Outline

- ✧ LR(0) 自动机*
- ✧ SLR(1) 分析*
- ✧ LR(1) 分析*

LR 分析基础

✧ LR 分析表举例

– 文法： G[E]

（1）$E \rightarrow E+T$ （2） $E \rightarrow T$
（3）$T \rightarrow T*F$ （4） $T \rightarrow F$
（5）$F \rightarrow (E)$ （6） $F \rightarrow v$ （7）$F \rightarrow d$

栈顶状态	ACTION							GOTO		
	v	*d*	*	+	(	)	#	*E*	*T*	*F*
0	s5	s6			s4			1	2	3
1				s7			acc			
2			s8	r2		r2	r2			
3			r4	r4		r4	r4			
4	s5	s6			s4			9	2	3
5			r6	r6		r6	r6			
6			r7	r7		r7	r7			
7	s5	s6			s4				10	3
8	s5	s6			s4					11
9				s7		s12				
10			s8	r1		r1	r1			
11			r3	r3		r3	r3			
12			r5	r5		r5	r5			

LR(0)自动机

✧ LR(0)项目

– LR（0）项目（*item*）或配置（*configuration*）

- 一个LR（0）项目或配置是在右端某一位置有圆点的产生式

  如，产生式 **A→xyz** 对应如下 4 个 LR（0）项目：

  A→.xyz

  A→x.yz

  A→xy.z

  A→xyz.

  圆点标志着已分析过的串与该产生式匹配的位置

- 内核项：圆点不在最左端
- 非内核项：圆点在最左端

LR(0)自动机

✧ 核心概念

– 拓广文法（*augmented grammar*）

对于文法 $G = (V_N, V_T, P, S)$，增加如下产生式

$$S' \rightarrow S$$

其中，$S' \notin V_N \cup V_T$，得到 G 的拓广文法

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

说明：(1) 拓广文法等价于原文法;
(2) 拓广文法的开始符号不会出现在任何产生式的右部;
(3) LR(0)自动机的构造用到拓广文法
(4) $S' \rightarrow .S$ 是内核项

