4.4.2 SLR 分析

例：LR(0) 分析过程中的冲突

文法：

(0) $E' \rightarrow E$

(1) $E \rightarrow E+T$

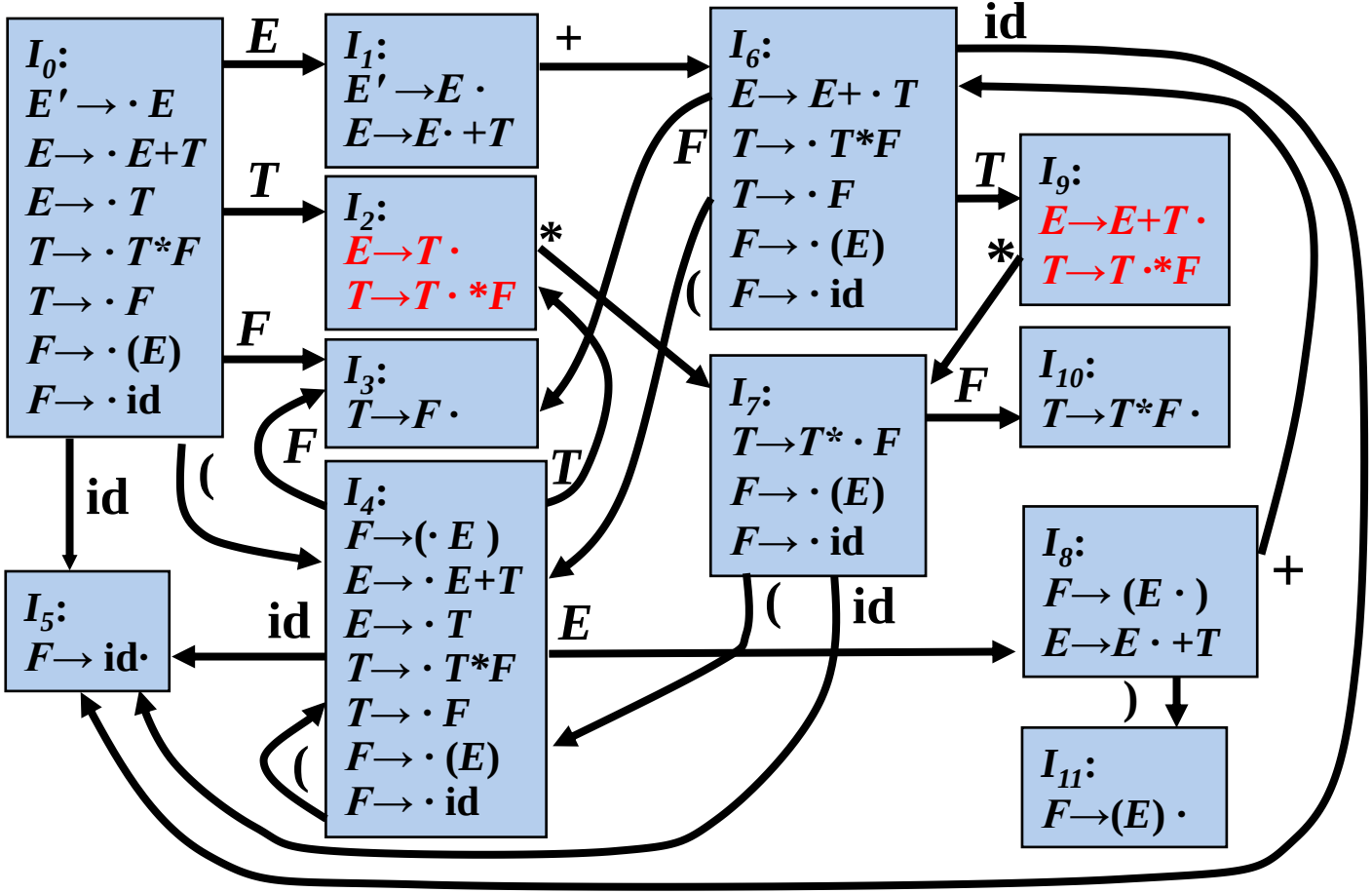
(2) $E \rightarrow T$

(3) $T \rightarrow T*F$

(4) $T \rightarrow F$

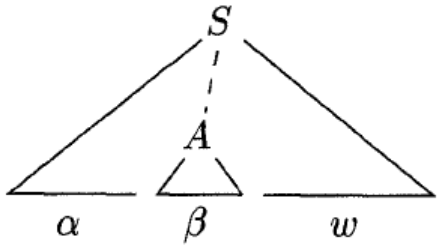
(5) $F \rightarrow (E)$

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



LR(0)自动机为什么消解不了某些冲突?

- 句柄都是相对一个句型而言的，因此应该将句柄的识别放在句型这样一个上下文环境中考虑
- 对于$A \rightarrow \beta$，是否应该将β归约为A取决于当前句型对应的分析树中A所处的上下文环境



- LR(0)考虑了A的上文（规范句型的前缀），但未考虑A的下文，因此消解冲突能力有限

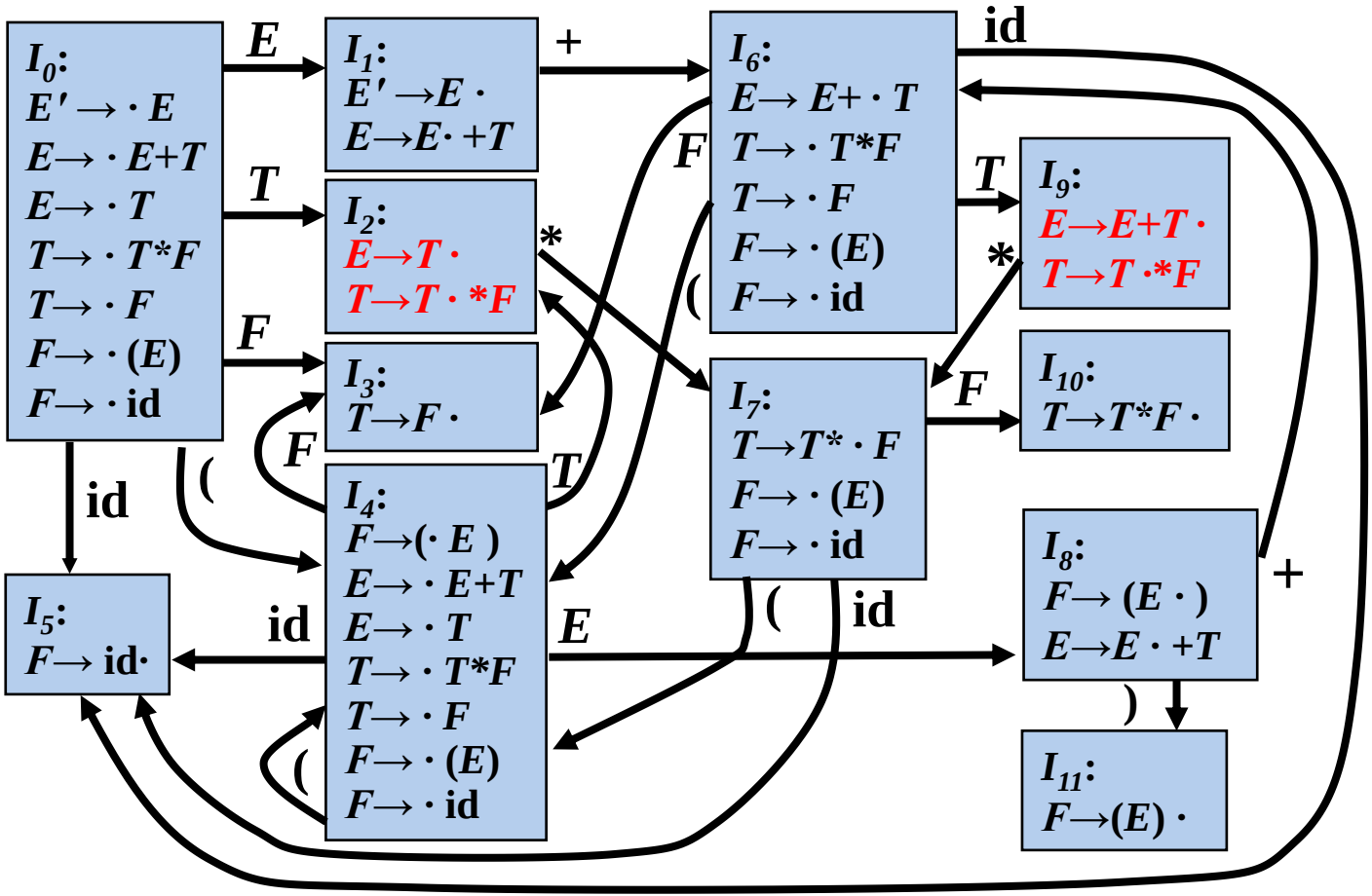
SLR 分析

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 分析

➢SLR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

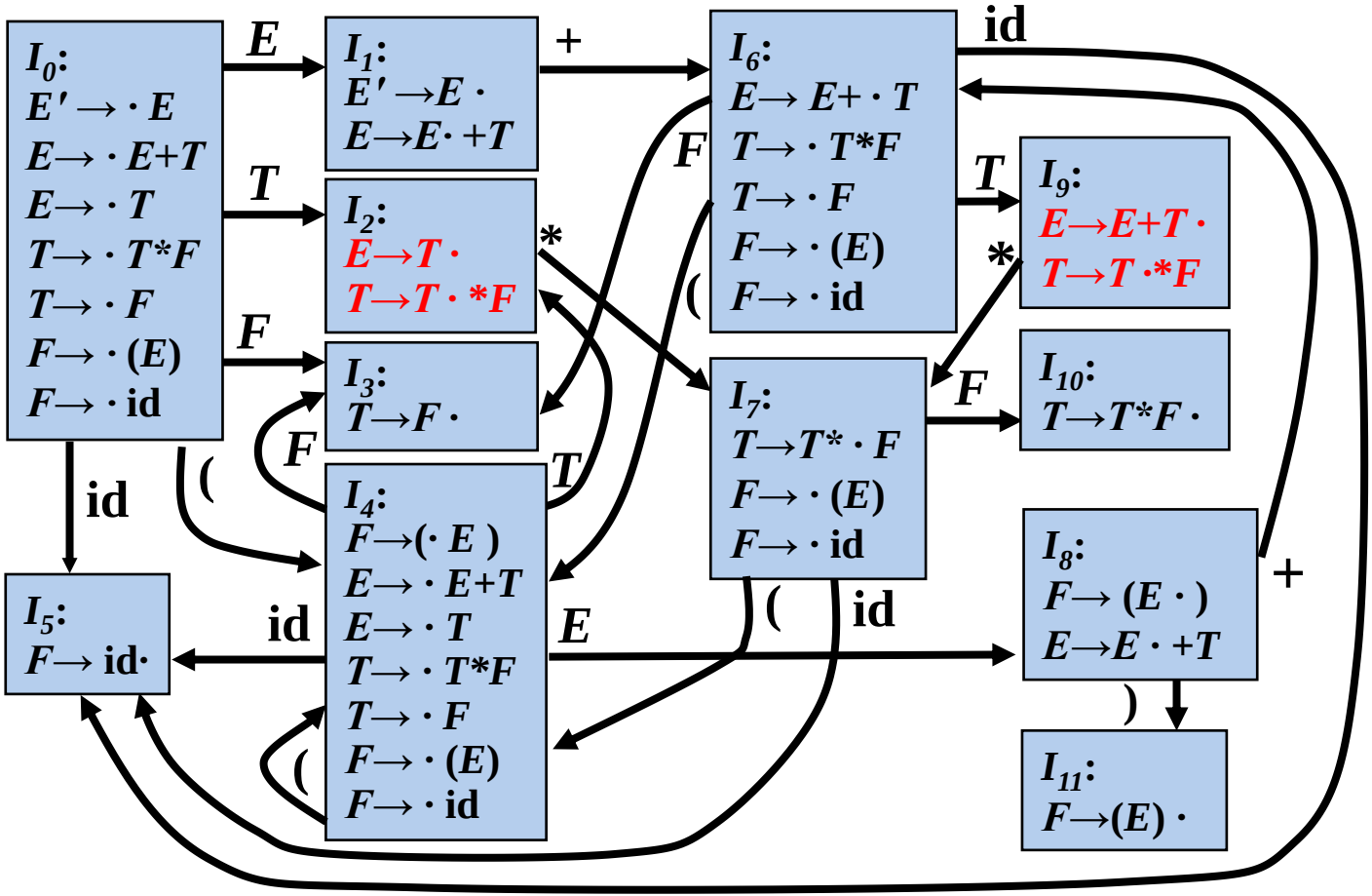
- 若$a \in \{a_1, a_2, \ldots, a_m\}$，则移进a
- 若$a \in FOLLOW(B_i)$，则用产生式$B_i \rightarrow \gamma_i$归约
- 此外，报错

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	), +, $, *



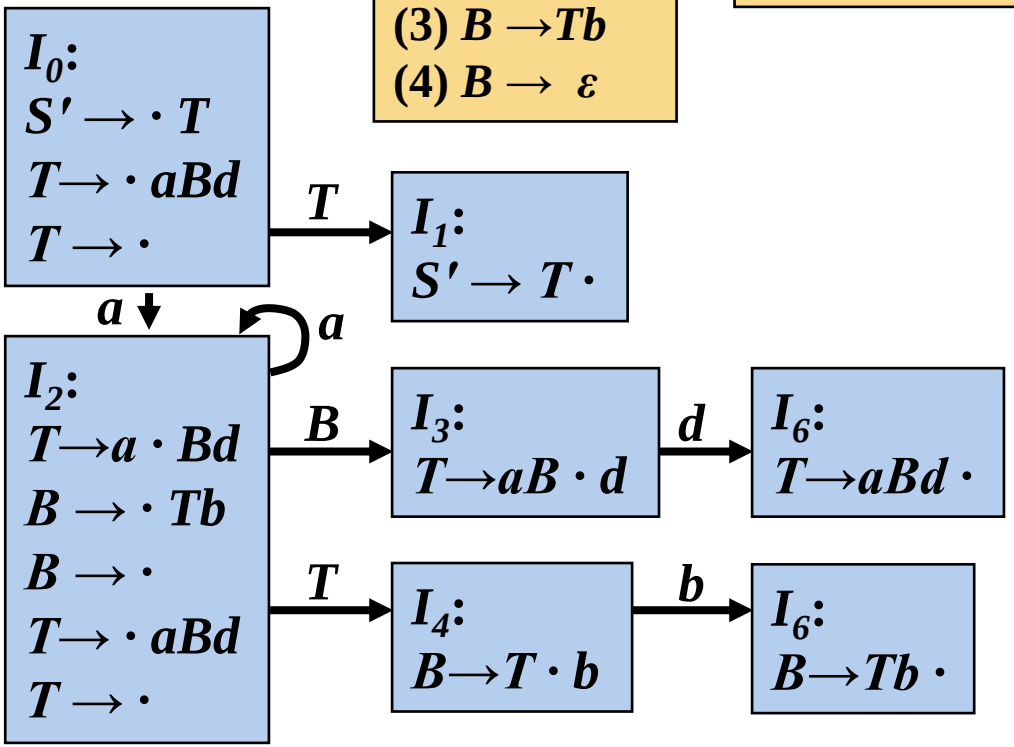
表达式文法的*SLR*分析表

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 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 $A \rightarrow \alpha \cdot a\beta \in I_i$ 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 $\forall a \in FOLLOW(A)$ 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”。

