

第6章 LR分析

6.1 自下而上分析及其LR分析概述

6.2 LR (0) 分析

6.3 SLR(1) 分析

6.4 LR (1) 分析

6.5 LALR(1)分析

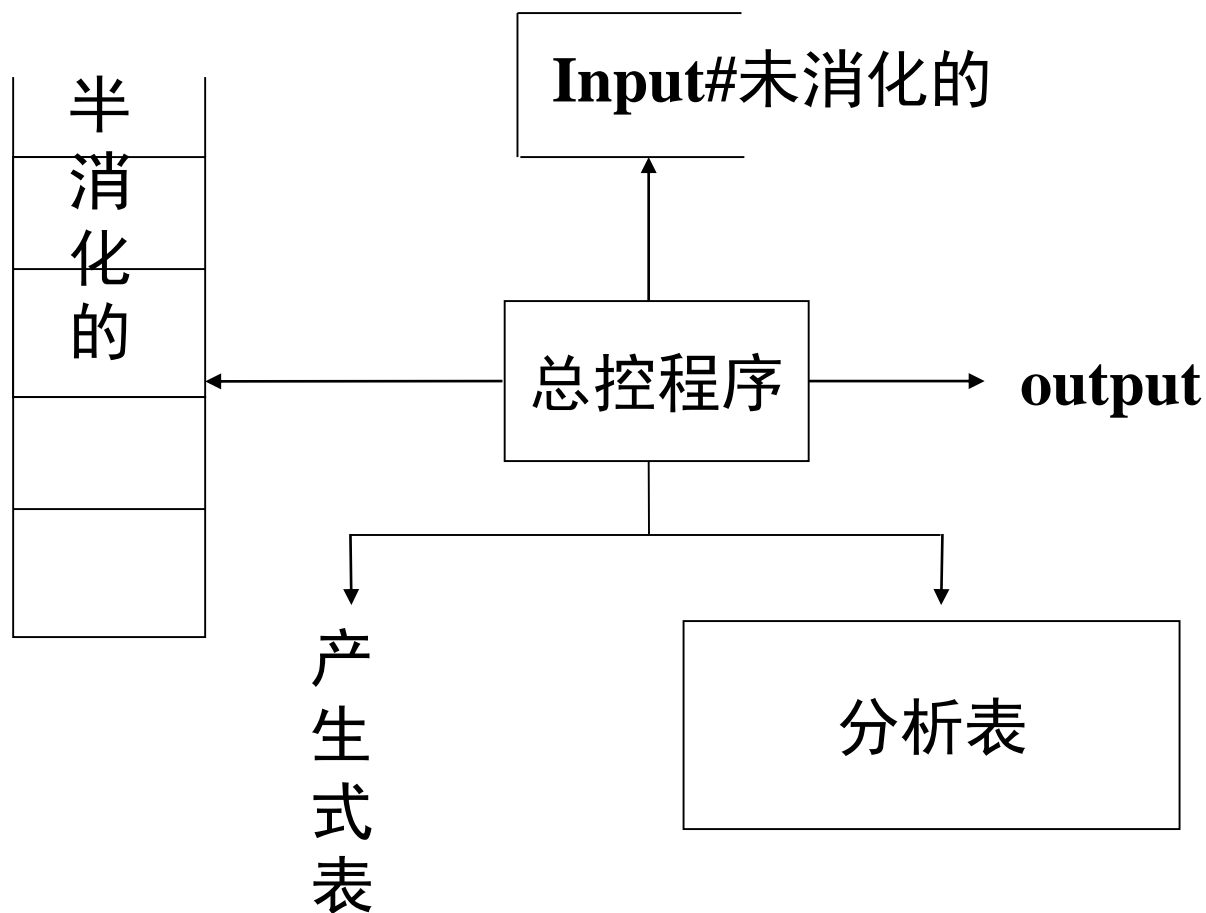
6.6 使用二义文法

6.7 语法分析程序的自动构造工具YACC

自下而上分析算法：能力强、构造复杂

最常用和最有效的模型---移进-归约

将输入分成两部分：**未消化的**和**半消化的**



$S \rightarrow E$ $E \rightarrow T \mid E + T$ $T \rightarrow \text{int} \mid (E)$

Reduce: 如能找到一产生式 $A \rightarrow w$ 且栈中的内容是 qw (q 可能为空), 则可以将其归约为 qA , 即倒过来用这个产生式。

如上例, 若栈中内容是 $(\text{int}$, 我们使用产生式 $T \rightarrow \text{int}$ 并把栈中内容归约为 $(T$ 。

Shift: 如不能执行一个归约且在未消化的输入中还有 token, 就把它从输入移到栈中。

如上例, 假定栈中内容是 $($, 输入中还有 $\text{int}+\text{int})\#$ 。不能对 $($ 执行一个归约, 因为它不和任何产生式的右端匹配。所以把输入的第一个符号移到栈中, 于是栈中内容是 $(\text{int}$, 而余留的输入是 $+\text{int})\#$ 。

Reduce的一个特殊情况：栈中的全部内容 w 归约为开始符号 S （即施用 $S \rightarrow w$ ），且没有余留输入了，意味着已成功分析了整个输入串。

移进归约分析中还会出现一种情况，就是**出错**，比如当前的token不能构成一个合法句子的一部分，例如上面的文法，试分析 int+)时就会发生错误。

移进-归约模型分析(int + int)的过程

	STACK	REMAINING INPUT	PARSER ACTION
1		(int + int)#	Shift
2	(int + int)#	Shift
3	(int	+ int)#	Reduce: $T \rightarrow \text{int}$
4	(T	+ int)#	Reduce: $E \rightarrow T$
5	(E	+ int)#	Shift
6	(E +	int)#	Shift
7	(E + int)#	Reduce: $T \rightarrow \text{int}$
8	(E + T)#	Reduce: $E \rightarrow E + T$
9	(E)#	Shift
10	(E)	#	Reduce: $T \rightarrow (E)$
11	T	#	Reduce: $E \rightarrow T$
12	E	#	Reduce: $S \rightarrow E$
13	S	#	

$S \rightarrow E$ $E \rightarrow T \mid E + T$ $T \rightarrow \text{int} \mid (E)$ $(E + T) \#$ Reduce: $E \rightarrow E + T$ why不用 $E \rightarrow T$? $(E) \#$

若使用了 $E \rightarrow T$, 在栈中形成的 $(E+E$ 不是规范句型的
活前缀 (viable prefixes)

$(E+E$ 不能和任何产生式的右端匹配 $(E+E)$ 不是规范
句型

活前缀 是规范句型的前缀, 但不超过句柄

移进归约分析的栈中出现的内容加上剩余输入构成
规范句型

规范推导 规范句型 规范归约

最右推导：在推导的任何一步 $\alpha \Rightarrow \beta$ ，其中 α 、 β 是句型，都是对 α 中的最右非终结符进行替换

最右推导被称为**规范推导**

由规范推导所得的句型称为**规范句型**

G[S]: $S \rightarrow E \quad E \rightarrow E+T \mid T \quad T \rightarrow (E) \mid \text{int}$
 $S \Rightarrow E \Rightarrow T \Rightarrow (E) \Rightarrow (E+T) \Rightarrow (E+\text{int})$
 $\Rightarrow (T+\text{int}) \Rightarrow (\text{int}+\text{int})$

规范归约

假定 α 是G的一个句子，称序列 $\alpha_n, \alpha_{n-1}, \dots, \alpha_0$ 是 α 的一个规范归约，

如果该序列满足：

- (1) $\alpha_n = \alpha$
- (2) α_0 为文法的开始符号
- (3) 对任何 j , $0 < j \leq n$, α_{j-1} 是从 α_j 经把句柄替换为相应产生式的左部而得到的

文法要求

shift-reduce or reduce-reduce 冲突 (conflicts)

分析程序不能决定是 *shift* 还是 *reduce*

或者分析程序归约时有多个产生式可选

例子 (*dangling else*) :

$S \rightarrow \text{if } E \text{ then } S \mid \text{if } E \text{ then } S \text{ else } S$

如输入 *if E then if E then S else S*,

分析某一时刻, 栈的内容: *if E then if E then S*

而 *else* 是下一 token, 归约还是移进?