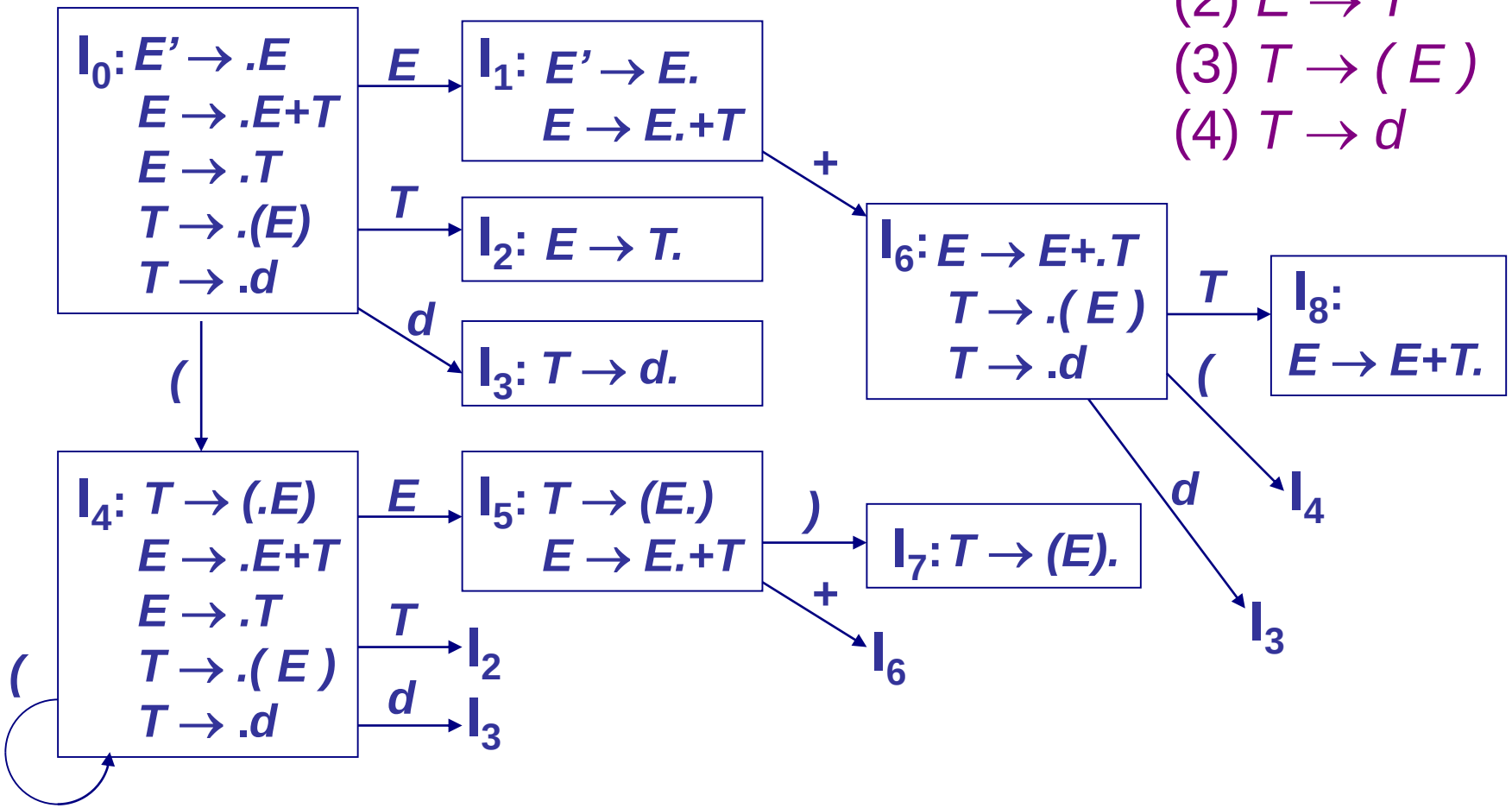
LR(0)自动机

✧ LR(0)自动机的构造举例

– 文法*G*[*E*]的拓广文法*G*'[*E*'] 的 LR(0)自动机

G[*E*]:

(1) $E \rightarrow E+T$
(2) $E \rightarrow T$
(3) $T \rightarrow (\ E\ )$
(4) $T \rightarrow d$



LR(0)自动机

✧ 核心概念

– LR(0)自动机

- 每个上下文无关文法 *G* 都对应一个**LR(0)**自动机
- 由 *G* 的拓广文法 *G'* 直接构造其 **LR(0)**自动机
- 文法 $G = (V_N, V_T, P, S)$ 的**LR(0)**自动机可以看作一个字母表为 $V_N \cup V_T$ 的 DFA（所有状态都是终态），可以证明该 DFA 的语言是 *G* 的所有活前缀的集合（后面讨论）

LR(0)自动机

✧ LR(0)自动机的构造

– LR(0)自动机的状态

- LR(0)自动机的状态是一个 LR（0）项目集的闭包（*closure*）
- 计算LR（0）项目集 I 的闭包 CLOSURE(I)的算法：

```
function  CLOSURE(I)
{   J:= I;
    repeat  for  J 中的每个项目A →α .Bβ  和 产生式 B→ γ
                 do 若B→ .γ 不在J中，则加 B→ .γ  到 J 中
    until   上一次循环不再有新项目加到J中
    return  J
};
```

LR(0)自动机

✧ LR(0)自动机的构造

– LR（0）项目解析

设 $G'[S']$ 是文法 $G=(V_N, V_T, P, S)$的拓广文法

根据圆点所在的位置和圆点后是终结符还是非终结符或为空，把项目分为以下几种：

移进项目: 形如 $A \rightarrow \alpha .a\beta$, 其中$a \in V_T$, $\alpha, \beta \in (V_N \cup V_T)^*$

待约项目: 形如 $A \rightarrow \alpha .B\beta$

归约项目: 形如 $A \rightarrow \alpha .$

接受项目: 形如 $S' \rightarrow S .$

LR(0)自动机

✧ LR(0)自动机的构造

– LR(0)自动机的初态

设文法 *G*[*S*] 的拓广文法为 *G'* [*S'*], 则 *G'*的**LR(0)**自动机的初态

$$\mathbf{I_0} = \text{CLOSURE}(\{\mathbf{S'\rightarrow .S}\})$$

例 右边文法*G*[*E*]的拓广文法为 *G'* [*E'*] ， 其 **LR(0)**自动机的初态

$\mathbf{I_0}$ = { *E'* → **.***E*,
E → **.***E*+*T*,
E → **.***T*,
T → **.**(*E*),
T → **.** *d*
}

G[*E*]:

(1) *E* → *E*+*T*
(2) *E* → *T*
(3) *T* → (*E*)
(4) *T* → *d*

LR(0)自动机

✧ LR(0)自动机的构造

– LR(0)自动机的状态转移函数

GO (I,X) = CLOSURE(J)

