Introduction To The Theory Of Computation Michael Sipser
计算理论导论

LAB	实验 C—-NFA 转换为 DFA
Name	屈德林
Student No.	201808010522
Class	计算机科学与技术 1805
Department	CSEE
Email	qdl.cs@qq.com
Date	2021 年 3 月 21 日

湖南大學

目录

1 Problem description

有限状态自动机（FSM ”finite state machine” 或者 FSA ”finite state automaton” ）是为研究有限内存的计算过程和某些语言类而抽象出的一种计算模型。有限状态自动机拥有有限数量的状态，每个状态可以迁移到零个或多个状态，输入字串决定执行哪个状态的迁移。有限状态自动机可以表示为一个有向图。有限状态自动机是自动机理论的研究对象。

1.1 定义

有限状态自动机 (FA—finite automaton) 是一个五元组：

- M=(Q, Σ, δ, q0, F), 其中:
- Q——状态的非空有穷集合。
- q∈Q，q 称为 M 的一个状态。
- Σ——输入字母表。
- δ——状态转移函数，有时又叫作状态转换函数或者移动函数，δ：Q×Σ→Q，δ(q,a)=p。
- q0——M 的开始状态，也可叫作初始状态或启动状态。q0 ∈ Q。
- F——M 的终止状态集合。F 被 Q 包含。任给 q ∈ F，q 称为 M 的终止状态。非确定有限状态自动机（NFA）与确定有限状态自动机（DFA）的唯一区别是它们的转移函数不同。确定有限状态自动机对每一个可能的输入只有一个状态的转移。非确定有限状态自动机对每一个可能的输入可以有多个状态转移，接受到输入时从这多个状态转移中非确定地选择一个。下图是一个非确定性有限状态自动机（NFA）的例子：

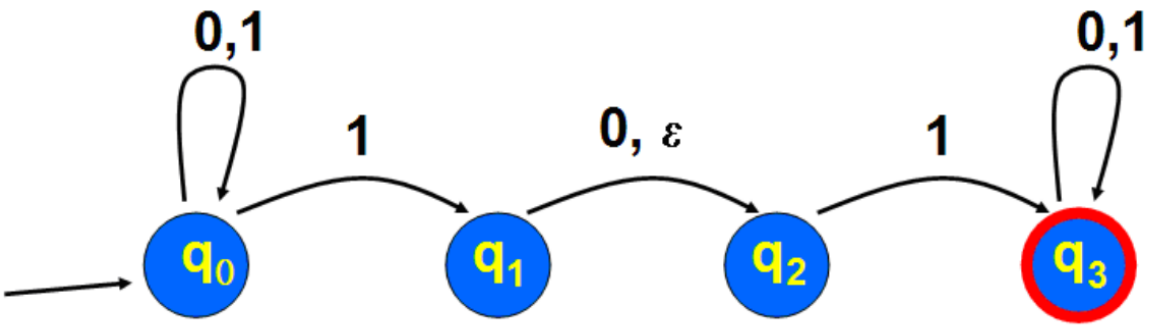

图 1: 一个 NFA 的图文表示

1.2 状态转移表

转移函数 δ 定义自下列状态转移表 1。表示状态集合的子集合，采用二进制（特征）串的方式，一个子集中包含该状态，对应的特征串就为 1，否则为 0，比如上面状态集合的子集 q0q1q2，其特征串就是 0111，而子集 q0，其特征串就是 0001。将对应的特征串转换为十进制的数字，得到转移函数 δ, 如表 2：你的任务，是要将一个给定的 NFA 转换

δ	0	1
q_0	$\{q_0\}$	$\{q_0q_1q_2\}$
q_1	$\{q_2\}$	$\{q_3\}$
q_2	{}	$\{q_3\}$
q_3	$\{q_3\}$	$\{q_3\}$

(a) 表 1#1

δ	0	1
q_0	1	7
q_1	4	8
q_2	0	8
q_3	8	8

(b) 表 2#2

图 2: 转移函数 δ 表格描述

为一个完全等价的 DFA（有限状态自动机等价的意思是识别相同的语言）。这里我们约定自动机识别的字符集为 0,1, 初始状态集合为 Q0，状态集为 q0,q1,⋯,qn-1。