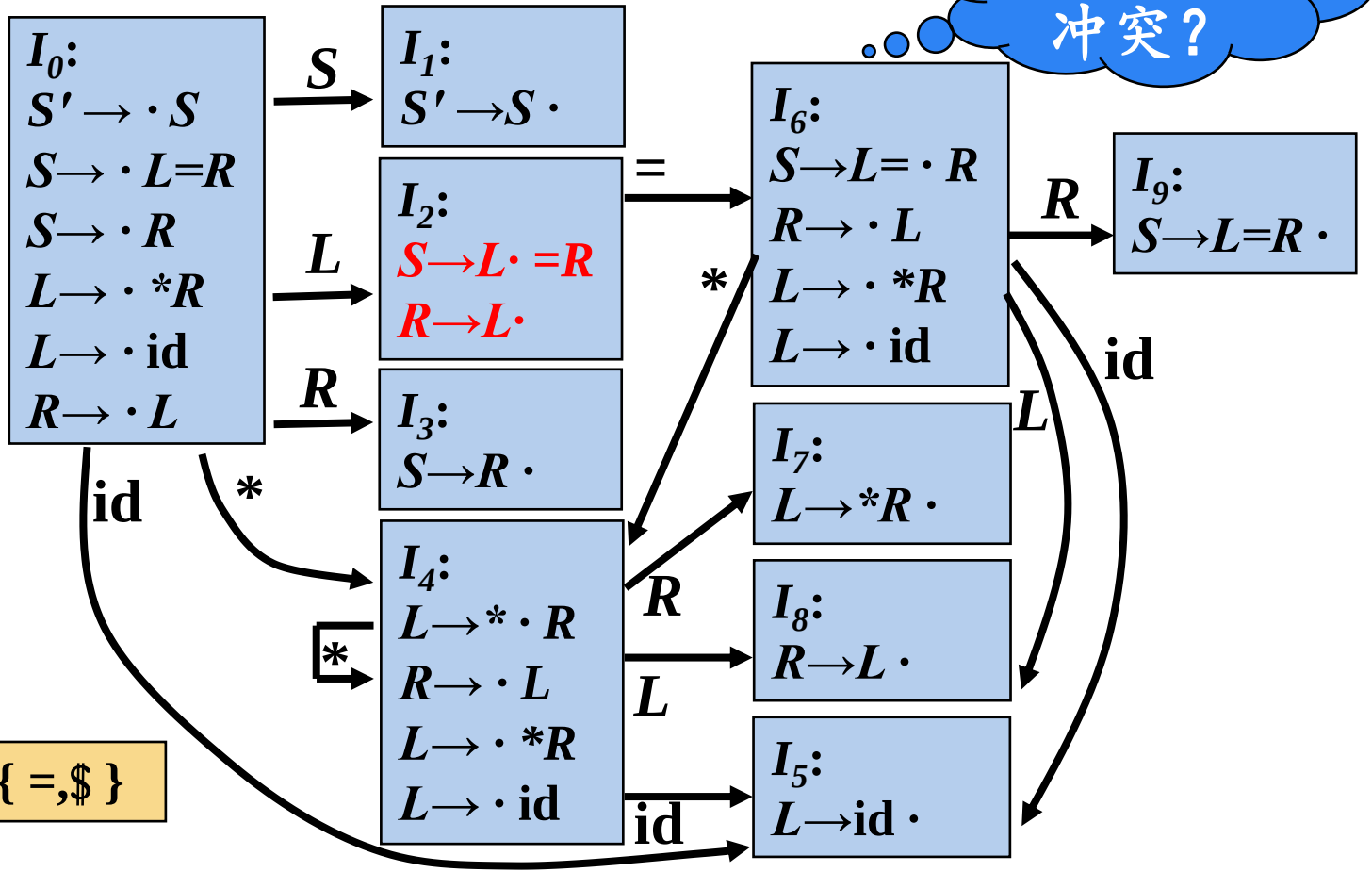
如果给定文法的SLR分析表中不存在有冲突的动作，
那么该文法称为SLR文法

SLR 分析中的冲突

如何消解冲突？

➢例

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



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

4.4.3 LR(1)分析

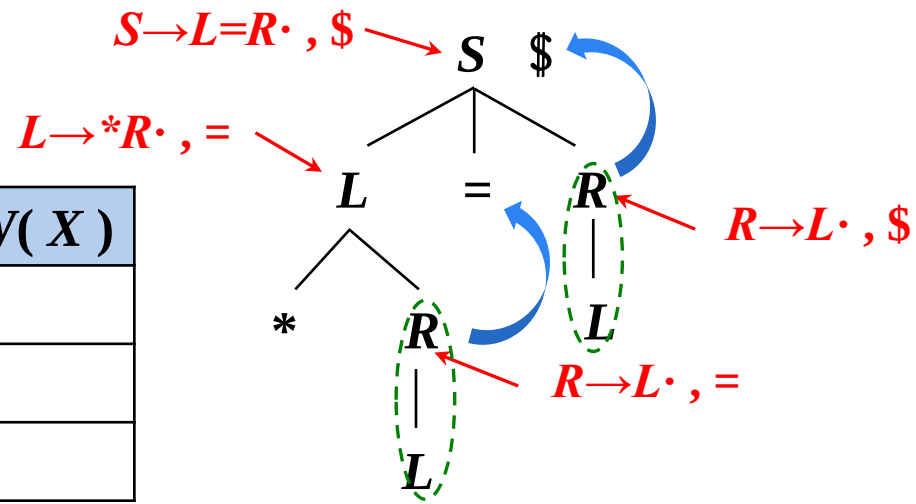
- SLR分析存在的问题
  - SLR只是简单地考察下一个输入符号b是否属于与归约项目$A\rightarrow\alpha$相关联的FOLLOW(A)，但$b\in FOLLOW(A)$只是归约α的一个必要条件，而非充分条件

LR(1)分析法的提出

➢对于产生式A→α的归约，在不同的使用位置，A会要求不同的后继符号

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

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



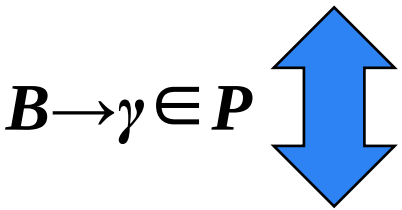
➢在特定位置，A的后继符集合是FOLLOW(A)的子集

规范*LR*(1)项目

- 将一般形式为 [$A \rightarrow \alpha \cdot \beta$, a]的项称为 ***LR*(1) 项**，其中$A \rightarrow \alpha\beta$ 是一个产生式，a 是一个终结符(这里将$视为一个特殊的终结符)它表示在当前状态下，*A*后面必须紧跟的终结符，称为该项的展望符(*lookahead*)
  - *LR*(1) 中的1指的是项的第二个分量的长度
  - 在形如[$A \rightarrow \alpha \cdot \beta$, a]且$\beta \neq \varepsilon$的项中，展望符a没有任何作用
  - 但是一个形如[$A \rightarrow \alpha \cdot$, a]的项在只有在下一个输入符号等于a时才可以按照$A \rightarrow \alpha$ 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

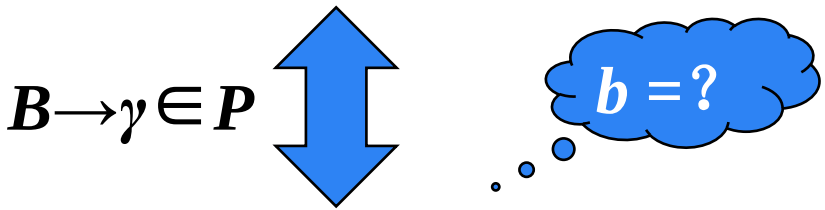
$[A \rightarrow \alpha \cdot B\beta, \mathbf{a}]$



?

等价$LR(1)$项目

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



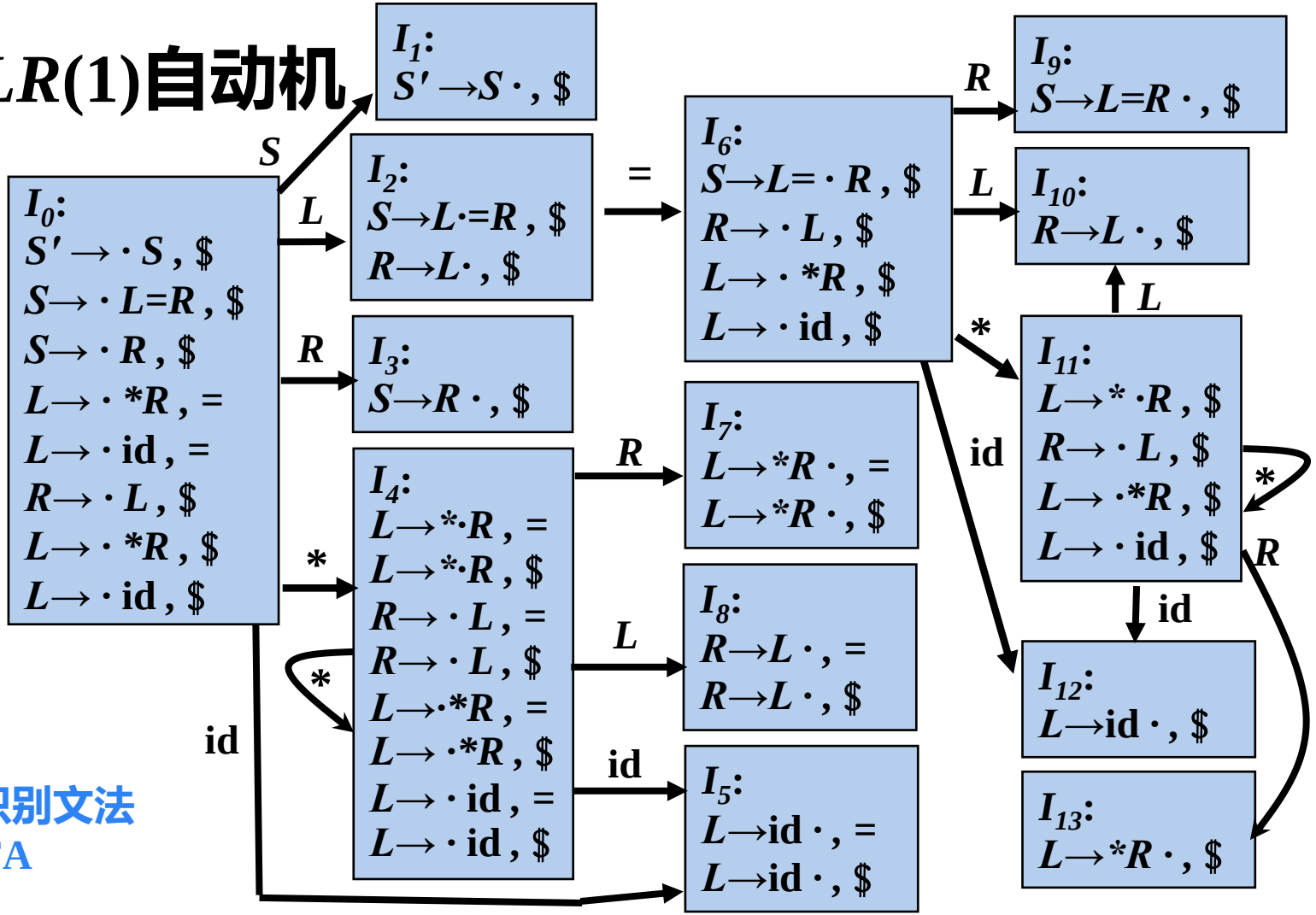
$[B \to \cdot\gamma, b]$

$b \in FIRST(\beta a)$

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

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) $R \rightarrow L$



基于LR(1)项目识别文法全部活前缀的DFA

赋值语句文法的 *LR*(1)分析表

文法

0) $S' \rightarrow S$

1) $S \rightarrow L=R$

2) $S \rightarrow R$

3) $L \rightarrow *R$

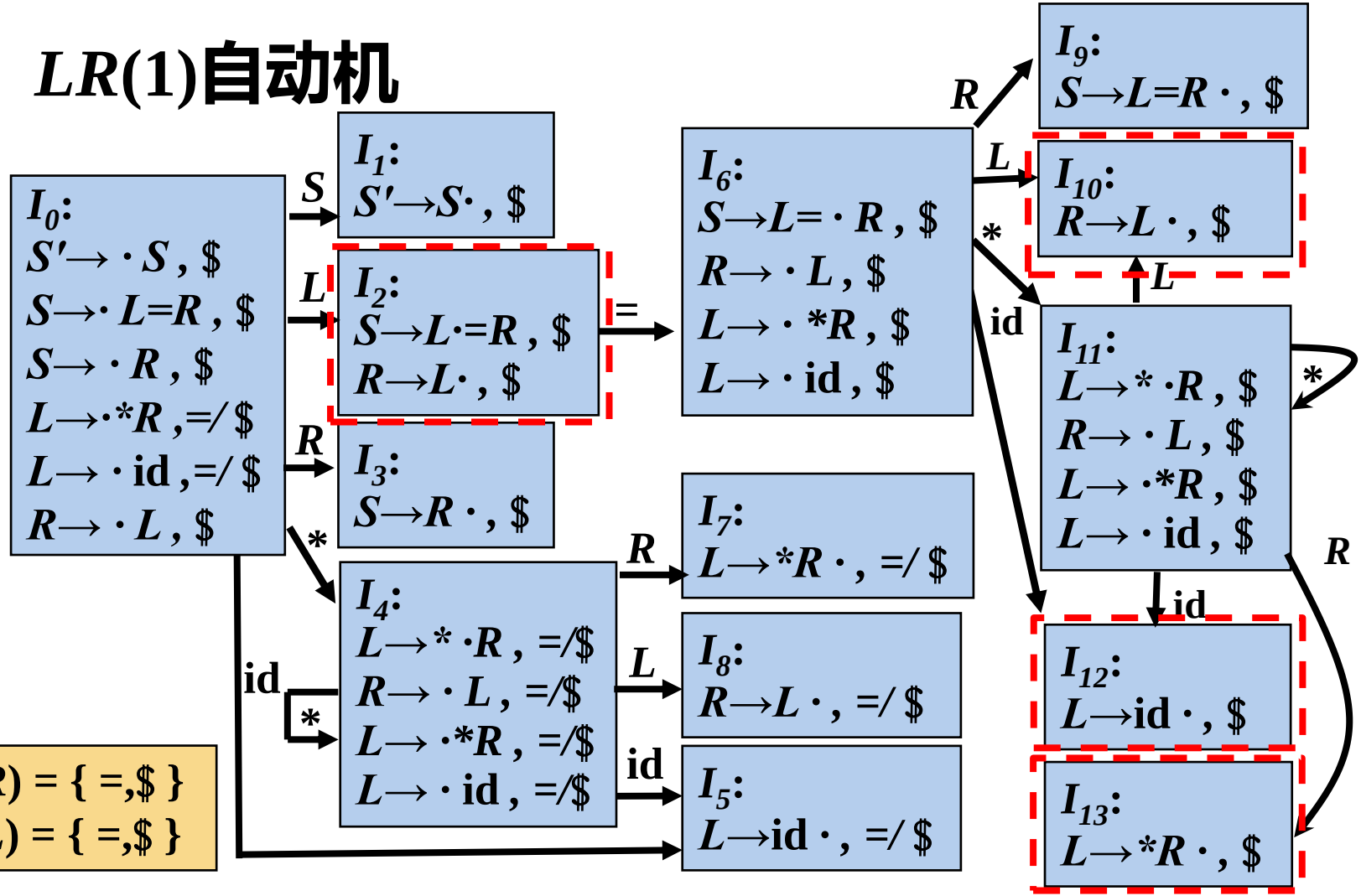
4) $L \rightarrow$ id

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			

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) = { =,# }
FOLLOW(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) $R \rightarrow L$