其中，**I**为**LR(0)**自动机的状态（项目集的闭包），**X** 为文法符号，**J={ A→αX.β | A→α.Xβ ∈I}**

– 从 **LR(0)**自动机的初态出发，应用上述转移函数，可逐步构造出完整的 **LR(0)**自动机

对于文法G，称其 LR(0)自动机的所有状态的集合为 G 的LR（0）项目集规范族

LR(0)自动机

✧ LR(0)自动机的构造

– 计算 LR（0）项目集规范族

设文法 *G*[*S*] 的拓广文法为 *G'* [*S'*], 则 *G'* 的LR（0）项目集规范族 **C** 可由如下算法计算：

C:= { CLOSURE ({S'→.S})}

Repeat

For C 中每一项目集 **I** 和每一文法符号**X**

Do if **GO(I,X)** 非空且不属于**C**

Then 把 **GO(I,X)** 放入**C**中

Until **C** 不再增大

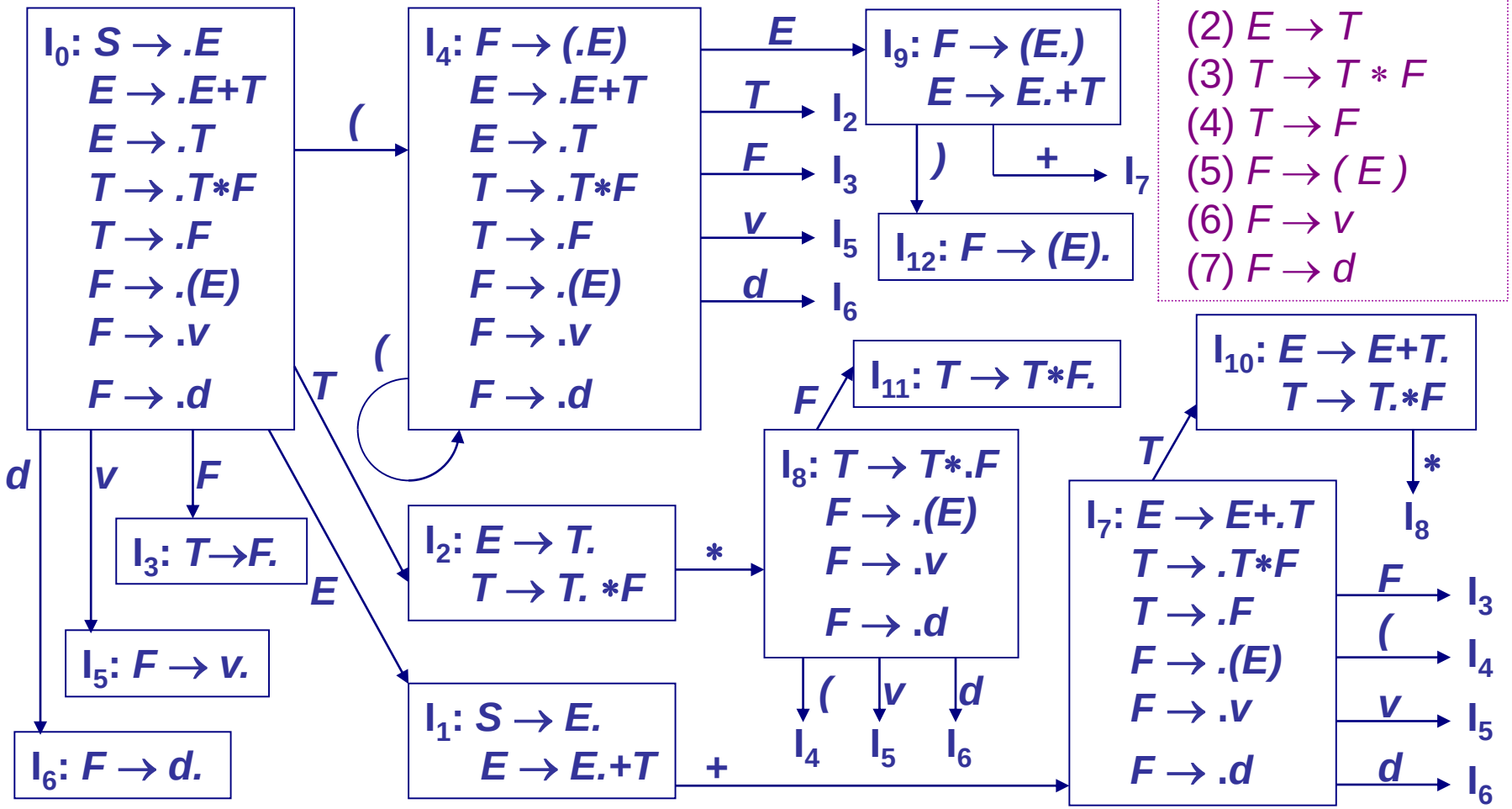
LR(0)自动机

✧ LR(0)自动机的构造举例

– 文法*G*[*E*]的拓广文法*G*'[*S*] 的 LR(0)自动机

拓广文法*G*'[*S*]:

(0) $S \to E$
(1) $E \to E + T$
(2) $E \to T$
(3) $T \to T * F$
(4) $T \to F$
(5) $F \to (E)$
(6) $F \to v$
(7) $F \to d$



LR(0)自动机

✧ 核心概念

– 文法的活前缀/可行前缀（*viable prefix*）

- 在移进–归约分析中，活前缀是可以出现在分析栈上的任何右句型之前缀
- 活前缀是任何右句型的前缀，它不超过该右句型最右句柄的右端
- 对于文法 $G = (V_N, V_T, P, S)$，以及
  $\alpha, \gamma \in (V_N \cup V_T)^*$，$\beta \in V_T^*$
  若 $S \overset{*}{\underset{rm}{\Rightarrow}} \alpha A \beta$ 且 $A \rightarrow \gamma$，则 $\alpha\gamma$ 的任何前缀 δ 都是文法 G 的活前缀

LR(0)自动机

✧ 活前缀举例

– 对于右边的文法G(S),

句子 aaab 是一个右句型，其唯一的句柄为：

ε： aaaεb;

所以 aaa 的任何前缀都是文法的活前缀：ε， a， aa， aaa

右句型 aaAb 的唯一的句柄为：

aA：aaAb;

