【文章编号】 1004-1540(2007)02-0127-04

# 有穷自动机计算中数据结构的设计

安立新,蓝向阳
(中国计量学院 信息工程学院,浙江 杭州 310018)

【摘 要】 尽管邻接矩阵是有穷自动机的一种常用存储方法,但是,邻接矩阵并不适合存储所有类型的有穷自动机.原因是两个状态间可能有两条以上同方向的弧,而邻接矩阵只能记录两个状态间存在的一条弧.所以,有穷自动机的存储结构最好采用邻接链表来存储.将此数据结构应用于 NFA 转换为 DFA 的计算,将传统的子集法中状态转换矩阵,由三维数组降低为二维数组.

【关键词】 有穷自动机;数据结构;邻接链表;邻接矩阵
【中图分类号】 TP301 【文献标识码】 A

## Data structure designed for finite automata computation

AN Li-xin, LAN Xiang-yang
(College of Information Engineering, China Jiliang University, Hangzhou 310018, China)

**Abstract:** Although the adjacency matrix was a storage structure of finite automaton in common use, it was not proper to store all kinds of finite automata. Perhaps existing two arcs above in the same direction between two states, adjacency matrix could only record one of them. So the storage structure of finite automaton should the choosed adjacency lists. We use it to realize the computation of transition from NFA to DFA by reducing the number of the transition matrix dimensions from a three-dimensional array in traditional subset approach to a two-dimensional array in the new algorithm.

**Key words:** finite automata; data structure; adjacency lists; adjacency matrix

有穷自动机是一种具有离散输入输出系统的数学模型.它具有任意有限数量的内部格局或状态,用此来记忆过去输入的有关信息,根据当前的输入可确定下一步的状态和行为.一个有穷自动机等价于一个状态转换图,由于状态转换图与程序有一定的对应关系,使得程序设计规范化.因此,有穷自动机理论在软件工程建模[1-4]、程序编译[5]和信息安全[6,7]等领域有着重要的应用价值.

可以应用有穷自动机的有关理论进行等价变换、状态最小化、程序自动生成等多种运算,这些运算具有通用性.参考文献中对有穷自动机的运算要么是只给出理论计算方法,没有数据结构[8],要么数据结构空间复杂度较高[5,9].本文以 DFA 到 NFA 的转换运算为例,对有穷自动机设计了合理的数据结构,并在此基础上设计了相关的算法,降低了转换矩阵的空间复杂度,通过实例说明合理的数据结构设计能够使算法简单易懂.

【收稿日期】 2007-01-30

【作者简介】 安立新(1967－),男,辽宁阜新人,副教授.主要研究方向为软件工程、计算理论等.

# 1 有穷自动机的基本概念

## 1.1 有穷自动机的定义

确定型有穷自动机(记为 DFA)是一个五元组 $M=(K,\sum,\delta,s,F)$. 其中,$K$ 是有穷状态的集合,$|K|$ 表示为状态个数;$\sum$是一个字母表(所有输入符号),$|\sum|$ 表示为字母表中字符个数;$s\in K$ 是初始状态;$F\subseteq K$ 是终结状态的集合;$\delta$是转移关系,它是从 $K\times\sum$到 $K$的函数,叫做转移函数.

如果把确定型有穷自动机中转移函数 $\delta$改成关系 $\Delta\subseteq K\times(\sum\cup\{e\})\times K$就得到了非确定型有穷自动机(记为 $NFA$),这里的 $e$ 表示空串.

## 1.2 DFA 到 NFA 的转换

根据自动机理论,对于每一台非确定型有穷自动机,有一台等价的确定型有穷自动机. 设 $M=(K,\sum,\Delta,s,F)$ 是一台非确定型有穷自动机. 要构造一台等价于 $M$ 的确定型有穷自动机 $M'=(K',\sum,\delta,s',F')$. 关键的想法是认为非确定型有穷自动机在任一时刻不是处于一个状态而是处于一个状态集合,即从初始状态出发通过到此为止消耗掉的输入能够达到的所有状态.

