编译原理

习题课 (2)

第四章 语法分析—自上而下分析

- 语法分析器的功能
- 自上而下分析面临的问题
- LL(1) 分析法
- 递归下降分析程序构造
- 预测分析程序

语法分析的方法

- 自上而下分析法 (Top-down)
  - 基本思想
    - 它从文法的开始符号出发，反复使用各种产生式，寻找 " 匹配 " 的推导
  - 递归下降分析法
    - 对每一语法变量 (非终结符) 构造一个相应的子程序，每个子程序识别一定的语法单位
    - 通过子程序间的相互调用实现对输入串的识别
  - 预测分析程序
    - 非递归实现
    - 直观、简单

P81-1. 考虑下面文法 $G_1(S)$:

$$S \rightarrow a \mid \wedge \mid (T)$$
$$T \rightarrow T, S \mid S$$

(1) 消去 G_1 的左递归。然后，对每个非终结符，写出不带回溯的递归子程序。

(2) 经改写后的文法是否是 LL （1）的？给出它的预测分析表。

- 思路
  - 消除左递归
  - 提取左公共因子
  - 计算非终结符的 FIRST 集合和 FOLLOW 集合
  - 检查 LL(1) 条件
  - 构造预测分析表或递归子程序

$G_1(S)$:

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow T, S \mid S$

- 消除左递归： 按照 T,S 的顺序消除左递归
- $G'_1(S)$:

  $S \rightarrow a \mid \wedge \mid (T)$

  $T \rightarrow S\ T'$

  $T' \rightarrow , S\ T' \mid \varepsilon$
- 无左公共因子

G'$_1$(S) :

S → a | ∧ | (T)

T → S T'

T' → , S T' | ε

$$FIRST(\alpha) = \{a \mid \alpha \overset{*}{\Rightarrow} a..., a \in V_T\}$$

$$FOLLOW(A) = \{a \mid S \overset{*}{\Rightarrow} ...Aa..., a \in V_T\}$$

- 计算非终结符的 FIRST 和 FOLLOW 集合
  - FIRST(S)={a, ^, (}
  - FIRST(T)={a, ^, (}
  - FIRST(T')={ , , ε }
  - FOLLOW(S)={), , ,# }
  - FOLLOW(T)={) }
  - FOLLOW(T')={) }
- 检查 LL(1) 条件

构造不带回溯的自上而下分析的文法条件

1. 文法不含左递归
2. 对于文法中每一个非终结符 A 的各个产生式的候选首符集两两不相交。即，若

$$A \rightarrow \alpha_1 | \alpha_2 | \ldots | \alpha_n$$

则 $FIRST(\alpha_i) \cap FIRST(\alpha_j) = \phi \quad (i \neq j)$

3. 对文法中的每个非终结符 A，若它存在某个候选首符集包含ε，则

$$FIRST(\alpha_i) \cap FOLLOW(A) = \phi$$

$$i=1,2,...,n$$

如果一个文法 G 满足以上条件，则称该文法 G 为 LL(1) 文法。

$G'_1(S)$:

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow S\ T'$

$T' \rightarrow ,\ S\ T' \mid \varepsilon$

- 计算非终结符的 FIRST 和 FOLLOW 集合
  - FIRST(S)={a, ^, (}
  - FIRST(T)={a, ^, (}
  - FIRST(T')={ , , ε }
  - FOLLOW(S)={), , ,# }
  - FOLLOW(T)={) }
  - FOLLOW(T')={) }
- 检查 LL(1) 条件
  - 满足

分析表 M[A， a] 的构造

- 在每个非终结符 A 及其任意候选α都构造出 FIRST(α) 和 FOLLOW(A) 的基础上
- 构造 G 的分析表 M[A， a]， 确定每个产生式 A→α 在表中的位置

