提纲

*LR*分析中的错误处理

- 语法错误的检测
  - 当*LR*分析器在查询分析表并发现一个报错条目时，就检测到了一个语法错误
- 错误恢复策略
  - 恐慌模式错误恢复
  - 短语层次错误恢复

恐慌模式错误恢复

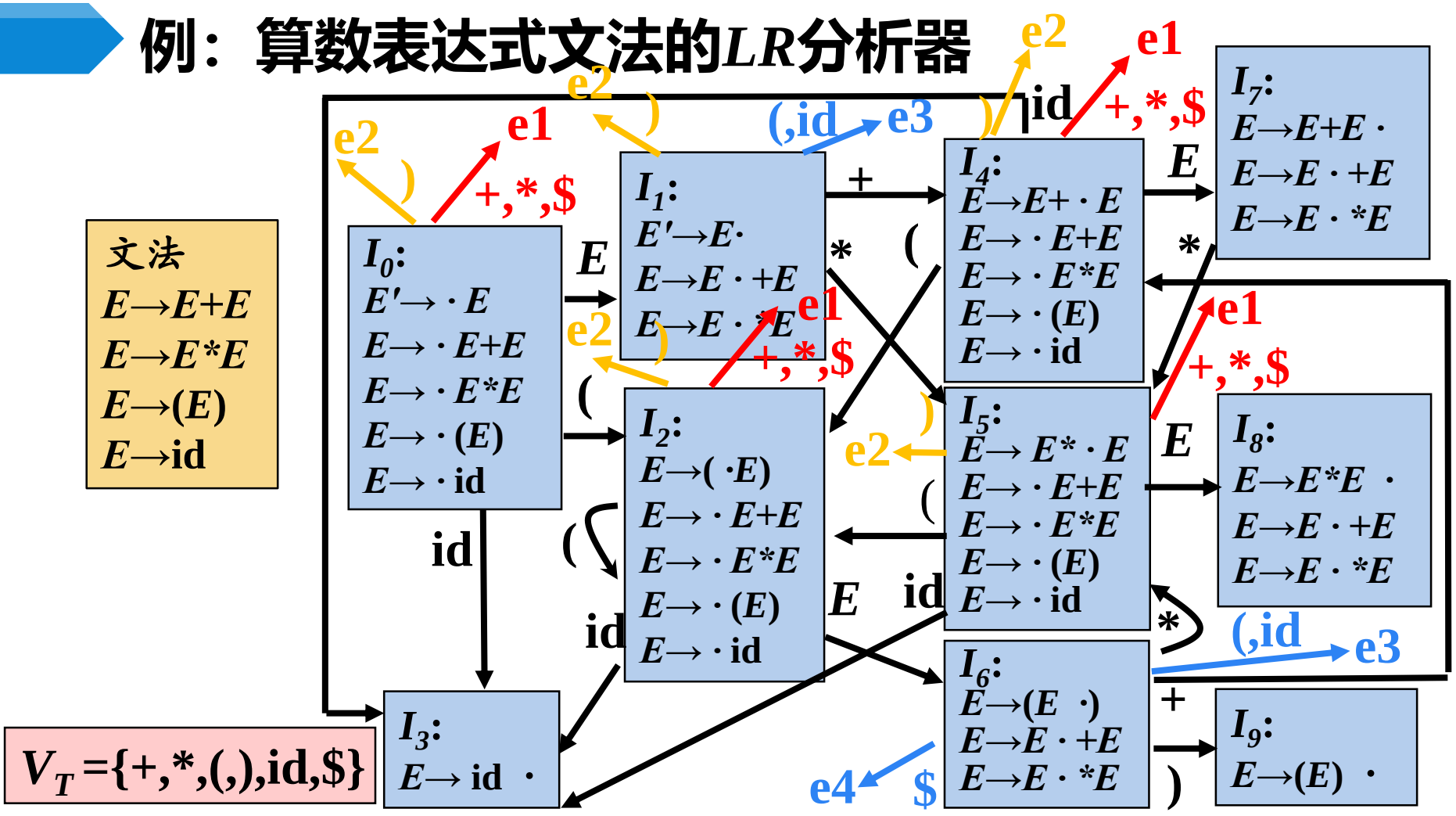


- 从栈顶向下扫描，直到发现某个状态s_i，它有一个对应于某个非终结符A的$GOTO$目标，可以认为从这个A推导出的串中包含错误
- 然后丢弃0个或多个输入符号，直到发现一个可能合法地跟在A之后的符号a为止。
- 之后将$s_{i+1} = GOTO(s_i, A)$压入栈中，继续进行正常的语法分析