构造 $M$ 就是把这个想法形式化. $M$ 的状态集合 $K$ 是 $M$ 的状态集合的幂集,即 $K'=2^K$. $M$ 的终结状态集合由 $K$ 中至少包含 $M$ 的一个终结状态的所有子集组成. 至于 $M$ 的转移函数可以看作:$M$ 对读入一个输入符号 $a\in\sum$的动作模仿 $M$ 对输入符号 $a$ 的动作,后面可能跟着 $M$ 的任意次 $e$ 动作.

对任一状态 $q\in K$,令 $E(q)$ 等于 $M$ 从状态 $q$ 开始、不读任何输入能够到达的所有状态的集合,即 $E(q)$ 是集合$\{q\}$在关系$\{(p,r):$有转移$(p,e,r)\in\Delta\}$下的闭包.

# 2 有穷自动机的数据结构设计

## 2.1 有穷自动机的存储结构

从数据结构的角度来看,有穷自动机的逻辑结构是一种图,更确切地说是网络. 对于图(或网络)的存储一般采用邻接矩阵和邻接链表. 尽管邻接矩阵是有穷自动机的一种常用存储方法[10],但是,邻接矩阵并不适合存储所有类型的有穷自动机. 原因是两个状态间可能有两条以上同方向的弧,而邻接矩阵只能记录两个状态间存在的一条弧. 所以,有穷自动机的存储结构最好采用邻接链表来存储. 本文采用的邻接链表的形式说明如下:

```
typedef struct node{      //边表结点
    int      adjvex;      //邻接点域
    char     read_char;   //读入字符(或理解为权)
    struct node  *next;   //链域
}edgenode;

typedef struct{           //顶点表结点
    vextype  vertex;      //顶点域
    edgenode  *link;      //边表头指针
}vexnode;
vexnode ga[|K|];          //ga 是邻接表类型
```

## 2.2 状态 i 的 e 闭包存储结构

状态 $i$ 的 $e$ 闭包 $E(i)$ 定义为:状态集合 $E(i)$ 中的任一状态经任意条 $e$ 弧而能达到的状态集合. 由于 $E(i)$ 的个数 $\leq|K|$,所以可以用向量来表示. 假设 $E(0)=\{0,1,2,4\}$ 且 $|K|=5$,则 $E(0)$ 在计算机中可存储为$(1,1,1,0,1)$.

为了存储每个状态的状态集合,采用二维数组整型数组 $E[|K|][|K|]$来实现.

## 2.3 状态转换矩阵的存储结构

状态转换矩阵就是 $\delta$转换函数. 传统的子集法[5]中状态转换矩阵 C 采用三维数组,即 int C$[|K'|][|\sum|+1][|K|]$,而在本文中采用了二维数组,即 int C$[|K'|][|K|+1]$来实现. 其中 $|K'|\leq 2^K$,表示 $M$ 的状态数,算法中可定义为一个常数. 其中,0 到 $|K|-1$ 列用来保存状态集合,第 $|K|$ 列用来保存该状态集合是否是新状态集.

# 3 NFA 转换 DNA 算法的设计

## 3.1 状态 i 的 e 闭包算法

通过对状态 $i$ 的深度优先搜索,可以求得状态 $i$ 的 $e$ 闭包,即是状态 $i$ 经任意条 $e$ 弧而能达到的状态集合,结果保存在 visited$[|K|]$数组中.

```
int visited[|K|];
vexnode gl[|K|];
void e_closure_i(int i){
    int j; edgenode  *p;
    visited[i]=1;   //标记 v_i 已访问
    p=ga[i].link;   //取 v_i 边表的头指针
    while(p){
      if (!visited[p->adjvex-1]&&p->
read_char=='e')     //若 v_i 尚未被访问
```

```
        e_closure_i (p - > adjvex - 1) ;
                    //以 V_j 为出发点向纵深搜索
        p = p - > next ;//找 v_i 的下一邻接点
      }
}
```

