有限状态自动机的并行确定化及过程分析*)

孙玉强[1,2] 刘三阳[2] 王明斐[1] 邹 凌[1]

(江苏工业学院计算机系 常州 213016)[1] (西安电子科技大学 西安 710071)[2]

摘 要 本文通过对并行环境下非确定有限自动机和确定有限自动机的等价性和转换进行研究,详细分析了非确定有限自动机到确定有限自动机的并行转换方法及算法,并以实例给出了其间并行转化的过程。

关键词 并行转换,非确定有限自动机,确定有限自动机

Parallel Conversion of NFA to DFA

SUN Yu-Qiang[1,2] LIU San-Yang[2] WANG Ming-Fei[1] ZOU Ling[1]

(Department of Computer, Jiangsu Polytechnic University, Changzhou213016)[1]

(School of Science, Xidian University, Xi'an710071)[2]

Abstract This paper through to research the equivalence and conversion of NFA and DFA in parallel environment, labor the method of NFA convert to DFA, and give a example of the process of parallel conversion.

Keywords parallel conversion, NFA, DFA

1 引言

自动机理论是编译程序设计的基础。在传统意义上的正规文法、正规式和自动机之间的等价关系及实现算法有较系统的描述[1],文法的并行识别与词语法分析是计算机软件理论界关注的研究方向[2~5],在其基本的并行结构和算法基础[6]之上,本文通过对并行环境下非确定有限自动机到确定有限自动机的转换方法及算法的设计与分析,给出一种非确定有限自动机到确定有限自动机的并行转换方法及算法,并以实例模拟其间并行转化的过程并加以验证。

2 基本概念与串行变换

有限自动机 FA(NFA 和 DFA)是具有有限个记忆的离散动态系统的形式模型,用于数字计算机、图形识别、信息和编码以及神经过程等的形式模型表示和研究,在编译原理的词法分析中,用于单词识别、生成过程的模型表示和实现。

非确定有限自动机(Non-Deterministic Finite Automata)简称为 NFA ,是一个五元组$(S, \Sigma, \delta, S_0, F)$,其中:$S$ 是有限状态集;Σ 是有穷输入字母表;δ 是从 $S\times\Sigma^* \to 2^S$ 上的映射;$S_0 \in S$ 是唯一的初始状态;F 是 S 的子集,它为终止状态集。

确定有限自动机(Deterministic Finite Automata)简称为 DFA ,是一个五元组$(S, \Sigma, \delta, S_0, F)$,其中:$S$ 是有限状态集;Σ 是有穷输入字母表;δ 是一个从 $S\times\Sigma$ 至 S 的单值部分映射;$S_0 \in S$ 是唯一的初始状态;F 是 S 的子集,它为终止状态集。

对于 Σ^* 中的任何一个字 x,若存在从某一初态结点到每一终态结点的通路,且这条通路上所有弧的标记字依序连接成的字等于 x,则称 x 可为 NFA M/DFA M 所识别。若 M 的初态结点同时又是终态结点,则空字 ε 可为 M 所识别。NFA M/DFA M 所能识别的字的全体记为 $L(M)$。

DFA 是 NFA 的特例。但是,对于每个 NFA M 存在一个 DFA M',使 $L(M)=L(M')$。因此对于 NFA M 可以将其转化为 DFA M。

证明过程如下:

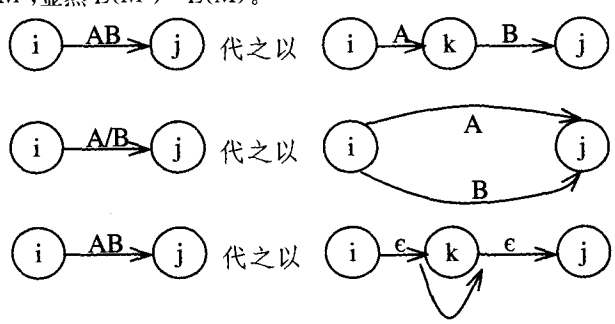
(1)假定 NFA $M=(S, \Sigma, \delta, S_0, F)$,我们对 M 的状态转换图进行以下改造:

①引进新的初态结点 X 和终态结点 $Y, X, Y \notin S$,

从 X 到 S_0 中任意状态结点连一条 ε 箭弧,从 F 中任意状态结点连一条 ε 箭弧到 Y。

②对 M 的状态转换图进一步施行下图所示的替换,其中 k 是新引入的状态。

重复这种分裂过程直至状态转换图中的每条箭弧上的标记或为 ε,或为 Σ 中的单个字母。将最终得到的 NFA 记为 M',显然 $L(M')=L(M)$。



(2)将 M' 进一步变换为 DFA,方法如下:

①假定 I 是 M' 的状态集的子集,定义 I 的 ε 闭包 ε-CLOSURE(I)为:

(a)若 $q\in I$,则 $q\in\varepsilon$-CLOSURE(I);

(b)若 $q\in I$,那么从 q 出发经任意条 ε 弧而能到达的任何状态 q' 都属于 ε-CLOSURE(I);

②假定 I 是 M' 的状态集的子集,$a\in\Sigma$,定义 $Ia=\varepsilon$-CLOSURE(J),其中,J 是那些可以从 I 中的某一状态结点出发经一条 a 弧而到达的状态结点的全体。

③假定 $\Sigma=\{a_1, \cdots, a_k\}$。我们构造一张表,此表的每一行含有 $k+1$ 列。置该表的首行首列为 ε-CLOSURE(X)。一般而言,如果某一行的第一列的状态子集已经确定,例如记为 I,那么,置该行的 $I+1$ 列为 $Ia_i (i=1, \cdots\cdots, k)$。然后,检查该行上的所有状态子集,看它们是否已在标的第一列中出现,将未曾出现这填入到后面空行的第一列。重复上述过程,

*)受河南省自然科学基金(0211021600 和 0324220079)资助。孙玉强 教授,博士生;刘三阳 教授,博导;邹 凌 博士;王明斐 硕士生。

直至出现在第 $I+1$ 列($i=1,\cdots\cdots,k$)上的所有状态子集均已在第一列上出现。因为 M' 的状态子集的个数是有限的，所以上述过程必定在有限步内终止。

我们将构造出来的表视为状态转换表，将表中的每个状态子集视为新的状态。显然，该表唯一的刻画了一个 DFA M'。它的初态是该表首行首列的那个状态，终态是那些含有远终态的状态子集。根据上述构造方法，不难得出：$L(M')=L(M')=L(M)$。

3 并行确定化算法设计

(1)对一个表达式 M 分析，得到其 NFA 图及其 NFA 的表列表示法。

NFA 的表列表示法构造方法如下：

Procedure gen(s(初态),sym(输入),$s1$(次态 1),$s2$(次态 2))

Begin

①if sym 是个字母 then symbol(s)←sym end if

②next 1(s)←$s1$

③if $s2$ 是定义了的 then next 2(s)←$s2$ end if

end

可产生如下列表：

State symbol next1 next2

1 a 3 2

.....

由 NFA 的表列表示法构造如下函数：

for all state par-do

构造 $W(x, y)=s$；其中 x 为 state 列元素，y 为该 state 状态对应的 next1/next2 的值，若 next1，next2 同时为空，则构造 $W(state,Y)=E$；即表示该 state 为终结状态。S 为该 state 状态对应的 symbol 值，用 θ 表示空值，并将 symbol 值为空的函数置于集合 A 中，其它的函数置于集合 B 中。

End for

(2)For 集合 B 中所有函数 par-do

对于每一个集合 B 中函数，在集合 A 中寻找所有 $\mathbf{Y}_A=\mathbf{X}_B$ 的函数 $W(\mathbf{X}_A,\mathbf{Y}_A)=\theta$

(其中，$\mathbf{Y}_A$ 表示集合 A 中的函数 $W(x,y)$ 的 y 值，$\mathbf{X}_B$ 表示集合 B 中的函数 $W(x,y)$ 的 x 值。s_B 表示集合 B 中的函数 $W(x,y)=s$ 中的 s 值。)

合并两函数得 $W(\mathbf{X}_A,\mathbf{Y}_B)=s_B$，并将其置于一新的集合 B' 中。

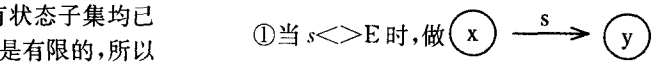
End for

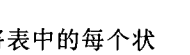
If $B<>B'$ then $B=B'$，将 B' 清空 ，goto (2)