1. 对文法 G 的每个产生式 A→α 执行第 2 步和第 3 步;
2. 对每个终结符 a ∈FIRST(α)，把 A→α 加至 M[A， a] 中;
3. 若ε∈ FIRST(α)，则对任何 b∈FOLLOW(A) 把 A→α 加至 M[A， b] 中。
4. 把所有无定义的 M[A， a] 标上“出错标志”。

$G'_1(S)$:

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow S\ T'$

$T' \rightarrow ,\ S\ T' \mid \varepsilon$

FIRST(S)={a, ^, (}
FIRST(T)={a, ^, (}
FIRST(T')={ , , ε }
FOLLOW(S)={), , ,# }
FOLLOW(T)={) }
FOLLOW(T')={) }

- 构造预测分析表

	a	^	(	)	,	#
S	$S \rightarrow a$	$S \rightarrow \wedge$	$S \rightarrow (T)$			
T	$T \rightarrow S\ T'$	$T \rightarrow S\ T'$	$T \rightarrow S\ T'$			
T'				$T' \rightarrow \varepsilon$	$T' \rightarrow ,\ S\ T'$	

$G'_1(S)$:

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow S\ T'$

$T' \rightarrow , S\ T' \mid \varepsilon$

- 构造递归子程序

```
procedure T';
begin
   if sym=',' then begin
                        advance;
                        S;T'
                   end
end;
```

```
procedure S;
begin
   if sym='a' or sym='^'
      then abvance
      else if sym='('
             then begin
                  advance;T;
                  if sym=')' then advance;
                             else error;
              end
              else  error
end;
```

```
procedure T;
begin
   S;T'
end;
```

第五章 语法分析——自下而上分析

- 自下而上分析的基本问题
- 算符优先分析算法
- LR 分析法

语法分析的方法

- 自下而上分析法 (Bottom-up)
  - 基本思想
    - 从输入串开始，逐步进行归约，直到文法的开始符号
    - 归约：根据文法的产生式规则，把产生式的右部替换成左部符号
    - 从树末端开始，构造语法树
  - 算符优先分析法
    - 按照算符的优先关系和结合性质进行语法分析
    - 适合分析表达式
  - LR 分析法
    - 规范归约

短语、直接短语、句柄和素短语

- 定义：令 G 是一个文法， S 是文法的开始符号，假定αβδ是文法 G 的一个句型，如果有

$$S \overset{*}{\Rightarrow} \alpha A \delta \text{且} \quad A \overset{+}{\Rightarrow} \beta$$