一种shift-reduce实现技术

LR 分析

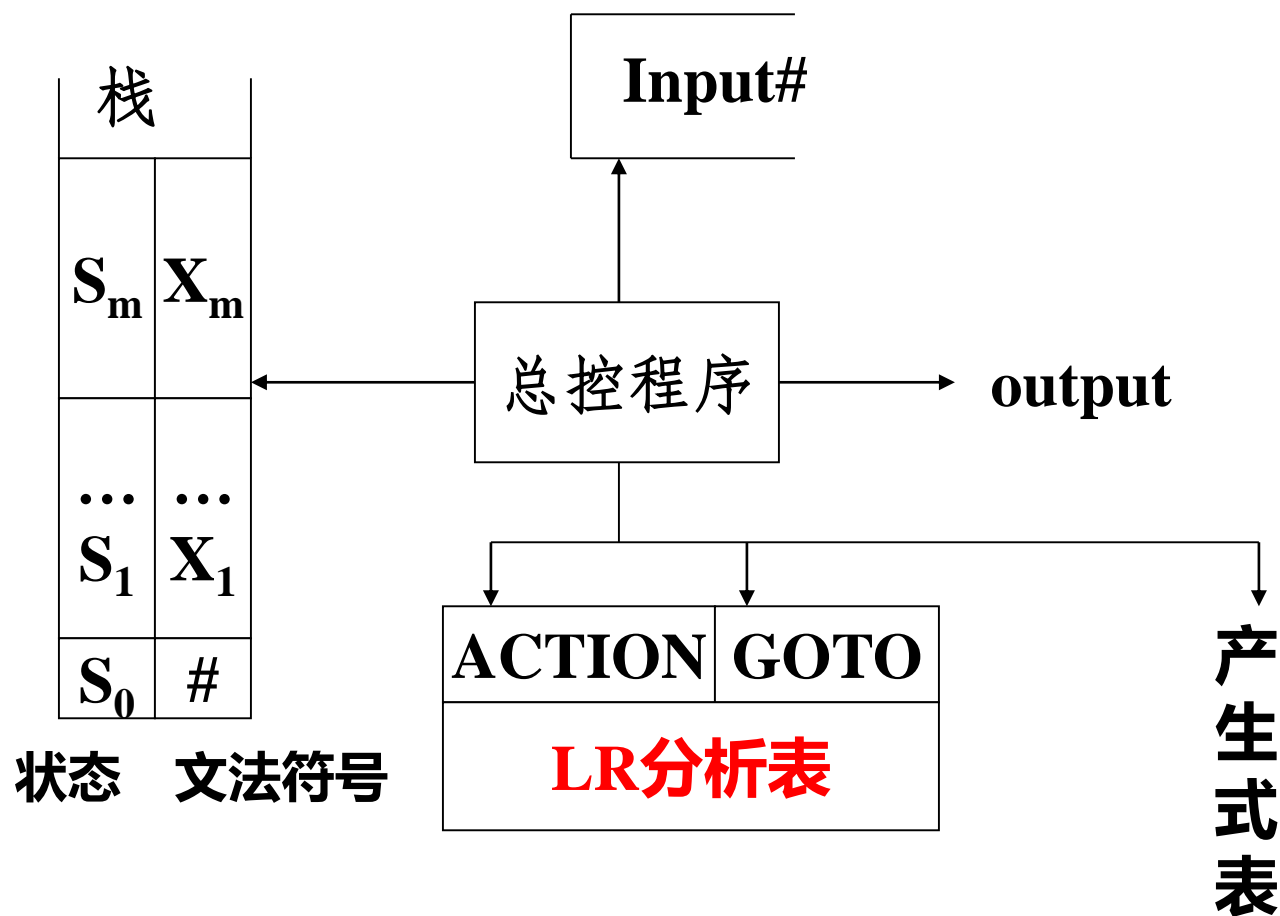
L

R 最右推导

分析器模型和分析算法

LR 分析特征讨论

LR分析器模型



LR分析表

[illegible]

LR分析算法

置ip指向输入串w的第一个符号

令S为栈顶状态

a是ip指向的符号

repeat begin

if ACTION[S,a]=S_j

then begin PUSH j,a(进栈)

ip 前进(指向下一输入符号)

end

else if ACTION[S,a]=r_j (第j条产生式为A→β)

LR分析算法

then begin

pop $|\beta|$ 项

令当前栈顶状态为 S'

push GOTO[S' , A]和A(进栈)

end

else if ACTION[s,a]=acc

then return (成功)

else error

end

例6.1:
$$G[S]: S \rightarrow aAcBe[1]$$
$$A \rightarrow b[2]$$
$$A \rightarrow Ab[3]$$
$$B \rightarrow d[4]$$
$$w = abbcde\#$$

Step	States	Syms	The rest of input	Action	Goto
1	0	#	abbcde#	s2	
2	02	#a	bbcde#	s4	
3	024	#ab	bcde#	r2	goto(2,A)
4	023	#aA	bcde#	s6	
5	0236	#aAb	cde#	r3	
6	023	#aA	cde#	s5	
7	0235	#aAc	de#	s8	
8	02358	#aAcd	e#	r4	
9	02357	#aAcB	e#	s9	
10	023579	#aAcBe	#	r1	
11	01	#S		acc	

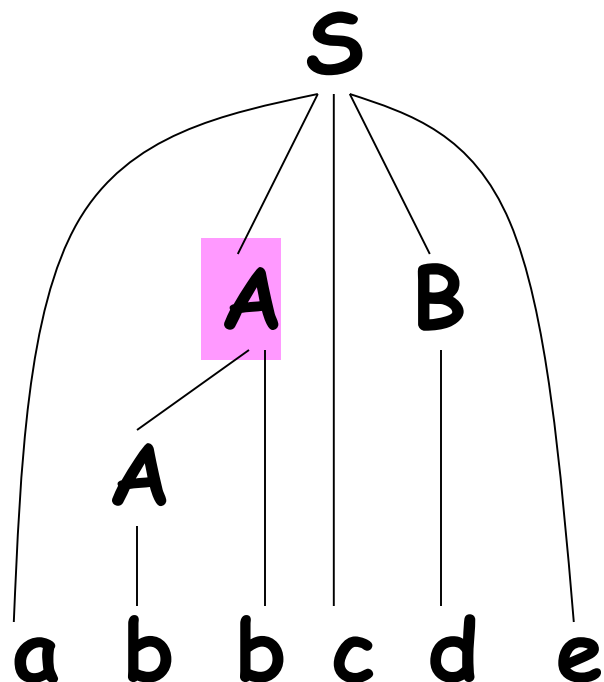
文法G[S]:

(1) $S \rightarrow aAcBe$

(2) $A \rightarrow b$

(3) $A \rightarrow Ab$

(4) $B \rightarrow d$



步骤	符号栈	输入符号串	动作
1)	#	abbcde#	移进
2)	#a	bbcd#	移进
3)	#ab	bcde#	归约($A \rightarrow b$)
4)	#aA	bcde#	移进
5)	#aAb	cde#	归约($A \rightarrow Ab$)
6)	#aA	cde#	移进
7)	#aAc	de#	移进
8)	#aAcd	e#	归约($B \rightarrow d$)
9)	#aAcB	e#	移进
10)	#aAcBe	#	归约
11)	#S	#	接受

对输入串abbcde#的移进-规约分析过程

字符串abbcde是否是G[S]的句子

$S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde \Rightarrow abbcde$

[illegible]

r_i :归约，用第 i 个产生式归约，同时状态栈与符号栈退出相应个符号，并把GOTO表相应状态和第 i 个产生式的左部非终结符入栈。

LR 文法

对于一个cfg 文法, 如果能够构造一张 LR 分析表, 使得它的每一个入口均是唯一的 ($S_j, r_j, \text{acc}, \text{空白}$), 则称该 cfg 是LR 文法.

LR分析

特征:

- 规范的
- 符号栈中的符号串是规范句型的前缀，且其最右符号不超过句柄的末端(活前缀)
- 分析决策依据——栈顶状态和现行输入符号、识别活前缀的 DFA

四种技术

LR(0) SLR(1) LR(1) LALR(1)

LR(0) 分析

LR(0)文法

能力最弱，理论上最重要

存在FA 识别活前缀

识别活前缀的DFA如何构造

(LR(0)项目集规范族的构造)

LR(0)分析表的构造

拓广文法

为使文法的初始符号不出现在任何产生式的右部，需对文法 $G[S]$ 进行拓广：在原文法 G 中增加 $S' \rightarrow S$ 产生式。

文法 $G[S]$:

- (1) $S \rightarrow aAcBe$
- (2) $A \rightarrow b$
- (3) $A \rightarrow Ab$
- (4) $B \rightarrow d$

文法 $G[S']$:

- (0) $S' \rightarrow S$
- (1) $S \rightarrow aAcBe$
- (2) $A \rightarrow b$
- (3) $A \rightarrow Ab$
- (4) $B \rightarrow d$

最右推导过程:

$$\begin{aligned} S' &\Rightarrow S[0] \Rightarrow aAcBe[1][0] \Rightarrow aAcd[4]e[1][0] \\ &\Rightarrow aAb[3]cd[4]e[1][0] \\ &\Rightarrow ab[2]b[3]cd[4]e[1][0] \end{aligned}$$

规约时在栈中的句型的前缀

ab[2]
aAb[3]
aAcd[4]
aAcBe[1]
S[0]

规约前可在栈中出现的规范句型(不含句柄)的前缀

a, ab
a, aA, aAb
a, aA, aAc, aAcd
a, aA, aAc, aAcB, aAcBe
a, aA, aAc, aAcB, S

活前缀

给定文法 $G=(V_n, V_t, P, S)$, 若有规范推导

$S' \xRightarrow{*}_R \alpha A \omega \Rightarrow \alpha \beta \omega$, γ 是 $\alpha \beta$ 的前缀, 则称 γ 是文法 G 的活前缀.

例如:

a, ab

a, aA, aAb

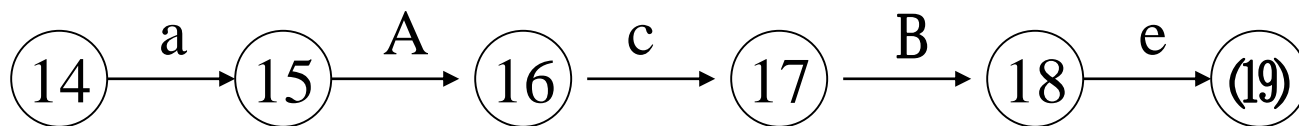
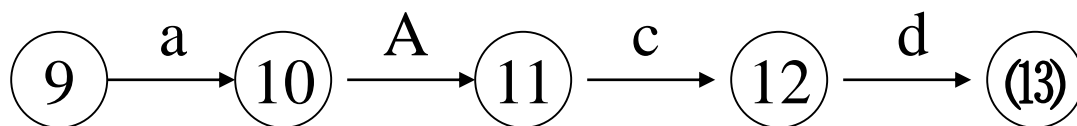
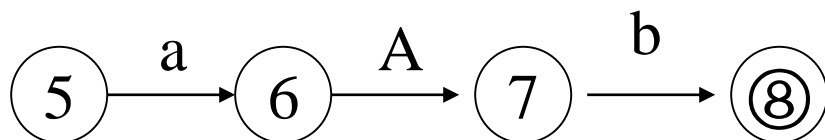
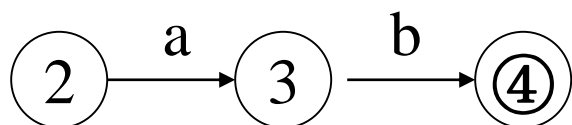
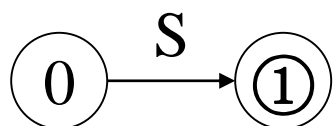
$a, aA, aAc, aAcd$