所以 aaA 的任何前缀都是文法的活前缀：ε， a， aa， aaA

文法 G（S）：

（1）S → AB
（2）A → aA
（3）A → ε
（4）B → b
（5）B → bB

LR(0)自动机

✧ 活前缀与句柄的关系

- 活前缀不含有句柄的任何符号

  此时期待从输入串中看到该句柄对应的产生式 $A\rightarrow\alpha$的右部所推导出的符号串

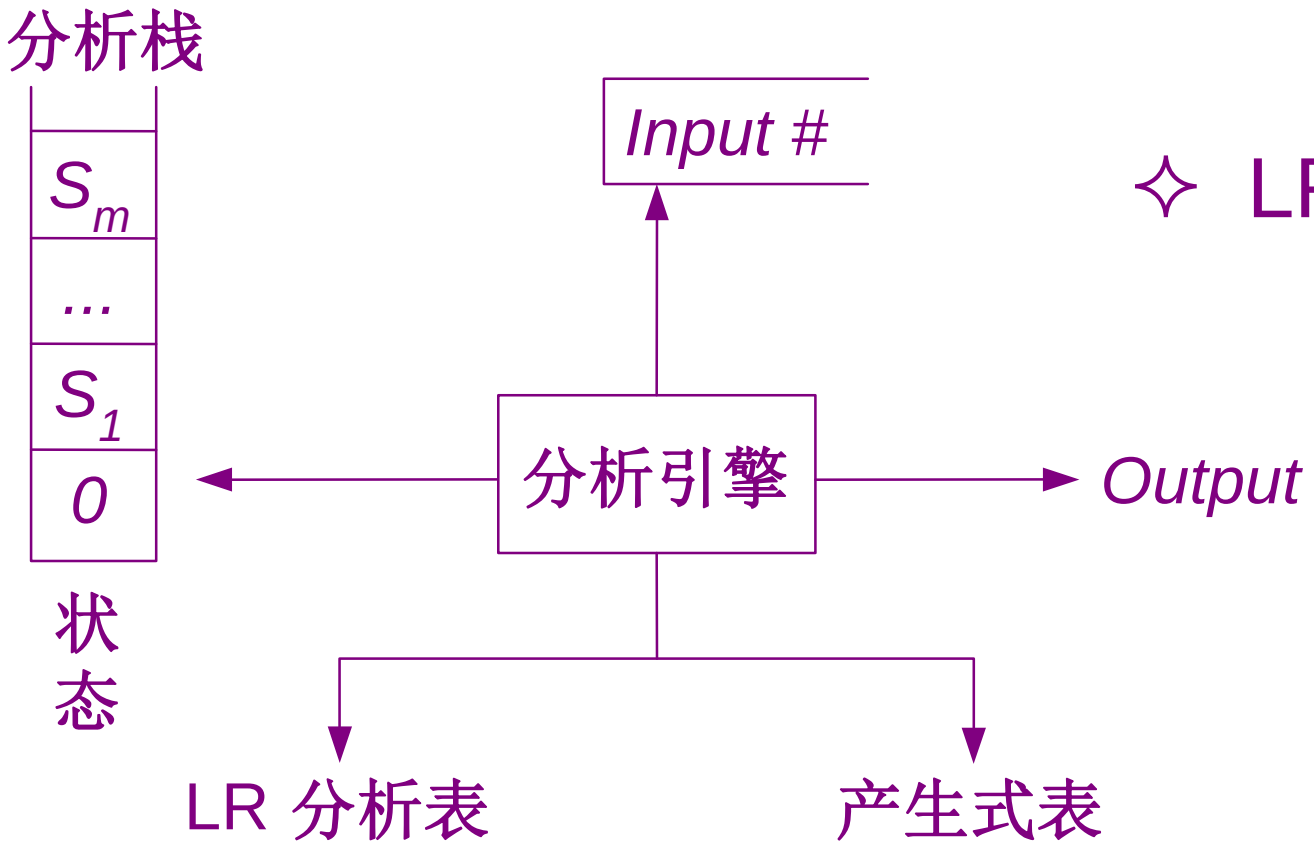
- 活前缀只含句柄的一部分符号

  表明该句柄对应的产生式 $A\rightarrow\alpha_1\alpha_2$ 的右部的子串α_1 已出现在栈顶，期待从输入串中看到α_2 推导出的符号串

- 活前缀已含有句柄的全部符号

  表明该句柄对应的产生式 $A\rightarrow\alpha$的右部α已出现在栈顶

SLR（1）分析



✧ LR 分析模型

✧ 如何获得 LR 分析表

– SLR（1）, LR（1）和 LALR（1）三种分析方法分别讨论

SLR（1）分析

✧ SLR（1）分析思想

– SLR（1）分析表的构造基于文法的 LR(0) 自动机

在SLR（1）分析表中，ACTION 表的归约表项只适用于相应非终结符Follow 集中的输入符号

即，根据下一个输入符号是否属于要归约的非终结符的 **Follow** 集来决定是否进行归约

SLR（1）分析

✧ SLR（1）分析表的构造

– 假定*G*[*S*]的拓广文法为*G'*[*S*]，其LR(0)A的状态集为**C**={I_0, I_1,…，I_n}；令状态I_k对应的 SLR（1）分析表的栈顶状态为k ；并令含有项目*S'*→.*S* 的项目集为I_0, 因此0为初态。ACTION表项和 GOTO 表项可按如下方法构造：

- 若GO (I_k, A)= I_j, A为非终结符，则置GOTO(k, A)=j;
- 若项目A→**α**.a**β**属于 I_k 且 GO (I_k, a)= I_j, a 为终结符，则置 ACTION[k, a] 为“把状态j移进栈”，简记为“sj”;
- 若项目A→**α**. 属于I_k, 那么，对任何a∈Follow(A), 置ACTION[k, a]为“用产生式A→**α**进行归约”，简记为“rj”;其中，假定A→**α**为文法*G'*的第j个产生式;
- 若项目*S'*→*S*. 属于I_k, 则置ACTION[k, #]为“接受”，简记为“acc”;
- 分析表中凡不能用上述规则填入信息的空白格均置上“出错标志”