1.3 Input

输入第一行只有一个正整数 t，表示有 t 个测试数据（意味着 t 个 NFA）t≤10; 对于每组测试数据（每个 NFA），首先是 3 个正整数 n,Q0,f，分别表示状态数、起始状态集合和接受状态集合的特征串对应的整数。n≤10;Q0,f < 2n; 接下来两行是 NFA 的转移函数矩阵，第一行是每个状态在输入为 0 的状态转移情况，用特征串对应的整数表示；第二行是每个状态在输入为 1 的状态转移情况。

1.4 Output

对于每个 NFA，输出四行表示与之等价的 DFA。输出格式如下：第一行 3 个空格隔开的整数 a b c，分别表示 DFA 的状态数，接受状态数，起始状态的编号（从 0 开始对状态编号）。要求 a < 65536。b，c ≤ a 第二行 b 个空格分隔的整数，表示每个接收状态的编号, 每个编号的值一定在 [0,a) 之间。第三行、第四行每行 a 个空格分隔的整数，表示 DFA 的转移函数矩阵，第三行第 i 个值 ui 表示状态转移函数的一项 δ（qi，0）→ui，第四行第 i 个值 vi 表示状态转移函数的一项 δ（qi，1）→vi，, 每个 ui,vi 的值一定在 [0,a) 之间。

1.5 Sample Input

1
4 1 8
1 4 0 8
7 8 8 8

1.6 Sample Output

16 8 1
8 9 10 11 12 13 14 15
0 1 4 5 0 1 4 5 8 9 12 13 8 9 12 13
0 7 8 15 8 15 8 15 8 15 8 15 8 15 8 15

1.7 Judge Tips

样例中的 NFA 如图一所示与某个 NFA 等价的 DFA 不一定是唯一的，比如和图一等价的 DFA 可以是样例的解答，也可以是如下的 DFA
4 1 0
3
0 2 0 3
1 3 3 3
本题会使用 special judge，只要是符合条件的解答都可以接受（Accept）。

2 Lab Environment 环境

- 操作系统：Arch Linux
- 程序运行环境：gcc (GCC) 10.2.0
- 报告编写环境：TeX Live 2020
- 开发工具：VSCode

3 Lab Steps 步骤

3.1 分析问题

对于 NFA 转 DFA 的问题，常用子集法求解，可以通过求闭包的方式求得 DFA 的状态，然后编号即可求解问题。但是在本题中，起始状态集合和接受状态集合都是用特征串对应的整数所表示的，举个例子: 状态集合的子集 q0q1q2，其特征串就是 0111，而子集 q0，其特征串就是 0001。将对应的特征串转换为十进制的数字
在本题中，我们直接使用子集法可能并不是最佳的解题方法，因为特征串用递归的方式处理往往会更加方便!

3.2 算法思想

1. 状态集合的子集合，采用二进制（特征）串的方式，一个子集中包含该状态，对应的特征串就为 1，否则为 0，比如上面状态集合的子集 q0q1q2，其特征串就是 0111，而子集 q0，其特征串就是 0001。

2. 采用 DFS 的方式搜索整个状态图。

3. 搜索过程中，对于每一个特征串 p，用 $p \oplus -p$ 取出 p 的最后一个状态（最低非零位），分别根据状态转换表格搜索 0，1 到达的集合，维护到达状态序列 Qlist 和下标。

4. 标记已访问过的状态，直到整个状态图搜索结束，根据记录的序列和下标调整标号，打印结果。

3.2.1 对特征串的处理:

- $x \wedge -x$: lowbit 运算，返回最低位 1 表示的值
- lowbit $x \oplus x$ 运算: 剔除最低位的 1

3.2.2 算法伪代码表述：

经过上述分析，算法伪代码可以表述为：

```
main():
    while(t--)
    {
        Input(DFA);                          // 输入DFA
        DFS();                               // 搜索状态图
        for(int i = 0; i < cnt; i++)         //
            任何包含接受状态的节点都是接受状态
            if(ans[i]&f)
                ac[i] = 1, sum++;
        for(int i = 0; i < cnt; i++)         // 调整序号
            change[ans[i]] = i;
        OutPut(DFA);                         // 输出DFA
    }

DFS(int p):
    while(p)
    {
        int x = p&(-p);                      //
            选择p的一个状态，我们才用与上p的补运算可以取出最后一个为1的位
        int y = indexOfBinary(x);            // 状态x的下标位置
        lsum = travel(lsum, transTable0[y]); // 经过0到达的子集合
        rsum = travel(rsum, transTable1[y]); // 经过1到达的子集合
        p ^= x;                              // 迭代，p =p^x
            剔除已经搜索的状态x，也就是最后一位
    }

    lft[cnt] = lsum;                         // 将DFA中的状态：(qcnt, 0)保存
    rgt[cnt] = rsum;                         // 将DFA中的状态：(qcnt, 1)保存
    cnt++;                                   // 指针移动到下一个DFA状态
    if(!visit[lsum])                         //
        如果lsum还没有被访问，则要迭代访问lsum
        visit[lsum] = 1, DFS(lsum);
    if(!visit[rsum])                         // 同上
        visit[rsum] = 1, DFS(rsum);
```

