第一章

概论

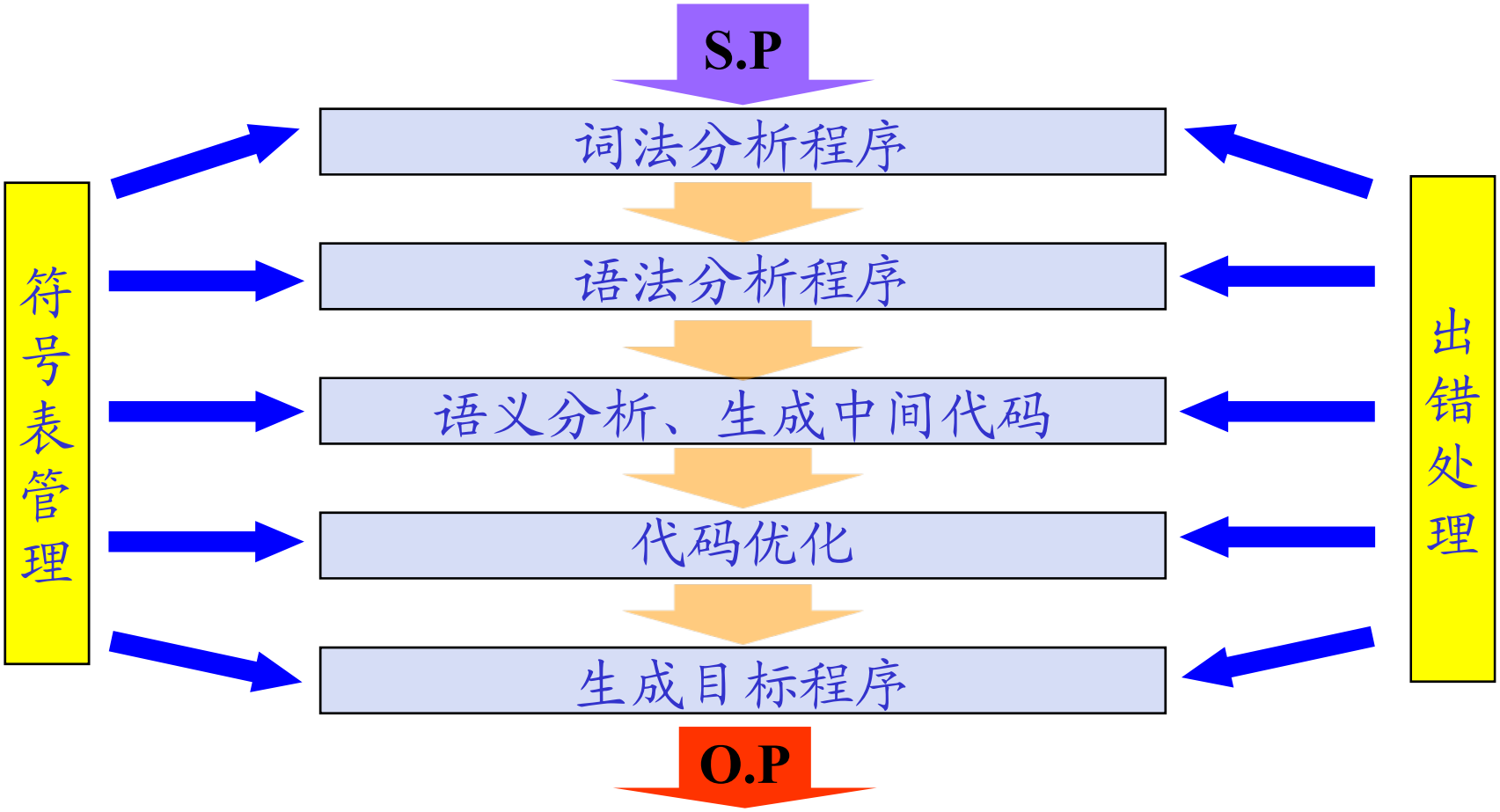
(介绍名词术语、了解编译系统的结构和编译过程）

1.2 编译过程

所谓编译过程是指将高级语言程序翻译为等价的目标程序的过程。

习惯上是将编译过程划分为5个基本阶段:

典型的编译程序具有7个逻辑部分



第二章

- 掌握符号串和符号串集合的运算、文法和语言的定义
- 几个重要概念：递归、短语、简单短语和句柄、语法树、文法的二义性、文法的实用限制等。
- 掌握文法的表示：BNF、扩充的BNF范式、语法图。
- 了解文法和语言的分类

