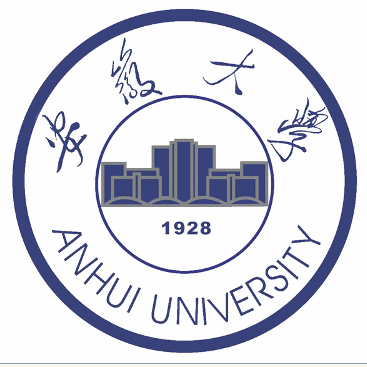
编译原理实验报告

****

**实验名称: 不确定有穷自动机的确定化**

**实验时间： 2020/05/20**

**院 系： 计算机科学与技术学院**

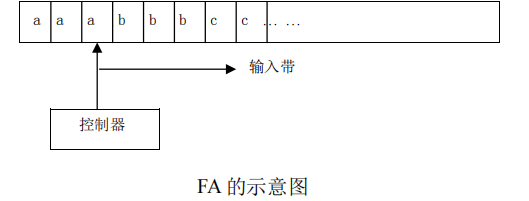
**班 级： 计算机科学与技术一班**

**学 号： E01814055**

**姓 名： 张亚强**

1. **背景资料**

有穷自动机（FA）可以看作是由一个带有读头的有穷控制器和一条字符输

入带组成，如图所示。

控制器的读头从左至右顺次扫描输入带，每当从输入带上读到一个符号时，便引起控制器状态的改变，同时读头右移一个符号位。

控制器包括有穷个状态，状态和状态之间存在转换关系。当处于某个状态，读入一个字符时，则使状态改变为另一个状态，从而形成状态转换，改变后的状态称为后继状态。状态转换后的后继状态有三种可能情况：（1）后继状态为自身；（2）后继状态为一个；（3）后继状态为若干个。

某个有穷自动机，如果每次状态转换的后继状态都是惟一的，则称它是确定有穷自动机（DFA）；如果转换后的后继状态并不都是惟一的，则称它是不确定有穷自动机（NFA）。

有穷自动机的开始工作状态称为初始状态，结束工作的状态称为终止工作状态或接收状态。如果把上一小节中的状态转换图的各个结点看成是某一个状态，初始结点为初始状态，终止结点为终止状态，并且每一条边表示一个转换关系，这样一个有穷自动机的工作状态就可以采用状态转换图来描述了，从而可以把前面的图看成是一个有穷自动机。

对于上图，有穷自动机处在初始状态 0，当读入符号 a 后，自动机便从状态0 转换到后继状态 1 中，再读入一个符号 b 后，自动机便从状态 1 转换到后继状态 2。当自动机读入一个符号串，自动机则从初始状态开始，经过一系列状态转换，最终若能够到达终止状态，则称这一符号串被该自动机所接收或识别，否则不能被该自动机所接收。