### 3.2　所有状态的 e 闭包算法

```
void e_closure_all(){
for(i = 0 ;i < | K| ;i + + ){
   e_closure_i(i) ;
   for(j = 0 ;j < | K| ;j + + )
   E[i][j] = visited[j] ;
   visited[j] = 0 ;}
}
```

### 3.3　状态集合的 e 闭包算法

状态集合存储在向量 visited 中,结果保存在向量 U 中.

```
void e_closure()
   { For(i = 0 ;i < | K| ;i + + )
      {if(visited[i] = = 1)  U = U ∪E[i] ;}
}
```

### 3.4　状态集合 I 的 a 弧转换

状态集合存储在 E[i]向量中,结果保存在向量 visited 中.

```
void Move(int i ,char a)
{for(j = 0 ;j < | K| ;j + + )
   {if(E[i][j] = = 1)
      {p = ga[i].link ;
      while(p)
      If(p - > readchar = = a)  visited[p - > adjvex] = 1 ;
      p = p - > next ;
      }
        }
      }
```

### 3.5　NFA 的状态转换算法

为了实现 $M$ 向 $M'$ 的转化,关键是得到 $\delta'$ 转换函数. 对于图 1 的 NFA,因为 $M$ 有 5 个状态,所以 $M'$ 最多有 32 个状态. 但是,这些状态中只有几个与 $M$ 运行有关,即它们能够从状态 $s$ 开始、通过读入某个输入串到达. 从 $s$ 开始,当对于某个已引入的状态 $q \in K'$ 和某个 $a \in \sum$,$\delta'(q,a)$ 是一个新的状态时引入这个状态,这样可以得到 $M'$ 的可达到的状态.

```
Void transition()
{   i = 0 ;j = 1 ;    //i 是双亲结点下标;j 是孩子结
                      //点下标
  C[0] = E[0] ;  //s' = E{s}
  C[0][| K| ] = 1 ; //标识为新结点
  while(i ! = j)
  {for(每一个输入字母 a)
    {if(C[j][| K| ] = 0)  {i + + ;break ;}
                      //老结点不再进行 a 弧转换
    Move(i ,a) ; //状态集合 I 的 a 弧转换,所到达
                      //的状态集保存在向量 visited 中.
    e_closure() ;//状态集合存储在向量 visited
                      //中,结果保存在向量 U 中.
    C[j] = U ;
    if(U 是新状态集) C[j][| K| ] = 1 ;
    j + + ;
    }
    i + + ;
  }
    }
```

## 4　实例分析

下面以图 1 的 NFA $M$ 为例,说明其数据结构和运算过程. 首先,要建立 $M$ 的邻接链表,由于其实现简单,故省略. 在图 1 中共有 5 个状态,通过调用 e_closure_all()算法可求得各个状态的 $e$ 闭包(即状态集合). 结果如表 1 所示. 其含义为:$E(0)=\{0,1,2,3\}$,$E(1)=\{1,2,3\}$,$E(2)=\{2\}$,$E(3)=\{3\}$,$E(4)=\{3,4\}$.

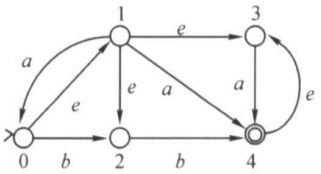


图 1　非确定型有穷自动机 M

表 1　每个状态的状态集合 E

| | | | | |
|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 0 |
| 0 | 1 | 1 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 0 | 1 | 1 |

在求得各个状态的 $e$ 闭包后,在二维数组 C 中求 $K'$ 和 $\delta'$,通过调用 alteration()算法可求得各个状态 $K'$,结果如表 2. 其求解过程采用二叉树(这里|∑| = 2)描述,如图 2 所示. 图 2 中的根

结点就是 $s'$,左孩子是 $a$ 弧转换得到的状态集,右孩子是 $b$ 弧转换得到的状态集. 结点序号表示计算顺序,与 C 数组的下标一致. 结点序号中带 * 号的结点表示不是新状态集,相当于 C 数组中 $|K|$ 列值为 0,无须再进行转换. 最后,通过 C 数组中的数据可求得 $M'$,如图 3.