短语层次错误恢复

- 检查*LR*分析表中的每一个报错条目，并根据语言的使用方法来决定程序员所犯的何种错误最有可能引起这个语法错误
- 然后构造出适当的恢复过程

例：算数表达式文法的*LR*分析器



带有错误处理子程序的算术表达式文法*LR*分析表

状态	*ACTION*						*GOTO*
	id	+	*	(	)	$	*E*
0	s3	e1	e1	s2	e2	e1	1
1	e3	s4	s5	e3	e2	acc	
2	s3	e1	e1	s2	e2	e1	6
3	r4	r4	r4	r4	r4	r4	
4	s3	e1	e1	s2	e2	e1	7
5	s3	e1	e1	s2	e2	e1	8
6	e3	s4	s5	e3	s9	e4	
7	r1	r1	s5	r1	r1	r1	
8	r2	r2	r2	r2	r2	r2	
9	r3	r3	r3	r3	r3	r3	

- 错误处理例程
  - *e*1：将状态3压入栈中，发出诊断信息“缺少运算分量”
  - *e*2：从输入中删除“)”，发出诊断信息“不匹配的右括号”
  - *e*3：将状态4压入栈中，发出诊断信息“缺少运算符”
  - *e*4：将状态9压入栈中，发出诊断信息“缺少右括号”

自底向上语法分析小结

	Top_down	Bottom_up
通用框架	递归下降分析	移入-规约分析
主要问题	候选式冲突	冲突：规约-规约冲突；移入-规约冲突

例：规约-规约冲突

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

例：移入-规约冲突

文法：

(1) $E \rightarrow E+T$

(2) $E \rightarrow T$

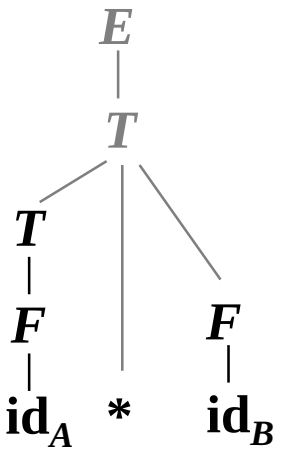
(3) $T \rightarrow T*F$

(4) $T \rightarrow F$

(5) $F \rightarrow (E)$

(6) $F \rightarrow \mathrm{id}$

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



自底向上语法分析小结

	Top_down	Bottom_up
通用框架	递归下降分析	移入-规约分析
主要问题	候选式冲突	冲突：规约-规约冲突；移入-规约冲突
关键问题	如何正确选择候选式	如何正确识别句柄（每次应该归约的符号串）

移入-规约:是句柄就规约,不是就移入
规约-规约:哪个是句柄就用哪个规约

如何正确地识别句柄？-- *LR*分析法

- 基本原理
  - 句柄是逐步形成的，用“状态”表示句柄识别的进展程度，状态是用项目来描述的
    - 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