Else 得到最终的集合 B。

按照如下规则画出其 DFA 图：

对函数 $W(x,y)=s$，

①当 $s<>$E 时，做 

②当 $s<>$E 时，做 

，(即 x 状态为终结状态)

(3)化为最简 DFA：

对求得的终态集合 B 中的函数分析：在集合 B 中，当有 $W(x_1,y)=s$，可找到 x_2，使得满足 $W(x_1,y)=s$ 的函数都有 $W(x_2,y)=s$，则所有的 $W(x_2,y)=s$ 可约去。

并行算法如下：按照 x 值的不同，用 k 个处理器将集合 B 中的函数分为 k 组，每组中函数按 y 值从小到大排序(Y 视为无穷大)。

即{$W(x_1,y_{11})=s_{11}$，$W(x_1,y_{12})=s_{12}$，……，$W(x_1,y_{1m})=s_{1m}$，}

{$W(x_2,y_{21})=s_{k1}$，$W(x_2,y_{22})=s_{22}$，……，$W(x_2,y_{2m})=s_{2m}$，}

.....

{$W(x_k,y_{k1})=s_{k1}$，$W(x_k,y_{k2})=s_{k2}$，……，$W(x_k,y_{km})=s_{km}$，}

逐个对函数个数相同的两组函数比较。

{$k=0$；

For 该组中每个函数 par-do //* * 对第 i,j 两组函数比较时

If($y_{ir}==y_{jr}$)∧($s_{ir}==s_{jr}$)=false then $k=1$；

End for

If $k=0$ then 约去第 j 组函数}

对集合中剩余函数按照②中所述绘制 DFA 图的方法绘出其最简 DFA 图。

4 实例分析

例如：表达式 $a^*\cdot b\cdot(a|c)^*$ 的 NFA 图和 NFA 的表列表示法如图 1 和表 1。

表 1 NFA 的列表

State	symbol	nextl	next2	
1			3	2
2			5	
3		a	4	
4			2	3
5		b	6	
6			7	
7			9	8
8				
9			11	13
10			8	9
11		a	12	
12			10	
13		c	14	
14			10	

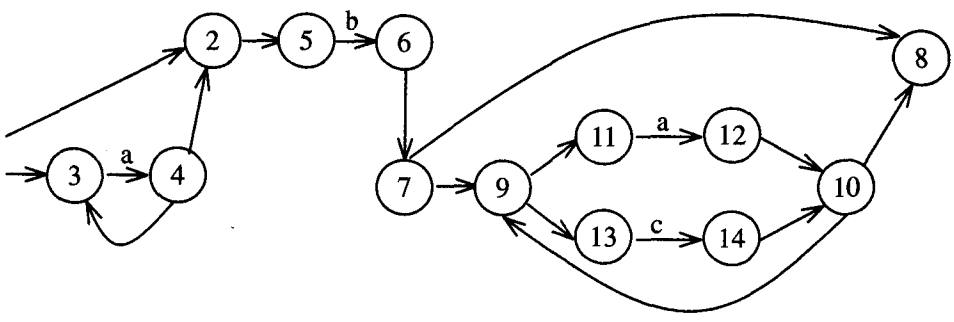


图 1 表达式 $a^*\cdot b\cdot(a|c)^*$ 的 NFA 图

上例经过上述算法(1)由表1可构造如下集合(见表2和表3)：

表2 集合A

W(1,3)=θ	W(7,8)=θ
W(1,2)=θ	W(9,11)=θ
W(2,5)=θ	W(9,13)=θ
W(4,2)=θ	W(10,8)=θ
W(4,3)=θ	W(10,9)=θ
W(6,7)=θ	W(12,10)=θ
W(7,9)=θ	W(14,10)=θ