第三章: 词法分析

3.1 词法分析的功能

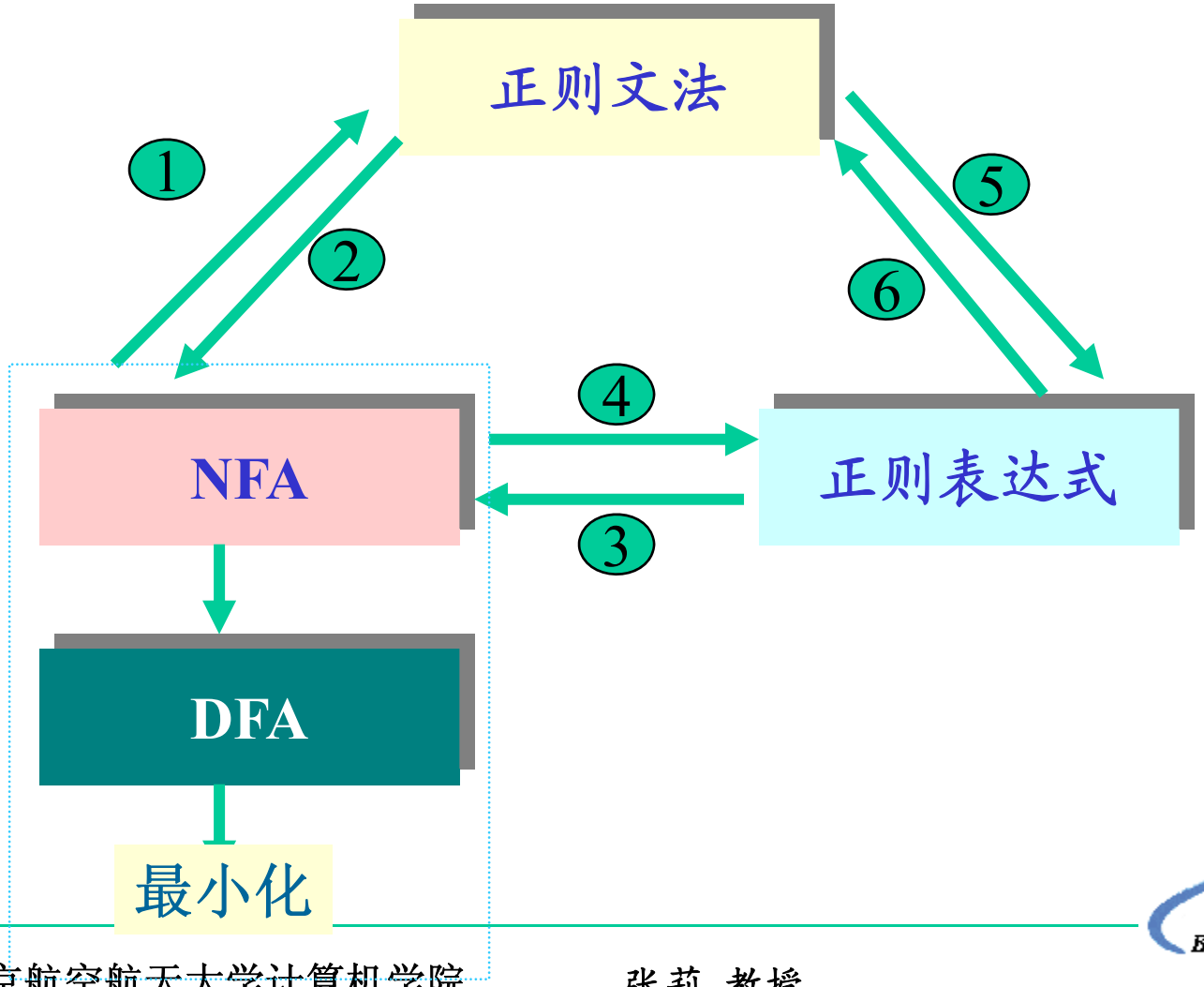
3.2 词法分析程序的设计与实现

- 状态图

3.3 词法分析程序的自动生成

- 有穷自动机、LEX

补充



第四章　语法分析

语法分析方法：
- 自顶向上分析法　$Z \overset{+}{\Rightarrow} S$
- 自顶向上分析法　$S \overset{+}{\Leftarrow} Z$

$S \in L[Z]$

(一) 自顶向下分析

①概述自顶向下分析的一般过程

存在问题：
- 左递归问题 ——— 消除左递归的方法
- 回溯问题 ———
  - 无回溯的条件
  - 改写文法
  - 超前扫描

②两种常用方法:

(1)递归子程序法
- a)改写文法,消除作递归,回溯
- b)写递归子程序

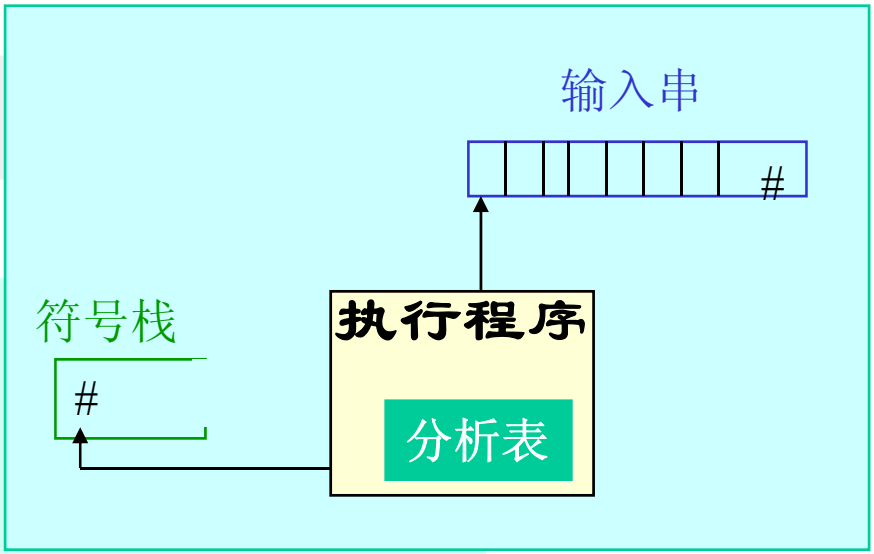
(2)LL(1)分析法
- LL(1)分析器的逻辑结构及工作过程
- LL(1)分析表的构造方法
  - 1.构造First集合的算法
  - 2.构造Follow集合的算法
  - 3.构造分析表的算法
- LL(1)文法的定义以及充分必要条件

4.2.5 LL分析法

LL－自左向右扫描、自左向右地分析和匹配输入串。

∴ 分析过程表现为最左推导的性质。

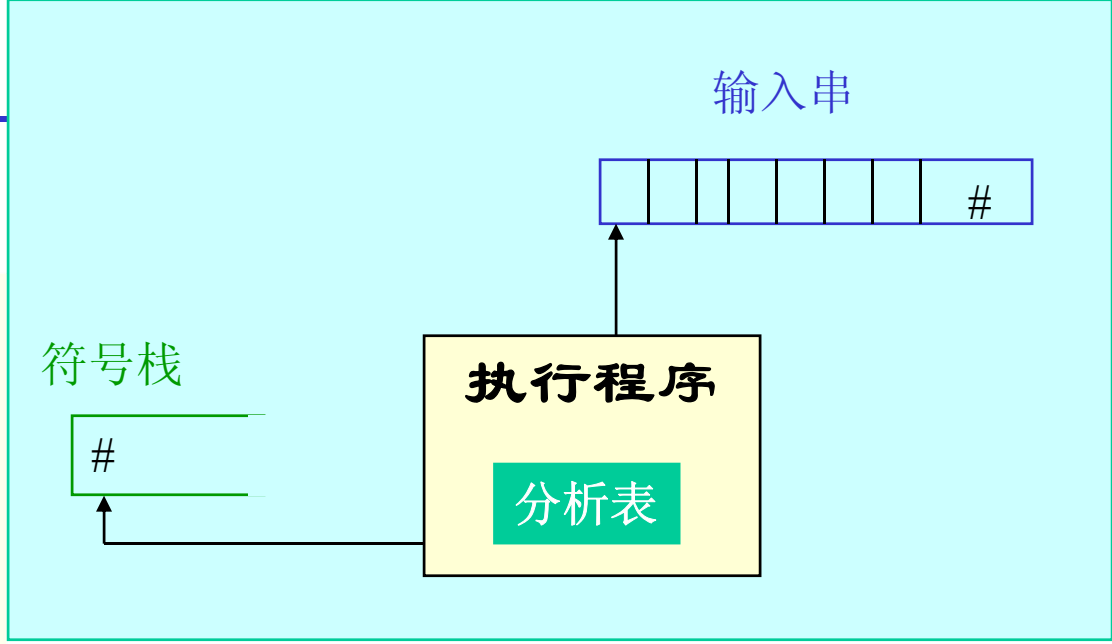
1、LL分析程序构造及分析过程



由三部分组成:

- 分析表
- 执行程序 (总控程序)
- 符号栈 （ 分析栈 ）

在实际语言中，每一种语法成分都有确定的左右界符，为了研究问题方便，统一以‘#’表示。



（1）、分析表：二维矩阵M

$$M[A,a]=\begin{cases} A::=\alpha_i & \alpha_i \in V^* \\ \text{或} & A \in V_n \\ \text{error} & a \in V_t \text{ or } \# \end{cases}$$

M[A,a] = A::=α_i

表示当要用A去匹配输入串时，且当前输入符号为a时，可用A的第i个选择去匹配。

即 当$\alpha_i \neq \varepsilon$时，有$\alpha_i \Rightarrow^{*} a\ldots$;

当$\alpha_i = \varepsilon$时，则a为A的后继符号。

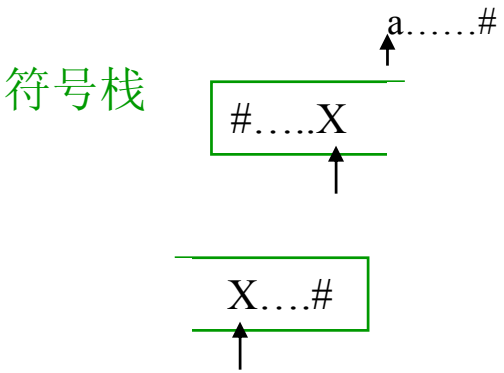
M[A,a] = error

表示当用A去匹配输入串时，若当前输入符号为a，则不能匹配，表示无$A \Rightarrow^{*} a\ldots$,或a不是A的后继符号。

（2）符号栈： 有四种情况

- 开始状态

- 工作状态



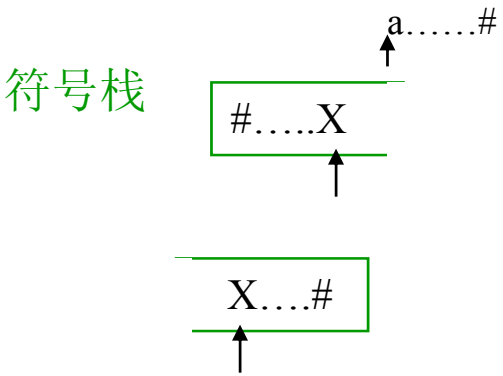
查分析表得:

$X \in V_n$, $M[X,a] = X::=\alpha_i$

$X \overset{+}{\Rightarrow} a\ldots$

$X \in V_t$, $X = a$

- 出错状态



查分析表得:

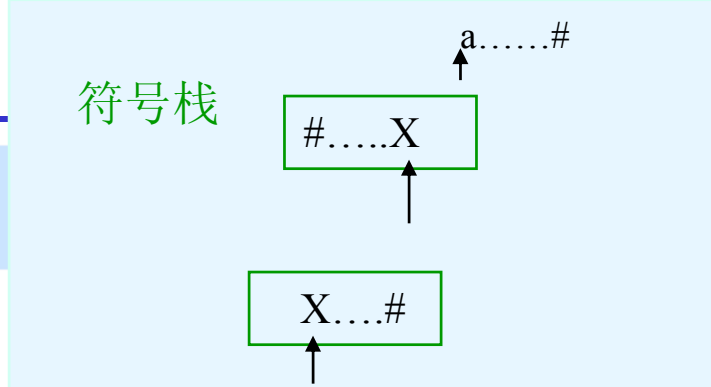
$X \in V_n$, $M[X,a] = error$

无$X \overset{+}{\Rightarrow} a\ldots$

$X \in V_t$, $X \neq a$

- 结束状态

（3）、执行程序



执行程序主要实现如下操作:

1. 把#和文法识别符号E推进栈, 读入下一个符号，重复下述过程直到正常结束或出错。

2. 测定栈顶符号X和当前输入符号a,执行如下操作:

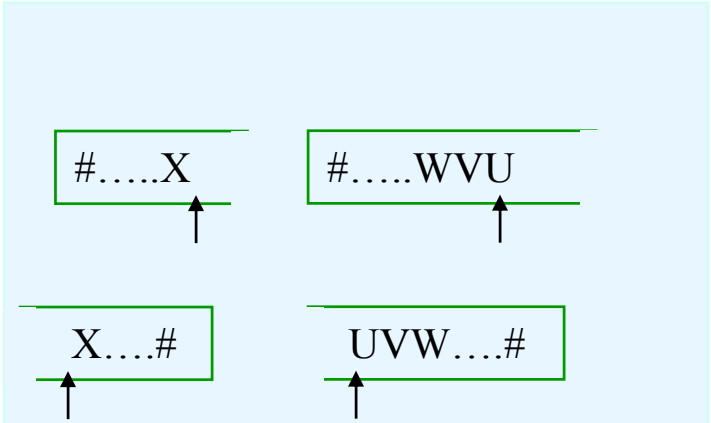
(1) 若X=a=#，分析成功，停止。E匹配输入串成功。
(2) 若X=a≠#，把X推出栈，再读入下一个符号。
(3) 若X∈V_n，查分析表M。

(3) 若X∈V_n，查分析表M。

a) M[X,a] = X::= UVW
则将X弹出栈，将UVW压入
注：U在栈顶 （最左推导）

b) M[X, a] = error 转出错处理

c) M[X, a] = X::=ε，
— a为X的后继符号
则将X弹出栈 (不读下一符号)
继续分析。



分析表

	i	+	*	(	)	#
E	E::=TE'			E::= TE'		
E'		E'::= +TE'			E'::= ε	E'::= ε
T	T::=FT'			T::= FT'		
T'		T'::= ε	T'::= *FT'		T'::= ε	T'::= ε
F	F::= i			F::= (E)		

★ 注：矩阵元素空白表示Error

3、LL(1)文法

定义：一个文法G，其分析表M不含多重定义入口(即分析表中无二条以上规则)，则称它是一个LL(1)文法.