*LR*分析器基于由项目集表示的状态构造自动机进行句柄的识别

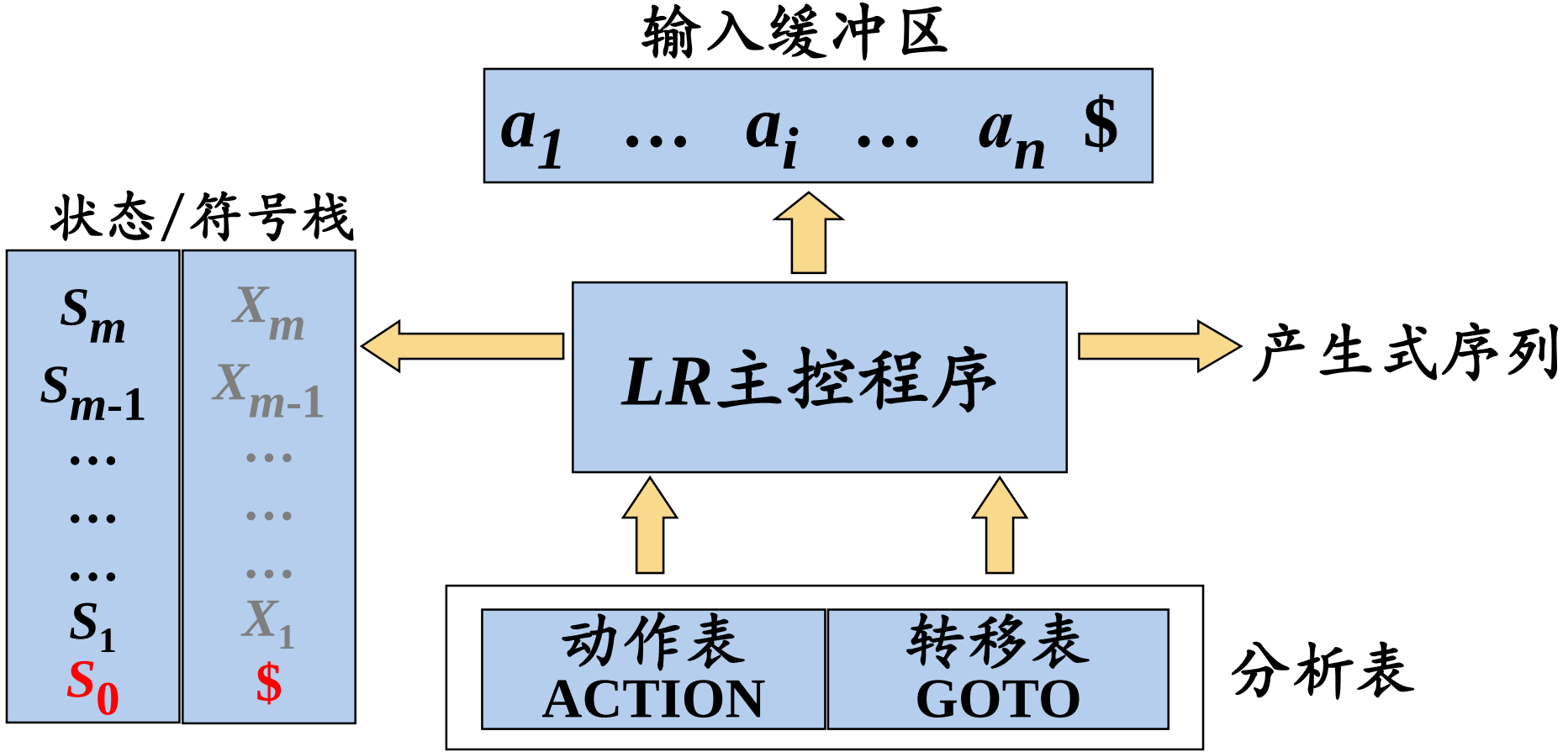
等价项目

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

(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) $I\rightarrow I,i\cdot$
(0) $S'\rightarrow \cdot S$
(1) $S'\rightarrow S\cdot$
(11) $I\rightarrow \cdot i$
(12) $I\rightarrow i\cdot$
(13) $T\rightarrow \cdot r$
(14) $T\rightarrow r\cdot$

把等价的项目组成一个项目集I，项目集作为状态来构造自动机

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		

sn： 将符号*a*、状态*n* 压入栈
rn： 用第*n*个产生式进行归约

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

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

$LR(0)$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* $A \to \alpha\ \mathrm{a}\beta \in I_i$ *and* $GOTO(I_i, \mathrm{a}) = I_j$ *then* $ACTION[i, \mathrm{a}] = \mathrm{sj}$
  - *if* $A \to \alpha.B\beta \in I_i$ *and* $GOTO(I_i, B) = I_j$ *then* $GOTO[i, B] = \mathrm{j}$
  - *if* $A \to \alpha \cdot \in I_i$ 且$A \neq S'$ *then for* $\forall \mathrm{a} \in V_T \cup \{\$\}$ *do* $ACTION[i, \mathrm{a}] = \mathrm{rj}$
    （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*”

各种LR分析表构造方法的不同之处在于规约项目的处理上

- *if* $[A \rightarrow \alpha\ ;\ c] \in I_i$

$A = S'$ ： $ACTION[\ i, \$\ ] = acc$

$A \neq S'$ ：

- **LR(0)** *for* $\forall a \in V_T \cup \{\$\}$ *do* $ACTION[\ i, a\ ] = r_j$ (j是产生式的编号)
- **SLR(1)** *for* $\forall a \in FOLLOW(A)$ *do* $ACTION[\ i, a\ ] = r_j$
- **LR(1)** 精准规约 $ACTION[\ i, a\ ] = r_j$