SLR（1）分析

- SLR（1）分析表的构造举例
  - 拓广文法：$G'[S]$

（0）$S \to E$ （1）$E \to E+T$ （2）$E \to T$
（3）$T \to T*F$ （4）$T \to F$
（5）$F \to (E)$ （6）$F \to v$ （7）$F \to d$

栈顶状态	ACTION							GOTO		
	v	*d*	*	+	(	)	#	*E*	*T*	*F*
0	s5	s6			s4			1	2	3
1				s7			acc			
2			s8	r2		r2	r2			
3			r4	r4		r4	r4			
4	s5	s6			s4			9	2	3
5			r6	r6		r6	r6			
6			r7	r7		r7	r7			
7	s5	s6			s4				10	3
8	s5	s6			s4					11
9				s7		s12				
10			s8	r1		r1	r1			
11			r3	r3		r3	r3			
12			r5	r5		r5	r5			

SLR（1）分析

✧ SLR（1）文法

- 按上述算法构造的分析表，如果各表项均无多重定义，则称它为文法 *G* 的一张 SLR（1）表，并称 *G* 为一个 SLR（1）文法
- SLR（1）文法的LR(0)A中，每个状态都满足：
  - 对该状态的任何项目A→u.av（为终结符），不存在项目 B→w. 使得 a∈Follow(B)
  - 对该状态的任何两个项目A→u.和B→v.，满足
    Follow(A) ∩ Follow(B) = Φ

SLR（1）分析

✧ SLR 分析表

– 使用两张表

- ACTION 表 告诉分析引擎：在栈顶状态为*k*, 当前输入符号是 **a** 时做什么
  ACTION [*k*,a]=*si*， *Shift*：状态 *i* 移进栈顶
  ACTION [*k*,a]=*rj*， *Reduce*：按第 *j* 条产生式归约
  ACTION [*k*,a]=*acc*， *Accept*：分析完成
  ACTION [*k*,a]=*err*，*Error*：发现错误（常标为空白）
- GOTO 表 GOTO[***i***,*A*]=***j*** 告诉分析引擎：
  在依产生式 $A\rightarrow\beta$ 归约之后，栈顶状态为***i*** 时，要将新状态为 ***j*** 移进栈顶
  （依产生式 $A\rightarrow\beta$ 归约时，要将栈顶的 $|\beta|$ 个状态弹出）

SLR（1）分析

✧ SLR 分析算法

– 置 *ip* 指向输入串 *w* 的首符号，置初始栈顶状态为 ***0***
令 *i* 为栈顶状态，**a** 是 *ip* 指向的符号，重复如下步骤：

```
if ( ACTION[i,a]=sj ) {
    PUSH j ; /*进栈*/      ip 前进 ; /*指向下一输入符号*/
}
else if （ACTION[i,a]=rj ) {     /*第 j 条产生式为 A→β */
        POP |β| 项; /*位于栈顶部的 |β| 个状态退栈*/
        令当前栈顶状态为 k; PUSH GOTO[k,A];
    }
    else if （ACTION[i,a]=acc ) return; /*成功*/
        else error;     /*报错/错误恢复*/
```

SLR（1）分析

✧ SLR 分析过程举例

– 文法：*G*[*E*]

– 输入串： ***v*** + ***v*** ∗ ***d***

（1）*E*→*E*+*T*　（2）*E*→*T*

（3）*T*→*T*∗*F*　（4）*T*→*F*

（5）*F*→(*E*)　（6）*F*→*v*　（7）*F*→*d*

分析栈	余留输入串	分析动作
0	***v*** + ***v*** ∗ ***d*** #	ACTION [*0*,*v*]=*s5*
0 5	+ ***v*** ∗ ***d*** #	ACTION [*5*,+]=*r6*, GOTO [*0*,*F*]=*3*
0 3	+ ***v*** ∗ ***d*** #	ACTION [*3*,+]=*r4*, GOTO [*0*,*T*]=*2*
0 2	+ ***v*** ∗ ***d*** #	ACTION [*2*,+]=*r2*, GOTO [*0*,*E*]=*1*
0 1	+ ***v*** ∗ ***d*** #	ACTION [*1*,+]=*s7*
0 1 7	***v*** ∗ ***d*** #	ACTION [*7*,*v*]=*s5*
0 1 7 5	∗ ***d*** #	ACTION [*5*,∗]=*r6*, GOTO [*7*,*F*]=*3*
0 1 7 3	∗ ***d*** #	ACTION [*3*,∗]=*r4*, GOTO [*7*,*T*]=*10*
0 1 7 10	∗ ***d*** #	ACTION [*10*,∗]=*s8*
0 1 7 10 8	***d*** #	ACTION [*8*,*d*]=*s6*
0 1 7 10 8 6	#	ACTION [*6*,#]=*r7*, GOTO [*8*,*F*]=*11*
0 1 7 10 8 11	#	ACTION [*11*,#]=*r3*, GOTO [*7*,*T*]=*10*
0 1 7 10	#	ACTION [*10*,#]=*r1*, GOTO [*0*,*E*]=*1*
0 1	#	ACTION [*1*,#]=*acc*

