一类 NFA 到 DFA 的直接转化方法①

程元斌

(江汉大学 数学与计算机科学学院, 武汉 430056)

摘 要: NFA 的确定化具有重要的理论和实际意义. 迄今为止, 普遍采用子集构造法将一个 NFA(非确定性自动机)转化为 DFA(确定性自动机),但这种方法需要引入空输入 ε 及状态子集 I 的 ε -闭包, 其计算过程相对繁琐. 而且在确定化过程中对于 NFA 状态集存在 ε -closure 重复计算和由于对非 ε 转换的判断而引起的重复计算等问题. 本文描述了一种将一类 NFA 直接转化为 DFA 的方法. 在本方法中, 不需要引入空输入 ε , 可根据原始的 NFA 状态图或状态转移表直接得出等价的 DFA 状态图或状态转移表, 而且所有状态都是单一的状态而非集合状态, 便于软硬件实现与测试.

关键词: 有限自动机; 非确定性有限自动机; 确定性有限自动机; 子集构造法; 直接转换法

Translating a Kind of NFA Into DFA Straightly

CHENG Yuan-Bin

(School of Mathematics & Computer Science, Jianghan University, Wuhan 430056, China)

Abstract: It is important in theory and practice to translate a NFA into DFA. So far, subset construction is the most popular method. The method, however, needs import a dummy input ε and ε-closure and has a complicated computing procedure. In this paper, a new method to translate a NFA into DFA straightly is described. The new method needn't import a dummy input ε and ε-closure, it accords to the original NFA state graph or state shift table to translate a NFA into DFA straightly.

Key words: finite automata; nondeterministic finite automata; deterministic finite automata; subset construction; convert straightly

1 引言

有限自动机(Finite Automata, FA)在计算机领域中有着广泛地运用. 有限自动机可分为确定性有限自动机(DFA, Deterministic Finite Automata)与非确定性有限自动机(NFA, Nondeterministic Finite Automata)两类. NFA 的特点是占用存储空间较小,但在运行过程中每读入一个字符,都要更新其所维护的全部状态,因此运行效率较低; 而 DFA 的运行效率则高得多[1]. 所以, 人们常常将 NFA 转化为 DFA. 迄今为止, 普遍采用子集构造法[2, 3]将一个 NFA 转化为 DFA. 不过, 子集构造法虽然能够正确地完成 NFA 的确定化,但是相当繁琐, 而且在确定化过程中对于 NFA 状态集存在 ε - closure 重复计算和由于对非 ε 转换的判断而引起的重复计算等问题[4,5,6]. 当 NFA 状态集元素数量较多时重复计算的问题非常严重[7]. 文献[3]和[7]从不同的角度对子集构造法进行了优化, 但仍然是在子集构造法的框架内. 本文提出一种非子集构造法的将NFA直接转化为DFA的新方法, 可以从根本上解决子集构造法的不足.

2 子集构造法一览

为了更好地说明问题, 有必要先对 NFA 及子集构造法做一个概要的描述.

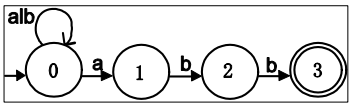
一个 NFA 是一个五元组, 即

$$M = (S, \Sigma, f, S0, Z) \quad (1)$$

① 收稿时间:2012-02-23;收到修改稿时间:2012-03-27

其中, Σ是有穷字符表, S 是有限状态集, f 是状态转移函数, f 定义了从 S×Σ到ρ(S)的映射, ρ(S)是 S 的幂集, S0 与 Z 分别是初态集与终态集.

图 1 是引自文献[2]的一个典型的 NFA, 分别用状态转移图(a)与状态转移表(b)表示. 为简单起见, 本文对所有的终态时的输入边都略去了.