LR(1)分析实际上是根据后继符集合的不同，
将原始的LR(0)状态分裂成不同的LR(1)状态

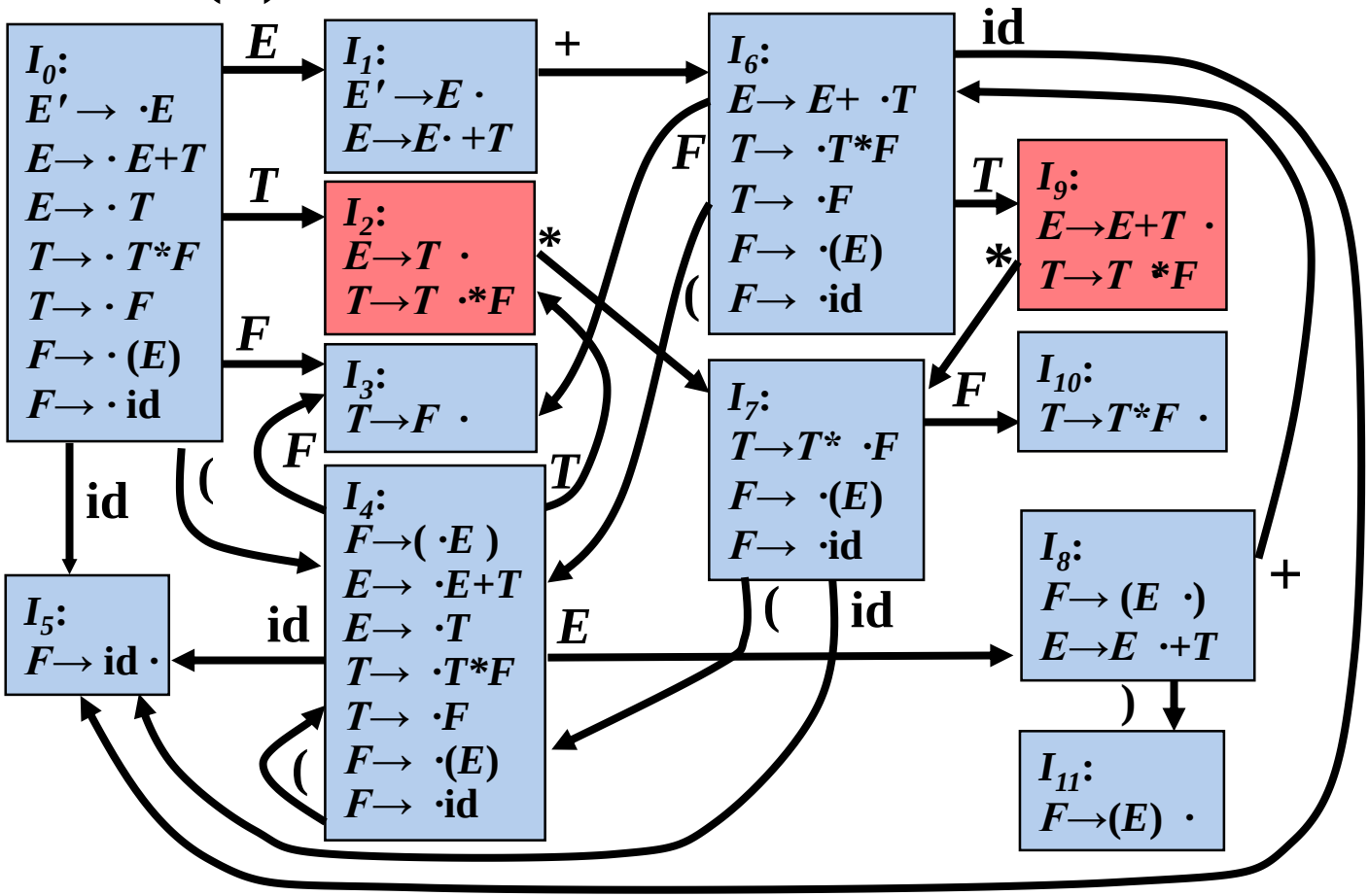
LR (0) → SLR (1)