**二、实验目的要求**

输入： 非确定有穷（穷）状态自动机。

输出： 确定化的有穷（穷）状态自动机

**三、实验原理**

一 个 确 定 的 有 穷 自 动 机 （ DFA ） M 可 以 定 义 为 一 个 五 元 组 ， M ＝

（K，∑，F，S，Z），其中：

a a a b b b c c … …

输入带

控制器（1） K 是一个有穷非空集，集合中的每个元素称为一个状态；

（2） ∑是一个有穷字母表，∑中的每个元素称为一个输入符号；

（3） F 是一个从 K×∑→K 的单值转换函数，即 F（R，a）＝Q，（R，Q∈K）表示当前

状态为 R，如果输入字符 a，则转到状态 Q，状态 Q 称为状态 R 的后继状态；

（4） S∈K，是惟一的初态；

（5） Z⊆ K，是一个终态集。

由定义可见，确定有穷自动机只有惟一的一个初态，但可以有多个终态，

每个状态对字母表中的任一输入符号，最多只有一个后继状态。

对于 DFA M，若存在一条从某个初态结点到某一个终态结点的通路，则称

这条通路上的所有弧的标记符连接形成的字符串可为 DFA M 所接受。若 M 的初态

结点同时又是终态结点，则称 ε 可为 M 所接受（或识别），DFA M 所能接受的

全部字符串（字）组成的集合记作 L（M）。

一 个 不 确 定 有 穷 自 动 机 （ NFA ） M 可 以 定 义 为 一 个 五 元 组 ， M ＝

（K，∑，F，S，Z），其中：

（1） k 是一个有穷非空集，集合中的每个元素称为一个状态；

（2） ∑是一个有穷字母表，∑中的每个元素称为一个输入符号；

（3） F 是一个从 K×∑→K 的子集的转换函数；

（4） S⊆ K，是一个非空的初态集；

（5） Z⊆ K，是一个终态集。

由定义可见，不确定有穷自动机 NFA 与确定有穷自动机 DFA 的主要区别是：

（1）NFA 的初始状态 S 为一个状态集，即允许有多个初始状态；

（2）NFA 中允许状态在某输出边上有相同的符号，即对同一个输入符号可

以有多个后继状态。即 DFA 中的 F 是单值函数，而 NFA 中的 F 是多值函数。

因此，可以将确定有穷自动机 DFA 看作是不确定有穷自动机 NFA 的特例。和

DFA 一样，NFA 也可以用矩阵和状态转换图来表示。

对于 NFA M，若存在一条从某个初态结点到某一个终态结点的通路，则称

这条通路上的所有弧的标记（ε 除外）连接形成的字符串可为 M 所接受。NFA M

所能接受的全部字符串（字）组成的集合记作 L（M）。

由于 DFA 是 NFA 的特例，所以能被 DFA 所接受的符号串必能被 NFA 所接受。

设 M1和 M2是同一个字母集∑上的有穷自动机，若 L（M1）＝L（M2），则称

有穷自动机 M1和 M2等价。

由以上定义可知，若两个自动机能够接受相同的语言，则称这两个自动机

等价。DFA 是 NFA 的特例，因此对于每一个 NFA M1 总存在一个 DFA M2，使得

L（M1）＝L（M2）。即一个不确定有穷自动机能接受的语言总可以找到一个等价

的确定有穷自动机来接受该语言。

NFA 确定化为 DFA

同一个字符串 α 可以由多条通路产生，而在实际应用中，作为描述控制过

程的自动机，通常都是确定有穷自动机 DFA，因此这就需要将不确定有穷自动

机转换成等价的确定有穷自动机，这个过程称为不确定有穷自动机的确定化，

即 NFA 确定化为 DFA。

下面介绍一种 NFA 的确定化算法，这种算法称为子集法：（1） 若 NFA 的全部初态为 S1，S2，…，Sn，则令 DFA 的初态为：

S＝[S1，S2，…，Sn]，

其中方括号用来表示若干个状态构成的某一状态。

（2） 设 DFA 的状态集 K 中有一状态为[Si，Si+1，…，Sj]，若对某符号 a∈∑，在 NFA

中有 F（{ Si，Si+1，…，Sj }，a）={ Si’，Si+1’，…，Sk’ }