4 Lab Results 结果

4.1 实验结果

HUNAN UNIVERSITY ACM/ICPC Judge Online

Realtime judge status

Solution	User	Problem	Language	Judge Result	Memory	Time Used	Code Length	Submit Time
697531	jsll201807030415	13120	GNU C++	Accepted	28KB	0ms	1717B	2021-03-19 21:31:42.0
697530	jsll201808010404	13120	GNU C++	Compile Error	0KB	0ms	861B	2021-03-19 17:23:33.0
697529	jsll201808010404	13120	GNU C++	Compile Error	0KB	0ms	1092B	2021-03-19 17:22:41.0
697528	jsll201808010404	13120	GNU C++	Compile Error	0KB	0ms	1092B	2021-03-19 17:22:39.0
697527	jsll201808010404	13120	GNU C++	Compile Error	0KB	0ms	1092B	2021-03-19 17:21:44.0
697525	jsll201808010522	13120	GNU C++	Accepted	1068KB	0ms	2258B	2021-03-19 10:08:00.0
692393	ACM202018110428	13120	GNU C++	Accepted	896KB	0ms	2260B	2020-12-16 20:28:20.0
692058	ACM202004061228	13120	GNU C++	Accepted	1068KB	0ms	2258B	2020-12-15 20:45:51.0
692057	ACM202004061228	13120	GNU C	Compile Error	0KB	0ms	2258B	2020-12-15 20:45:27.0
683067	ACM201911020127	13120	GNU C++	Accepted	1296KB	0ms	2258B	2020-09-11 16:19:27.0
572835	ACM201608030205	13120	GNU C++	Accepted	1260KB	0ms	2259B	2017-09-18 19:13:40.0
572371	ACM201608030208	13120	GNU C++	Accepted	1516KB	0ms	2257B	2017-09-15 09:27:20.0
555142	jsll201408010328	13120	GNU C++	Accepted	928KB	0ms	2255B	2017-01-01 20:08:51.0
555135	jsll201408010304	13120	GNU C++	Accepted	1300KB	0ms	2258B	2017-01-01 16:31:46.0
555035	jsll201408010326	13120	GNU C++	Accepted	928KB	0ms	2257B	2016-12-31 20:10:27.0
555034	jsll201408010326	13120	GNU C++	Accepted	892KB	0ms	2257B	2016-12-31 20:09:52.0
554985	jsll201408010310	13120	GNU C++	Accepted	1300KB	15ms	2237B	2016-12-31 19:05:58.0
554875	jsll201408010311	13120	GNU C++	Accepted	1300KB	0ms	2258B	2016-12-31 14:19:32.0
554866	jsll201408010424	13120	GNU C++	Accepted	28KB	0ms	2258B	2016-12-31 12:44:26.0
554865	jsll201408010424	13120	GNU C++	Accepted	1300KB	0ms	2258B	2016-12-31 12:44:23.0

图 3: http://acm.hnu.cn/online 提交结果

在 http://acm.hnu.cn/online 提交代码，AC 通过. SolutionID 697525，User jsll201808010522，Memory 3072KB Time Used 0ms。因此，实验正确。

5 Lab Experience 心得

5.1 实验心得

1. 本次实验难度适中，因为在教材中都能找到 NFA 确定化和 DFA 最小的的算法。子集法虽然很容易理解，但是在算法实现过程中仍然有很多需要注意的地方。在本题中，由于起始状态集合和接受状态集合都是用特征串对应的整数所表示，子集法可能并不是最佳的解题方法，因为特征串用递归的方式处理往往会更加方便。