(a)状态图

现态 \ 输入 次态	a	b
0	{0, 1}	0
1	①	2
2	①	3
3	3	3

(b)状态转移表

图 1　非确定性有限自动机 M1

从图 1 可见, 当系统当前状态处于状态 0 时, 如果输入字符 a, 则有两种可能的次态: 0 或 1. 而对于现态 1 和 2, 如果输入字符 a, 则次态为空. NFA 的确定化就是要消除这两种情况. 基于子集构造法的 NFA 确定化算法首先引入空输入 ε, 并由此得到图 2 所示的状态图. 图 2 仍然是一个 NFA, 但其不确定边已转变为 ε. 然后, 引入状态子集 I 的 ε-闭包, 即 ε-closure(I), 并将每一个 ε-closure(I)看作一个状态. 便可进一步得出图 3a 所示的状态转移表, 但其计算过程相对繁琐[2]. 然后, 再对图 3a 中的每一个 ε-closure(I)进行重命名, 得到图 3b 的状态转移表及对应的图 3c 所示的状态图, 易见这是一个 DFA, 但存在多余的状态. 最后, 将该 DFA 化简为图 3d 所示的最小化 DFA. 这一过程亦比较繁琐[2, 3, 7].

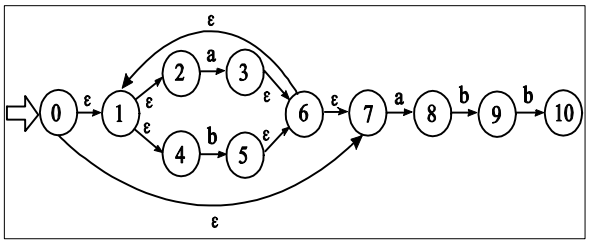


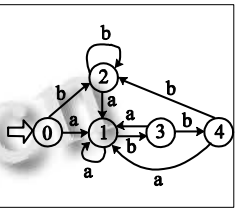
图 2　引入空输入后的 NFAM1'

S	a	b
0,1,2,4,7	3,8,6,7,1,4,2	5,6,71,2,4
3,8,6,7,1,2,4	3,8,6,7,1,2,4	5,9,6,7,1,2,4
5,6,71,2,4	3,8,6,7,1,2,4	5,6,71,2,4
5,9,6,7,1,2,4	8,3,6,7,1,2,4	10,5,6,7,1,2,4
10,5,6,7,1,2,4	8,3,6,7,1,2,4	5,6,7,1,2,4

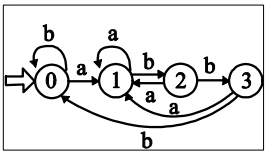
(a)状态转移表

s	a	b
0	1	2
1	1	3
2	1	2
3	1	4
4	1	2

(b)重命名的状态转移表



(c)DFA 状态图



(d)最小化 DFA 状态图

图 3　基于子集构造法的 NFA 到 DFA 的转化

3　基于状态图的直接变换法

所谓直接变换法, 就是根据原始的 NFA 状态图或其他表示方法的 NFA, 直接转化为 DFA, 而不需要引入辅助的手段.

直接转化法既可以根据 NFA 状态图进行, 也可以根据 NFA 状态表进行. 两种方法在本质上是一样的. 本节描述基于状态图的转化规则与步骤.

考察图 1a, 当自动机状态未到达终态前, 无论现态是什么, 只要输入是 a, 次态就可以是 1. 输入 a 之后如果紧跟着两个连续的输入 b, 则自动机将到达终态 3, 否则回到状态 1. 所以, 图 1a 的 NFA 等价于图 4 所示的 DFA.

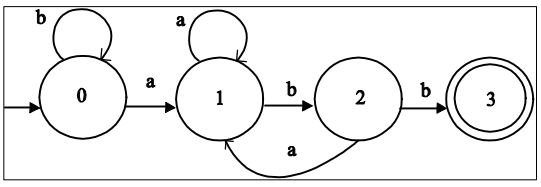
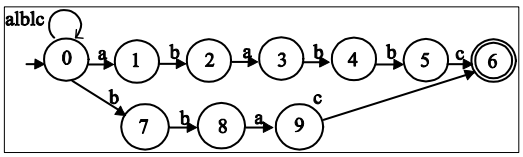