定理：文法G是LL(1)文法的充分必要条件是：对于G的每一个非终结符A的任意两条规则A::=α|β,下列条件成立：

1、FIRST(α) ∩ FIRST(β) = Φ

2、若β=*=> ε, 则FIRST(α) ∩ FOLLOW(A) = Φ

(二) 自底向上分析

规约过程:

(1)一般过程: 移进—规约过程

问题:如何寻找句柄

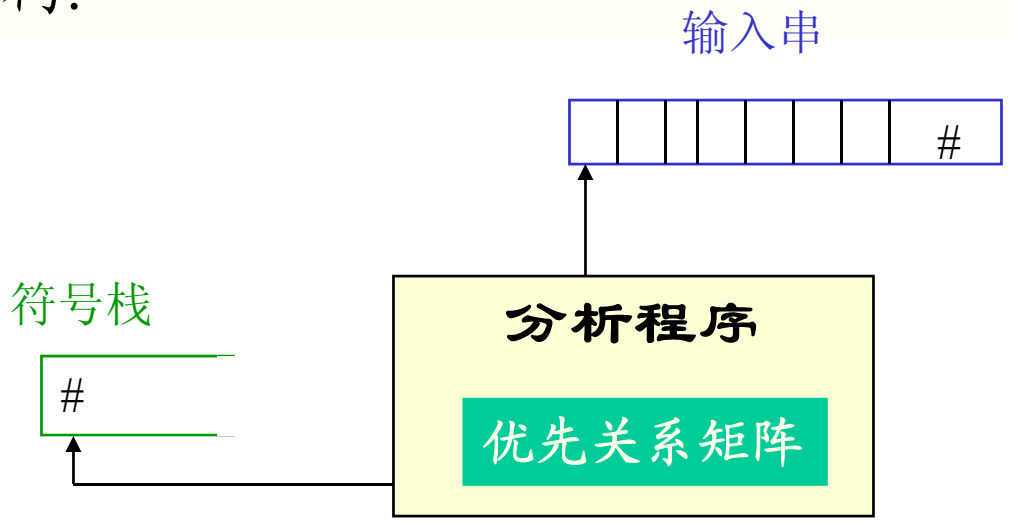
(2)算法:

i)算符优先分析法:

1.分析器的构造,分析过程.

根据算符优先关系矩阵来决定是移进还是规约.

- 分析器结构:



a \ b	+	*	i	(	)	#
+	$\dot{>}$	<	<	<	$\dot{>}$	$\dot{>}$
*	$\dot{>}$	>	<	<	$\dot{>}$	$\dot{>}$
i	$\dot{>}$	>			$\dot{>}$	$\dot{>}$
(	<	<	<	<	$\doteq$	
)	$\dot{>}$	>			$\dot{>}$	$\dot{>}$
#	<	<	<	<		

2.算符优先法的进一步讨论

1.适用的文法类----------引出的算符优先文法的定义
2.优先关系矩阵地构造
3.什么是“句柄”,如何找
有句柄引出的最左素短语的概念.
最左素短语的定理,如何找.

算符优先文法（OPG）的定义

设有一OG文法，如果在任意两个终结符之间，至多只有上述关系中的一种，则称该文法为算符优先文法(OPG)

（2）构造优先关系矩阵

- 求 “ = ” 检查每一条规则，若有U::=…ab…或U::=…aVb…, 则 a=b

- 求 “ < ”、 “ > ”复杂一些，需定义两个集合

$$FIRSTVT(U) = \{b \mid U \overset{+}{\Rightarrow} b\ldots \text{或} U \overset{+}{\Rightarrow} Vb\ldots, b \in V_t, V \in V_n\}$$

$$LASTVT(U) = \{a \mid U \overset{+}{\Rightarrow} \ldots a \text{或} U \overset{+}{\Rightarrow} \ldots aV, a \in V_t, V \in V_n\}$$

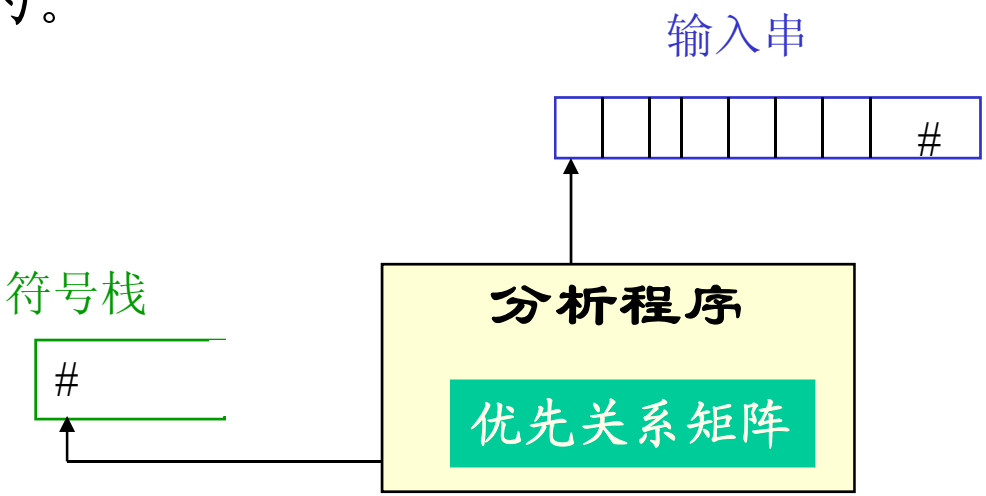
- 求 “$\lessdot$”、“$\gtrdot$”：

若文法有规则
W::=...aU... ， 对任何b, b∈FIRSTVT(U)
则有： a $\lessdot$ b

若文法有规则
W::=...Ub... ， 对任何a, a∈LASTVT(U)
则有： a $\gtrdot$ b

算符优先分析法的实现:

基本部分是找句型的最左子串（最左素短语）并进行归约。



当栈内终结符的优先级＜＝栈外的终结符的优先级时，移进；栈内终结符的优先级＞栈外的终结符的优先级时，表明找到了素短语的尾，再往前找其头，并进行归约。

ii)LR分析法

1.概述----概念、术语　(活前缀、项目)

- 2.LR分析
  - 1)逻辑结构
    - 状态栈
    - 分析表
      - GOTO表
      - 分析动作表
    - 控制程序
  - 2)分析过程

- 3.如何构造SLR分析表
  - 1)构造DFA
  - 2)由DFA构造分析表

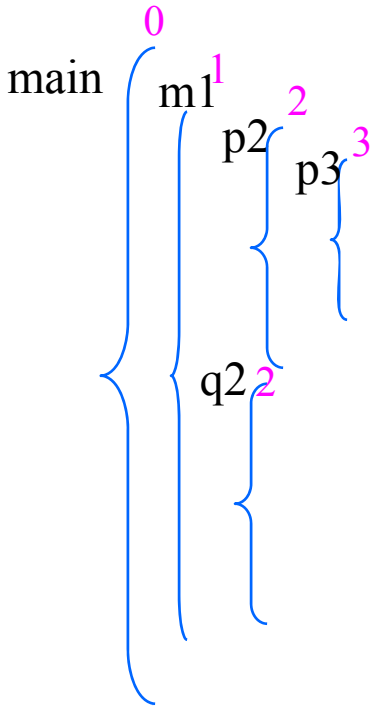
第五章:语法制导翻译技术

- 5.1 翻译文法(TG)和语法制导翻译
- 5.2 属性翻译文法(ATG)
- 5.3 自顶向下语法制导翻译
  - 翻译文法的自顶向下语法制导翻译
  - 属性文法的自顶向下语法制导翻译
- 5.4 自底向上的语法制导翻译（自学）