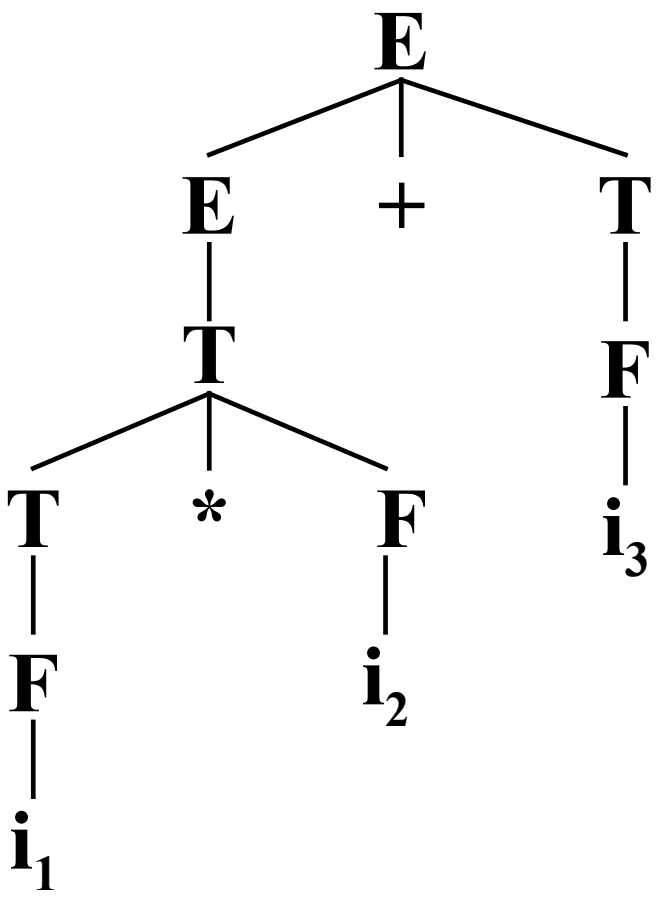
则β称是句型αβδ相对于非终结符 A 的短语。

特别是，如果有 A⇒β, 则称β是句型αβδ相对于规则 A→ β 的直接短语。一个句型的最左直接短语称为该句型的句柄。

一个文法 G 的句型的素短语是指这样一个短语，它至少含有一个终结符，并且，除它自身之外不再含任何更小的素短语

最左素短语是指处于句型最左边的那个素短语

短语、直接短语和句柄



- 在一个句型对应的语法树中
  - 以某非终结符为根的两代以上的子树的所有末端结点从左到右排列就是相对于该非终结符的一个短语
  - 如果子树只有两代，则该短语就是直接短语

P133-1. 令文法 G1 为：

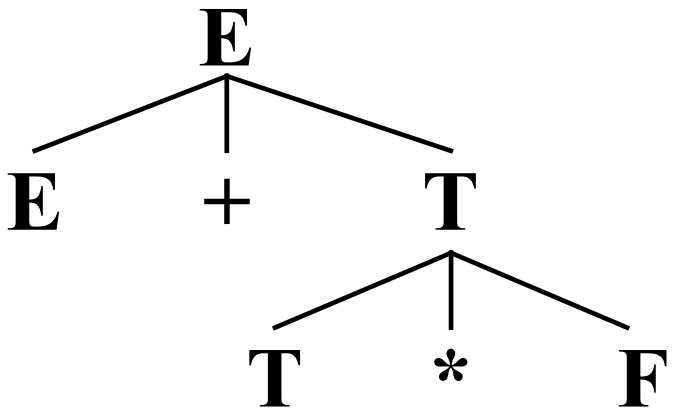
$E \rightarrow E+T \mid T$

$T \rightarrow T*F \mid F$

$F \rightarrow (E) \mid i$

证明 E+T*F 是它的一个句型，指出这个句型的所有短语，直接短语和句柄。

- 短语： E+T*F, T*F
- 直接短语 : T*F
- 句柄 : T*F



构造集合 FIRSTVT(P) 的算法

$$FIRSTVT(P)=\{a|P\overset{+}{\Rightarrow}a\cdots,\text{或}P\overset{+}{\Rightarrow}Qa\cdots,a\in V_T\text{而}Q\in V_N\}$$

- 反复使用下面两条规则构造集合 FIRSTVT(P)

  1. 若有产生式 P→a… 或 P→Qa…，则 a∈FIRSTVT(P)

  2. 若 a∈FIRSTVT(Q)，且有产生式 P→Q…，

将对推导的遍历转换成对产生式的反复遍历

构造集合 LASTVT(P) 的算法

$$LASTVT(P)=\{a \mid P\overset{+}{\Rightarrow}\cdots a,\text{或}P\overset{+}{\Rightarrow}\cdots aQ,a\in V_T\text{而}Q\in V_N\}$$

- 反复使用下面两条规则构造集合 LASTVT(P)

  1. 若有产生式 P→… a 或 P→ … aQ，则 a∈ LASTVT(P)；
  2. 若 a∈ LASTVT(Q)，且有产生式 P→… Q，则 a∈ LASTVT(P)。

构造优先关系表算法

- 通过检查 G 的每个产生式的每个候选式，可找出所有满足 a　b 的终结符对
- 通过检查每个产生式的候选式确定满足关系　和　的所有终结符对
  - 假定有个产生式的一个候选形为

    …aP…

    那么，对任何 b∈FIRSTVT(P)，有 a　b
  - 假定有个产生式的一个候选形为

    …Pb…

    那么，对任何 a∈LASTVT(P)，有 a　b

P133-3.