例：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

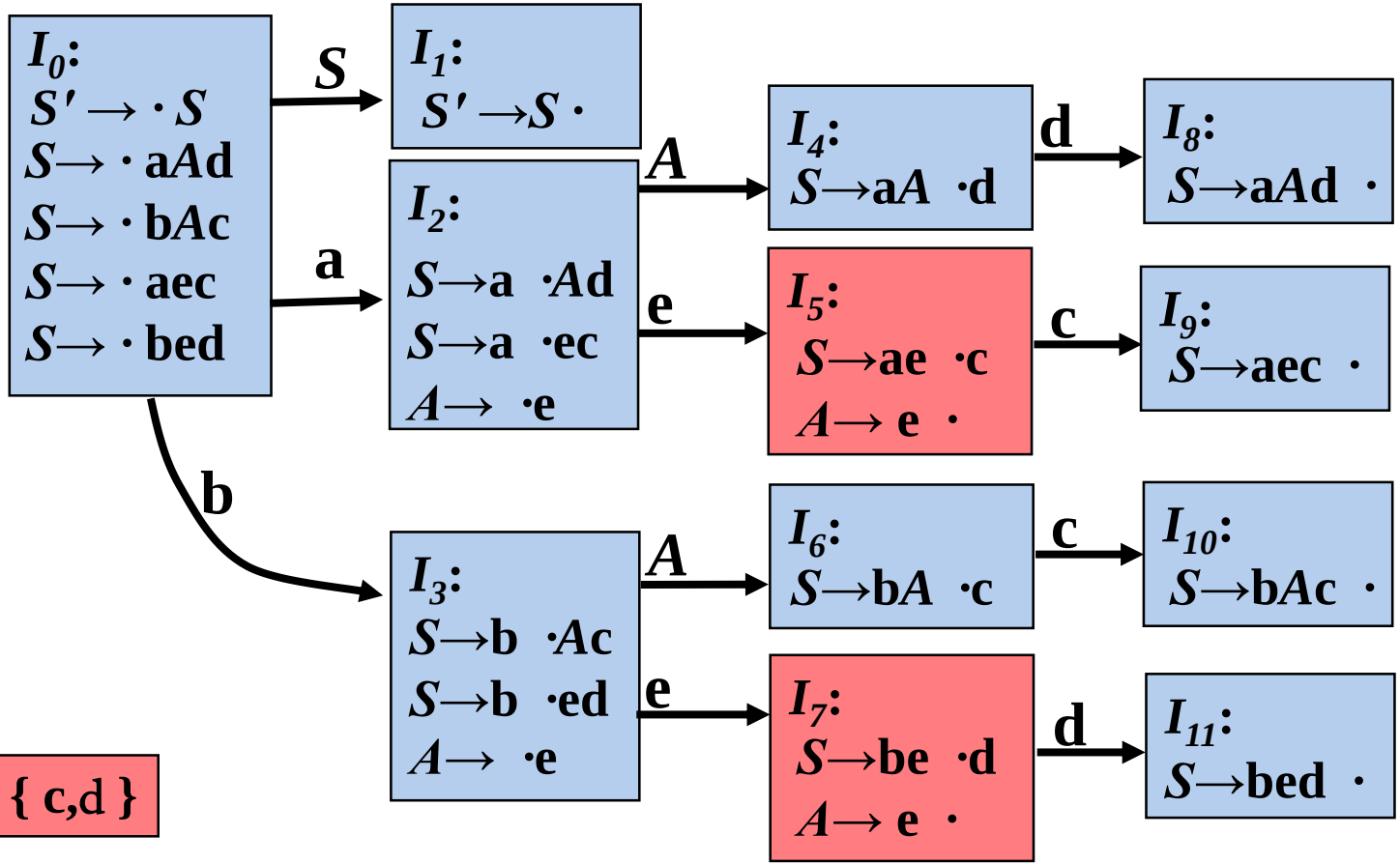
X	FOLLOW(X)
E	), +, $
T	), +, $, *
F	), +, $, *



SLR (1) → LR (1)

➢例

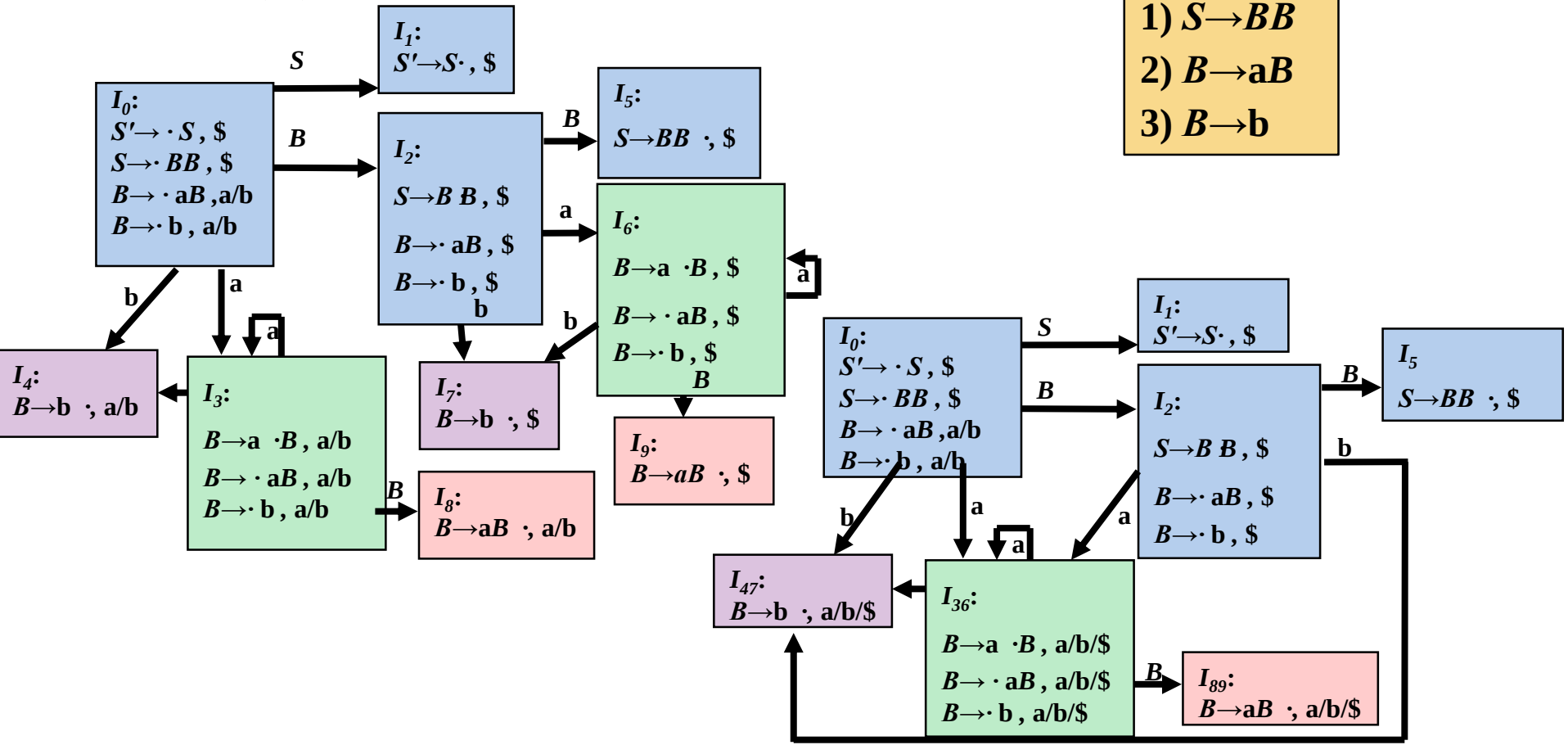
0) S′→S
1) S→aAd
2) S→bAc
3) S→aec
4) S→bed
5) A→e