第六章: 符号表管理技术

- 6.1 概述
- 6.2 符号表的组织与内容
- 6.3 非分程序结构语言的符号表组织
- 6.4 分程序结构语言的符号表组织

```
Progiam  mail (…);                                   0
  var
     x, y : real;   i, k: integer;
     name:  array [1…16] of char;
        :
     procedure  M1 (ind:integer);                    1
          var    x : integer
           procedure  P2 (j : real);                 2
                    :
                  procedure   P3;                    3
                     var
                      f : array [1…5] of intrger
                        test1:  boolean;
                        begin
                          :
                     end;     {P3}
              begin
                 :
             end;{p2}
           procedure  q2;                            2
              var   r1,r2 : real;
               begin
                 :
               p2(r1+r2);
                 :
              end;  {q2}
       begin
          :
         P2(x/y);
          :
     end;{M1}
begin
   :
  M1(i+k);
   :
End    {mail}
```



教授

符号表

	name	kind	type	lev	other inf
1	x	var	real	0	
2	y	var	real	0	
3	i	var	int	0	
4	k	var	int	0	
5	name	var	array	0	
6	M_1	proc		0	
7	ind	para	int	1	
8	x	var	int	1	
9	P_2	proc		1	
10	j	para	real	2	
11	P_3	proc		2	
12	f	var	array	3	
13	test1	var	boolean	3	

main

M_1

P_2

P_3

分程序索引表

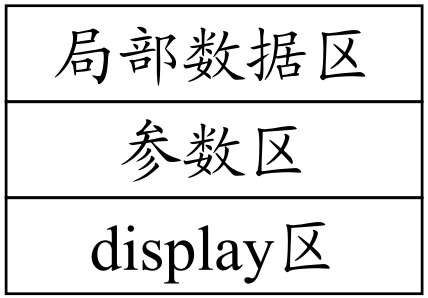
0	1
1	7
2	10
3	12

第七章:运行时的存储组织及管理

- 7.1 概述
- 7.2 静态存储分配
- 7.3 动态存储分配

7.3.1 活动记录

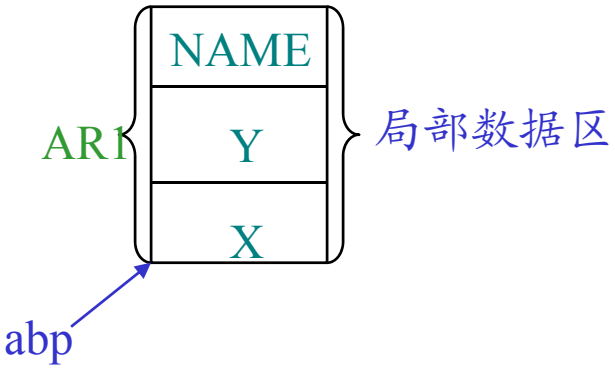
一个典型的活动记录可以分为三部分:



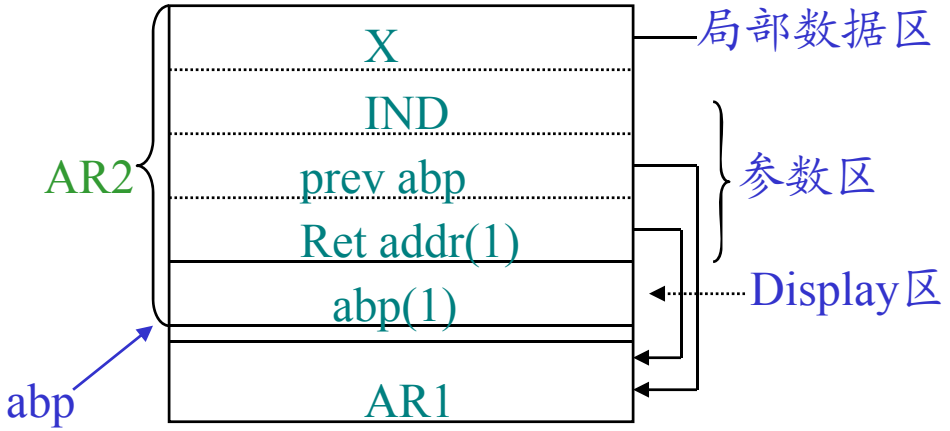
(1)局部数据区:

存放模块中定义的 各个局部变量。

例：下面给出源程序的目标程序运行时，运行栈(数据区栈)的跟踪情况



(a)进入模块1



(b)M1被调用

第八章: 源程序的中间形式

- 8.1 波兰表示
- 8.2 N－元表示
- 8.3 抽象机代码

第九章: 错误处理

- 9.1 概述
- 9.2 错误分类
- 9.3 错误的诊察和报告
- 9.4 错误处理技术

第十章: 语义分析和代码生成

- 10.1 语义分析的概念
- 10.2 栈式抽象机及其汇编指令
- 10.3 声明的处理
- 10.4 表达式的处理
- 10.5 赋值语句的处理
- 10.6 控制语句的处理
- 10.7 过程调用和返回

第11章:代 码 优 化

- 11.1 概述
- 11.2 优化的例子
- 11.3 基本块的优化
- 11.4 循环优化

总结：

优化分为两大类

- 与机器无关的优化独立于机器的（中间）代码优化
  - 局部优化：（一个入口，一个出口，线性）——基本块
    - 方法：常数合并
      - 冗余子表达式的消除等
  - 循环优化：对循环语句所生成的中间代码序列上所进行的优化
    - 方法：循环展开
      - 频度削弱（循环不变表达式的外提）
      - 强度削弱
  - 全局优化：顾名思义，跨越多个基本块的全局范围内的优化。因此它是在非线性程序段上（包括多个基本块,GOTO循环）的优化。
- 与机器有关的优化目标代码上的优化（与具体机器有关）

考试讲解

2.3.3 语言的形式定义

文法G[Z]所产生的所有句子的集合

定义6：文法G[Z]

（1）句型：x是句型 $\Leftrightarrow Z \overset{*}{\Rightarrow} x$，且 $x \in V^*$；

（2）句子：x是句子 $\Leftrightarrow Z \overset{+}{\Rightarrow} x$，且 $x \in V_t^*$；

（3）语言：$L(G[Z]) = \{x \mid x \in V_t^*, Z \overset{+}{\Rightarrow} x\}$；

形式语言理论可以证明以下两点：

（1）G → L(G)；

（2）L(G) → G1，G2，……，Gn;