表3 集合B

W(8,Y)=E
W(3,4)=a
W(5,6)=b
W(11,12)=a
W(13,14)=c

上述结果经过上述算法(2)可得到终态集合C(表4)和DFA图，如图2。

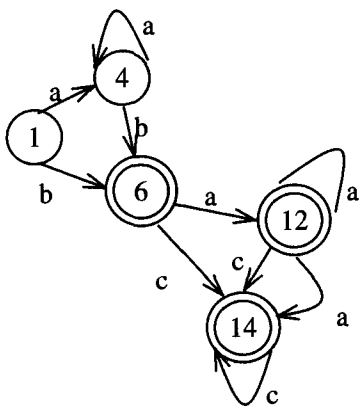


图2 DFA图

表4 集合C

W(1,4)=a	W(12,12)=a
W(1,6)=b	W(12,14)=c
W(4,4)=a	W(12,Y)=E
W(4,6)=b	W(14,12)=a
W(6,Y)=E	W(14,14)=c
W(6,12)=a	W(14,Y)=E
W(6,15)=c	

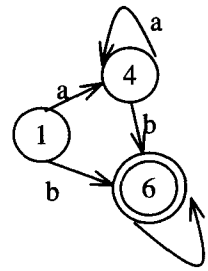


图3 最简DFA图

经过上述算法(3)可得到表达式 $a^* \cdot b \cdot (a|c)^*$ 的最简DFA图，如图3。

结束语 有限状态自动机的并行确定化算法的设计及过程分析有其理论和实践意义，其算法思想及表列中间存储结构也可用于类似的并行转换。在并行环境中，多机的并行结构模型和中间存储结构都直接影响算法的运行结果。此外，并行确定化转换算法并未被证明是最优的，这方面还有许多工作可做，优化和存储结构的改进以及与不同并行结构模型的匹配等需要进一步的研究与讨论。

参 考 文 献

1 陈火旺，刘春林，谭庆平，等. 程序设计编译原理. 北京：国防工业出版社，2003

2 Gibbons A, Rytter W. Efficient parallel algorithms. Cambridge University Press ,1990

3 Ra Dong-Yul, Kim Jong-Hyun. A parallel parsing algorithm for arbitrary contexe-free grammars. Information Processing Letters, 1996,58:87～96

4 Matsumoto Y. A parallel parsing system for natural language analysis. New Generation Comput,1987,15:63～78

5 Wyard P J, Ninghtingale C. A single layer higher order neural net and its application to context-free grammar recognition. In: Sharkey N, ed

6 陈国良. 并行计算—结构、算法、编程. 北京：高等教育出版社，1999

(上接第292页)

```
    ……
    JNI_GetDefaultJavaVMInitArgs(&vm_args);
        sprintf (classpath, "%s%c%s",
                 vm_args. classpath, PATH_SEPARATOR,
USER_CLASSPATH);
    vm_args. classpath = classpath;
    /* Create the Java VM */
    res= JNI_CreateJavaVM(&jvm,&env,&vm_args);
    if (res < 0) {
        fprintf(stderr, "Can't create Java VM\n");
        exit(1);
    }
     cls = (*env)->FindClass(env, "Prog");
    ……
     (*env)->CallStaticVoidMethod(env, cls, mid, args);
     (*jvm)->DestroyJavaVM(jvm);
}
//Prog. Java
public class Prog {
    public static void main(String [] args) {
        System. out. println("Hello World" + args [0]);
}
}
```

invoke. c代码开始调用JNI_GetDefaultJavaVMInitArgs获得初始化设置，接着调用JNI_CreateJavaVM加载和初始化虚拟机。在JNI_CreateJavaVM成功返回之后，当前native线程已将自身捆绑在Java虚拟机上，但无结果返回JVM。

结束语 本文围绕在Java中调用native方法，native方法与Java的整合，在native方法中调用Java虚拟机等方面阐述了技术实现方案，使Java通过JNI调用本地的库文件的内部方法，实现和本地机器的紧密联系，调用系统级的各接口。

Java JNI技术的不断完善给基于Java技术的应用系统开发带来了便利，随着Java软件技术的不断提高，“一次编写，多处可用”的理念将会发挥巨大作用。

参 考 文 献

1 飞思科技产品研发中心. JavaWeb服务应用开发指南. 电子工业出版社，2002

2 靳慧峰，等. 新概念JSP网络应用. 北京科海集团公司，2001

3 王立冬，张凯. Java虚拟机分析. 北京理工大学学报，2002(8)

4 陈传波，蒋湘. 网络管理中JAVA技术应用的探讨. 湖北工学院学报，2001(3)