则令 F（{ Si，Si+1，…，Sj }，a）={ Si’，Si+1’，…，Sk’ }为 DFA 的一个转换函

数。若[ Si’，Si+1’，…，Sk‘ ]不在 K 中，则将其作为新的状态加入到 K 中。

（3） 重复第 2 步，直到 K 中不再有新的状态加入为止。

（4） 上面得到的所有状态构成 DFA 的状态集 K，转换函数构成 DFA 的 F，DFA 的字母表

仍然是 NFA 的字母表∑。

（5） DFA 中凡是含有 NFA 终态的状态都是 DFA 的终态。

对于上述 NFA 确定化算法——子集法，还可以采用另一种操作性更强的描

述方式，下面我们给出其详细描述。首先给出两个相关定义。

假 设 I 是 NFA M 状 态 集 K 的 一 个 子 集 （ 即 I∈K ） ， 则 定 义 ε-

closure（I）为：

（1） 若 Q∈I，则 Q∈ε-closure（I）；

（2） 若 Q∈I，则从 Q 出发经过任意条 ε 弧而能到达的任何状态 Q’，则 Q’∈ε-

closure（I）。

状态集 ε-closure（I）称为状态 I 的 ε 闭包。

假设 NFA M＝（K，∑，F，S，Z），若 I∈K，a∈∑，则定义 Ia＝ε-

closure（J），其中 J 是所有从 ε-closure（I）出发，经过一条 a 弧而到达的

状态集。

NFA 确定化的实质是以原有状态集上的子集作为 DFA 上的一个状态，将原状

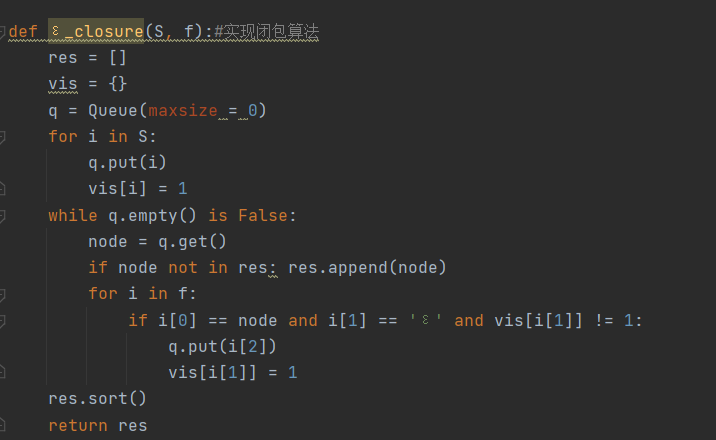
态间的转换为该子集间的转换，从而把不确定有穷自动机确定化。经过确定化后，

状态数可能增加，而且可能出现一些等价状态，这时就需要简化。

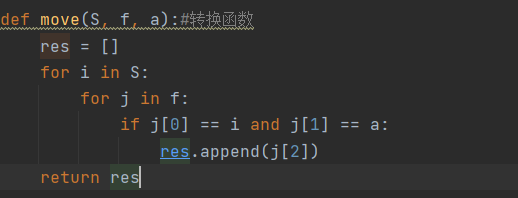
**四、实验内容**

本次实验使用python进行编程。

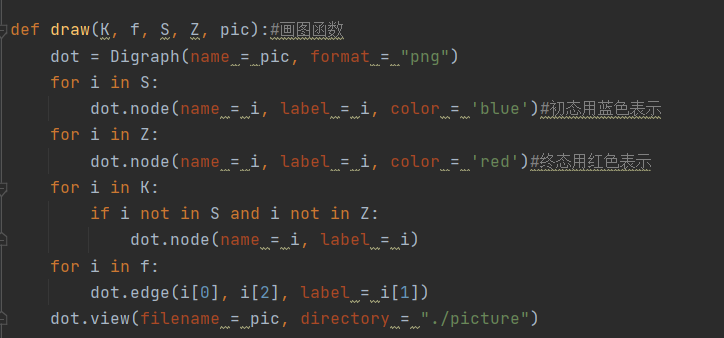
1. 实现计算闭包 closure（I）的算法；



1. 实现转换函数 move（q，a）的算法；



1. 可视化函数



**五、实验心得**

通过本次实验加深了对有穷自动机定义及其相关概念的理解，掌握了从NFA到DFA的转化，学习了图形化工作工具GRAPHVIZ 的使用。

**六、实验代码与结果**

1.代码

**from graphviz import Digraph**

**from queue import Queue**

**def ε\_closure(S, f):#实现闭包算法**

**res = []**

**vis = {}**

**q = Queue(maxsize = 0)**

**for i in S:**

**q.put(i)**

**vis[i] = 1**

**while q.empty() is False:**

**node = q.get()**

**if node not in res: res.append(node)**

**for i in f:**

**if i[0] == node and i[1] == 'ε' and vis[i[1]] != 1:**

**q.put(i[2])**

**vis[i[1]] = 1**

**res.sort()**

**return res**

**def move(S, f, a):#转换函数**

**res = []**

**for i in S:**

**for j in f:**

**if j[0] == i and j[1] == a:**

**res.append(j[2])**

**return res**

**def judge\_end(u, z):**

**for i in u:**

**for j in z:**

**if (i == j): return True**

**return False**

**def draw(K, f, S, Z, pic):#画图函数**