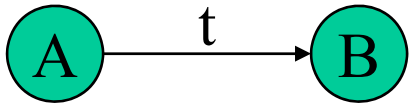
已知文法，求语言，通过推导;

已知语言，构造文法，无形式化方法，更多是凭经验。

第二次

（1）有穷自动机⇒正则文法

算法：



1. 对转换函数f (A, t)=B，可写成一个产生式: A→ tB

2. 对可接受状态Z, 增加一个产生式: Z→ ε

3. 有穷自动机的初态对应于文法的开始符号(识别符号），
有穷自动机的字母表为文法的终结符号集。

为了使得NFA确定化，首先给出两个定义:

定义1、集合I的ε-闭包:

令I是一个状态集的子集，定义ε-closure（I）为:
1）若s∈I，则s∈ε-closure（I）；
2）若s∈I，则从s出发经过任意条ε弧能够到达的任何状态都属于ε-closure（I）。
状态集ε-closure（I）称为I的ε-闭包。

可以通过一例子来说明状态子集的 ε -闭包的构造方法

定义2：令I是NFA M'的状态集的一个子集，$a \in \Sigma$

定义：$I_a = \varepsilon\text{-closure}(J)$

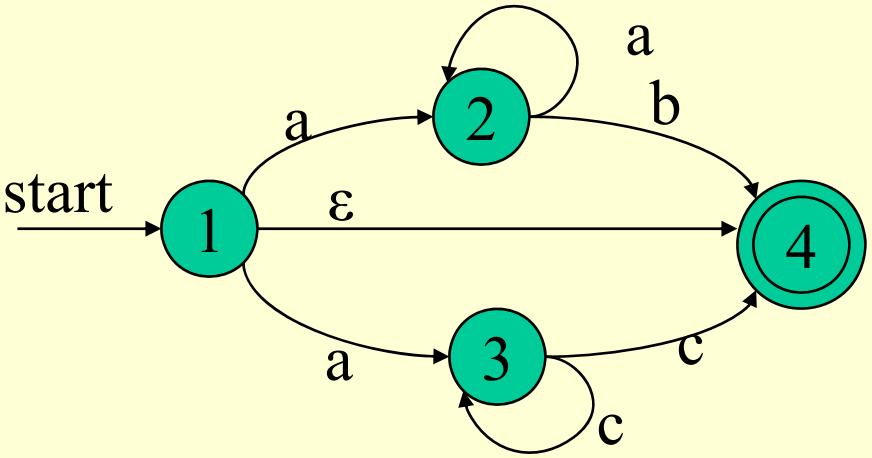
其中 $J = \bigcup_{S \in I} \delta(s,a)$

-- J是从状态子集I中的每个状态出发,经过标记为a的弧而达到的状态集合。

-- I_a是状态子集,其元素为J中的状态,加上从J中每一个状态出发通过ε弧到达的状态。

同样可以通过一例子来说明上述定义,仍采用前面给定的状态图为例

例：有NFA M’



I=ε-closure({1})={1,4}

I_a=ε-closure(δ(1,a)∪δ(4,a))

= ε-closure({2,3}∪φ)

= ε-closure ({2,3})

={2,3}

I_b= ε-closure(δ(1,b)∪δ(4,b))

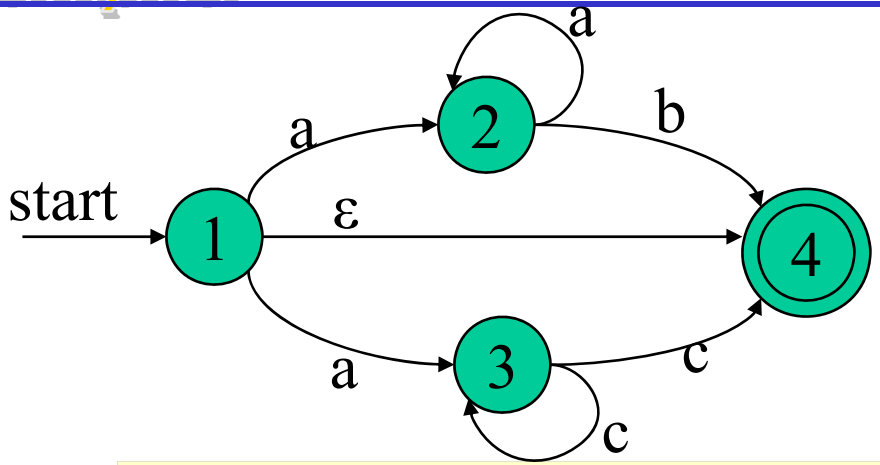
= ε-closure(φ)

=φ

I_c= ε-closure(δ(1,c)∪δ(4,c))

= φ

I={2,3}, I_a={2},I_b={4},I_c={3,4}…



I	I_a	I_b	I_c
{1,4}	{2,3}	φ	φ
{2,3}	{2}	{4}	{3,4}
{2}	{2}	{4}	φ
{4}	φ	φ	φ
{3,4}	φ	φ	{3,4}

I	I_a	I_b	I_c
{1,4}	{2,3}	φ	φ
{2,3}	{2}	{4}	{3,4}
{2}	{2}	{4}	φ
{4}	φ	φ	φ
{3,4}	φ	φ	{3,4}