(1) 计算 $G_2(S)$:

$$S \rightarrow a \mid \wedge \mid (T)$$
$$T \rightarrow T, S \mid S$$

的 FIRSTVT 和 LASTVT 。

(2) 计算 G_2 的优先关系。 G_2 是一个算符优先文法吗？

(3) 计算 G_2 的优先函数。

(4) 给出输入串 (a, (a, a)) 的算符优先分析过程。

思路

- 计算非终结符的 FIRSTVT 和 LASTVT
- 计算终结符之间的优先关系
- 检查算符优先文法的条件
- 构造算符优先优先函数

$G_2(S)$:

$S \rightarrow a \mid \wedge \mid (T)$

$T \rightarrow T, S \mid S$

- FIRSTVT(S)={ a, ^, (}
- FIRSTVT(T)={ ,, a, ^, (}
- LASTVT(S) ={ a, ^,) }
- LASTVT(T) ={ ,, a, ^,) }

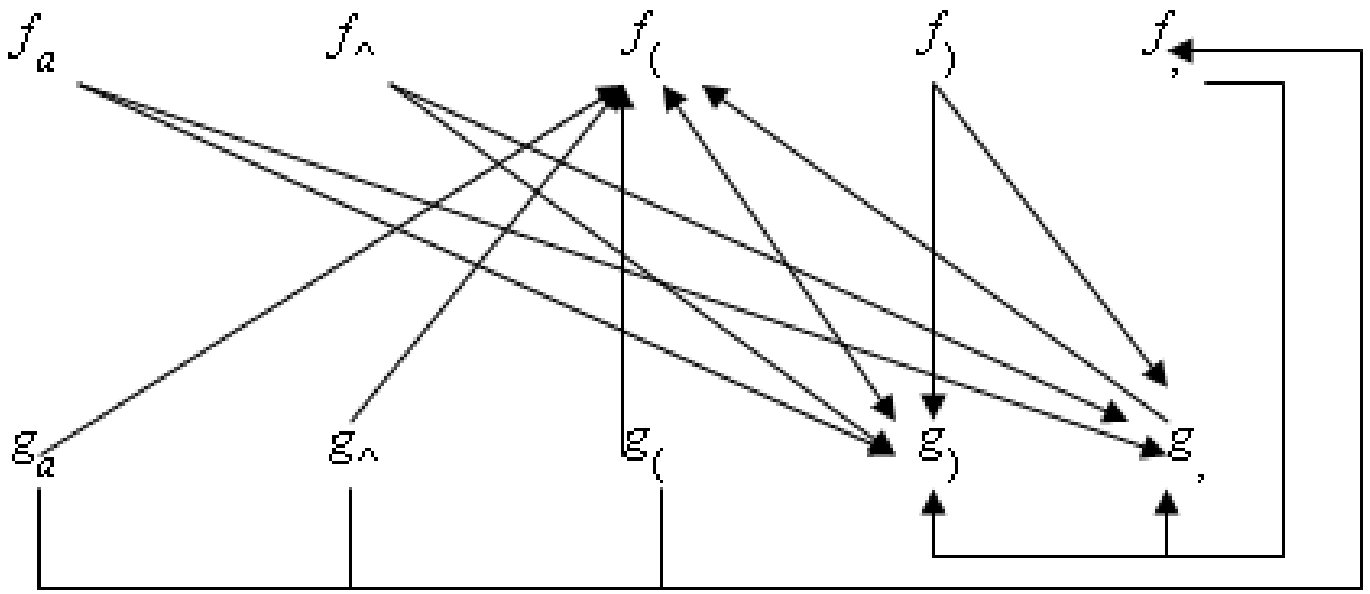
	a	∧	(	)	,
a					
∧					
(					
)					
,					