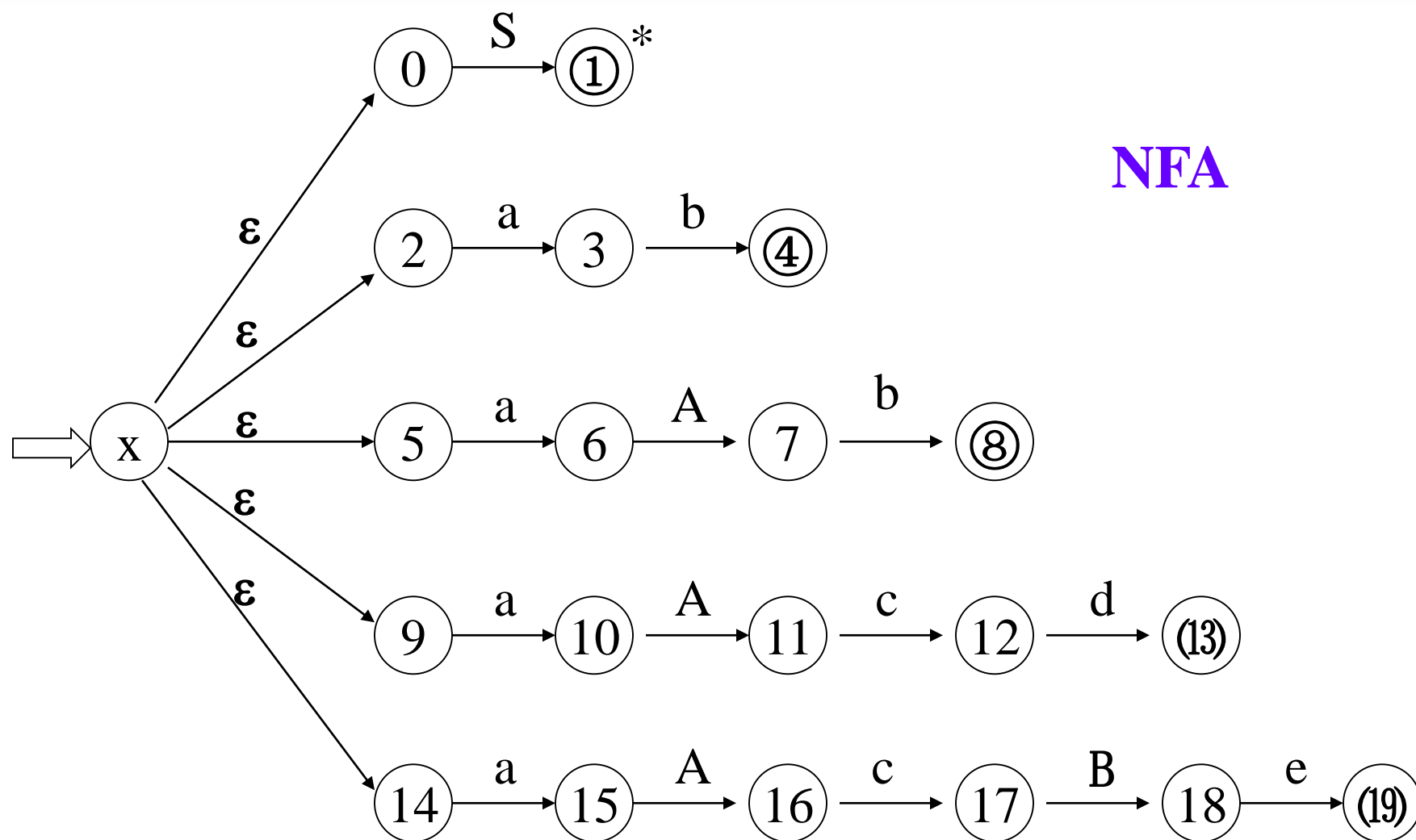
$a, aA, aAc, aAcB, aAcBe$

$a, aA, aAc, aAcB, S$

$S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde \Rightarrow abbcde$

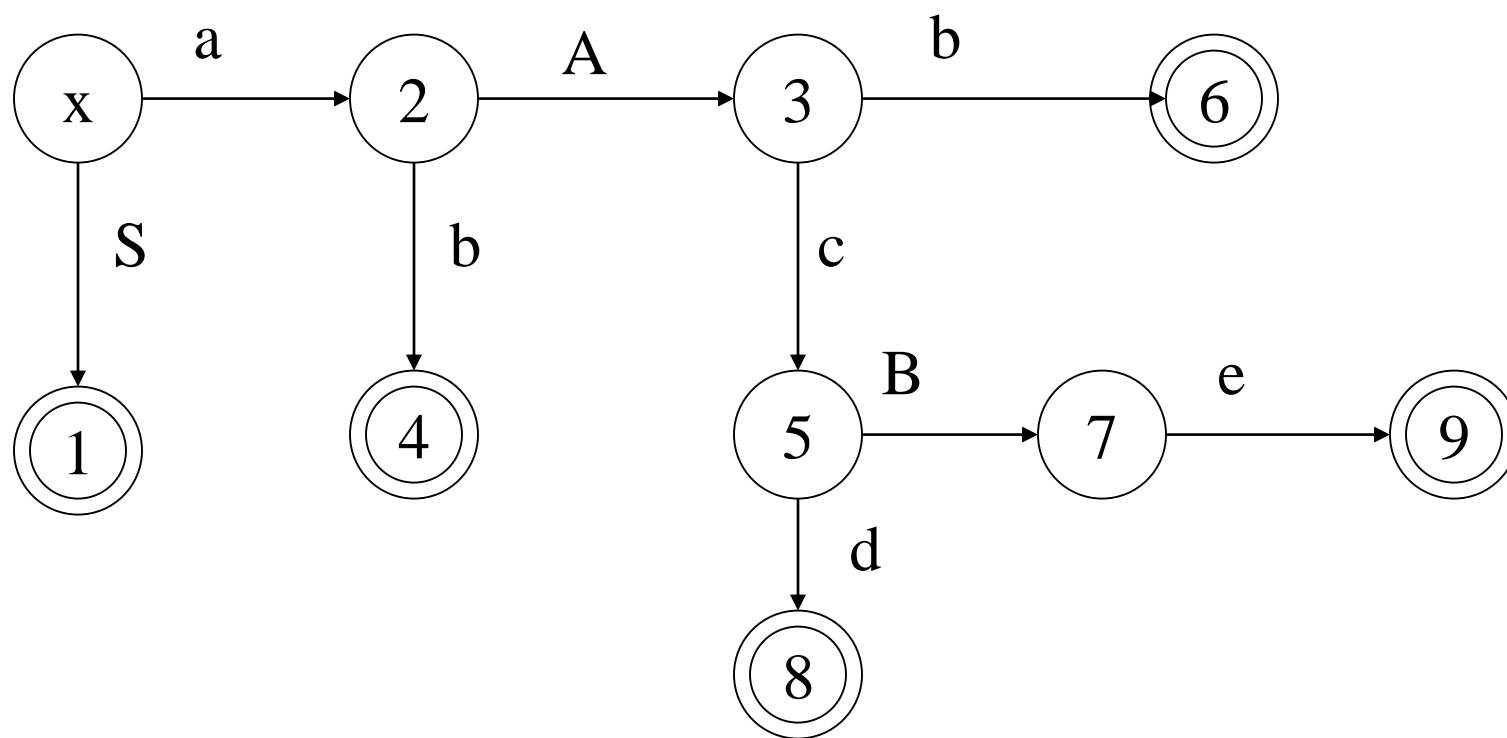
识别活前缀的NFA





$S' \Rightarrow S \Rightarrow aAcBe \Rightarrow aAcde \Rightarrow aAbcde \xRightarrow{i} ab^ibcde$

DFA



活前缀及可归前缀的计算

定义(非终结符的左文)

$$LC(A) = \{\beta \mid S' \xRightarrow[R]{*} \beta A \omega, \beta \in V^*, \omega \in V_t^*\},$$

对拓广文法的开始符号 S' :

$$LC(S') = \{\varepsilon\}$$

若 $B \rightarrow \gamma A \delta$, 则: $LC(A) \supseteq LC(B) \cdot \{\gamma\}$

**G[S]: (0) $S' \rightarrow S$ (1) $S \rightarrow a A c B e$
(2) $A \rightarrow b$ (3) $A \rightarrow Ab$ (4) $B \rightarrow d$**