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

I_1:
$S' \rightarrow S \cdot$, $\$$

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

I_3:
$S \rightarrow R \cdot$, $\$$

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

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

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

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

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

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

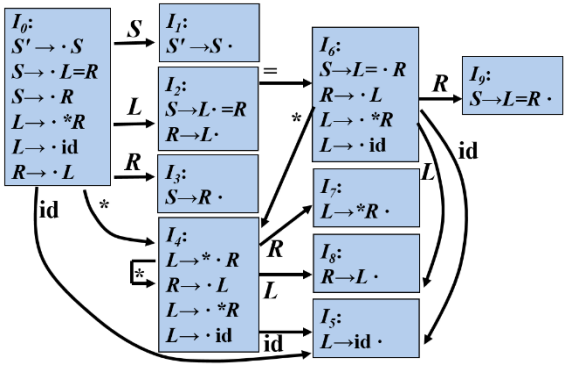
I_{10}:
$R \rightarrow L \cdot$, $\$$

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

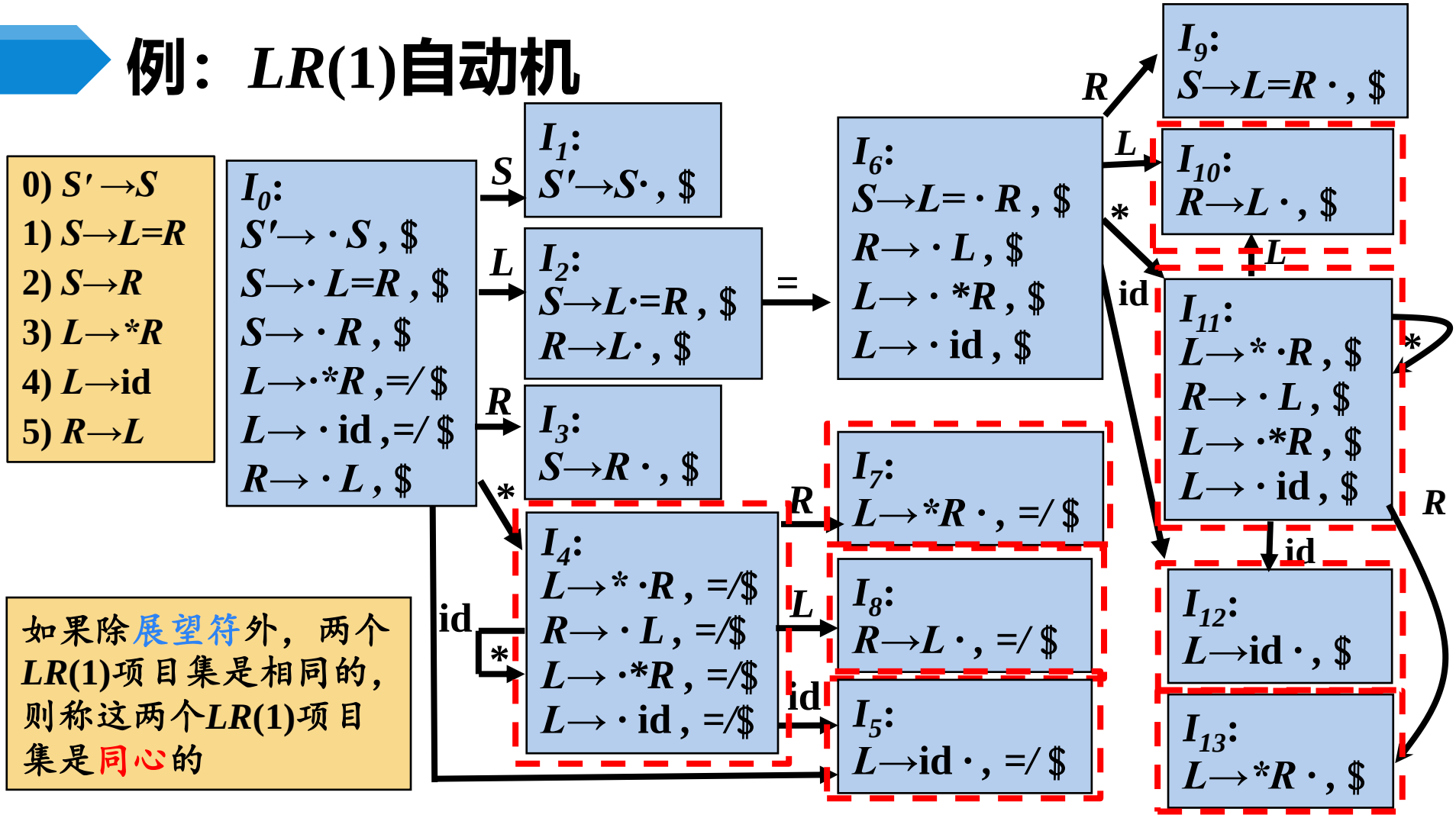
I_{12}:
$L \rightarrow id \cdot$, $\$$

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

SLR



例：LR(1)自动机


0) S′→S
1) S→L=R
2) S→R
3) L→*R
4) L→id
5) R→L
I0: S′→·S, $; S→·L=R, $; S→·R, $; L→·*R, =/$; L→·id, =/$; R→·L, $
I1: S′→S·, $
I2: S→L·=R, $; R→L·, $
I3: S→R·, $
I4: L→*·R, =/$; R→·L, =/$; L→·*R, =/$; L→·id, =/$
I5: L→id·, =/$
I6: S→L=·R, $; R→·L, $; L→·*R, $; L→·id, $
I7: L→*R·, =/$
I8: R→L·, =/$
I9: S→L=R·, $
I10: R→L·, $
I11: L→*·R, $; R→·L, $; L→·*R, $; L→·id, $
I12: L→id·, $
I13: L→*R·, $
S
L
R
*
id
=
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

LR(1)项目集闭包

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

```
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

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

```
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]加入到集合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

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

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

LR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* $[A \to \alpha \cdot a\beta, b] \in I_i$ *and* $GOTO(I_i, a) = I_j$ *then* $ACTION[i, a] = sj$
  - *if* $[A \to \alpha \cdot B\beta, b] \in I_i$ *and* $GOTO(I_i, B) = I_j$ *then* $GOTO[i, B] = j$
  - *if* $[A \to \alpha \cdot, a] \in I_i$且$A \neq S'$ *then* $ACTION[i, a] = rj$
  - *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*”

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

各种LR分析表构造方法的不同之处在于归约项目的处理上

- *if* $[A \to \alpha \cdot, c] \in I_i$

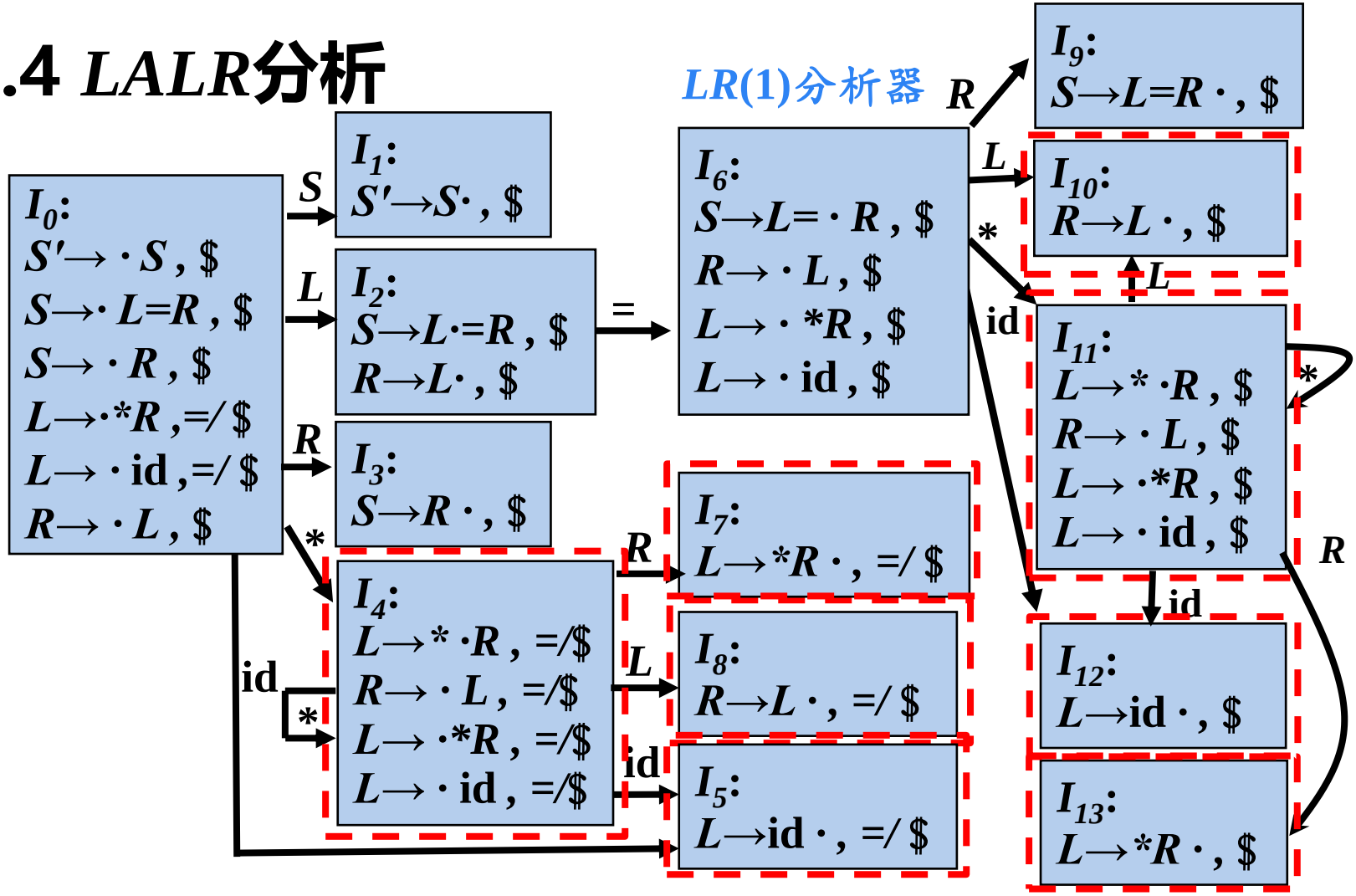
  - $A = S'$: $ACTION[i, \$] = acc$
  - $A \neq S'$:
    - LR(0) : $for\ \forall a \in V_T \cup \{\$\}$ $do\ ACTION[i, a] = r_j$ (j是产生式$A \to \alpha$的编号)
    - SLR(1) : $for\ \forall a \in FOLLOW(A)$ $do\ ACTION[i, a] = r_j$
    - LR(1) : 精准归约 $ACTION[i, c] = r_j$