该文法是算符文法，并且是算符优先文法

根据优先表构造优先函数

1. 画图：对于每个终结符 a，令其对应两个符号 f_a 和 g_a，画一以所有符号和为结点的方向图。如果 a　　b，则从 f_a 画一条弧至 g_b，如果 a　　b，则画一条弧从 g_b 至 f_a。
2. 数数：对每个结点都赋予一个数，此数等于从该结点出发所能到达的结点（包括出发点自身）。赋给 f_a 的数作为 f(a)，赋给 g_a 的数作为 g(a)。
3. 验证：检查所构造出来的函数 f 和 g 是否与原来的关系矛盾。若没有矛盾，则 f 和 g 就是要求的优先函数，若有矛盾，则不存在优先函数。

优先函数



	a	∧	(	)	,
f	4	4	2	4	4
g	5	5	5	2	3

算符优先分析

- 算符优先文法句型 (括在两个 # 之间) 的一般形式 :

$$\#N_1a_1N_2a_2\ldots N_na_nN_{n+1}\#$$

其中， a_i 是终结符， N_i 是可有可无的非终结符。

- 定理：一个算符优先文法 G 的任何句型的最左素短语是满足如下条件的最左子串 $N_ja_j\ldots N_ia_iN_{i+1}$ ，

$$a_{j-1}\blacklozenge a_j$$

$$a_j\blacklozenge a_{j+1}\ ,\ \cdots,\ a_{i-1}\blacklozenge a_i$$

$$a_i\square a_{i+1}$$

栈	输入字符串	动作
#	(a, (a,a))#	预备
#(	a, (a,a)) #	进
#(a	, (a,a)) #	进
#(S	, (a,a)) #	归
#(S,	(a,a))#	进
#(S,(	a,a))#	进
#(S,(a	,a))#	进
#(S,(S	,a))#	归
#(S,(S,	a))#	进
#(S,(S,a	))#	进
#(S,(S,S	))#	归
#(S,(T	))#	归
#(S,(T)	)#	进
#(S, S	)#	归
#(T	)#	归
#(T)	#	进
# S	#	归

success

	a	∧	(	)	,
a					
∧					
(					
)					
,					

P134-5. 考虑文法

$$S \rightarrow AS \mid b$$

$$A \rightarrow SA \mid a$$

(1) 列出这个文法的所有 LR(0) 项目。

(2) 构造这个文法的 LR(0) 项目集规范族及识别活前缀的 DFA 。

(3) 这个文法是 SLR 的吗？若是，构造出它的 SLR 分析表。

P134-5. 考虑文法

$$S \rightarrow AS \mid b$$

$$A \rightarrow SA \mid a$$

(1) 列出这个文法的所有 LR （0）项目。

- 对文法进行拓广 : S’→ S
- LR （0）项目 :