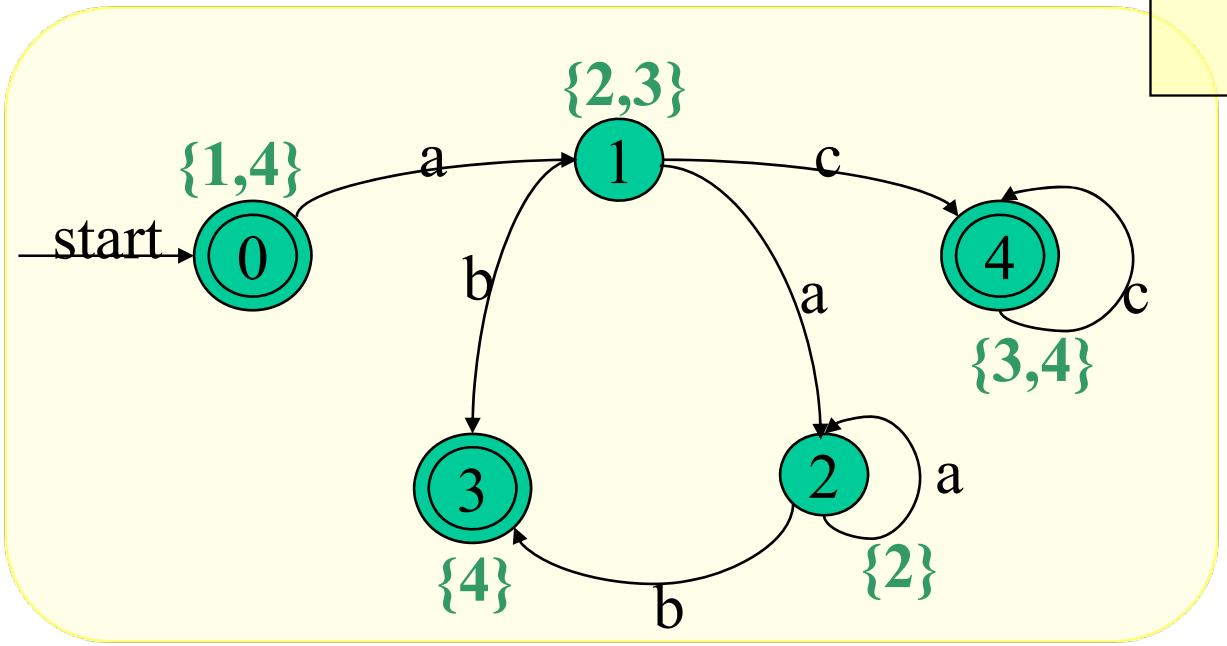
将求得的状态转换矩阵重新编号

DFA M状态转换矩阵:

状态 \ 符号	a	b	c
0	1	–	–
1	2	3	4
2	2	3	–
3	–	–	–
4	–	–	4

状态 \ 符号	a	b	c
0	1	–	–
1	2	3	4
2	2	3	–
3	–	–	–
4	–	–	4

DFA M的状态图:



★ 注意: 包含原初始状态1的状态子集为DFA M的初态
包含原终止状态4的状态子集为DFA M的终态。

三题

补充: DFA的简化(最小化)

“对于任一个DFA，存在一个唯一的状态最少的等价的DFA”

一个有穷自动机是化简的 ⇔ 它没有多余状态并且它的状态中没有两个是互相等价的。

一个有穷自动机可以通过消除多余状态和合并等价状态而转换成一个最小的与之等价的有穷自动机

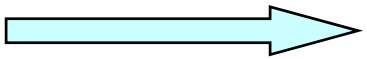
定义:

(1) 有穷自动机的多余状态: 从该自动机的开始状态出发,任何输入串也不能到达那个状态

例:

	0	1
S_0	S_1	S_5
S_1	S_2	S_7
S_2	S_2	S_5
S_3	S_5	S_7
S_4	S_5	S_6
S_5	S_3	S_1
S_6	S_8	S_0
S_7	S_0	S_1
S_8	S_3	S_6

画状态图可以看出S_4,S_6,S_8为不可达状态应该消除



	0	1
S_0	S_1	S_5
S_1	S_2	S_7
S_2	S_2	S_5
S_3	S_5	S_7
S_5	S_3	S_1
S_7	S_0	S_1

(2)等价状态⟺状态s和t的等价条件是:

1)一致性条件: 状态s和t必须同时为可接受状态或不接受状态。

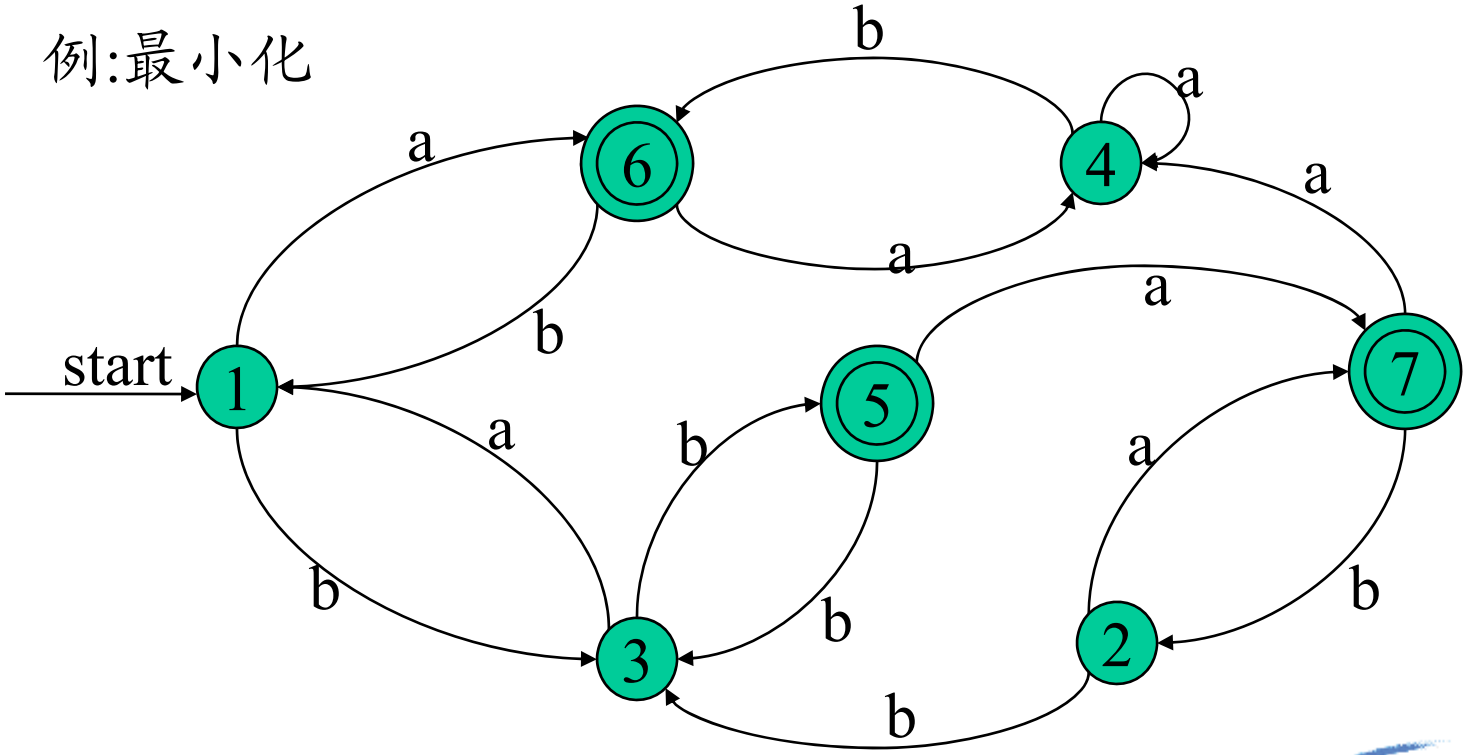
2)蔓延性条件: 对于所有输入符号,状态s和t必须转换到等价的状态里。

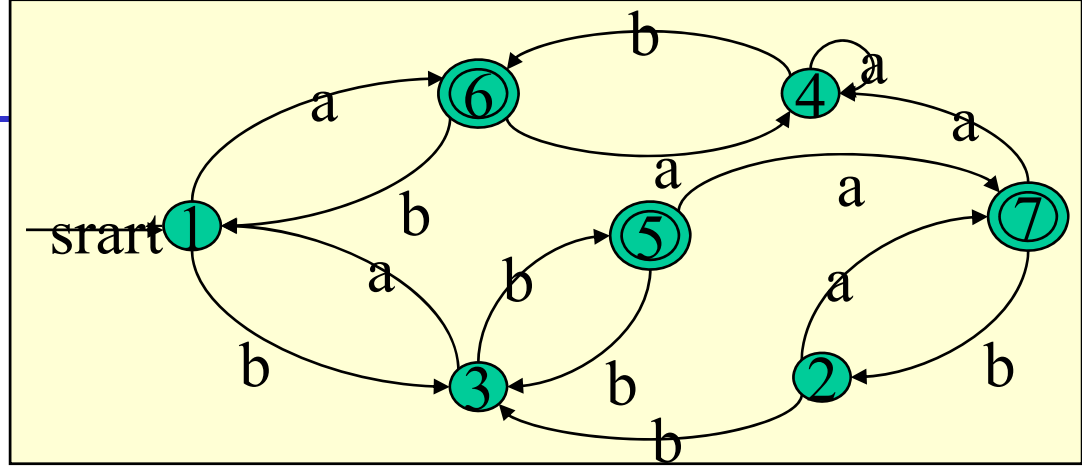
对于所有输入符号c，$I_c(s)=I_c(t)$，即状态s、t对于c具有相同的后继，则称s，t是等价的。
（任何有后继的状态和任何无后继的状态一定不等价）

有穷自动机的状态s和t不等价,称这两个状态是可区别的。

“分割法”：把一个DFA(不含多余状态)的状态分割成一些不相关的子集，使得任何不同的两个子集状态都是可区别的，而同一个子集中的任何状态都是等价的。

例:最小化



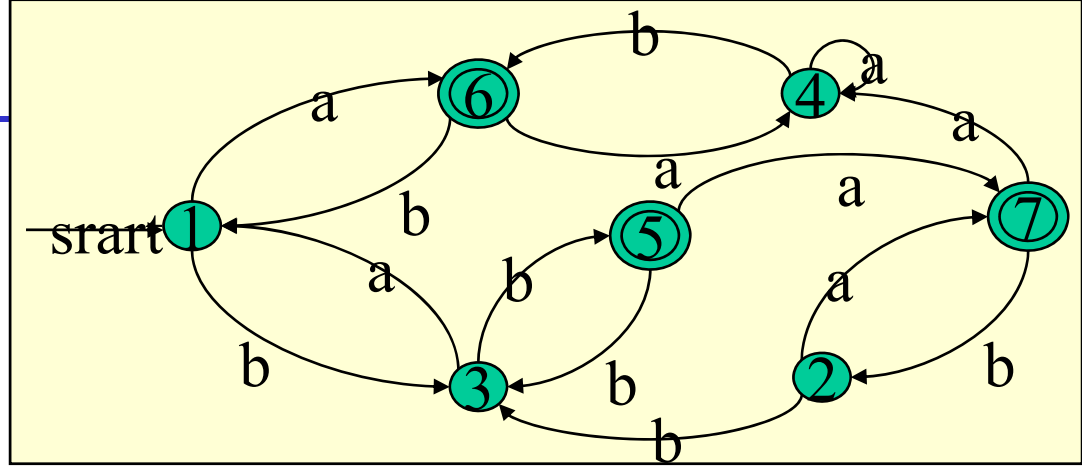


解：

(一) 区分终态与非终态

	a	b	区号
1	6	3	1
2	7	3	
3	1	5	
4	4	6	
5	7	3	2
6	4	1	
7	4	2	

	a	b	区号
1	6	3	1
2	7	3	
3	1	5	2
4	4	6	
5	7	3	3
6	4	1	
7	4	2	



	a	b	区号
1	6	3	1
2	7	3	
3	1	5	2
4	4	6	3
5	7	3	4
6	4	1	
7	4	2	

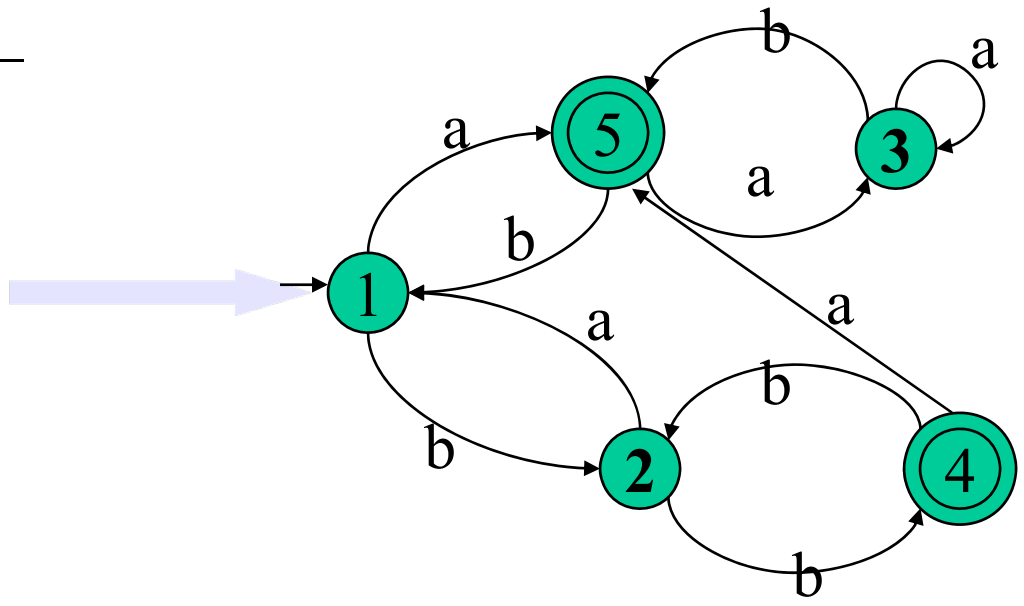
	a	b	区号
1	6	3	1
2	7	3	
3	1	5	2
4	4	6	3
5	7	3	4
6	4	1	5
7	4	2	

	a	b	区号
1	6	3	1
2	7	3	
3	1	5	2
4	4	6	3
5	7	3	4
6	4	1	5
7	4	2	

	a	b
1	5	2
2	1	4
3	3	5
4	5	2
5	3	1

将区号代替状态号得:

	a	b
1	5	2
2	1	4
3	3	5
4	5	2
5	3	1



- 认真复习，有重点，
- 概念清楚、例题自己会做、习题会做。

- 考过了，还会考
- 没有考过的，也会考。