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

4.4.4 LALR分析

➢例

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

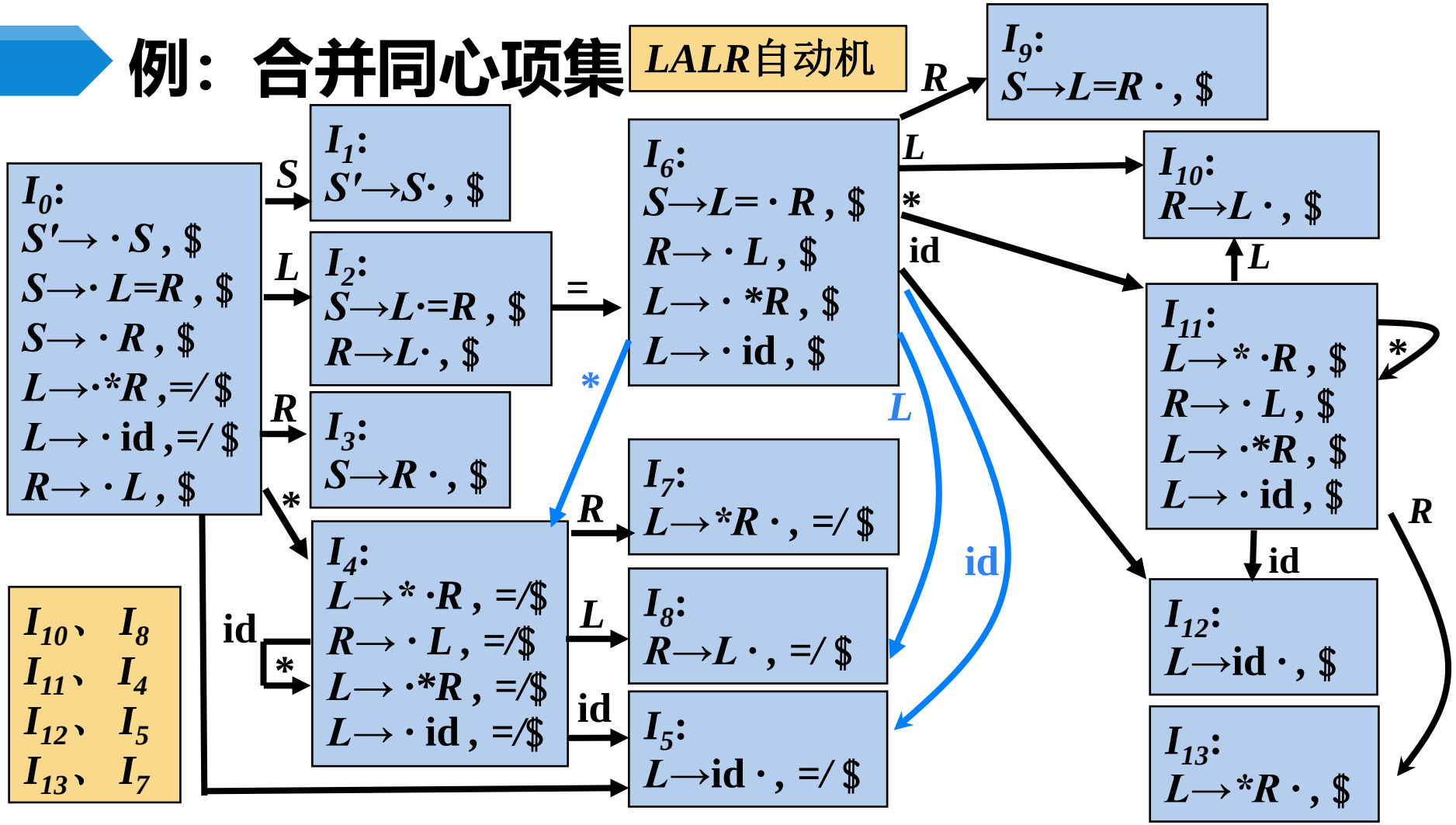


*LALR (lookahead-LR)*分析的基本思想

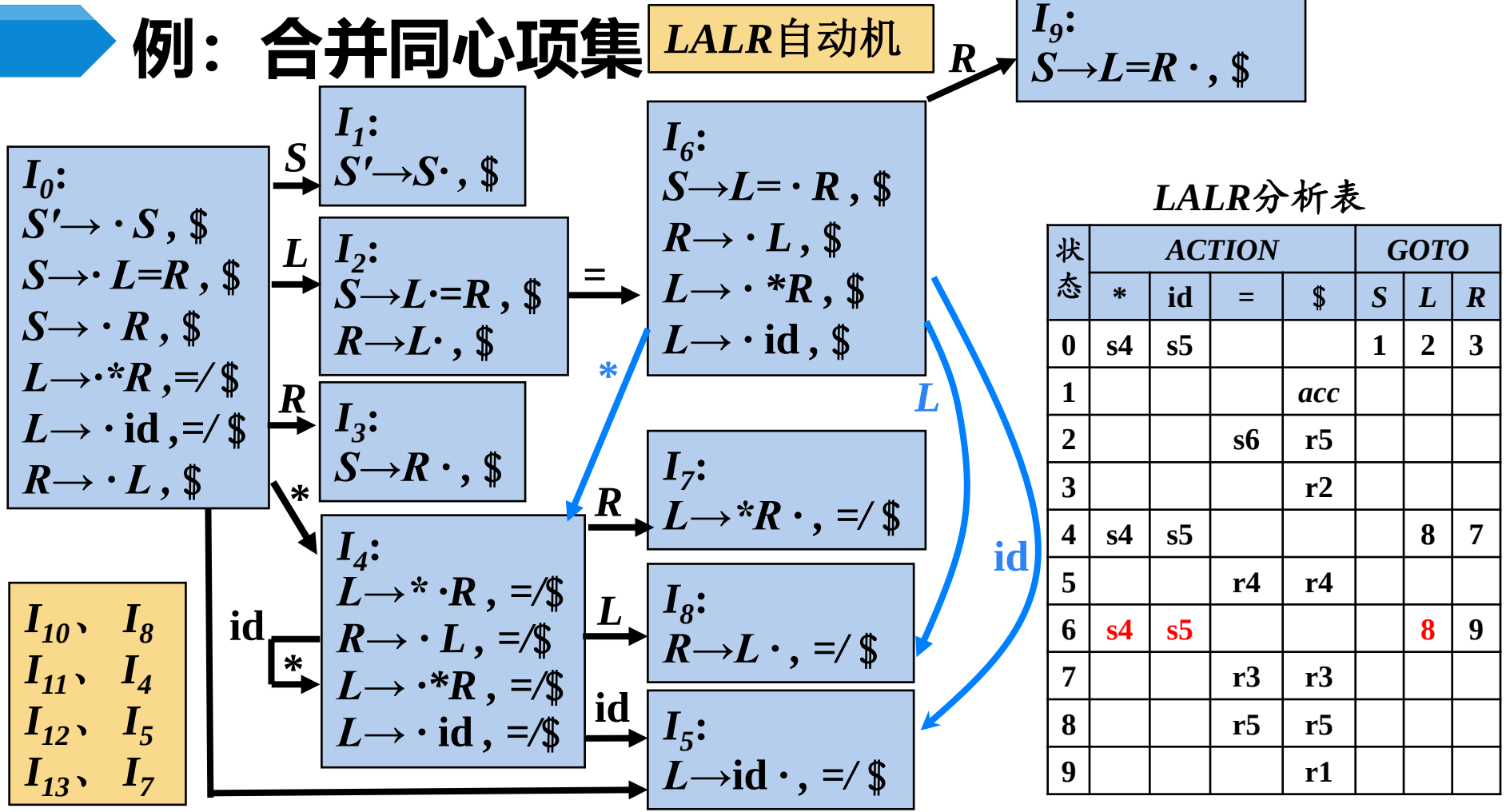
- 寻找具有相同核心的*LR* (1) 项集，并将这些项集合并为一个项集。 所谓项集的核心就是其第一分量的集合
- 然后根据合并后得到的项集族构造语法分析表
- 如果分析表中没有语法分析动作冲突，给定的文法就称为*LALR* (1) 文法，就可以根据该分析表进行语法分析