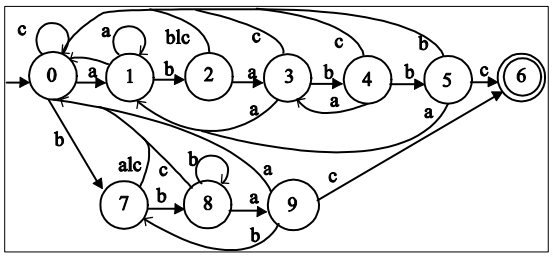
图 4　NFA 到 DFA 的直接转化(1)

图 5(a)是一个更一般同时也更全面的 NFA.

下面结合图 5(b)所示, 给出基于 NFA 状态图的直接转化法的一般规则.



(a)NFA M2 状态图



(b)NFA M2 的等价 DFA 状态图

图 5 NFA 到 DFA 的直接转化(2)

为描述方便, 我们先给出以下定义:

定义 1. 若一个状态存在这样的输入, 其次态可能是该状态本身, 也可能是另一个状态, 则称该状态为基态.

定义 2. 若一个非基态状态是基态的次态, 则称该状态为启动态. 称基态下次态可能为启动态的输入为启动输入; 基态下的其余输入称为踏步输入.

定义 3. 称启动态到终态的状态转移路径上的所有状态为途中态.

定义 4. 称启动态到终态的一条移除了回退部分的状态转移路径为前向分支或前向路径; 其中的输入称为前向输入; 启动输入与该前向分支上的前向输入构成的输入序列合称为分支输入序列.

显然, 从状态图的角度考察, NFA 与 DFA 的区别就在于 NFA 具有基态而 DFA 没有基态. 因此, 只要消除了 NFA 中的基态, NFA 就变成了 DFA.

为叙述简明且不失一般性, 本文所述方法的描述对象是这样一类 NFA, 它仅有一个基态状态 0, 即途中态中不再有基态; 且启动态到终态的状态转移路径上不存在回路. 并假定基态下的所有 n 个输入中有 m(1≤m≤n)个启动输入.

将 NFA 转化为 DFA 的步骤如下:

步骤1: 从基态的自循环边的n个输入中删去可进入启动态的那 m 个输入, 即使得基态的次态都成为确定性的. 经此处理后, 原有的 NFA 状态图已成为一个具有 m 条从基态出发的前向分支的 DFA. 图 4 是具有一个前向分支的实例, 图 5 是具有 2 个前向分支的实例. 例如, 图 5a 中, 输入 a 与 b 分别是从基态 0 到启动态 1 与 7 的启动输入, 所以, 要从基态 0 的自循环边的输入 a|b|c 中删去 a 与 b, 得到图 5b 中基态 0 的自循环边的输入为 c.

步骤 2: 对每一条前向分支上的除了基态和终态外的状态都补上一条到该分支的启动态的边, 对应的输入为该分支的启动输入, 例如图 5b 中从状态 1、3、5 出发的 a 边及从状态 9 出发的 b 边.

这里有一个例外: 如果某条分支从启动态开始有直到状态 K 的连续的启动输入边, 则状态 K 补上一条输入为启动输入的自循环边, 而状态 K 之前的状态显然不再需要补充启动输入边. 例如图 5b 中从状态 8 出发的 b 边展现了这种情况.

还有一种例外将在步骤 3 中陈述.

步骤 3: 对每一条前向分支上的除了基态和终态外的状态都补上一条到基态的边, 对应的输入为除了该分支的启动输入及该状态的前向输入外的所有输入, 例如图 5b 中从状态 1、3、4、8 出发的 c 边, 从状态 2 出发的 b|c 边、从状态 5 出发的 b 边、从状态 7 出发的 a|c 边以及从状态 9 出发的 a 边.

这里也有一个还涉及到步骤 2 的例外: 如果某条分支的分支输入序列为$\{I_0xI_1I_2yI_3\}$, 其中序列 I_2=I0, x 的次态为 S_x; 而实际有输入序列$\{I_0xI_1I_2x\}$时, 则其次态是 S_x, 而不是基态. 图 5b 状态 4 的 a 输入边展现了这种情况.