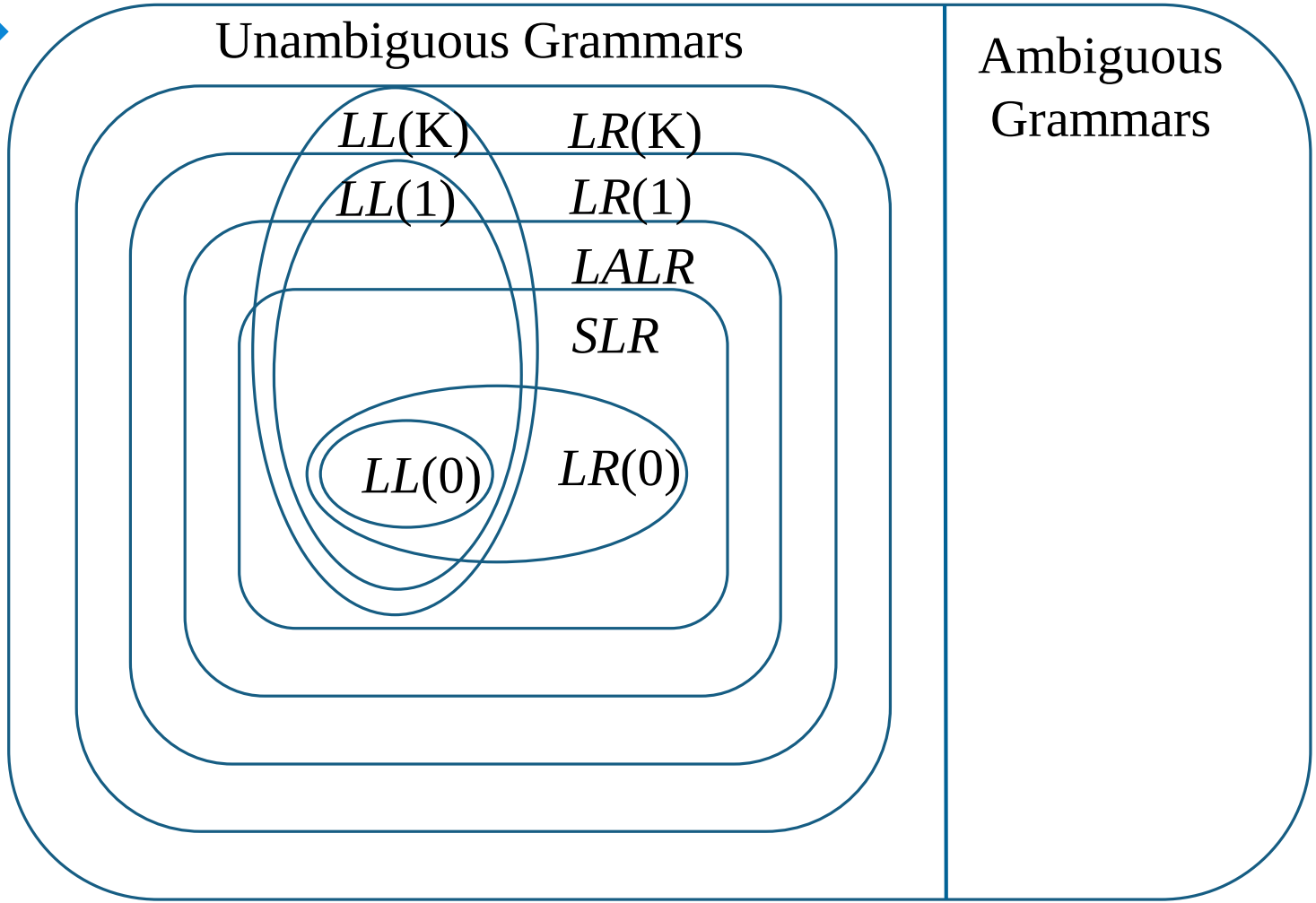


FOLLOW(A) = { c,d }

LR (1) → LALR

0) $S' \rightarrow S$
1) $S \rightarrow BB$
2) $B \rightarrow aB$
3) $B \rightarrow b$