例：合并同心项集

LALR自动机



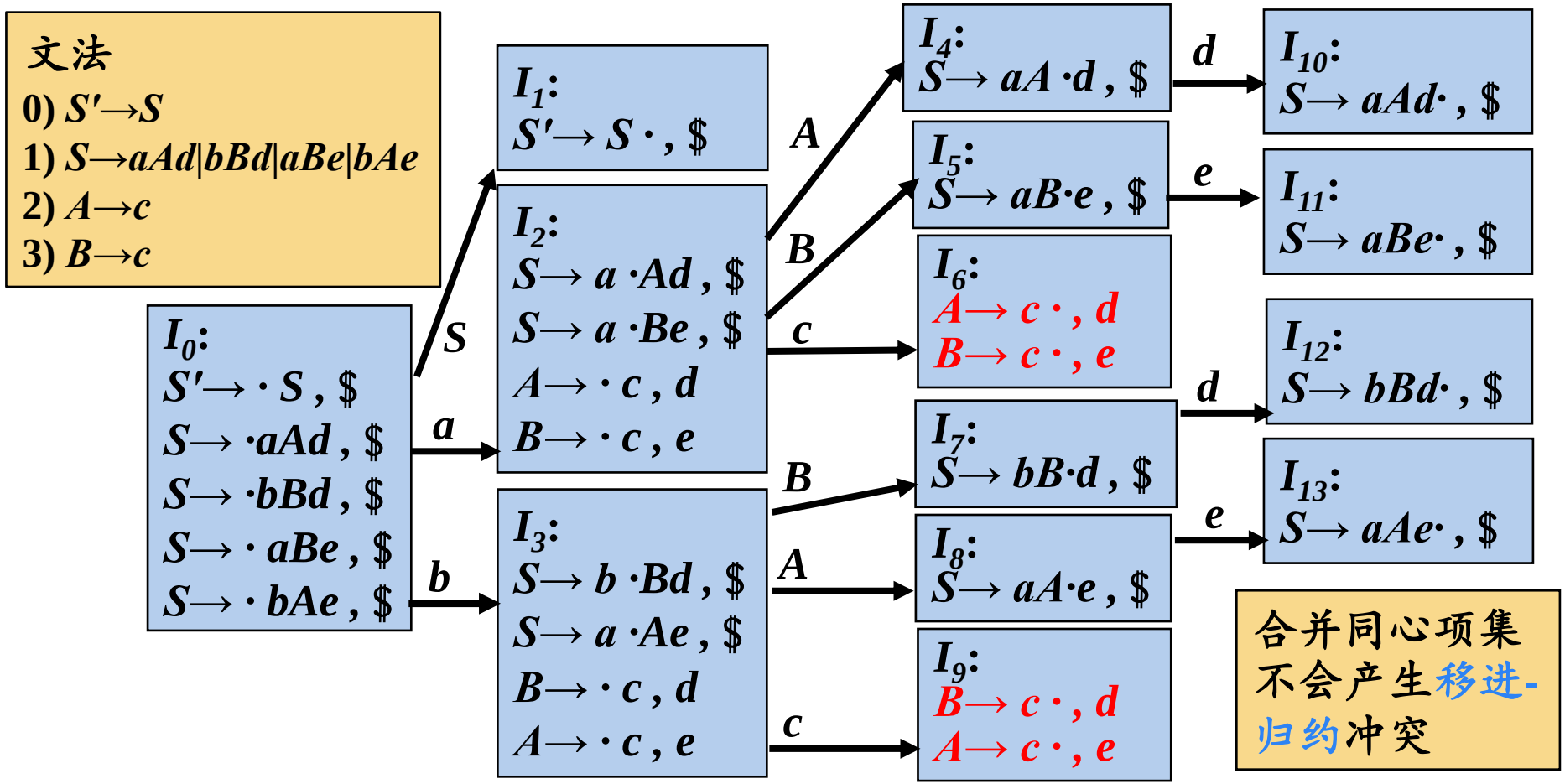
例：合并同心项集



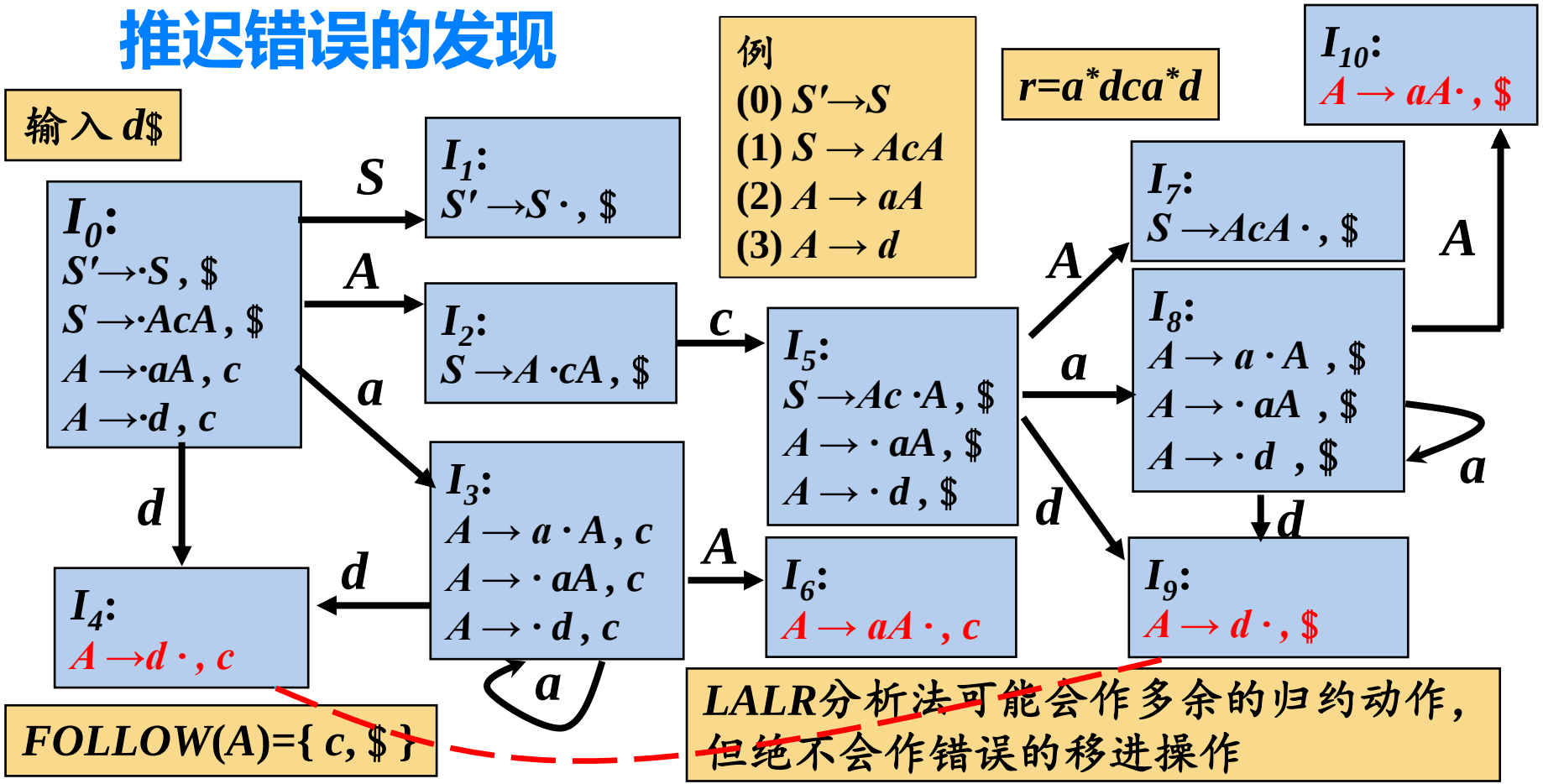
LALR分析表

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			

合并同心项集时产生归约-归约冲突的例子

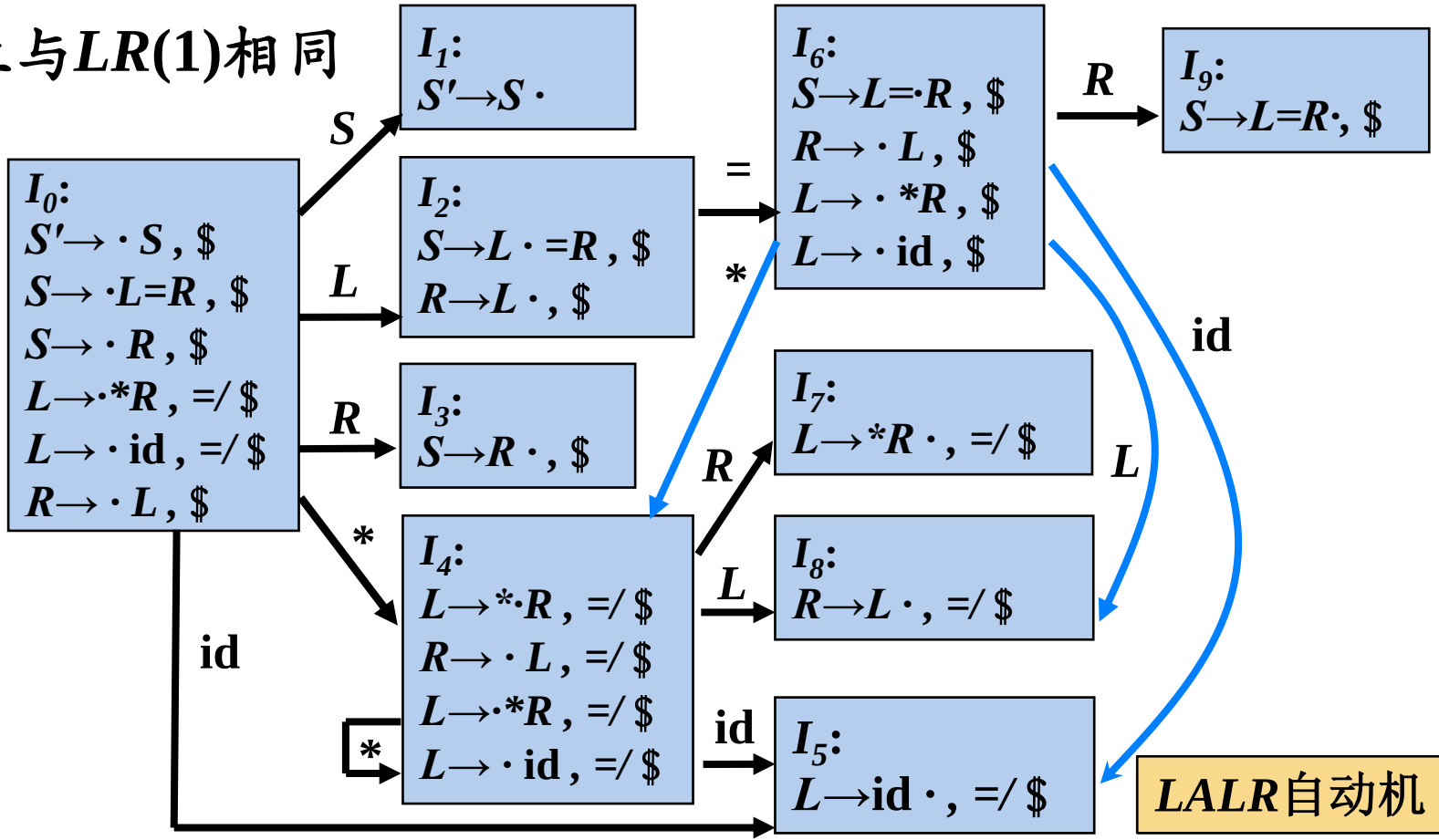


合并同心项集后，虽然不产生冲突，但可能会推迟错误的发现



LALR(1)的特点

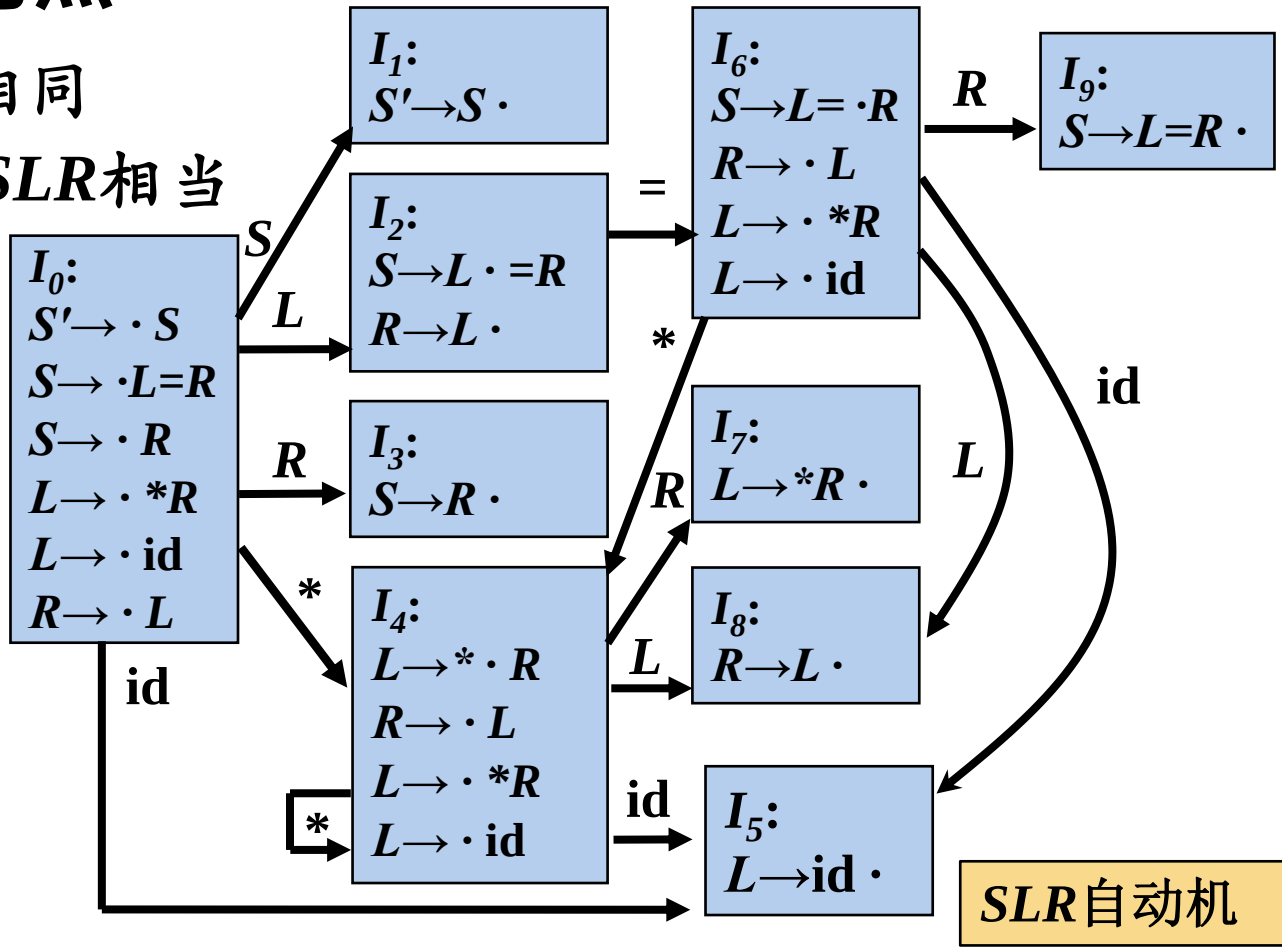
- 形式上与LR(1)相同



LALR自动机

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

$$LR(0) < SLR < LALR(1) < LR(1)$$

- 合并后的展望符集合仍为*FOLLOW*集的子集

4.4.5 二义性文法的LR分析

- 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、更自然
    - 例

二义性文法

① $E \rightarrow E+E$

② $E \rightarrow E*E$

③ $E \rightarrow (E)$

④ $E \rightarrow \mathbf{id}$

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 $F \rightarrow \mathbf{id}$

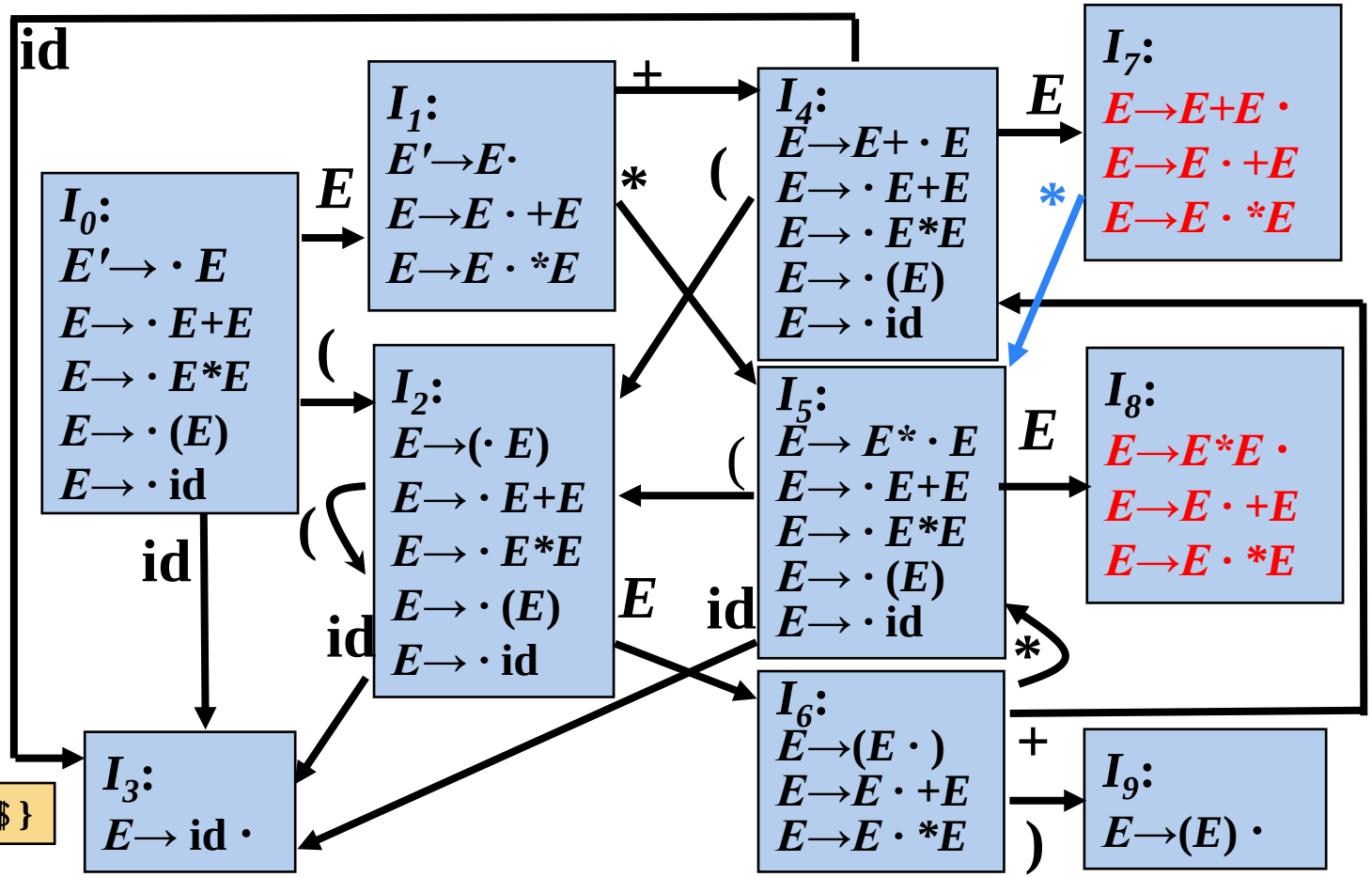
二义性算术表达式文法的$LR(0)$分析器

➢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

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



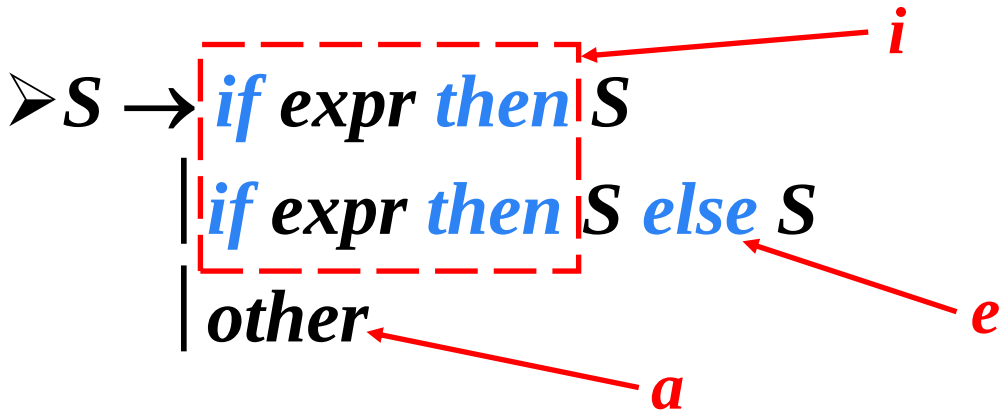
二义性算术表达式文法的*SLR*分析表

文法
E→*E*+*E*
E→*E***E*
E→(*E*)
E→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	

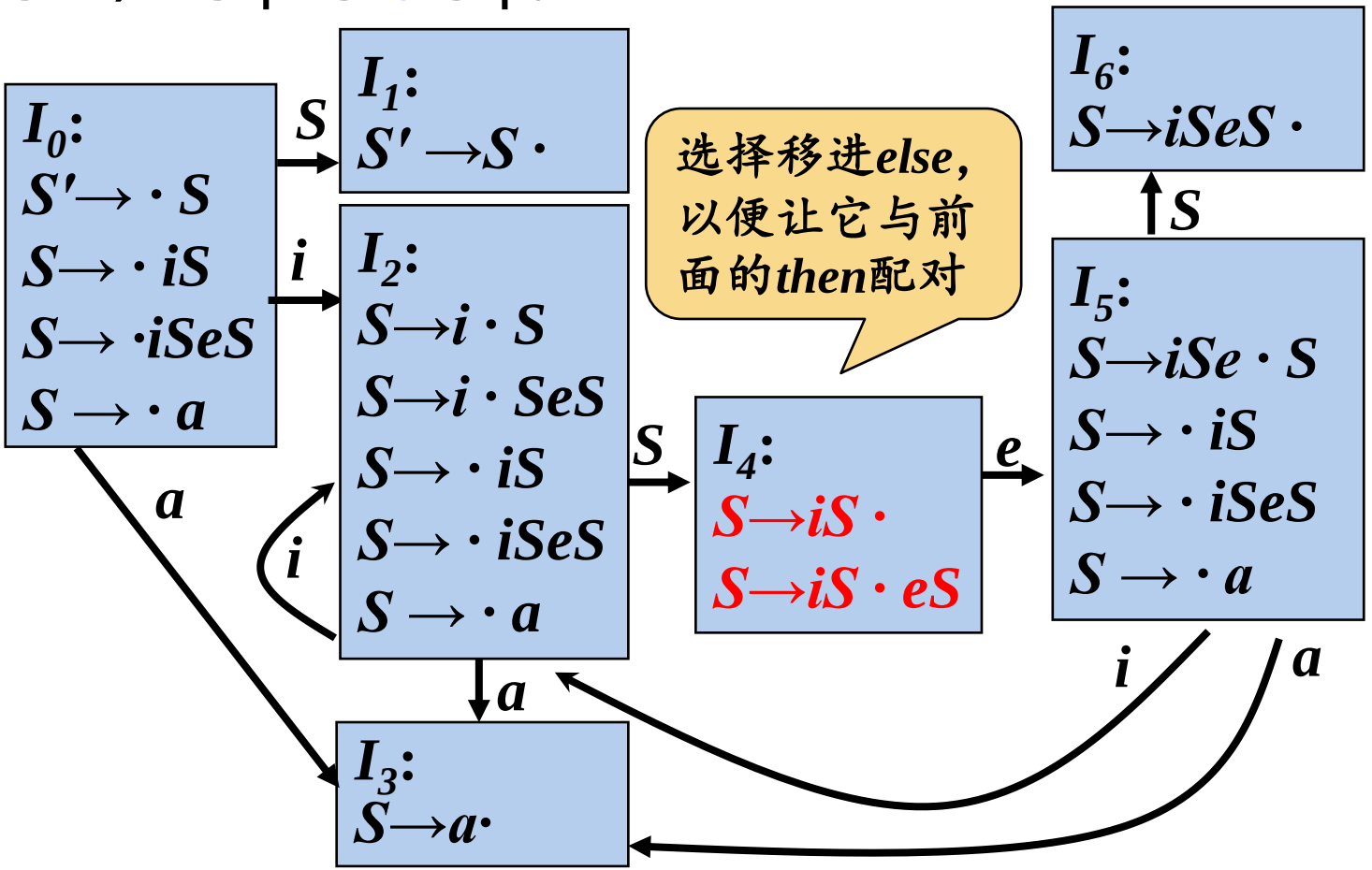
例：二义性*if* 语句文法的*LR*分析



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

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

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



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	

FOLLOW(*S*)={ e, $ }

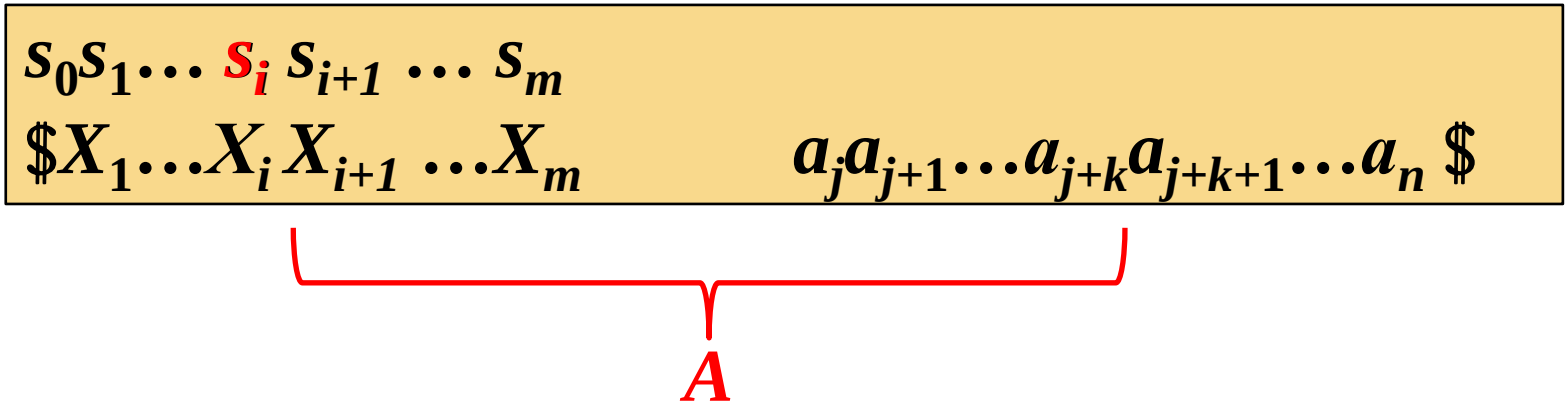
二义性文法的使用

- 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

4.4.6 *LR*分析中的错误处理

- 语法错误的检测
  - 当*LR*分析器在查询语法分析动作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