0. S’→ .S　　1. S’→ S.　　2. S → .AS

3. S → A.S　　4. S → AS.　　5. S → .b

6. S → b.　　7. A→ .SA　　8. A → S.A

9. A→ SA.　　10. A → .a　　11. A→ a.

P134-5. 考虑文法

$$S \rightarrow AS \mid b$$

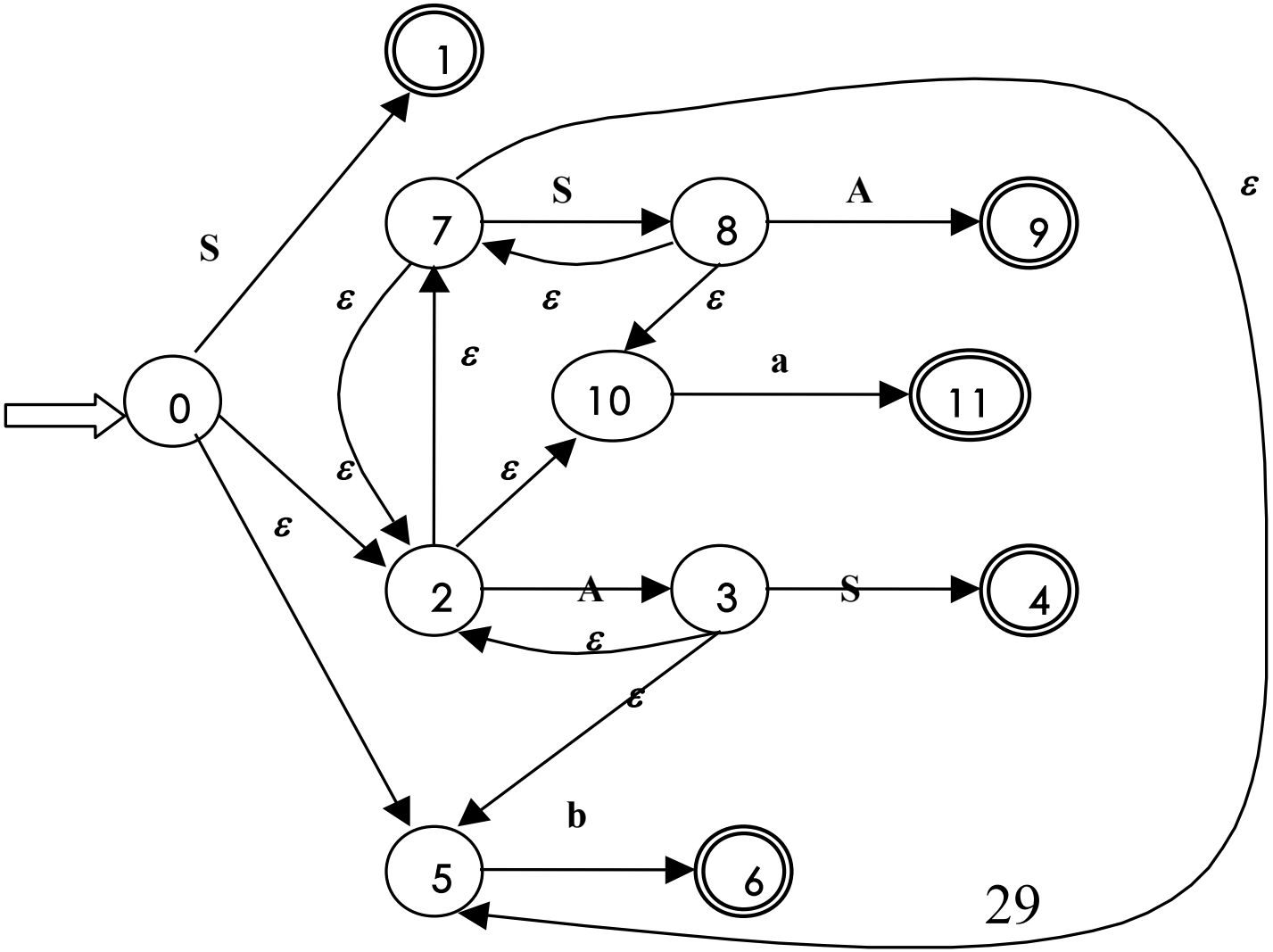
$$A \rightarrow SA \mid a$$

(1) 列出这个文法的所有 LR （0）项目。

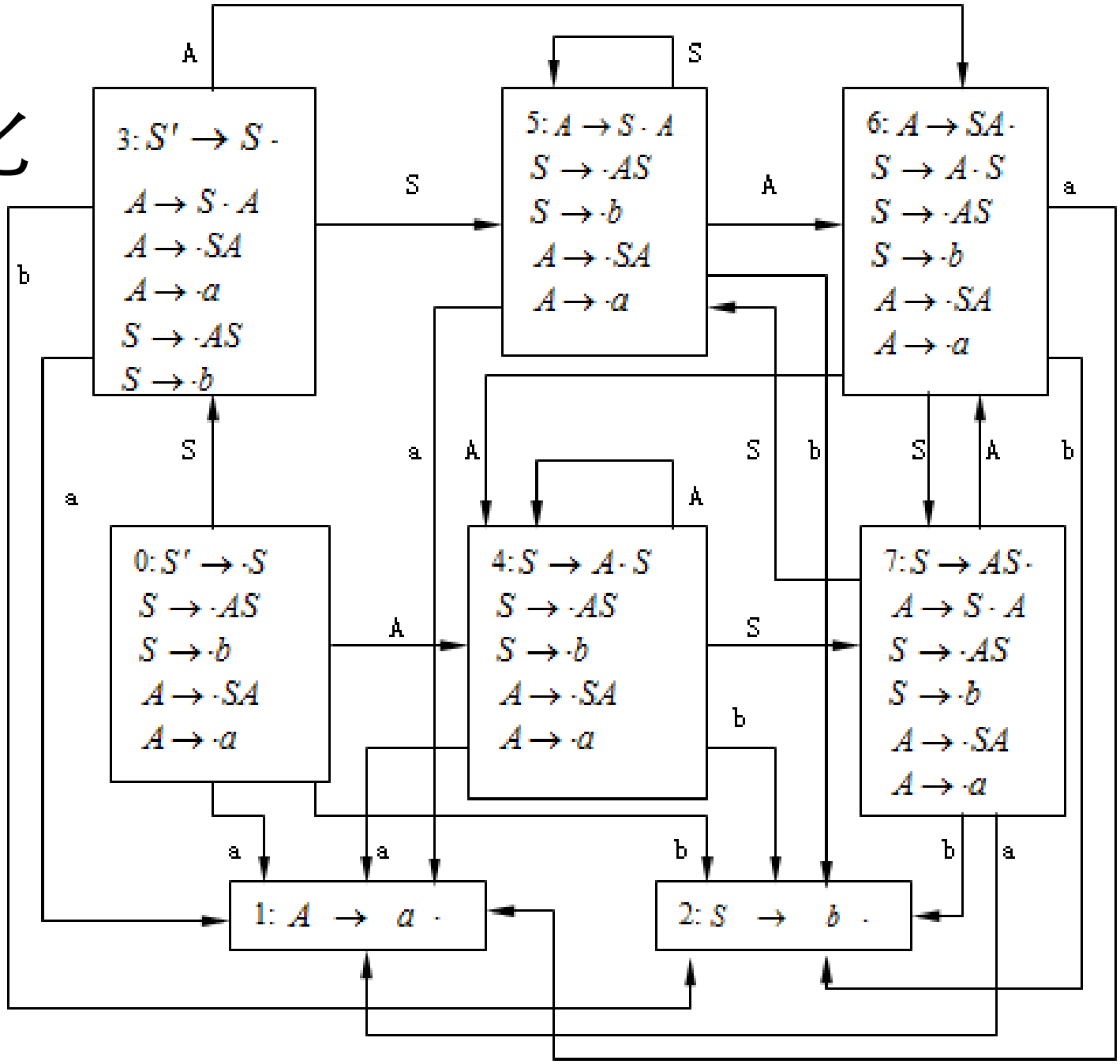
(2) 构造这个文法的 LR （0）项目集规范族及识别活前缀的 DFA 。

- 构造 LR(0) 项目规范族，两种方法:
  - 利用有限自动机来构造
  - 利用函数 CLOSURE 和 GO 来构造

识别活前缀的 DFA



确定化


3: S′ → S ·
A → S · A
A → · SA
A → · a
S → · AS
S → · b
5: A → S · A
S → · AS
S → · b
A → · SA
A → · a
6: A → SA ·
S → A · S
S → · AS
S → · b
A → · SA
A → · a
0: S′ → · S
S → · AS
S → · b
A → · SA
A → · a
4: S → A · S
S → · AS
S → · b
A → · SA
A → · a
7: S → AS ·
A → S · A
S → · AS
S → · b
A → · SA
A → · a
1: A → a ·
2: S → b ·
A
S
a
b