每个非终结符的左文方程组

$$LC(S') = \{\varepsilon\}$$

$$LC(S) = LC(S') \cdot \{\varepsilon\}$$

$$LC(A) = LC(S) \cdot \{a\} \cup LC(A) \cdot \{\varepsilon\}$$

$$LC(B) = LC(S) \cdot \{aAc\}$$

化简为:

$$[S'] = \varepsilon$$

$$[S] = [S']$$

$$[A] = [S]a + [A]$$

$$[B] = [S]aAc$$

用代入法求解

$$[S'] = \varepsilon$$

$$[S] = \varepsilon$$

$$[A] = a + [A]$$

$$[B] = aAc$$

令 $\Sigma = \{[S'], [S], [A], [B], a, A, c\}$

则方程两边都是 Σ 上的正规式

而 $[A] = a + [A]$ 即为 $[A] = a \mid [A]$ 由正规式所表示的正规集

得: $[A] = a$

G[S]: (0) $S' \rightarrow S$ (1) $S \rightarrow a A c B e$

(2) $A \rightarrow b$ (3) $A \rightarrow Ab$ (4) $B \rightarrow d$

定义(产生式的LR(0)左文)

$LR(0)C(A \rightarrow \alpha) = \{\gamma \mid \gamma = \beta\alpha \text{ 且 } S' \xRightarrow[R]{*} \beta A \omega \xRightarrow[R]{} \beta\alpha\omega, \omega \in V_t^*\}$

推论: $LR(0)C(A \rightarrow \alpha) = LC(A) \cdot \{\alpha\}$

则有:

$LR(0)C(S' \rightarrow S) = S$

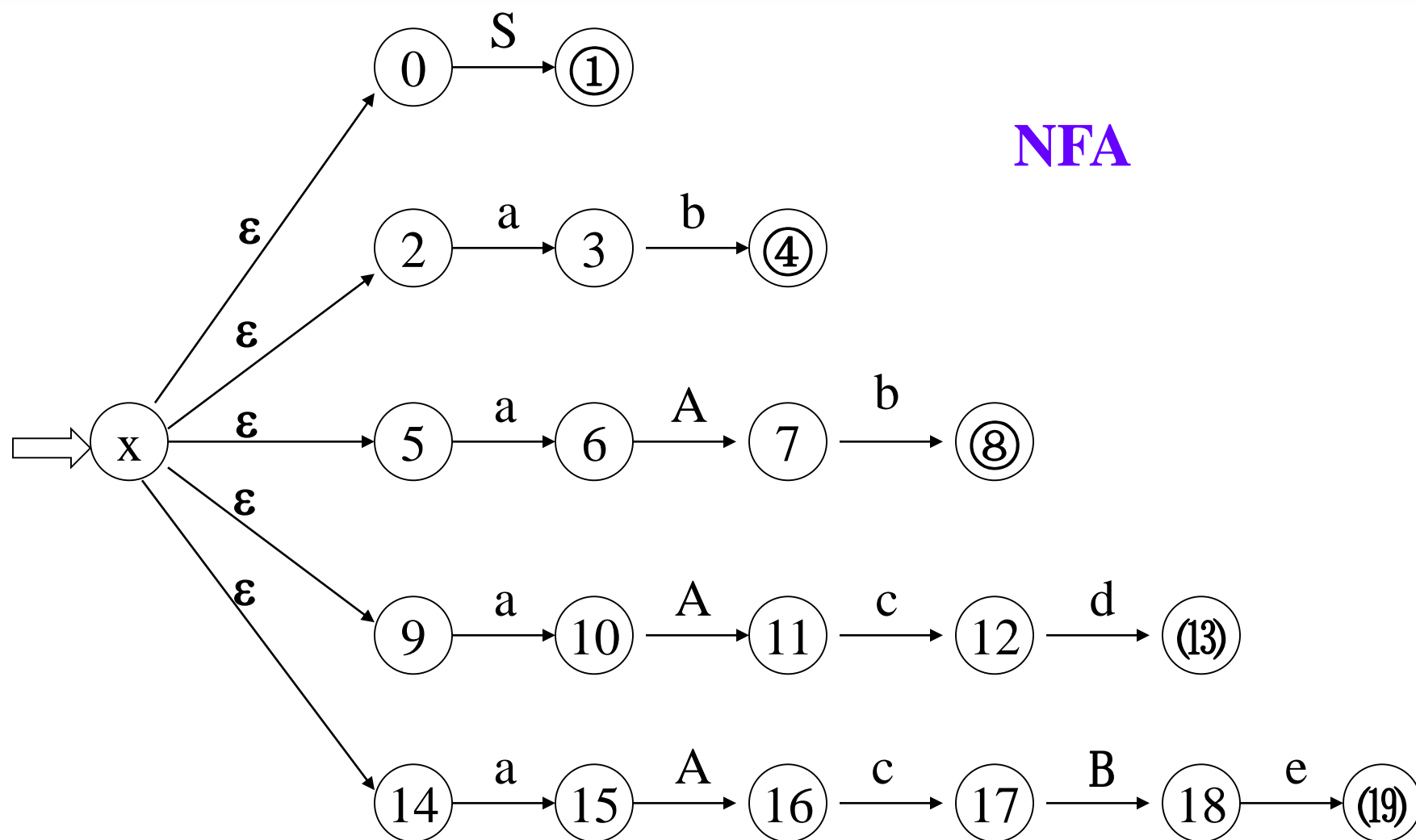
$LR(0)C(S \rightarrow aAcBe) = aAcBe$

$LR(0)C(A \rightarrow b) = ab$

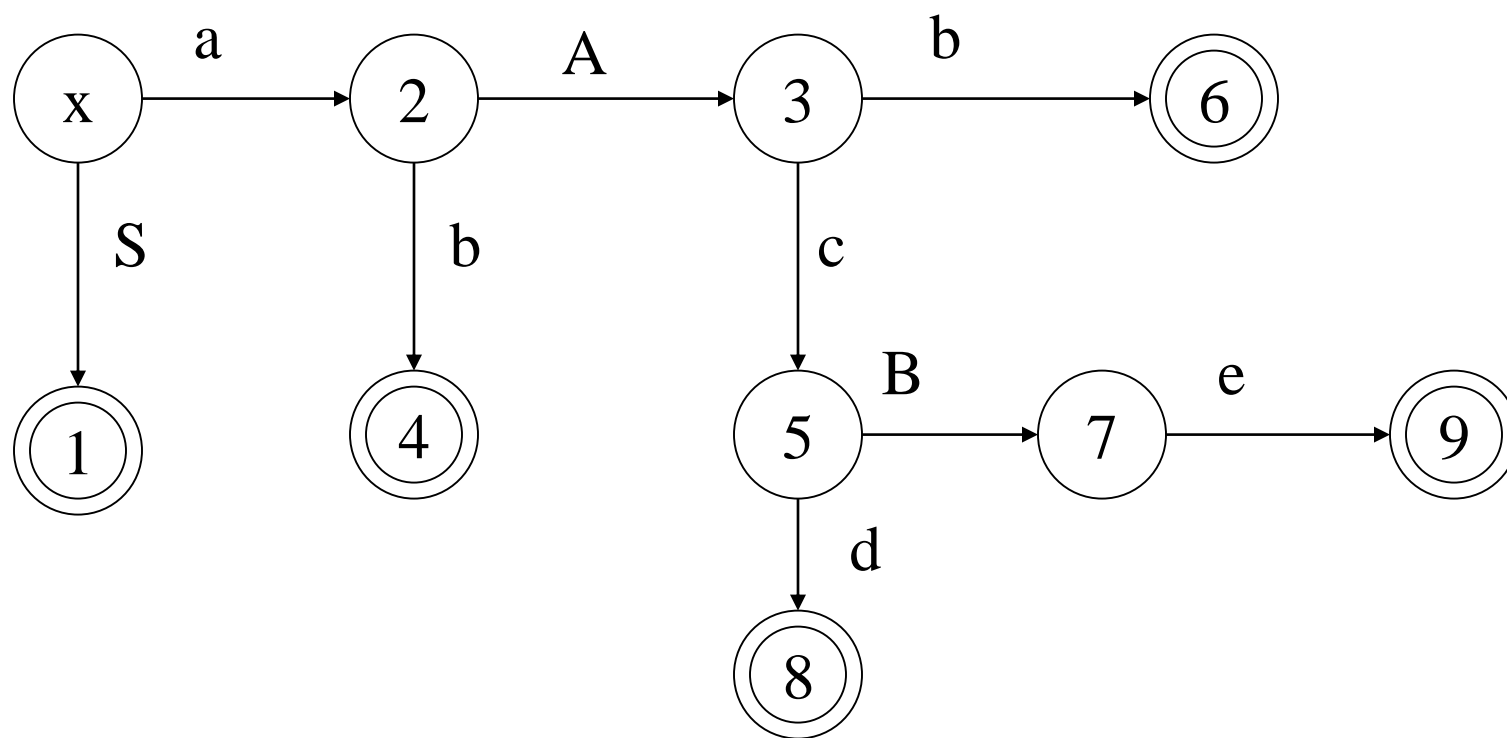
$LR(0)C(A \rightarrow Ab) = aAb$

$LR(0)C(B \rightarrow d) = aAcd$

$(\Sigma = V_n \cup V_t)$ 上的正规式



DFA



LR(0)项目

构造LR(0)项目

LR(0)项目或配置 (*item or configuration*)

---在右端某一位置有圆点的文法G的产生式

$A \rightarrow xyz$ $A \rightarrow .xyz$

$A \rightarrow x.yz$

$A \rightarrow xy.z$

$A \rightarrow xyz.$

如: $S \rightarrow aAd$

$S \rightarrow .aAd$ $S \rightarrow a .Ad$ $S \rightarrow aA .d$ $S \rightarrow aAd .$

活前缀与句柄的关系

$G[S]$:

若 $S \xRightarrow{*}_R \alpha A \omega \xRightarrow{R} \alpha \beta \omega$ r 是 $\alpha\beta$ 的前缀, 则称

r 是 G 的一个活前缀。

1. 活前缀已含有句柄的全部符号, 表明产生式 $A \rightarrow \beta$ 的右部 β 已出现在栈顶
2. 活前缀只含句柄的一部分符号表明 $A \rightarrow \beta_1 \beta_2$ 的右部子串 β_1 已出现在栈顶, 期待从输入串中看到 β_2 推出的符号
3. 活前缀不含有句柄的任何符号, 此时期望 $A \rightarrow \beta$ 的右部所推出的符号串

活前缀、句柄、LR(0)项目

为刻划这种分析过程中的文法G的每一个产生式的右部符号已有多大一部分被识别（出现在栈顶）的情况，分别用标有圆点的产生式来指示位置。

$A \rightarrow \beta \cdot$ 刻划产生式 $A \rightarrow \beta$ 的右部 β 已出现在栈顶

$A \rightarrow \beta_1 \cdot \beta_2$ 刻划 $A \rightarrow \beta_1 \beta_2$ 的右部子串 β_1 已出现在栈顶，期待从输入串中看到 β_2 推出的符号

$A \rightarrow \cdot \beta$ 刻划没有句柄的任何符号在栈顶，此时期望 $A \rightarrow \beta$ 的右部所推出的符号串

对于 $A \rightarrow \varepsilon$ 的LR(0)项目只有 $A \rightarrow \cdot$ 。

由LR(0)项目构造 识别活前缀的NFA

$G[S]: (0) S' \rightarrow S \quad (1) S \rightarrow a A c B e$
 $(2) A \rightarrow b \quad (3) A \rightarrow Ab \quad (4) B \rightarrow d$

文法的项目为:

1. $S' \rightarrow .S$ 2. $S' \rightarrow S .$
3. $S \rightarrow .aAcBe$ 4. $S \rightarrow a.AcBe$ 5. $S \rightarrow aA.cBe$
6. $S \rightarrow aAc.Be$ 7. $S \rightarrow aAcB.e$ 8. $S \rightarrow aAcBe.$
9. $A \rightarrow .b$ 10. $A \rightarrow b.$
11. $A \rightarrow .Ab$ 12. $A \rightarrow A.b$ 13. $A \rightarrow Ab.$
14. $B \rightarrow .d$ 15. $B \rightarrow d.$

项目就是状态!

项目(状态)之间的转换

转换方法如下:

若有项目i: $X \rightarrow X_1 X_2 \dots X_{i-1} \cdot X_i \dots X_n$

项目j: $X \rightarrow X_1 X_2 \dots X_{i-1} X_i \cdot X_{i+1} \dots X_n$

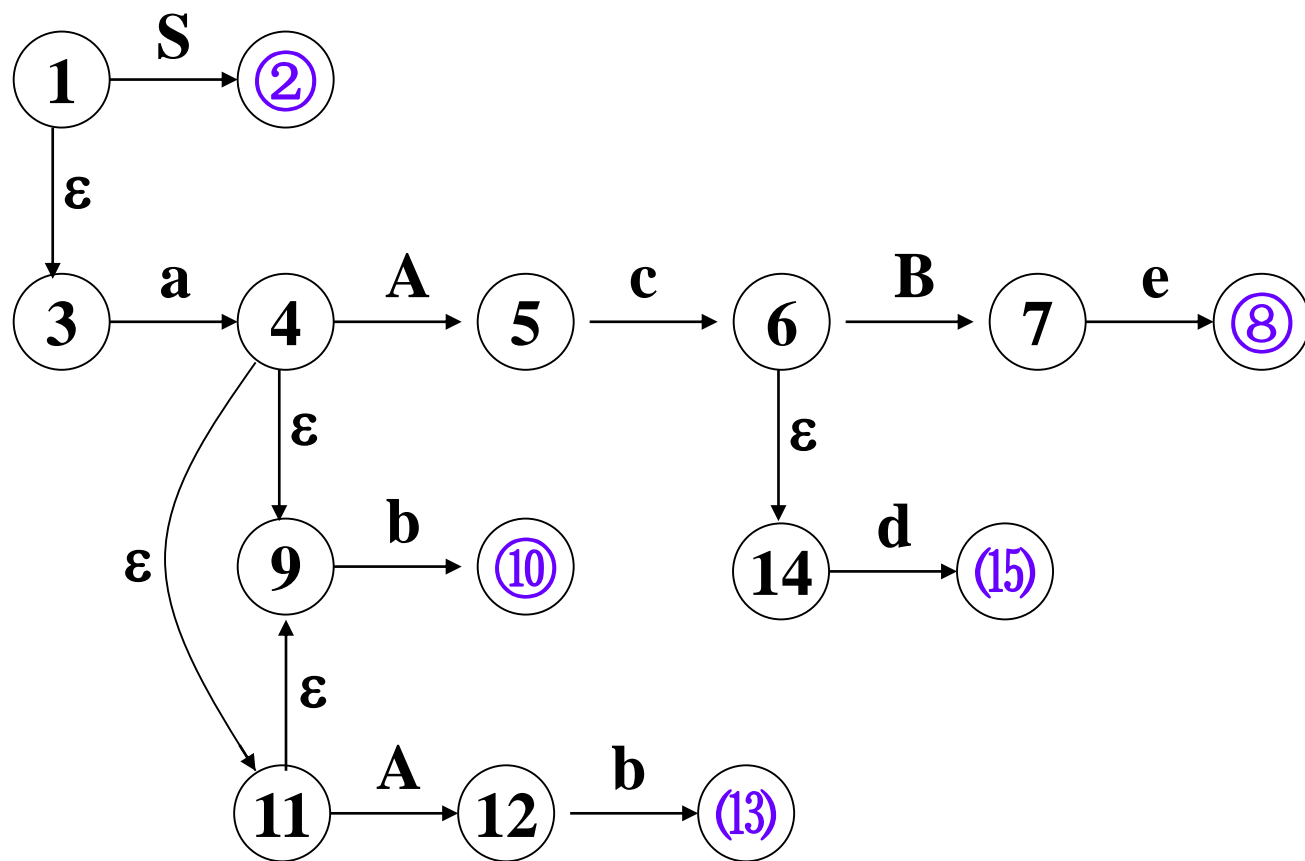
则从状态i到状态j连一条标记为 X_i 的箭弧。

若有项目i: $X \rightarrow \gamma \cdot A \delta$

项目k: $A \rightarrow \cdot \beta$

则从状态i画标记为 A 的箭弧到状态k

点在最右边的项目为句柄识别态, 即NFA的终态。

NFA?!看看原来的**再将NFA转换成DFA?**Forward(DFA)

LR(0)项目集的规范族

LR(0) 项目集的闭包CLOSURE

若当前处于 $A \rightarrow X \bullet YZ$ 刻划的情况，期望移进 $\text{First}(Y)$ 中的某些符号，假如有产生式

$Y \rightarrow u \mid w$ 。那么 $Y \rightarrow \bullet u$ 和 $Y \rightarrow \bullet w$ 这两个项目便是刻划期望移进 $\text{First}(Y)$ 中的某些符号的情况。

$$A \rightarrow X \bullet YZ$$

$$Y \rightarrow \bullet u$$

$$Y \rightarrow \bullet w$$

这三个项目对应移进归约分析的同一个状态，这三个项目构成一个**配置集（项目集）**，对应每个配置集，分析表将有一个状态。

LR(0)项目集闭包的构造

LR(0)项目集的闭包CLOSURE

```
function CLOSURE (I); /* I 是项目集*/  
{ J:= I;  
  repeat for J 中的每个项目  $A \rightarrow \alpha .B \beta$  和产生式  
     $B \rightarrow \gamma$  , 若  $B \rightarrow .\gamma$  不在J中  
    do 将  $B \rightarrow .\gamma$  加到J中  
  until 再没有项目加到J中  
  return J  
};
```

转换函数 $GO(I, X)$

GO 函数

$GO(I, X) === CLOSURE(J)$;

其中, I : 项目集, X : 文法符号,

$J = \{ \text{任何形如 } A \rightarrow \alpha X \beta \text{ 的项目} \mid A \rightarrow \alpha \cdot X \beta \in I \}$

LR(0)项目集规范族的构造

计算LR(0)项目集规范族

$C = \{I_0, I_1, \dots, I_n\}$

Procedure Itemsets(G');

Begin $C := \{ \text{CLOSURE}(\{S' \rightarrow \cdot S\}) \}$

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

End;

LR(0)项目集的规范族构成识别一个文法活前缀的DFA的状态的全体。

文法G:

(0) $S' \rightarrow E$ (1) $E \rightarrow aA$ (2) $E \rightarrow bB$

(3) $A \rightarrow cA$ (4) $A \rightarrow d$ (5) $B \rightarrow cB$

(7) $B \rightarrow d$

LR(0) 项目集规范族 (识别G的活前缀的DFA):

$I_0: S' \rightarrow \cdot E$ $I_1: S' \rightarrow E \cdot$ $I_2: E \rightarrow a \cdot A$

$E \rightarrow \cdot aA$ $A \rightarrow \cdot cA$

$E \rightarrow \cdot bB$ $A \rightarrow \cdot d$

$I_3: E \rightarrow b.B$

$B \rightarrow .cB$

$B \rightarrow .d$

$I_4: A \rightarrow c.A$

$A \rightarrow .cA$

$A \rightarrow .d$

$I_5: B \rightarrow c.B$

$B \rightarrow .cB$

$B \rightarrow .d$

$I_6:$

$E \rightarrow aA.$

$I_7:$

$E \rightarrow bB.$

$I_8:$

$A \rightarrow cA.$

$I_9: B \rightarrow cB.$

$I_{10}: A \rightarrow d.$

$I_{11}: B \rightarrow d.$

LR(0)分析表的构造

假定 $C = \{I_0, I_1, \dots, I_n\}$, 令每个项目集 I_k 的下标 k 为分析器的一个状态, 因此, G' 的LR(0)分析表含有状态 $0, 1, \dots, n$ 。令那个含有项目 $S' \rightarrow \cdot S$ 的 I_k 的下标 k 为初态。ACTION和GOTO可按如下方法构造:

若项目 $A \rightarrow \alpha \cdot a\beta$ 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 和符号 a 移进栈”, 简记为 “sj”;

若项目 $A \rightarrow \alpha \cdot$ 属于 I_k , 那么对任何终结符 a , 置 $ACTION[k, a]$ 为 “用产生式 $A \rightarrow \alpha$ 进行规约”, 简记为 “rj”; 其中, 假定 $A \rightarrow \alpha$ 为文法 G' 的第 j 个产生式;

若项目 $S' \rightarrow \cdot S$ 属于 I_k , 则置 $ACTION[k, \#]$ 为 “接受”, 简记为 “acc”;

若 $GO(I_k, A) = I_j$, A 为非终结符, 则置 $GOTO(k, A) = j$; 分析表中凡不能用规则1至4填入信息的空白格均置上 “出错标志”。

按上述算法构造的含有ACTION和GOTO两部分的分析表，如果每个入口不含多重定义，则称它为文法G的一张LR(0)表。具有LR(0)表的文法G称为一个LR(0)文法。

LR(0)文法是无二义的。

文法G: (0) $S' \rightarrow E$ (1) $E \rightarrow aA$ (2) $E \rightarrow bB$
 (3) $A \rightarrow cA$ (4) $A \rightarrow d$ (5) $B \rightarrow cB$ (6) $B \rightarrow d$

ACTION							GOTO		
	a	c	b	d	#		E	A	B
0	S2		S3				1		
1					acc				
2		S4		S10				6	
3		S5		S11					7
4		S4		S10				8	
5		S5		S11					9
6	r1	r1	r1	r1	r1				
7	r2	r2	r2	r2	r2				
8	r3	r3	r3	r3	r3				
9	r5	r5	r5	r5	r5				
10	r4	r4	r4	r4	r4				
11	r6	r6	r6	r6	r6				

LR(0)项目

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 $A \rightarrow \alpha \cdot a\beta$ a 是终结符, $\alpha, \beta \in V^*$ 下同.