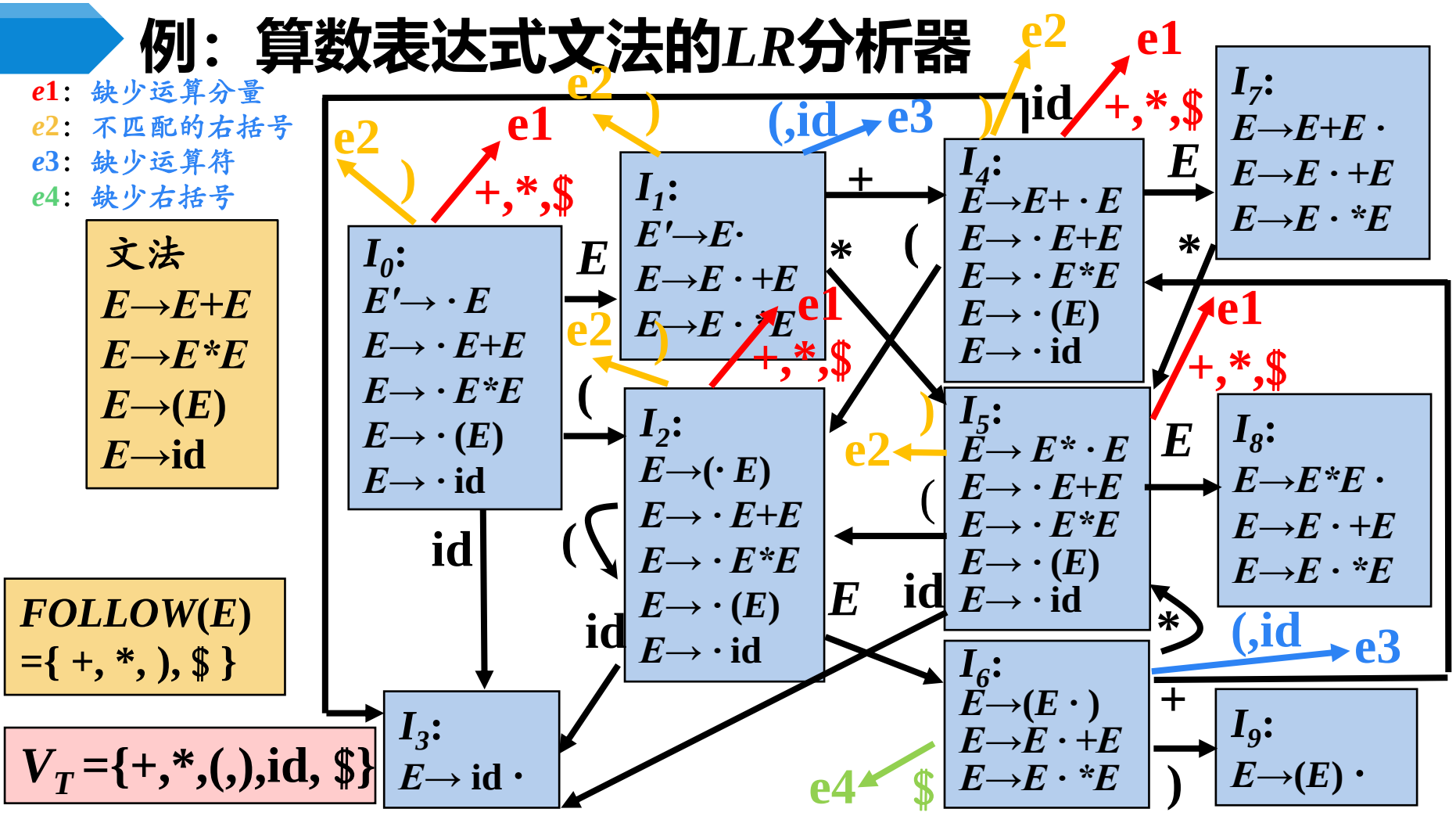


- 从栈顶向下扫描，直到发现某个状态s_i，它有一个对应于某个非终结符A的$GOTO$目标，可以认为从这个A推导出的串中包含错误
- 然后丢弃0个或多个输入符号，直到发现一个可能合法地紧跟在A之后的符号a为止
- 之后将$s_{i+1} = GOTO(s_i, A)$压入栈中，继续进行正常的语法分析
- 实践中可能会选择多个这样的非终结符A。通常这些非终结符代表了主要的程序段，比如表达式、语句或块

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

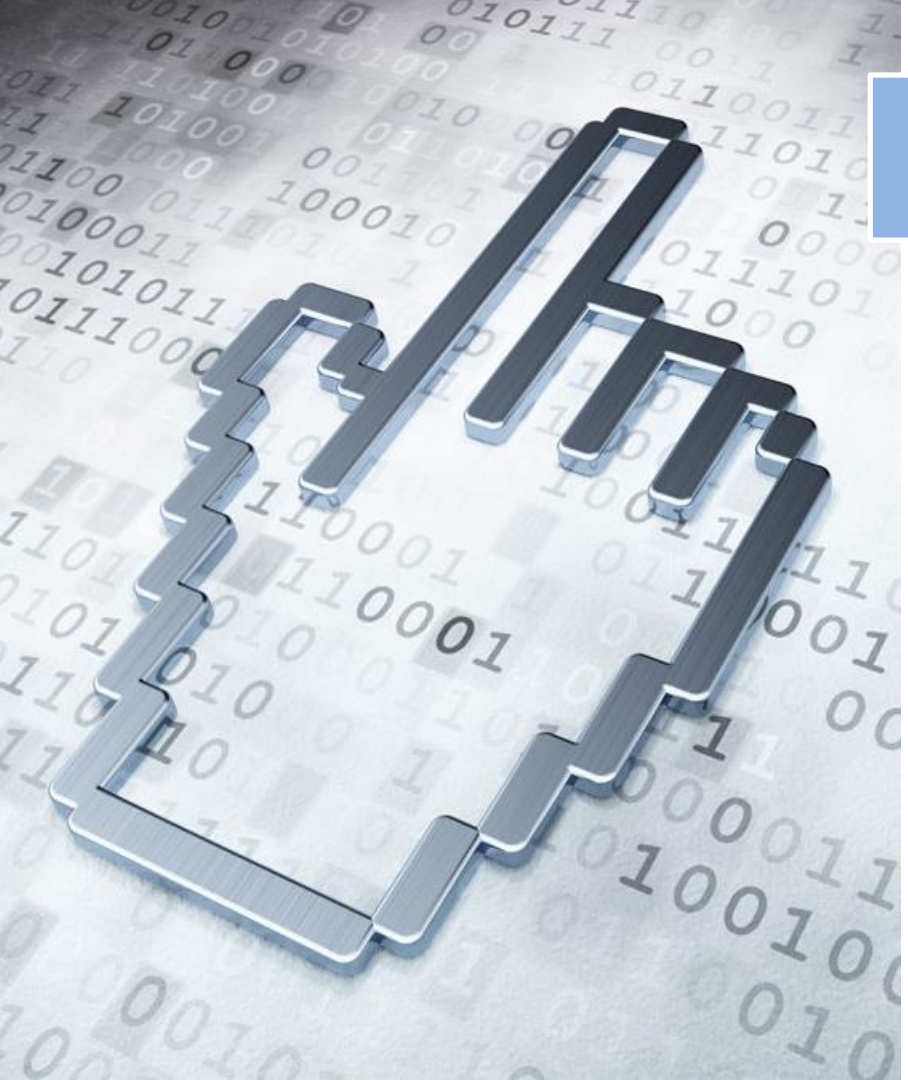
例：算数表达式文法的*LR*分析器



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
  - *e1*：将状态3压入栈中，发出诊断信息“缺少运算分量”
  - *e2*：从输入中删除“)”，发出诊断信息“不匹配的右括号”
  - *e3*：将状态4压入栈中，发出诊断信息“缺少运算符”
  - *e4*：将状态9压入栈中，发出诊断信息“缺少右括号”



提纲

4.5 算符优先分析法

- 优先分析法（precedence parsing）
  - 基本思想
    - 根据归约的先后次序为句型中相邻的文法符号规定优先关系
      - 句柄内的各个符号同时归约，因此规定句柄内各相邻符号的优先级相同，用符号≡表示
      - 句柄要先归约，因此规定句柄两端符号的优先级要高于句柄外与之相邻的符号的优先级，用符号$\lessdot$和$\gtrdot$分别表示“低于”和“高于”关系

$$X_1 \cdots X_{i-1} \lessdot X_i \equiv X_{i+1} \equiv \cdots \equiv X_{j-1} \equiv X_j \gtrdot X_{j+1} \cdots X_n$$

4.5 算符优先分析法

什么样的文法可以使用优先分析技术?

- 优先分析法（precedence parsing）
  - 基本思想
    - 根据归约的先后次序为句型中相邻的文法符号规定优先关系
    - 利用优先关系识别句柄并归约：利用⋗识别句柄尾，利用⋖识别句柄头
      - 利用分析栈存放已识别部分，比较栈顶和下一输入符号的关系，如果是句柄尾，则沿栈顶向下寻找句柄头，找到后弹出句柄，归约为非终结符
    - 重复上述过程，直到输入串扫描结束，且栈中只有开始符号为止

优先文法

- 对任意文法符号（终结符和非终结符），若它们可能在某一句型中相邻，则按上述方法为其定义优先关系。如果各文法符号之间的优先关系不冲突（即至多存在一种优先关系），则可以利用这种优先关系识别任意句型的句柄。满足这种优先关系的文法称为优先文法（precedence grammar），基于这种文法及其文法符号间优先关系的分析方法称为优先分析法（precedence parsing）

算符优先分析法

- Floyd在1963年首先提出，Greis于1971年将其形式化
- 主要用来分析程序设计语言中的各类表达式

算符优先分析法

$$E \rightarrow E+T \mid E-T \mid T$$
$$T \rightarrow T*F \mid T/F \mid F$$
$$F \rightarrow (E) \mid id$$

- 算术表达式文法的特点
  - 运算次序只与运算符（终结符）有关，而与运算对象（非终结符）无关 → 可以仅对文法中可能在句型中相邻的终结符定义优先关系
    - 先乘除，后加减
    - 优先级相同的运算符采取左结合原则
  - 产生式右部没有相邻的非终结符（运算对象） → 算符文法（operatot grammar, OG）

算符优先文法

为了防止运算对象为空，限制OG文法不含空产生式

算符优先文法

- 假设$G=(V_N, V_T, P, S)$为OG，A、B、$C \in V_N$，a、$b \in V_T$，a和b之间的关系定义为：
  - $a \equiv b \Leftrightarrow A \to \ldots ab \ldots \in P$或$A \to \ldots aBb \ldots \in P$
  - $a \lessdot b \Leftrightarrow A \to \ldots aB \ldots \in P$且$(B \Rightarrow^+ b \ldots$ 或 $B \Rightarrow^+ Cb \ldots)$

    FIRSTOP(B)={ $b \mid B \Rightarrow^+ b \ldots$ 或 $B \Rightarrow^+ Cb \ldots$
  - $a \gtrdot b \Leftrightarrow A \to \ldots Bb \ldots \in P$且$(B \Rightarrow^+ \ldots a$ 或 $B \Rightarrow^+ \ldots aC)$

    LASTOP(B) ={ $a \mid B \Rightarrow^+ \ldots a$ 或 $B \Rightarrow^+ \ldots aC$
  - a与b无关$\Leftrightarrow$$a$与$b$在G的任何句型中都不相邻

$a \equiv b \Leftrightarrow a \equiv b$

$a \lessdot b \Leftrightarrow b \gtrdot a$

$a \gtrdot b \Leftrightarrow b \lessdot a$

如果G的任何一对终结符a和b之间只满足上述某一种关系，则称文法G为算符优先文法（Operator Precedence Grammar，OPG）

例

E’ →E

E → E+T | E-T | T
T → T*F | T/F | F
F → (E) | id

- FIRSTOP(*E*) = { +， -， *， /， (， id }
- FIRSTOP(*T*) = { *， /， (， id }
- FIRSTOP(*F*) = { (， id }
- LASTOP(*E*) = { +， -， *， /，)， id }
- LASTOP(*T*) = { *， /，)， id }
- LASTOP(*F*) = {)， id }

算符优先关系矩阵

	+	-	*	/	(	)	id	$
+	⋗	⋗	⋖	⋖	⋖	⋗	⋖	⋗
-	⋗	⋗	⋖	⋖	⋖	⋗	⋖	⋗
*	⋗	⋗	⋗	⋗	⋖	⋗	⋖	⋗
/	⋗	⋗	⋗	⋗	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	⋖	⋖	≡	⋖	
)	⋗	⋗	⋗	⋗		⋗		⋗
id	⋗	⋗	⋗	⋗		⋗		⋗
$	⋖	⋖	⋖	⋖	⋖		⋖	≡

算符优先分析——例

- 问题?
  - 不是严格最左归约

$E \rightarrow E+T \mid E-T \mid T$
$T \rightarrow T*F \mid T/F \mid F$
$F \rightarrow (E) \mid id$

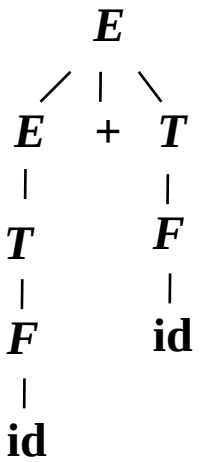
算符优先关系矩阵

	+	-	*	/	(	)	id	$
+	⋗	⋗	⋖	⋖	⋖	⋗	⋖	⋗
-	⋗	⋗	⋖	⋖	⋖	⋗	⋖	⋗
*	⋗	⋗	⋗	⋗	⋖	⋗	⋖	⋗
/	⋗	⋗	⋗	⋗	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	⋖	⋖	≡	⋖	
)	⋗	⋗	⋗	⋗		⋗		⋗
id	⋗	⋗	⋗	⋗		⋗		⋗
$	⋖	⋖	⋖	⋖	⋖		⋖	≡

栈	剩余输入	动作
$	id+id $	
$ ⋖id	+id $	移入
$ *F*	+id $	归约: $F \rightarrow id$
$ ⋖*F*+	id $	移入
$ ⋖*F*+⋖id	$	移入
$ ⋖*F*+ *F*	$	归约: $F \rightarrow id$
$ *E*	$	归约: $E \rightarrow E+T$

算符优先分析——例

$E \to E+T \mid E-T \mid T$
$T \to T*F \mid T/F \mid F$
$F \to (E) \mid id$



- 问题?
  - 不是严格最左归约
  - 有时未归约真正的句柄
    - 归约的符号串有时与产生式右部不同
- 原因?
  - OPG未定义非终结符之间的优先关系，不能识别由单非终结符组成的句柄

栈	剩余输入	动作
$	id+id $	
$ ⋖id	+id $	移入
$ F	+id $	归约: $F \to id$
$ ⋖F+	id $	移入
$ ⋖F+⋖id	$	移入
$ ⋖F+ F	$	归约: $F \to id$
$ E	$	归约: $E \to E+T$