利用函数 CLOSURE 和 GO 来构造项目集规范族

I_0 = { S'→ .S , S → .AS , S → .b , A→ .SA , A→. a }

GO(I_0 , a)= { A→ a. }= I_1
GO(I_0 , b)= { S → b. }= I_2
GO(I_0 , S)= { S'→ S. , A → S.A , A→ .SA , A→. a , S → .AS ,
S → .b }= I_3
GO(I_0 , A)={ S → A.S , S → .AS , S → .b , A → .AS , A→ .a }= I_4

GO(I_3 , a)= { A→ a. }= I_1
GO(I_3 , b)= { A→ b. }= I_2
GO(I_3 , S)= { A → S.A , S → .AS , S → .b , A→ .SA , A→ .a }= I_5
GO(I_3 , A)={ A→ SA. , A → S.A , S → .AS , S → .b , A→ .SA ,
A→ .a }= I_6

GO(I_4 , a)= { A→ a. }= I_1

GO(I_4 , b)= { S → b. }= I_2

GO(I_4 , S)= { S → AS. , A → S.A , S→ .AS , S→. b , A → .AS , A → . a }= I_7

GO(I_4 , A)={ S → A.S , S → .AS , S → .b , A → .AS , A→ .a }= I_4

GO(I_5 , a)= { A→ a. }= I_1

GO(I_5 , b)= { A→ b. }= I_2

GO(I_5 , S)= { A → S.A , S → .AS , S → . b , A→ .SA , A→ . a }= I_5

GO(I_5 , A)={ A→ SA. , A → S.A , S → .AS , S → .b , A→ .SA , A→ .a }= I_6

GO(I_6 , a)= { A→ a. }= I_1

GO(I_6 , b)= { S → b. }= I_2

GO(I_6 , S)= { S → AS. , A → S.A , S→ .AS , S→. b , A → .AS , A → . a }= I_7

GO(I_6 , A)={ S → A.S , S → .AS , S → .b , A → .AS , A→ .a }= I_4

GO(I_7 , a)= { A→ a. }= I_1

GO(I_7 , b)= { A→ b. }= I_2

GO(I_7 , S)= { A → S.A , S → .AS , S → . b , A→ .SA , A→ . a }= I_5

GO(I_7 , A)={ A→ SA. , A → S.A , S → .AS , S → .b , A→ .SA , A→ .a }= I_6

项目集规范族为 C={I_0, I_1 , I_2 , I_3 , I_4 , I_5 , I_6 , I_7 }

P134-5. 考虑文法

$$S \rightarrow AS \mid b$$
$$A \rightarrow SA \mid a$$

(1) 列出这个文法的所有 LR （0）项目。

(2) 构造这个文法的 LR （0）项目集规范族及识别活前缀的 DFA 。

(3) 这个文法是 SLR 的吗？若是，构造出它的 SLR 分析表。

SLR 文法判断

- 状态 3， 6， 7 有移进归约冲突
- I_3 = { S'→ S. , A → S.A , A→ .SA , A→ .a ， S → .AS, S → .b}
  - FOLLOW(S')={#} 不包含 a,b ； 冲突可以消解
- I_6 = { A→ SA . , A → S.A , S → .AS , S → .b , A→ .SA , A→ .a}
  - FOLLOW(A)∩{a,b}={a,b} ≠Φ， 冲突无法消解
- I_7 = { S → AS., A → S.A， S→ .AS ， S → .b , A → .AS, A→ .a}
  - FOLLOW(S)={#,a,b}，冲突无法消解
- 所以不是 SLR 文法。
- 这个文法也不是 LR(1) 文法。

小结

- 自上而下分析
  - LL(1) 分析条件
  - 递归下降分析程序
  - 预测分析程序
- 自下而上分析
  - 可归约串：句柄、最左素短语
  - 算符优先分析
  - LR 分析