Unambiguous Grammars
Ambiguous Grammars
LL(K)
LR(K)
LL(1)
LR(1)
LALR
SLR
LL(0)
LR(0)

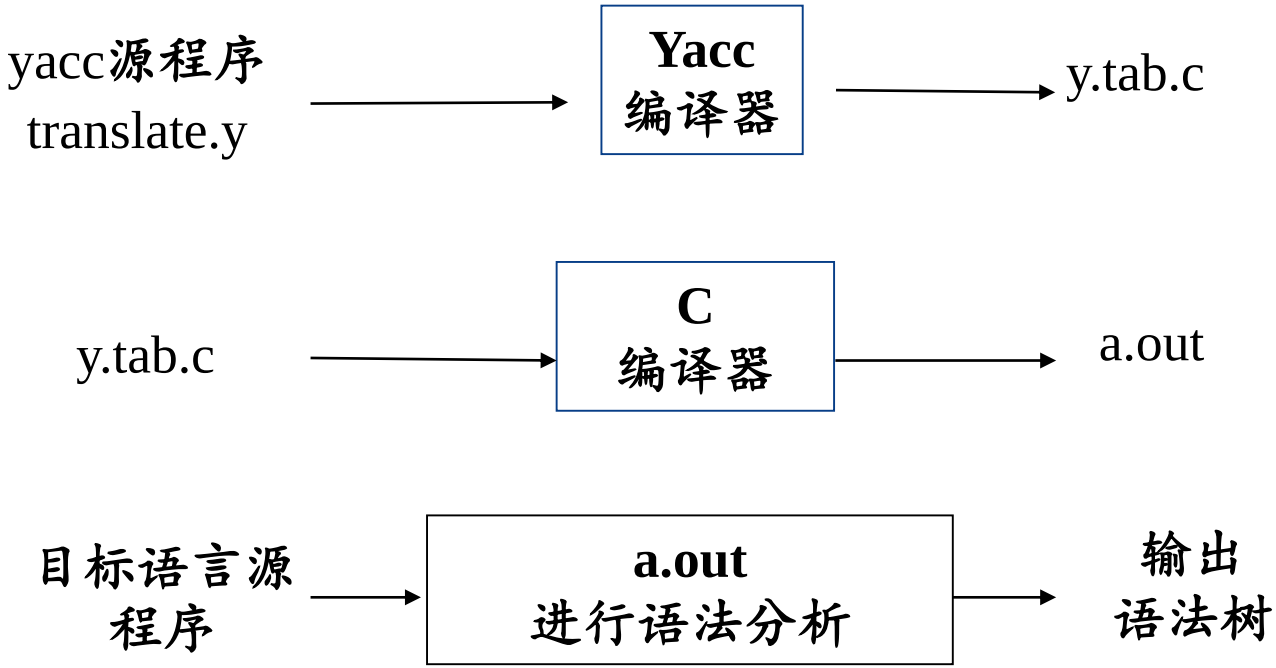
文法类的谱系

语法分析程序的自动构造工具Yacc

- YACC（Yet Another Compiler-Compiler）是美国贝尔实验室开发的语法分析程序自动生成器。
- 输入是某个语言的语法规则，输出是该语言的语法分析器。目前YACC生成的是一个LALR（1）分析器。
- 常用的版本：Berkeley大学的BYACC，GNU工程的BISON。

语法分析程序的自动构造工具Yacc

一. 用生成器Yacc构造翻译器的过程



语法分析程序的自动构造工具Yacc

二．Yacc源程序有三部分组成

声明

%%

翻译规则

%%

C写的子程序

三．　例

G[E]:　E→E+T | T

T →T*F | F

F →(E) | digit

读入一个表达式，计算它的值并输出。

语法分析程序的自动构造工具Yacc

```
%{
# include   <ctype.h>
%}
% token   digit

%%
line      :   expr´\n´        {printf("%d\n",$1);}
          ;
expr      :   expr´+´term       {$$=$1+$3;}
          :   term
          ;
```

语法分析程序的自动构造工具Yacc

```
term      :   term ´*´facter    {$$=$1*$3;}
          :   facter
          ;
facter    :   ´(´expr ´)´       {$$=$2;}
          :   digit
          ;
%%
yylex(){  int  c;
           c=getchar();
           if  (isdigit( c ){  yylval=c- ´0´;
                                 return  digit   ;     }
            return   c  ;      }
```