待约项目，形如 $A \rightarrow \alpha \cdot B\beta$

归约项目，形如 $A \rightarrow \alpha \cdot$

接受项目，形如 $S' \rightarrow S \cdot$

$A \rightarrow \epsilon$ 的LR(0)项目只有 $A \rightarrow \cdot$ 是归约项目

作用？

例7.1 $G[S]$ 为:

$S \rightarrow \mathbf{a} A \mathbf{c} B \mathbf{e}$

$A \rightarrow \mathbf{b}$

$A \rightarrow A \mathbf{b}$

$B \rightarrow \mathbf{d}$

1)构造识别活前缀的DFA

2)构造它的LR(0)分析表。

3)分别给出对输入符号串abbcde和
abbce的LR(0)分析步骤。

G[S]拓广为:

$S' \rightarrow S$

$S \rightarrow a A c B e$

$A \rightarrow b$

$A \rightarrow Ab$

$B \rightarrow d$

TSINGHUA UNIVERSITY

$G[L] = ab^+cde$

$I_1: S' \rightarrow S \cdot$

$I_0: S' \rightarrow \cdot S$

$S \rightarrow \cdot a A c B e$

$I_2: S \rightarrow a \cdot A c B e$

$A \rightarrow \cdot b$

$A \rightarrow \cdot Ab$

$I_5: S \rightarrow a A c \cdot B e$

$B \rightarrow \cdot d$

$I_8: B \rightarrow d \cdot$

$I_3: S \rightarrow a A \cdot c B e$

$A \rightarrow A \cdot b$

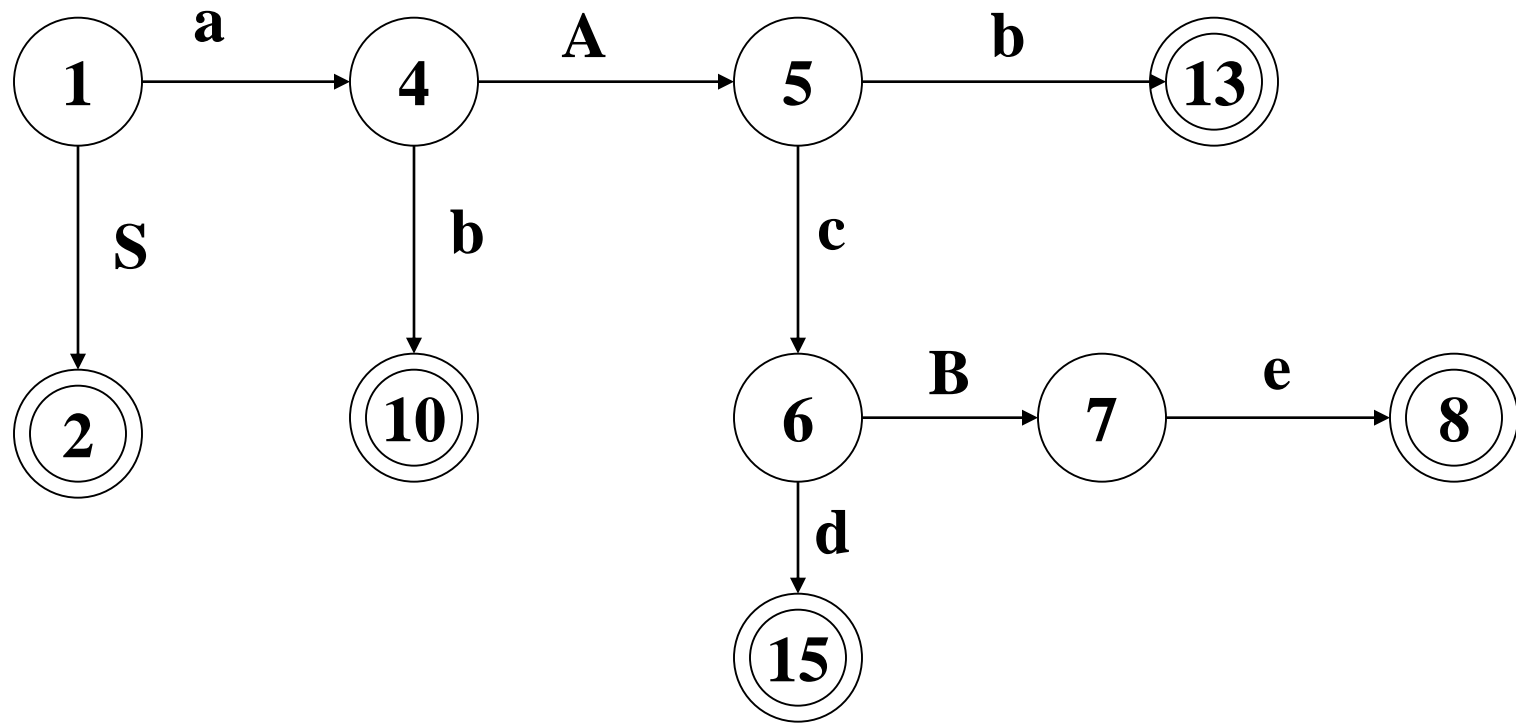
$I_7: S \rightarrow a A c B \cdot e$

$I_9: S \rightarrow a A c B e \cdot$

$I_4: A \rightarrow b \cdot$

$I_6: A \rightarrow A b \cdot$

DFA



看看NFA

主

																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																																					</
--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	--	----

对输入串abbcde#的分析过程

<u>Step</u>	<u>states.</u>	<u>Syms.</u>	<u>The rest of input</u>	<u>action goto</u>	
1	0	#	abbcde#	s2	
2	02	#a	bbcde#	s4	
3	0 24	#a b	bcde#	r2	3
4	023	#aA	bcde#	s6	
5	0 236	#a A b	cde#	r3	3
6	023	#aA	cde#	s5	
7	0235	#aAc	de#	s8	
8	02358	#aAcd	e#	r4	7
9	02357	#aAcB	e#	s9	
10	023579	#aAcBe	#	r1	1
11	01	#S	#	acc	

对输入串abbce#的分析过程

<u>Step</u>	<u>states.</u>	<u>Syms.</u>	<u>The rest of input</u>	<u>action</u>	<u>goto</u>
1	0	#	abbce#	s2	
2	02	#a	bbce#	s4	
3	0 24	#a b	bce#	r2	3
4	023	#aA	bce#	s6	
5	0 236	#a A b	ce#	r3	3
6	023	#aA	ce#	s5	
7	0235	#aAc	e#	出错	

说明abbce#不是例7.1 文法 $G[S]$ 的句子

The end of part 1