2. 对特征串的处理也值得注意：

   - x∧ −x: lowbit 运算，返回最低位 1 表示的值
   - lowbit x $\oplus$ x 运算: 剔除最低位的 1

A 附录 1：Solution

```
#include<iostream>
#include<cmath>
#include<cstring>
#define M 1000000
using namespace std;
int Qlist[M];                    //状态序列，记录的是NFA状态号
int transTable0[M],transTable1[M]; //读取0 / 1对应的转移矩阵
int lft[M];                      //左子集合
int rgt[M];                      //右子集合
int QlistAdjust[M];              //状态序列，记录的是DFA状态号
bool visit[M];                   //标记为已经访问
bool ac[M];                      //接受状态集合
int cnt, n, q, f;
// 将十进制p转化为二进制位对应的下标
int indexOfBinary(int p){
    int x = 1;
    if(p == 1)
        return 0;
    int i = 0;
    while(++i)
    {
        x <<= 1;
        if(p == x) return i;
    }
    return 0;
}
// a:sum , b: 状态号
int travel(int a, int b){
    while(b){
        int x = b&(-b);                  // b的一个状态x = q
        if(!(a&x))                       // 如果a&x =
           0,表示sum中不含x状态，用 ^ 添加
            a ^= x;
        b ^= x;                          // 剔除b中的的x状态
    }
```

```
    return a;                                //
       返回a的结果，表示b能到达的子集合
}
// 搜索状态图，从起始位置p开始
void DFS(int p){
    Qlist[cnt] = p;
    int lsum = 0, rsum = 0;
    while(p){
        int x = p&(-p);                      //
           选择p的一个状态，我们才用与上p的补运算可以取出最后一个为1的位
        int y = indexOfBinary(x);            // 状态x的下标位置
        lsum = travel(lsum, transTable0[y]);// 经过0到达的子集合
        rsum = travel(rsum, transTable1[y]);// 经过1到达的子集合
        p ^= x;                              // 迭代，p =p^x
           剔除已经搜索的状态x，也就是最后一位
    }
    lft[cnt] = lsum;                         // 将DFA中的状态：（qcnt，0)保存
    rgt[cnt] = rsum;                         // 将DFA中的状态：（qcnt，1)保存
    cnt++;                                   // 指针移动到下一个DFA状态
    if(!visit[lsum])                         //
       如果lsum还没有被访问，则要迭代访问lsum
        visit[lsum] = 1, DFS(lsum);
    if(!visit[rsum])                         // 同上
        visit[rsum] = 1, DFS(rsum);
}
int main()
{
    int t;                                   // t个测试数据
    scanf("%d", &t);
    while(t--)                               // 输入测试数据
    {
        scanf("%d%d%d", &n, &q, &f);         //
           状态数,起始状态集合,接受状态集合的特征串对应的整数
        for(int i = 0; i < n; i++)           // 输入转移矩阵0 / 1
            scanf("%d", &transTable0[i]);
        for(int i = 0; i < n; i++)
            scanf("%d", &transTable1[i]);
```

```
            cnt = 0;
            memset(visit, 0, sizeof(visit));
            memset(ac, 0, sizeof(ac));
            visit[q] = 1;
            // 从起始状态开始搜索状态图
            DFS(q);

            // 遍历标记接收状态集合，sum表示当前接受状态个数
            int sum = 0;
            for(int i = 0; i < cnt; i++)
                if(Qlist[i]&f) ac[i] = 1, sum++; // 标记接受状态
            for(int i = 0; i < cnt; i++)
                QlistAdjust[Qlist[i]] = i;      //
                    将ans记录的NFA号调整到DFA上来
            printf("%d %d %d\n", cnt, sum, 0); //
                DFA的状态数，接受状态数，起始状态的编号
            for(int i = 0, j = 0; i < cnt; i++){ // 接收状态的编号
                if(ac[i]){
                    if(j)  printf(" ");
                    printf("%d", i);
                    j++;
                }
            }
            printf("\n");
            for(int i = 0; i < cnt; i++) {      //   (qi, 0) →ui
                if(i)  printf(" ");
                printf("%d", QlistAdjust[lft[i]]);
            }
            printf("\n");
            for(int i = 0; i < cnt; i++){       //   (qi, 1) →vi
                if(i) printf(" ");
                printf("%d", QlistAdjust[rgt[i]]);
            }
            printf("\n");
        }
        return 0;
    }
```