表 2 状态转换矩阵 C

| 1 | 1 | 1 | 1 | 0 | T |
|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | T |
| 0 | 0 | 1 | 1 | 1 | T |
| 1 | 1 | 1 | 1 | 1 | F |
| 0 | 0 | 1 | 1 | 1 | F |
| 0 | 0 | 0 | 1 | 1 | T |
| 0 | 0 | 0 | 1 | 1 | F |
| 0 | 0 | 0 | 1 | 1 | F |
| 0 | 0 | 0 | 0 | 0 | T |
| 0 | 0 | 0 | 0 | 0 | F |
| 0 | 0 | 0 | 0 | 0 | F |

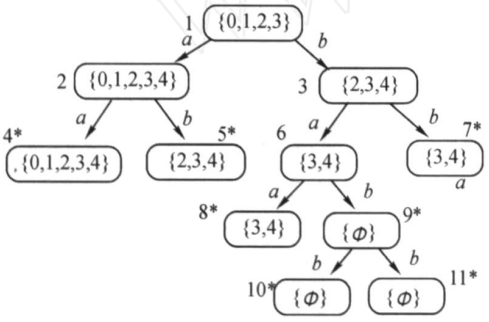


图 2 $\delta$的求解过程

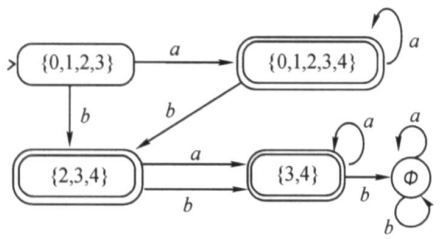


图 3 确定型有穷自动 M′

## 5 结 论

在本文中状态转换矩阵 C 采用了二维数组,实现了树形的运算过程,降低了空间复杂度,使得算法易于实现. 同时,对于一般的有穷自动机采用邻接链表来作为其存储结构,状态 $i$ 的 $e$ 闭包运算通过对状态 $i$ 的深度优先搜索得到,进一步说明合理的数据结构设计能够使算法简单易懂.

对于有穷自动机计算的一个很好的做法是把有穷自动机的存储结构和算法封装成一个抽象类型,例如是类的形式,便于采用面向对象的方法继承使用.

## 【参 考 文 献】

[1] 徐小良,汪乐宇,周 泓. 有限状态机的一种实现框架[J]. 工程设计学报,2003(10):251-255.

[2] 魏先民. 有限状态机在嵌入式软件中的应用[J]. 潍坊学院学报,2006,6(4):24-25.

[3] 肖健宇,张德运,陈海诠,等. 基于 UML 状态机与 B 方法的高可信嵌入式软件开发[J]. 计算机工程,2006,32(8):64-66.

[4] 姚鑫骅,潘雪增,傅建中,等. 微制造数控系统的实时有限状态机建模研究[J]. 浙江大学学报(工学版),2005,39(12):1965-1968.

[5] 陈火旺. 程序设计语言编译原理[M]. 3 版. 北京:国防工业出版社,2000:49-51.

[6] 王英梅,程湘云,刘增良. 基于有限状态机的多阶段网络攻击方法研究[J]. 空军工程大学学报,2006,7(1):31-34.

[7] 訾小超,姚立红,李 斓. 一种基于有限状态机的隐含信息流分析方法[J]. 计算机学报,2006,29(8):1460-1466.

[8] LEWIS H R, PAPADIMITRIOU C H. Elements of the theory of computation[M]. 北京:清华大学出版社,1999:69-73.

[9] 毛红梅,聂承启. 一种将 NFA 到最小化 DFA 的方法[J]. 计算机与现代化,2004(10):6-7.

[10] 朱征宇,付关友,赵银春. 矩阵模型表示下有限自动机等价判定方法[J]. 计算机工程与应用,2004,34:54-56.