语法分析程序的自动构造工具Yacc

声明部分

有任选的两节。

第一节处于分界符%{和%}之间，它是一些普通的C的声明；第二节是文法记号的声明。

翻译规则部分 每条翻译规则由一个文法产生式和有关的语义动作组成。

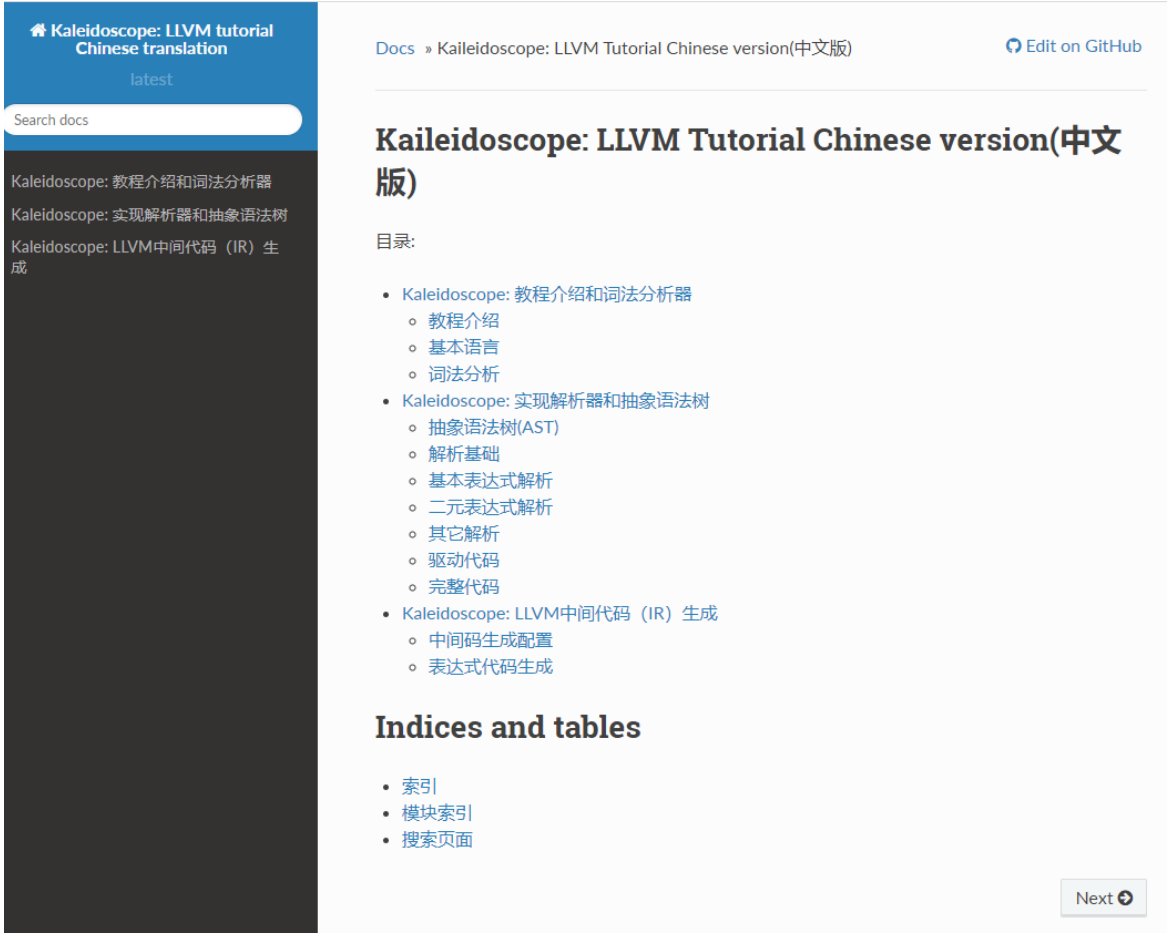
支持例程部分 一些C写的子程序。**例：**词法分析器，错误恢复例程等。

典型学习案例

https://llvm-tutorial-cn.readthedocs.io/en/latest/index.html

典型学习案例

https://kaleidoscope-llvm-tutorial-zh-cn.readthedocs.io/zh_CN/latest/



Exercise

1.文法G[S]为：

S→AB

A→aBa | ε

B→bAb | ε

该文法是SLR(1)文法吗？若是，构造其分析表并给出输入串baab$的分析过程。

2.若有文法G[S]：

S→S;M | M

M→MbD | D

D→D(S) | ε

给出G[S]的LR(1)项目集规范族中的I_0。

3.给定文法G[S]：

S→AdD | ε

A→aAd | ε

D→DdA | b | ε

(1)证明G[S]不是LR(0)和SLR(1)文法。

(2)判断G[S]是LR(1)和LALR(1)文法，并构造相应的分析表。