显然, 可以将步骤 2 与步骤 3 中的 2 种例外的处理作为步骤 4. 所以, 实施中可以将不考虑例外情况的步骤 1、2、3 作为基本过程, 然后用步骤 4 进行修正.

4 基于状态图的直接变换法

基于状态图的描述方法比较直观. 但从计算机处理的角度来说, 基于状态表的规则描述更容易进行转化软件的设计.

与图 5a 等价的用状态转移表描述的 NFA M2 如表 1 所示.

基于 NFA 状态表的直接转化法的一般步骤与规则如下, 其中所有的示例都取自表 1.

步骤 1: 删去基态行中所有次态为子集的子集中的基态本身. 例如 0 行中的子集型次态中的状态 0. 然后将更改后的表 1 复制为表 2. 在以后的步骤中, 表 1

总是保持不变, 作为计算的依据, 所得的结果总是存入表 2.

表 1　NFA M2 状态转移表

Sc \ I / Sx	a	b	c
0	{0,1}	{0,7}	0
1		2	
2	3		
3		4	
4		5	
5			6
6	6	6	6
7		8	
8	9		
9			6

I：输入
Sx：次态
Sc：现态

表 2　DFA M2 状态转移表

Sc \ I / Sx	a	b	c
0	1	7	0
1	*1*	2	*0*
2	3	*0*	*0*
3	*1*	4	*0*
4	*3*	5	*0*
5	*1*	*0*	6
6	6	6	6
7	*0*	8	*0*
8	9	*8*	*0*
9	*0*	*8*	6

I：输入
Sx：次态
Sc：现态

步骤 2: 从基态行的各启动态出发, 记录下每一个分支输入序列. 例如, 从 a 列出发的分支输入序列为 ababbc. 然后找出该序列中与起始子串 I_0 相同的子串 I_2, 此例中为后一个 ab, 将紧跟 I_0 的下一个输入 x 的次态复制到 I_2 到达的下一行的 x 列中. 本例中, I_0 “ab” 的下一个字符是 a, 次态是 3, 而输入序列 abab 后到达 4 行, 所以 4 行中 a 列的次态为 3(结果, 用斜体表示, 下同).

步骤 3: 从基态行的各启动态出发, 一直行进到所引到的次态行的非空次态位于非出发列停止. 将该停止行及继续行进直到终态前的那些行的位于出发列的次态均置为停止行对应的现态. 但已有的非空态不变. 例如, 从 0 行 a 列出发, 停止行为 1, 继续行进直到终态前的行包括 2、3、4、5, 所以 1 至 5 行的 a 列均应置为 1, 但 2、4 行 a 列已有的值 3, 所以实际上只有 1、3、5 行的 a 列置为 1. 而从 0 行 b 列出发的行进路线, 停止行为 8, 继续行进直到终态前的行只有 9, 所以只有 8、9 两行的 b 列置为 8.

步骤 4: 将表 2 中除终态行外的所有空次态均改为基态.

按照以上步骤对表 1 进行变换, 所得结果如表 2 所示, 易见其与图 5b 完全对应.

5　转化算法

根据上一节描述的步骤与规则, 可以给出下列类 C++语言的算法描述. 其中, 变量 s 是存储每一前向分支的数组, 每一数组元素 st 表示一节点, st.in 是输入(列号), st.sx 是状态编号(行号). 将 s 中所有元素的 sx 依次串联起来即是一前向分支状态序列, 而将 in 依次串联起来即是该分支的分支输入序列.

```
输入: NFA 状态表 t1;
输出: DFA 状态表 t2;
NFA2DFA( ) //主函数, 为描述简洁而略去参数
{
DeleteBaseStateInSubset( t1);//步骤 1 之 1
Copy(t2,t1); //步骤 1 之 2
nI = t1.nInput; //获取输入字符个数
for ( j = 0; j <nI ; j++)
  if ( t1.Sx[0][j]) //基态行 j 列次态不为 0
  {
    //初始化带有次态信息的输入序列 s
    st.in = t1.in[j], st.sx = t1.Sx[0][j];
    Inits(s,st );
    //调用对应步骤 2、3 的递归函数
    Route(GetRow(t1.Sx[0][j]), s , col=j );
    }
  FillWithBaseState(); //步骤 4
}
```