最左素短语（LPP）——算符优先分析过程中识别出来的“句柄”

- 素短语（Prime Phase）
  - 素短语是一个短语
  - 它至少包含一个终结符
  - 除自身外，不再包含其它含终结符的短语

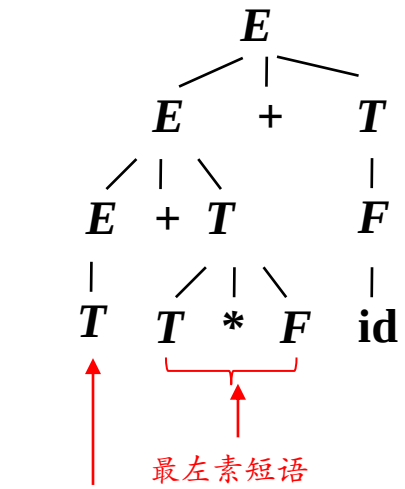
文法

$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T*F \mid T/F \mid F$

$F \rightarrow (E) \mid \mathrm{id}$

句型：$T+T*F+\mathrm{id}$



- 短语
  - T
  - $T*F$（素短语）
  - $T+T*F$
  - id （素短语）
  - $T+T*F+\mathrm{id}$

最左素短语（LPP）——算符优先分析过程中识别出来的“句柄”

- 素短语（Prime Phase）
  - 素短语是一个短语
  - 它至少包含一个终结符
  - 除自身外，不再包含其它含终结符的短语

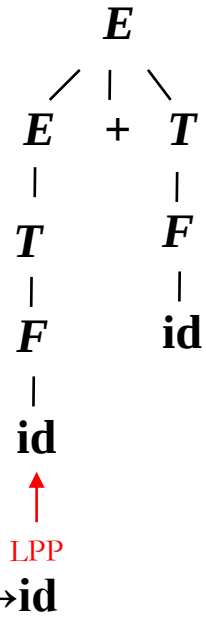
文法

$E \rightarrow E+T \mid E-T \mid T$

$T \rightarrow T*F \mid T/F \mid F$

$F \rightarrow (E) \mid id$

句型：id+id



栈	剩余输入	动作
$	id+id $	
$ id	+id $	移入
$ F	+id $	归约: $F \rightarrow id$

最左素短语（LPP）——算符优先分析过程中识别出来的“句柄”

- 素短语（Prime Phase）
  - 素短语是一个短语
  - 它至少包含一个终结符
  - 除自身外，不再包含其它含终结符的短语

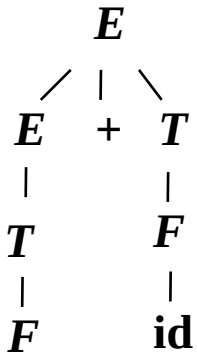
文法

$E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid F$

$F \to (E) \mid id$

句型：id+id



栈	剩余输入	动作
$	id+id $	
$ ⋖id	+id $	移入
$ F	+id $	归约: F→id
$ ⋖F+	id $	移入
$ ⋖F+⋖id	$	移入
$ ⋖F+ F	$	归约: F→id

最左素短语（LPP）——算符优先分析过程中识别出来的“句柄”

- 素短语（Prime Phase）
  - 素短语是一个短语
  - 它至少包含一个终结符
  - 除自身外，不再包含其它含终结符的短语

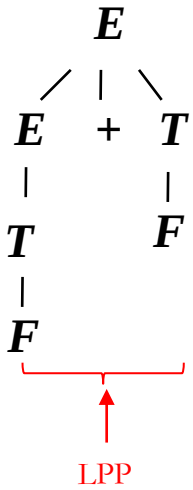
文法

$E \to E+T \mid E-T \mid T$

$T \to T*F \mid T/F \mid F$

$F \to (E) \mid id$

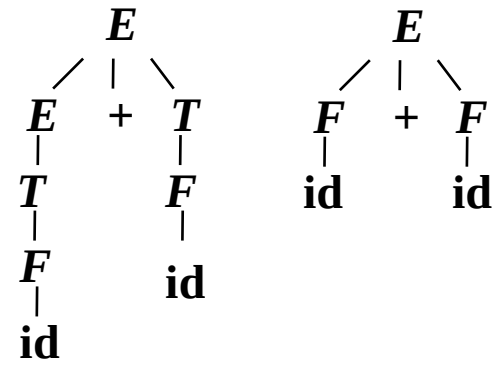
句型：id+id



栈	剩余输入	动作
$	id+id $	
$ ⋖id	+id $	移入
$ F	+id $	归约: $F \to id$
$ ⋖F+	id $	移入
$ ⋖F+⋖id	$	移入
$ ⋖F+ F	$	归约: $F \to id$
$ E	$	归约: $E \to E+T$

应该把LPP归约为哪一个语法变量?

- LPP: $N_i a_i N_{i+1} a_{i+1} \cdots N_j a_j N_{j+1}$ $(N_i \in V_N \cup \{\varepsilon\}, a_i \in V_T)$
- 某产生式右部: $U_i a_i U_{i+1} a_{i+1} \cdots U_j a_j U_{j+1}$
- 由于算符优先分析法不能识别单语法变量组成的句柄，所以最后归约出的分析树和按最左归约形成的分析树不同，可以成其为“语法树架子”



语法分析树　　语法树架子

算符优先分析vs.LR分析

- 优点
  - 速度快得多（跳过了所有单语法变量产生式所对应的归约步骤）
- 缺点
  - 有可能将本来不是句子的输入串误认为是句子
  - 效率较低（需要不断确定分析栈中终结符的位置）

算符优先关系矩阵的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$|V_T|\times|V_T|$的算符优先矩阵M
- 步骤：
  - begin
    - for $\forall A\rightarrow X_1X_2\ldots X_n \in P$ do
      - for i:=1 to n-1 do
        - if ($X_iX_{i+1}\in V_TV_T$) then $M[X_i, X_{i+1}]$:='≡'
        - else if ($i<n$-2 & $X_iX_{i+1}X_{i+2}\in V_TV_NV_T$) then $M[X_i, X_{i+2}]$:='≡'
        - else if ($X_iX_{i+1}\in V_TV_N$) then for$\forall a\in$FIRSTOP(X_{i+1}) do $M[X_i, a]$:='⋖'
        - else if ($X_iX_{i+1}\in V_NV_T$) then for $\forall a\in$LASTOP(X_i)do $M[a, X_{i+1}]$:='⋗'
  - end

FIRSTOP和LASTOP的构造

- 若有产生式$B \to b\ldots$或$B \to Cb\ldots$，则有$b \in$ FIRSTOP(B)
- 若有产生式$A \to B\ldots$，则有FIRSTOP(B)$\subseteq$ FIRSTOP(A)

$$F[A,a] = \begin{cases} true, & a \in FIRSTOP(A) \\ false, & else \end{cases}$$

- 若有产生式$B \to \ldots b$或$B \to \ldots bC$，则有$b \in$ LASTOP(B)
- 若有产生式$A \to \ldots B$，则有LASTOP(B)$\subseteq$ LASTOP(A)

FIRSTOP的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$\forall A \in V_N$，FIRSTOP(A)
- 步骤：

```
begin
    for ∀(A,a) ∈V_N×V_T do F[A,a]:= false
    for ∀A→a…∈P或 A→Ba…∈P do insert(A,a)
    while 栈非空 do
        begin
            pop(B,a)
            for ∀A→B…∈P do insert(A,a)
        end
    for ∀ A ∈V_N do FIRSTOP(A):=Φ
    for ∀(A,a) ∈V_N×V_T do
        begin
            if F[A，a] then FIRSTOP(A):= FIRSTOP(A)∪{a}
        end
end
```

```
procedure insert(A,a)
/* F[A, a]置为true并将其压入栈*/
begin
    if not F[A, a] then
        begin
            F[A,a]:= true
            push(A,a)
        end
end
```

FIRSTOP的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$\forall A \in V_N$，FIRSTOP(A)
- 步骤：

begin

for $\forall(A,a) \in V_N \times V_T$ **do** F[A,a]:= **false**

for $\forall A \to a\ldots \in P$或$A \to Ba\ldots \in P$ **do** insert(A,a)

while 栈非空 **do**

begin

pop(B,a)

for $\forall A \to B\ldots \in P$ **do** insert(A,a)

end

for $\forall A \in V_N$ **do** FIRSTOP(A):=Φ

for $\forall(A,a) \in V_N \times V_T$ **do**

begin

if F[A，a] **then** FIRSTOP(A):= FIRSTOP(A)∪{a}

end

end

例：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | id

	+	-	*	/	(	)	id
E	1	1					
T			1	1			
F					1		1

(F, id)

(F, ( )

(T, /)

(T, *)

(E, -)

(E,+)

FIRSTOP的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$\forall A \in V_N$，FIRSTOP(A)
- 步骤：

begin

for $\forall(A,a) \in V_N \times V_T$ **do** F[A,a]:= **false**

for $\forall A \to a\ldots \in P$或$A \to Ba\ldots \in P$ **do** insert(A,a)

while 栈非空 **do**

begin

pop(B,a)

for $\forall A \to B\ldots \in P$ **do** insert(A,a)

end

for $\forall A \in V_N$ **do** FIRSTOP(A):=Φ

for $\forall(A,a) \in V_N \times V_T$ **do**

begin

if F[A，a] **then** FIRSTOP(A):= FIRSTOP(A)∪{a}

end

end

例：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | id

	+	-	*	/	(	)	id
E	1	1					
T			1	1			1
F					1		1

(T, id)

(F, ( )

(T, /)

(T, *)

(E, -)

(E,+)

FIRSTOP的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$\forall A \in V_N$，FIRSTOP(A)
- 步骤：

begin

for $\forall(A,a) \in V_N \times V_T$ **do** F[A,a]:= **false**

for $\forall A \to a\ldots \in P$或$A \to Ba\ldots \in P$ **do** insert(A,a)

while 栈非空 **do**

begin

pop(B,a)

for $\forall A \to B\ldots \in P$ **do** insert(A,a)

end

for $\forall A \in V_N$ **do** FIRSTOP(A):=Φ

for $\forall(A,a) \in V_N \times V_T$ **do**

begin

if F[A，a] **then** FIRSTOP(A):= FIRSTOP(A)∪{a}

end

end

例：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | id

	+	-	*	/	(	)	id
E	1	1					1
T			1	1			1
F					1		1

(E, id)

(F, ( )

(T, /)

(T, *)

(E, -)

(E,+)

FIRSTOP的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$\forall A \in V_N$，FIRSTOP(A)
- 步骤：

begin

for $\forall(A,a) \in V_N \times V_T$ **do** F[A,a]:= **false**

for $\forall A \to a\ldots \in P$或$A \to Ba\ldots \in P$ **do** insert(A,a)

while 栈非空 **do**

begin

pop(B,a)

for $\forall A \to B\ldots \in P$ **do** insert(A,a)

end

for $\forall A \in V_N$ **do** FIRSTOP(A):=Φ

for $\forall(A,a) \in V_N \times V_T$ **do**

begin

if F[A，a] **then** FIRSTOP(A):= FIRSTOP(A)∪{a}

end

end

例：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | id

	+	-	*	/	(	)	id
E	1	1					1
T			1	1	1		1
F					1		1

(T, ( )

(T, /)

(T, *)

(E, -)

(E,+)

FIRSTOP的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$\forall A \in V_N$，FIRSTOP(A)
- 步骤：

begin

for $\forall(A,a) \in V_N \times V_T$ **do** F[A,a]:= **false**

for $\forall A \to a\ldots \in P$或$A \to Ba\ldots \in P$ **do** insert(A,a)

while 栈非空 **do**

begin

pop(B,a)

for $\forall A \to B\ldots \in P$ **do** insert(A,a)

end

for $\forall A \in V_N$ **do** FIRSTOP(A):=Φ

for $\forall(A,a) \in V_N \times V_T$ **do**

begin

if F[A，a] **then** FIRSTOP(A):= FIRSTOP(A)∪{a}

end

end

例：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | id

	+	-	*	/	(	)	id
E	1	1			1		1
T			1	1	1		1
F					1		1

(E, ( )
(T, /)
(T, *)
(E, -)
(E,+)

FIRSTOP的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$\forall A \in V_N$，FIRSTOP(A)
- 步骤：

begin

for $\forall(A,a) \in V_N \times V_T$ **do** F[A,a]:= **false**

for $\forall A \to a\ldots \in P$或$A \to Ba\ldots \in P$ **do** insert(A,a)

while 栈非空 **do**

begin

pop(B,a)

for $\forall A \to B\ldots \in P$ **do** insert(A,a)

end

for $\forall A \in V_N$ **do** FIRSTOP(A):=Φ

for $\forall(A,a) \in V_N \times V_T$ **do**

begin

if F[A，a] **then** FIRSTOP(A):= FIRSTOP(A)∪{a}

end

end

例：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | id

	+	-	*	/	(	)	id
E	1	1		1	1		1
T			1	1	1		1
F					1		1

(E, /)

(T, *)

(E, -)

(E,+)

FIRSTOP的构造算法

- 输入：文法$G=(V_N, V_T, P, S)$
- 输出：$\forall A \in V_N$，FIRSTOP(A)
- 步骤：

begin

for $\forall(A,a) \in V_N \times V_T$ **do** F[A,a]:= **false**

for $\forall A \to a\ldots \in P$或$A \to Ba\ldots \in P$ **do** insert(A,a)

while 栈非空 **do**

begin

pop(B,a)

for $\forall A \to B\ldots \in P$ **do** insert(A,a)

end

for $\forall A \in V_N$ **do** FIRSTOP(A):=Φ

for $\forall(A,a) \in V_N \times V_T$ **do**

begin

if F[A，a] **then** FIRSTOP(A):= FIRSTOP(A)∪{a}

end

end

例：

E → E+T | E-T | T

T → T*F | T/F | F

F → (E) | id

- FIRSTOP(E) = { +， -， *， /， (， id }
- FIRSTOP(T) = { *， /， (， id }
- FIRSTOP(F) = { (， id }

	+	-	*	/	(	)	id
E	1	1	1	1	1		1
T			1	1	1		1
F					1		1

(E, *)

(E, -)

(E,+)

优先函数

- 为了节省存储空间，提高效率，通常用“优先数”来表示算符之间的优先关系。
- 用两个优先函数f和g来代替优先矩阵
  - f表示算符的栈内优先数，g表示算符的栈外优先数
- 具体地，将每个终结符与两个整函数$f(a)$和$g(a)$的值相对应，使得
  - 如果$a \lessdot b$，则$f(a)<g(b)$
  - 如果$a \doteq b$，则$f(a)=g(b)$
  - 如果$a \gtrdot b$，则$f(a)>g(b)$

例

$E \rightarrow E+T \mid E-T \mid T$
$T \rightarrow T*F \mid T/F \mid F$
$F \rightarrow (E) \mid id$

算符优先关系矩阵

	+	-	*	/	(	)	id	$
+	⋗	⋗	⋖	⋖	⋖	⋗	⋖	⋗
-	⋗	⋗	⋖	⋖	⋖	⋗	⋖	⋗
*	⋗	⋗	⋗	⋗	⋖	⋗	⋖	⋗
/	⋗	⋗	⋗	⋗	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	⋖	⋖	≡	⋖	
)	⋗	⋗	⋗	⋗		⋗		⋗
id	⋗	⋗	⋗	⋗		⋗		⋗
$	⋖	⋖	⋖	⋖	⋖		⋖	≡