**dot = Digraph(name = pic, format = "png")**

**for i in S:**

**dot.node(name = i, label = i, color = 'blue')#初态用蓝色表示**

**for i in Z:**

**dot.node(name = i, label = i, color = 'red')#终态用红色表示**

**for i in K:**

**if i not in S and i not in Z:**

**dot.node(name = i, label = i)**

**for i in f:**

**dot.edge(i[0], i[2], label = i[1])**

**dot.view(filename = pic, directory = "./picture")**

**if \_\_name\_\_ == "\_\_main\_\_":**

**nfa\_k = ['0', '1', '2', '3', '4']**

**nfa\_t = ['a', 'b']**

**nfa\_f = [('0', 'a', '0'), ('0', 'a', '3'),**

**('0', 'b', '0'), ('0', 'b', '1'),**

**('1', 'b', '2'), ('2', 'a', '2'),**

**('2', 'b', '2'), ('3', 'a', '4'),**

**('4', 'a', '4'), ('4', 'b', '4')]**

**nfa\_s = ['0']**

**nfa\_z = ['2', '4']#初始化一个nfa（书本例3.7）**

**draw(nfa\_k, nfa\_f, nfa\_s, nfa\_z, 'nfa')**

**dfa\_to\_nfa = {}**

**vis = {}**

**node\_id = 0**

**dfa\_f = []**

**dfa\_z = []**

**dfa\_s = []**

**dfa\_k = []**

**dfa\_s.append(str(node\_id))**

**dfa\_k.append(str(node\_id))**

**tmp = ε\_closure(nfa\_s, nfa\_f)**

**dfa\_to\_nfa[str(node\_id)] = tmp**

**c = Queue(maxsize = 0)**

**c.put(tmp)**

**vis[str(tmp)] = str(node\_id)**

**while c.empty() is False:**

**t = c.get()**

**for a in nfa\_t:**

**u = ε\_closure(move(t, nfa\_f, a), nfa\_f)**

**if str(u) not in vis:**

**c.put(u)**

**node\_id += 1**

**dfa\_to\_nfa[str(node\_id)] = u**

**vis[str(u)] = str(node\_id)**

**if judge\_end(u, nfa\_z): dfa\_z.append(str(node\_id))**

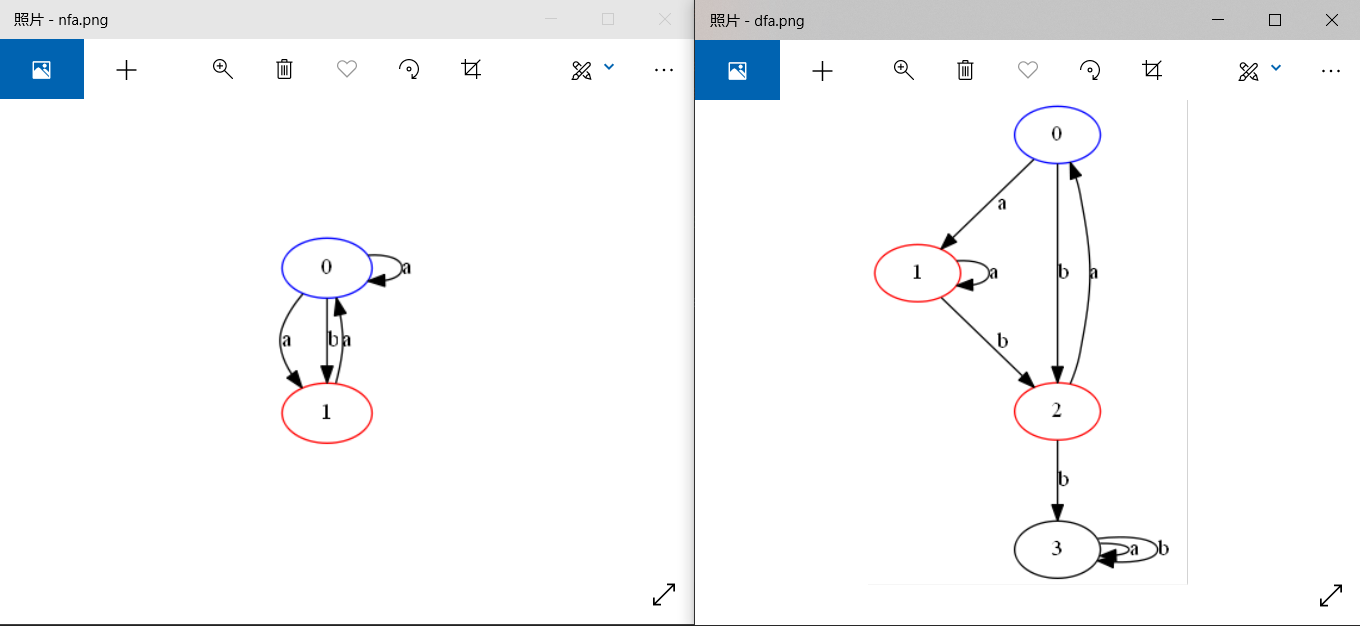
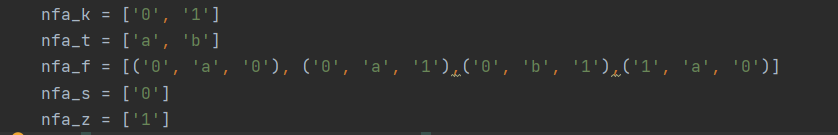
**dfa\_k.append(str(node\_id))**

**dfa\_f.append((vis[str(t)], a, vis[str(u)]))**

**draw(dfa\_k, dfa\_f, dfa\_s, dfa\_z, 'dfa')**

2.运行结果

（1）



（2）课本例3.7