上述算法中, 最核心最复杂的是对应步骤 2、3 的递归函数 Route(row, s).. 该函数以上游传来的部分序列 s 及该序列所到达的行 row 为参数, 继续步骤 2 所描述的输入路径的搜索与处理, 由于步骤 3 的处理过程有许多与步骤 2 相似, 故一并进行, 相应地, 增加一个参数 col, 表示本路径的启动输入是哪一个.

下面给出 Route()函数的描述.

```
Route(row, s , col )
{
  sr = s ; //保留 s 副本, 以处理分支
```

```
n = NumOfInput( row,&p); //获取行输入数据
switch( n ){
  case 0:   //已到终态行
    Treat( s , col); //进行步骤 2、3 的处理
    Break;
  case 1:   //无分支
    GetaNode(st, row, p[0]); //获取本行输入
    Appends( s, st );//添加到 s 末尾
    Route(st.sx, s , col); //处理次态行
    Break;
  default:   //有分支
    while( p ) //p 数组中有所有输入的列号
    {
      GetaNode(st, row, *p); //Get a node
      s =sr;
      Appends( s, st );
      Route(st.sx, s , col);
      p++;
    }
    Break;
}
```

不难看出，上述算法中，case 1 是可以合并到 default 中去的. 分开来写，只是为了读起来更清晰.

6 结语

本文描述的算法虽然只是针对只具有一个位于最开始位置的基态的 NFA，且前向分支中不存在回路，但也可推广到其他情况. 首先，对于含有多个基态的 NFA，如果各基态的输入集相同，显然只要分段转化即可；而对于以系统的输入集合的子集为输入集的基态构成的 NFA 分段，则应先转化该分段. 其次，前向分支上一条回路的存在只是增加了一个分支，仅仅影响到检查是否存在与起始子串 I_0 相同的子串 I_2. 不过应当注意，这几种情况对算法的细节还是有比较大的影响.

参考文献

1 张伟,薛一波,嵩天.一种支持多正则表达式匹配的硬件结构.清华大学学报,2009,49(10):132−135.

2 周经野,张继福.编译原理.武汉:武汉理工大学出版社,2003.

3 毛红梅,聂承启.一种将 NFA 到最小化 DFA 的方法.计算机与现代化,2004,10:6−7.

4 周启海.NFA→FA→GFA 自动机转换算法.电子科技大学学报,2005.3:363−365.

5 孙玉强,刘三阳,王明斐,等.有限状态自动机的并行确定化及过程分析.计算机科学,2006,10:293−295.

6 van Noord G. Treatment of Epsilon Moves in Subset Construction. Computational Linguistics, 2000,1: 61−76.

7 任平红,陈矗,曹宝香,禹继国.基于子集构造法的优化的 NFA 确定化算法.计算机技术与发展,2011,1:70−73.

（上接第 90 页）

IMS:移动领域的 IP 多媒体概念和服务.北京:机械工业出版社,2005.

2 Open IMS Core Project. http://openimscore.org.

3 Inexbee:MecuroIMSClient.http://www.mercuro.net

4 PT.Inovacao.“IMSCommunicator”. http://imscommunicator.berlios.de.

5 UniversityofCapeTown. UCTIMSClient. http://uctimsclient.berlios.de.

6 “Doubango Project”.http://www.doubango.org.

7 “Boghe”.http://code.google.com/p/boghe/.

8 “IMSDroid”.http://code.google.com/p/imsdroid/.

9 Rosenberg J. Schulzrinne H. “SIP:Session Initial Protocol”, RFC 3261, IETF. June 2002.