优先函数

	f	g
+	2	1
-	2	1
*	4	3
/	4	3
(	o	5
)	4	o
id	4	5
$	o	o

优先函数

- 为了节省存储空间，提高效率，通常用“优先数”来表示算符之间的优先关系。
- 用两个优先函数f和g来代替优先矩阵
  - f表示算符的栈内优先数，g表示算符的栈外优先数
- 具体地，将每个终结符与两个整函数$f(a)$和$g(a)$的值相对应，使得
  - 如果$a \lessdot b$，则$f(a)<g(b)$
  - 如果$a \doteq b$，则$f(a)=g(b)$
  - 如果$a \gtrdot b$，则$f(a)>g(b)$
- 使用优先函数后，用$f(a)>f(b)$来寻找LPP的尾，用$f(a)<f(b)$来寻找LPP的头

优先函数的缺点

- 由于是用整数表达优先级的，而整数的大小总是可以比较的，这就使得原先不具有优先关系的两个终结符现在变成可比较的了。
- 不具有优先关系意味着它们不能按照相应的顺序紧邻出现，现在“变成可以比较的了”，因而将会掩盖输入串中的这种错误。
- 解决办法
  - 通过检查栈顶符号Q和输入符号R的具体内容来发现那些原先不可比较的情形
- 注意：并非所有的优先关系都能用优先函数来表示

从算符优先矩阵构造优先函数的简单方法

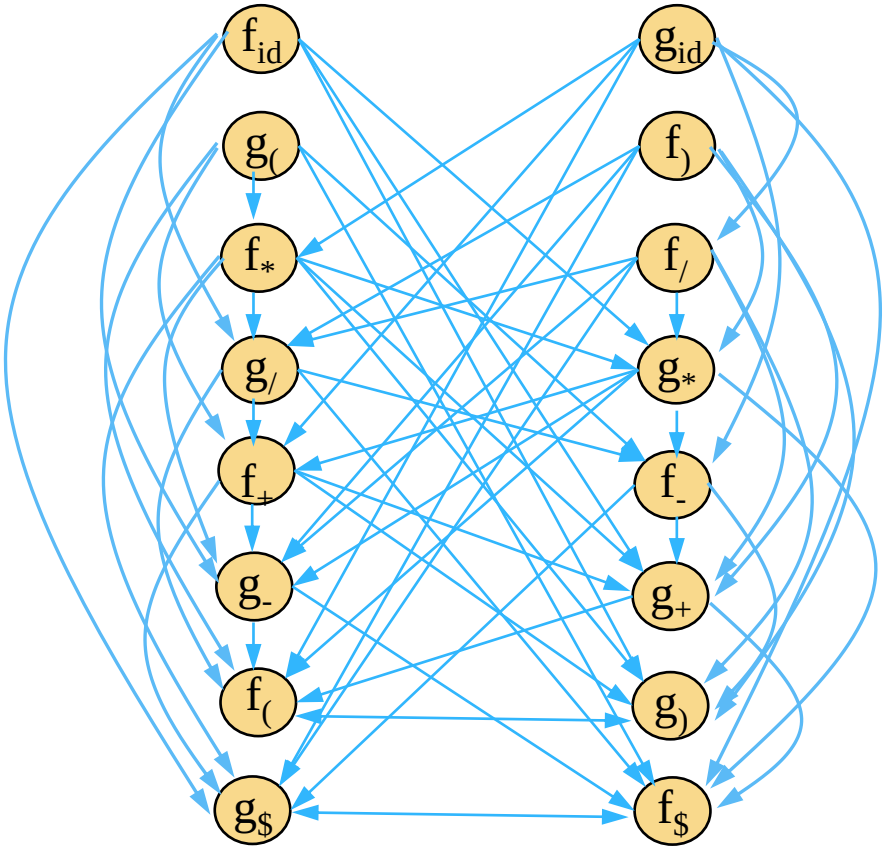
- 算法　优先函数的构造
  - 输入：算符优先矩阵
  - 输出：表示输入矩阵的优先函数，或指出其不存在
  - 方法：
    - 1. 对$\forall a \in V_T \cup \{\$\}$，建立以$f_a$和$g_a$为标记的顶点
    - 2. 对$\forall a \in V_T \cup \{\$\}$
      - 若$a \gtrdot b$或$a \equiv b$，则从f_a至g_b画一条弧
      - 若$a \lessdot b$或$a \equiv b$，则从g_b至f_a画一条弧
    - 3. 若图中无环，则存在优先函数，$f(a)$和$g(b)$等于从f_a和g_b出发的最长路径

例

$E \rightarrow E+T \mid E-T \mid T$
$T \rightarrow T*F \mid T/F \mid F$
$F \rightarrow (E) \mid id$

算符优先关系矩阵

	+	-	*	/	(	)	id	$
+	⋗	⋗	⋖	⋖	⋖	⋗	⋖	⋗
-	⋗	⋗	⋖	⋖	⋖	⋗	⋖	⋗
*	⋗	⋗	⋗	⋗	⋖	⋗	⋖	⋗
/	⋗	⋗	⋗	⋗	⋖	⋗	⋖	⋗
(	⋖	⋖	⋖	⋖	⋖	≡	⋖	
)	⋗	⋗	⋗	⋗		⋗		⋗
id	⋗	⋗	⋗	⋗		⋗		⋗
$	⋖	⋖	⋖	⋖	⋖		⋖	≡

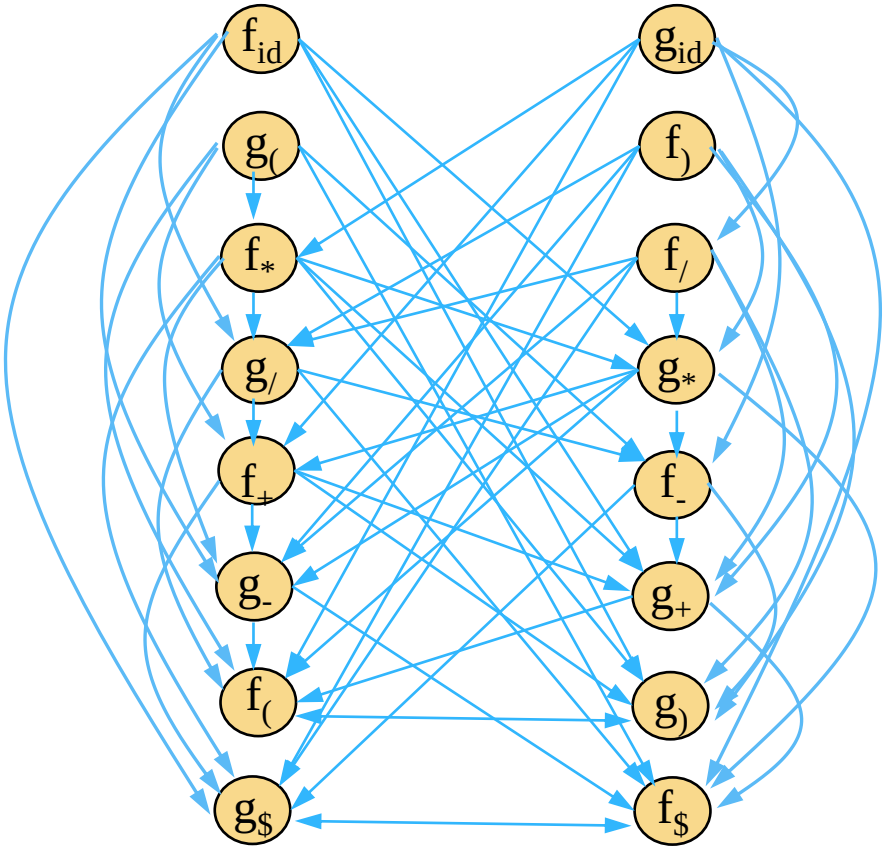


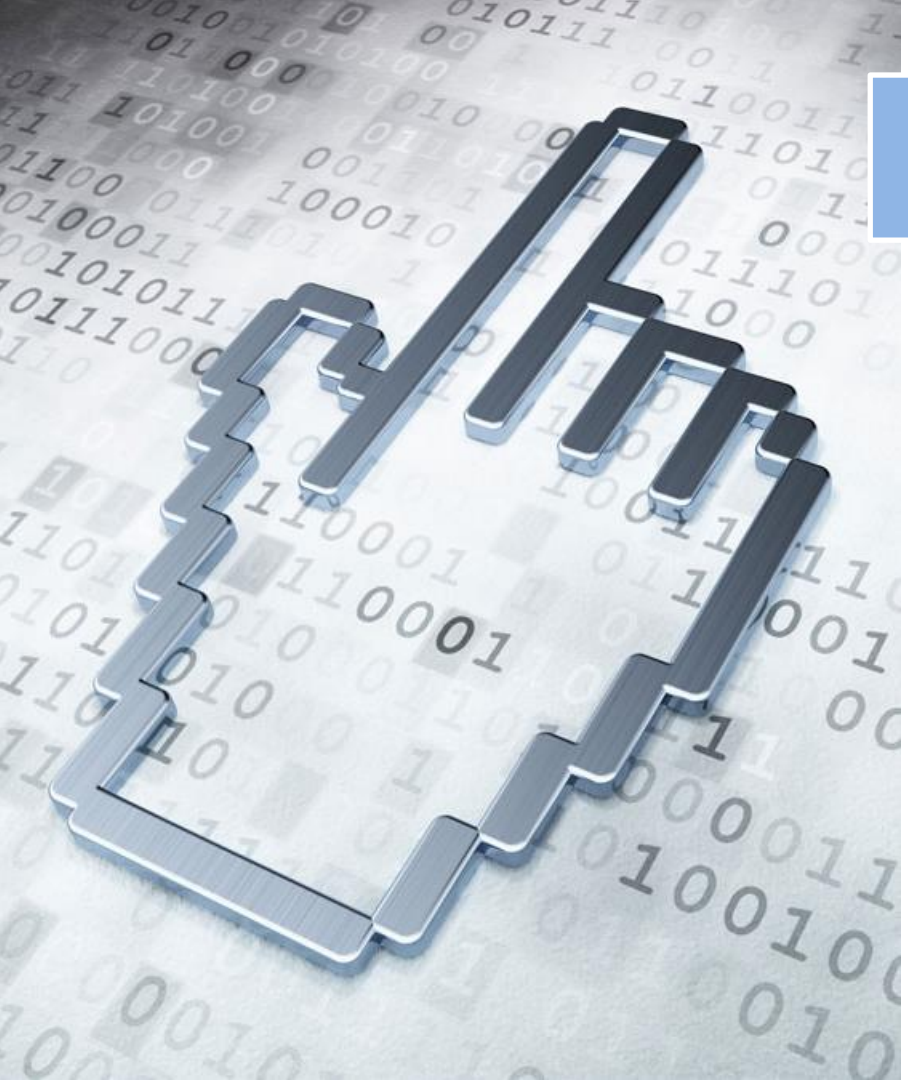
例

$E \rightarrow E+T \mid E-T \mid T$
$T \rightarrow T*F \mid T/F \mid F$
$F \rightarrow (E) \mid id$

优先函数

	f	g
+	2	1
-	2	1
*	4	3
/	4	3
(	0	5
)	4	0
id	4	5
$	0	0





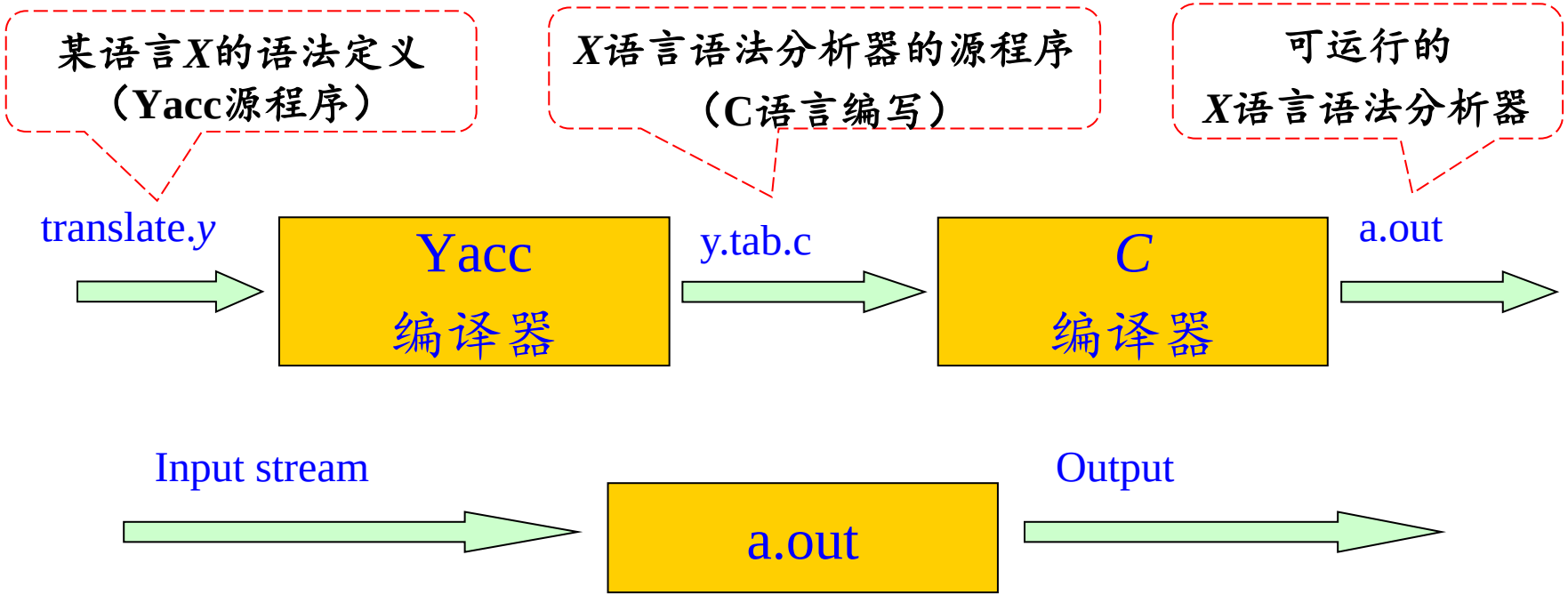
提纲

4.6 语法分析器自动生成工具

- Yacc (Yet Another Compiler's Compiler)

S.C.Johnson, Bell Laboratories, 1975~1978

  - LALR语法分析器生成工具



Yacc程序的结构

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

%token DIGIT

%%
line    : expr '\n'          { printf("%d\n", $1); }
        ;
expr    : expr '+' term      { $$ = $1 + $3; }
        | term
        ;
term    : term '*' factor    { $$ = $1 * $3; }
        | factor
        ;
factor  : '(' expr ')'       { $$ = $2; }
        | DIGIT
        ;
%%
yylex() {
    int c;
    c = getchar();
    if (isdigit(c)) {
        yylval = c-'0';
        return DIGIT;
    }
    return c;
}
```

声明部分

翻译规则

产生式 {语义动作}

辅助性C语言例程

本章小结

- 自顶向下的分析
- 预测分析法
  - LL(1) 文法
  - 递归的预测分析法
  - 非递归的预测分析法
  - 预测分析中的错误处理
- 自底向上的分析
- LR分析法
  - LR(0) 分析
  - SLR 分析
  - LR(1)分析
  - LALR (lookahead-LR)分析法
  - 二义性文法的LR分析
  - LR分析中的错误处理
- 语法分析器自动生成工具

结束