SLR（1）分析

✧ SLR（1）文法

- 按上述算法构造的分析表，如果各表项均无多重定义，则称它为文法 *G* 的一张 SLR（1）表，并称 *G* 为一个 SLR（1）文法
- SLR（1）文法的LR(0)A中，每个状态都满足：
  - 对该状态的任何项目A→u.av（为终结符），不存在项目 B→w. 使得 a∈Follow(B)
  - 对该状态的任何两个项目A→u.和B→v.，满足
    Follow(A) ∩ Follow(B) = Φ

LR（1）分析

✧ SLR（1）分析的局限性举例

– 验证如下文法不是 SLR（1）的

文法 *G*[*E*]:

(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$

G[*E*] 的拓广文法 *G*’[*S*]:

(0) $S \rightarrow E$
(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$

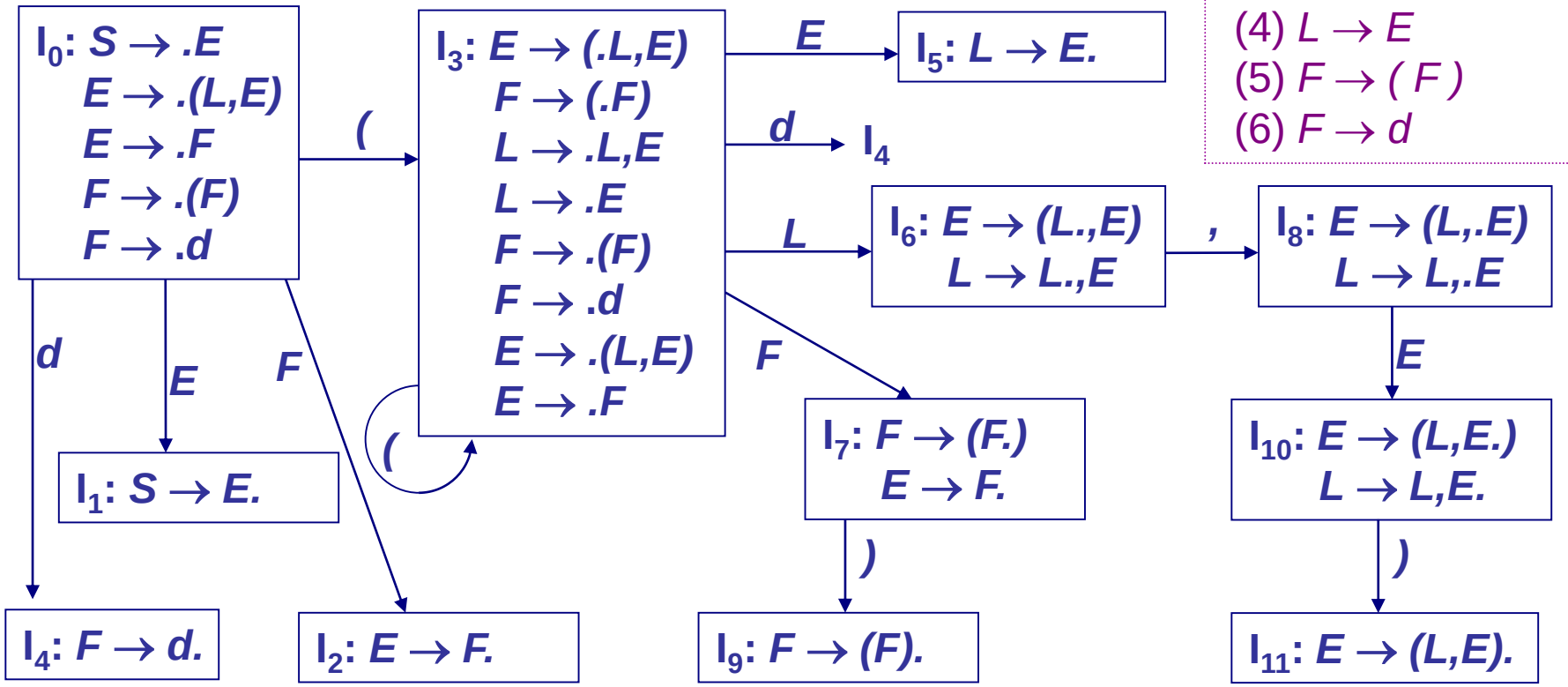
LR（1）分析

✧ SLR（1）分析的局限性举例

– 构造拓广文法*G'*[S] 的 LR(0)A

拓广文法*G'*[S]:

(0) $S \rightarrow E$
(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$



LR（1）分析

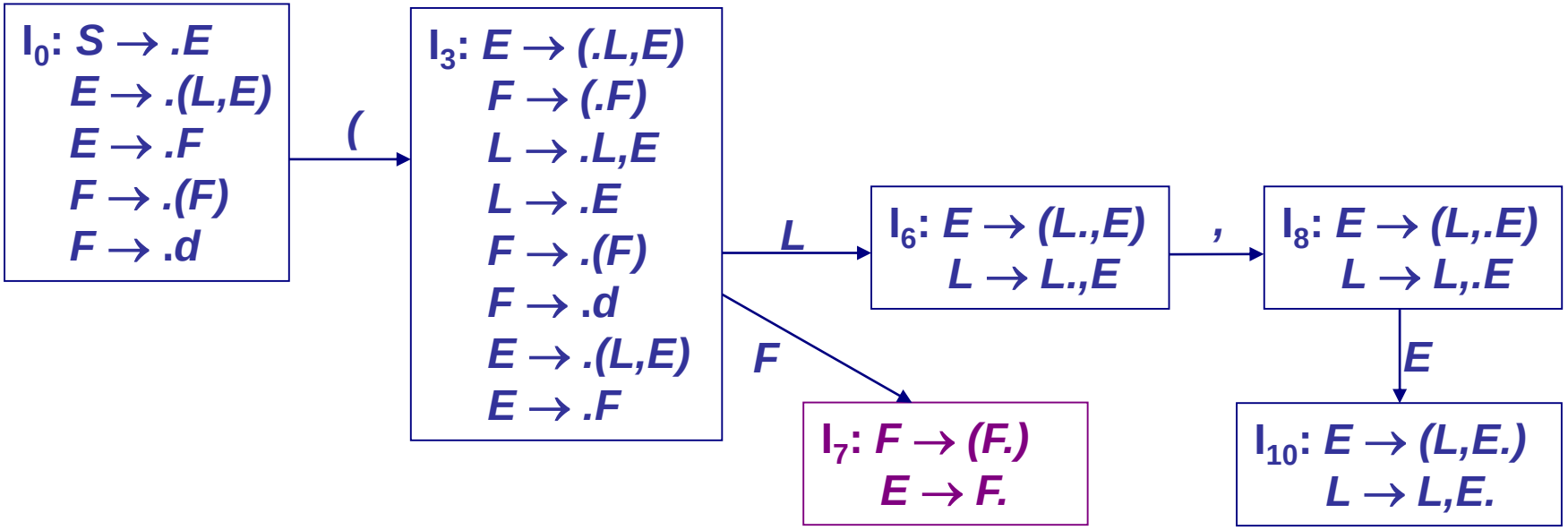
✧ SLR（1）分析的局限性举例

– 文法*G'*[*S*] 不是SLR（1）文法

状态 I_7 的移进–归约冲突无法用 SLR（1）分析方法解决

拓广文法*G'*[*S*]:

(0) $S \rightarrow E$
(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$



LR（1）分析

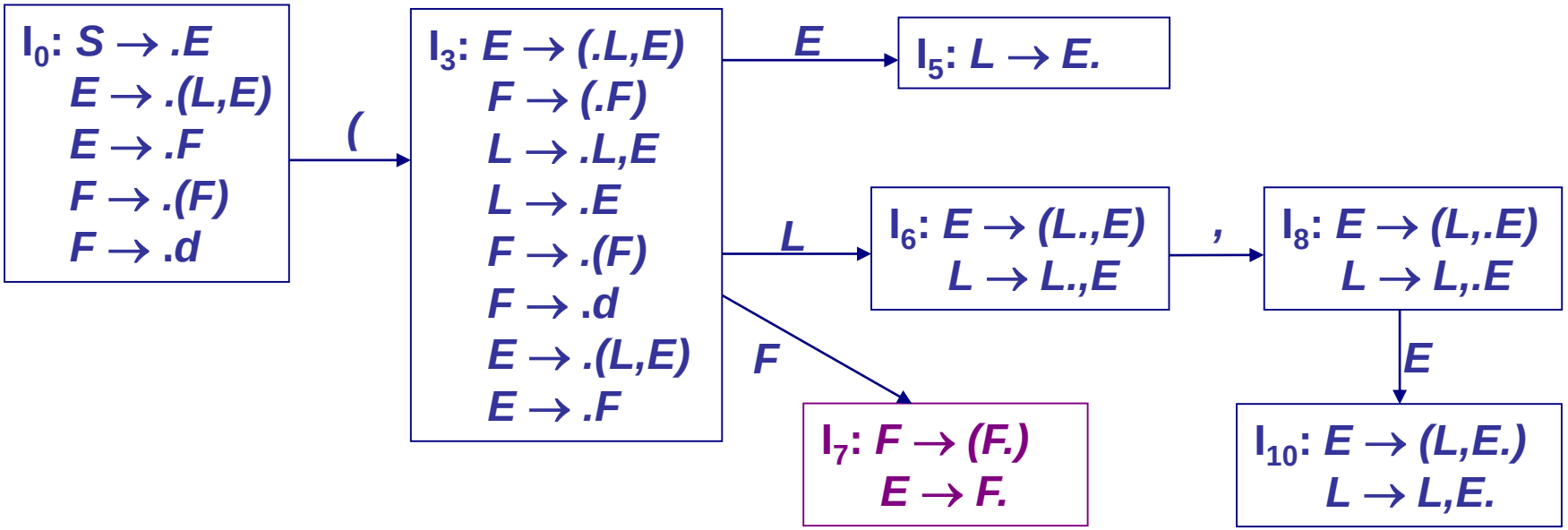
✧ SLR（1）分析的局限性举例

– 状态 I_7 的冲突是可以解决的

从 I_3 和 I_5 容易看出，到达状态 I_7（栈上的活前缀为*(F*）时，句柄 *F* 所期望的下一个输入符号实际上是 *,*，而不是 *)*

拓广文法*G'*[*S*]:

(0) $S \rightarrow E$
(1) $E \rightarrow (L\ ,\ E\ )$
(2) $E \rightarrow F$
(3) $L \rightarrow L\ ,\ E$
(4) $L \rightarrow E$
(5) $F \rightarrow (\ F\ )$
(6) $F \rightarrow d$



LR（1）分析

✧ SLR（1）分析的局限性

– 只考虑到所归约非终结符的 Follow 符号

虽然是向前查看一个输入符号，但只要输入符号属于所归约非终结符的 Follow 集合，就可进行归约

– 未考虑非终结符 Follow 集中的符号是否也是句柄的 Follow 符号

一个输入符号属于所归约非终结符的 Follow 集合，未必就是句柄可以后跟的符号

LR（1）分析

✧ LR（1）项目

– 在 LR（0）项目基础上增加一个终结符

所增加的终结符称为向前搜索符（*lookahead*）
表示产生式右端完整匹配后所允许在余留符号串中的下一个终结符

– LR（1）项目形如：

$$A \rightarrow \alpha . \beta , a$$

$A \rightarrow \alpha . \beta$ 同 LR（0）项目， **a** 为向前搜索符。

这里，**a** 或为终结符，或为输入结束标志符 #

LR（1）分析

✧ LR（1）项目解析

– 对于形如：

$$A \rightarrow \alpha\,.\,,\ a$$

的 LR（1）项目，对应 LR（0）的归约项目，但只有当下一个输入符是 a 时才能进行归约

对于其它形式的LR（1）项目， a 只起到信息传承的作用

（参见随后的 LR（1） 自动机构造过程）

LR（1）分析

✧ 记号

– 若形如

$$A \to \alpha . \beta,\ a_1$$
$$A \to \alpha . \beta,\ a_2$$
$$\ldots$$
$$A \to \alpha . \beta,\ a_m$$

的 LR（1）项目序列需要出现在同一个项目集中时，将其简记为

$$A \to \alpha . \beta,\ a_1 / a_2 / \ldots / a_m$$

LR（1）分析

✧ LR（1）A

– 类似于 LR(0) A, 可以在 LR（1）项目的基础上为上下文无关文法 *G* 构造一个类似的有限状态机，称为 LR(1)自动机（LR(1) A）

LR（1）分析

✧ LR（1）A的构造

– LR（1）A的状态

- LR（1） A 的状态是 LR（1）项目集的闭包（*closure*）
- 计算LR（1）项目集 **I** 的闭包 CLOSURE(**I**) 的算法：

```
function  CLOSURE(I)
{   J:= I;
    repeat  for  J 中的每个项目 [A→α . Bβ ， a] 和产生式 B→ γ
                 do 将所有形如 [B→ .γ , b]  的项目加到 J 中, 这里
                      b ∈ First（βa）
    until   上一次循环不再有新项目加到 J 中
    return  J
};
```

LR（1）分析

✧ LR（1）A的构造

– LR（1）A的初态

设文法 $G[S]$ 的拓广文法为 $G'[S']$, 则 G'的 LR（1）A 的初态　　$\mathbf{I_0 = CLOSURE(\{[S' \rightarrow .S, \#]\})}$

例 设右边文法$G[S]$的拓广文法为 $G'[S]$，其 LR（1）A 的初态

$\mathbf{I_0}$:　$S \rightarrow .E, \#$

$E \rightarrow .(L, E), \#$

$E \rightarrow .F, \#$

$F \rightarrow .(F), \#$

$F \rightarrow .d, \#$

文法 $G[E]$:

(1) $E \rightarrow (L, E)$

(2) $E \rightarrow F$

(3) $L \rightarrow L, E$

(4) $L \rightarrow E$

(5) $F \rightarrow (F)$

(6) $F \rightarrow d$

LR（1）分析

✧ LR（1）A的构造

– LR（1）A的状态转移函数

GO (I,X) = CLOSURE(J)

其中，**I**为自动机的状态（闭包的项目集），**X** 为文法符号，**J={ [A→αX . β , a] | [A→α . Xβ , a] ∈ I}**

– 从 LR（1）A 的初态出发，应用上述转移函数，可逐步构造出完整的 LR（1）A

对于文法*G*，称其 LR（1）A的所有状态的集合为 *G* 的 LR（1）项目集规范族

LR（1）分析

✧ LR（1）A的构造

– 计算 LR（1）项目集规范族

设文法 *G*[*S*] 的拓广文法为 *G'* [*S'*], 则 *G'* 的LR（1）项目集规范族 **C** 可由如下算法计算：

C:= { CLOSURE ({ [S'→**.**S, #] }) }

Repeat

For **C** 中每一项目集 **I** 和每一文法符号**X**

Do if **GO(I,X)** 非空且不属于 **C**

Then 把 **GO(I,X)** 放入 **C** 中

Until **C** 不再增大

Conclusions

- LR(0) 自动机
  - 活前缀
- SLR(1)分析
  - SLR(1)分析表
  - SLR(1)文法
  - SLR(1)分析算法
- LR(1)分析
  - SLR(1)分析的局限性
  - SLR(1)分析

✧ § 4.6 Introduction to LR Parsing

✧ § 4.7 More Powerful LR Parsers

Thank you!