DFA 算法 1 详细设计

朱磊 **2009212618**

王建 **2009212581**

目录

1. 开发环境

- 操作系统：Microsoft Windows XP Professional Service Pack 3
- 内存：1.96GB
- CPU：Pentium(R) Dual-Core CPU E5300 2.60GHz
- 编程语言：JAVA
- JDK 版本：JDK 1.6
- IDE：Eclipse 3.2

2. 项目要求

- 要求： 用Java 编写，保证可扩展性
- 判断一个自动机是不是DFA？是不是NFA？
- 实现自动机的交并补操作。
- 实现自动机最小化的算法。

3. 类设计

类名（主要）	描述
MyState	状态，每个状态都有状态名，以 ID 保证每个状态的唯一
MyAutomaton	包含状态，转移函数，输入字符，初始状态，终止状态，实现自动机的判断，DFA 的交并补运算以及自动机最小化算法，同时实现了去除不可到达状态，以及不可接受状态的算法。
Transition	包含一个状态在一个转移条件下能够到达的所有状态
StateTransition	状态以及由此状态出发的所有状态转移函数
DFATOOL1	包装所有的算法，提供使用算法的接口
TestUI	用于算法测试的界面
Error	错误信息
其他类	存储临时数据结构和提供算法需要的子过程

4. 功能描述

4.1DFA 和 NFA 的判断

判定标准：

- 存在转移条件为ε的状态转移函数则自动机为 NFA；
- 存在状态转移在相同的状态和转移条件下转移到不同的状态为 NFA；
- 存在某个状态在某个输入字符下不存在状态转移，则为 NFA；
- 如果不存在上述任何一种情况则为 DFA。

算法的伪代码如下：

输入：状态集合 Q　　转移函数集合δ

输出：是 DFA 返回 true,是 NFA 返回 false。

```
DFAORNFA(Q, δ)
  for each state P∈Q
      do Mack a set for P
  for each transition f(P,a)=T∈δ
      do S←find the set of P
          if a∈S or a=ε
             then return false
             else insert a into S
  for each set S in Sets
      if size[S]!=size[input_symbols]
         then return false
  return true
```

4.2DFA 并运算

对于两个 DFA 进行并运算，输入字符为两个自动机输入字符的并集。增加一个初始状态，增加它到两个 DFA 初始状态的转移函数，转移条件为ε,增加一个终止状态，增加转移函数两个 DFA 的所有终止状态到此状态的转移，转移条件为ε。

将两个 DFA 的初始状态和终止状态标记为普通状态。如果输入存在 NFA 则应调用 NFA 转 DFA 算法，再使用本算法，同时算法返回的是一个 NFA。

算法的伪代码描述：

输入：DFA_1 和 DFA_2，状态集合 Q_1,Q_2,字母表Σ_1,Σ_2,状态转移函数集合δ_1，δ_2，终止状态集合 F_1，F_2

DFAUNION

p←DFA_1 的开始状态

q←DFA_2 的开始状态

Q← $Q_1 \cup Q_2 \cup \{s,e\}$

$\Sigma \leftarrow \Sigma_1 \cup \Sigma_2$

$\delta \leftarrow \delta_1 \cup \delta_2 \cup \{f(s,\varepsilon)=p\} \cup \{f(s,\varepsilon)=p\}$

F←{e}

for each t in $F_1 \cup F_2$

do $\delta \leftarrow \delta \cup \{f(t,\varepsilon)=e\}$

return (Q,Σ,δ,s,F)

4.3DFA 交运算

对于两个 DFA 进行交运算，输入字符为两个自动机输入字符的交集。使用乘法构造法计算两个 DFA 的交。

算法伪代码描述：

输入：状态集合 Q_1,Q_2,字母表Σ_1,Σ_2,状态转移函数集合δ_1，δ_2

DFAINTERSECTION

for each $(p,q) \in Q_1 \times Q_2$

do for each $a \in \Sigma_1 \cap \Sigma_2$

do $p' \leftarrow f(p,a)$

$q' \leftarrow f(q,a)$

if $p' \neq \varepsilon$ and $q' \neq \varepsilon$

then insert $f(pq,a)= p' q'$ into δ^*

4.4DFA 补运算

将 DFA 中的接受状态修改为非接受状态，非接受状态修改为接受状态。

算法伪代码描述：

输入：给定的 DFA(Q,Σ,δ,p,F)

```
DFACOM
  if 需要补充陷阱状态
    then 添加陷阱 q 状态以及转移函数
         Q←Q∪{q}
  F*←Q-F
  return (Q,Σ,δ,p,F*)
```

4.6DFA 最小化算法

使用填表法进行最小化，通过虚拟一个失败状态，扩展算法可以对缺少某些转移的 DFA 进行最小化。

算法的伪代码描述为：

输入：给定的 DFA

输出：可区分状态表

```
MINDFA()
  for each (q,p)∈F×(Q-F)
     do 标记可区分状态表中的表项(q,p)
  for each (q,p)∈F×F∪(Q-F)×(Q-F) and q≠p
     do if a∈Σ 表项(f(q,a),f(p,a))已被标记
          then 标记表项(q,p)
               递归标记本次被标记的状态对的关联链表上的各个状态对对应
               的表项
          else for each a∈Σ
                 do if f(q,a)≠f(p,a) and (q,p)与(f(q,a),f(p,a))不是同一状态对
                    then 将(q,p)放在(f(q,a),f(p,a))的关联链表上
```

使用可区分状态表构造最小化 DFA

5. 新增功能

5.1 删除 DFA 中的不可到达状态

DFA 中可能存在某些状态，由初始状态始终无法到达该状态，因此该状态可以删除。通过将 DFA 抽象为有向图，由初始状态开始进行广度优先搜索，标记可达的状态，最后删除所有未标记的状态，以及包含该状态的转移。

5.2 删除 DFA 中的不可接受状态

DFA 中可能存在某些状态，从这些状态出发无法到达接受状态，因此这些状态可以删除。将 DFA 抽象为有向图，对有向图转置，从每个接收状态出发做广度优先搜索，标记所有可到达的状态，最后删除所有未被标记的状态，以及包含这些状态的转移。

6. 测试结果显示

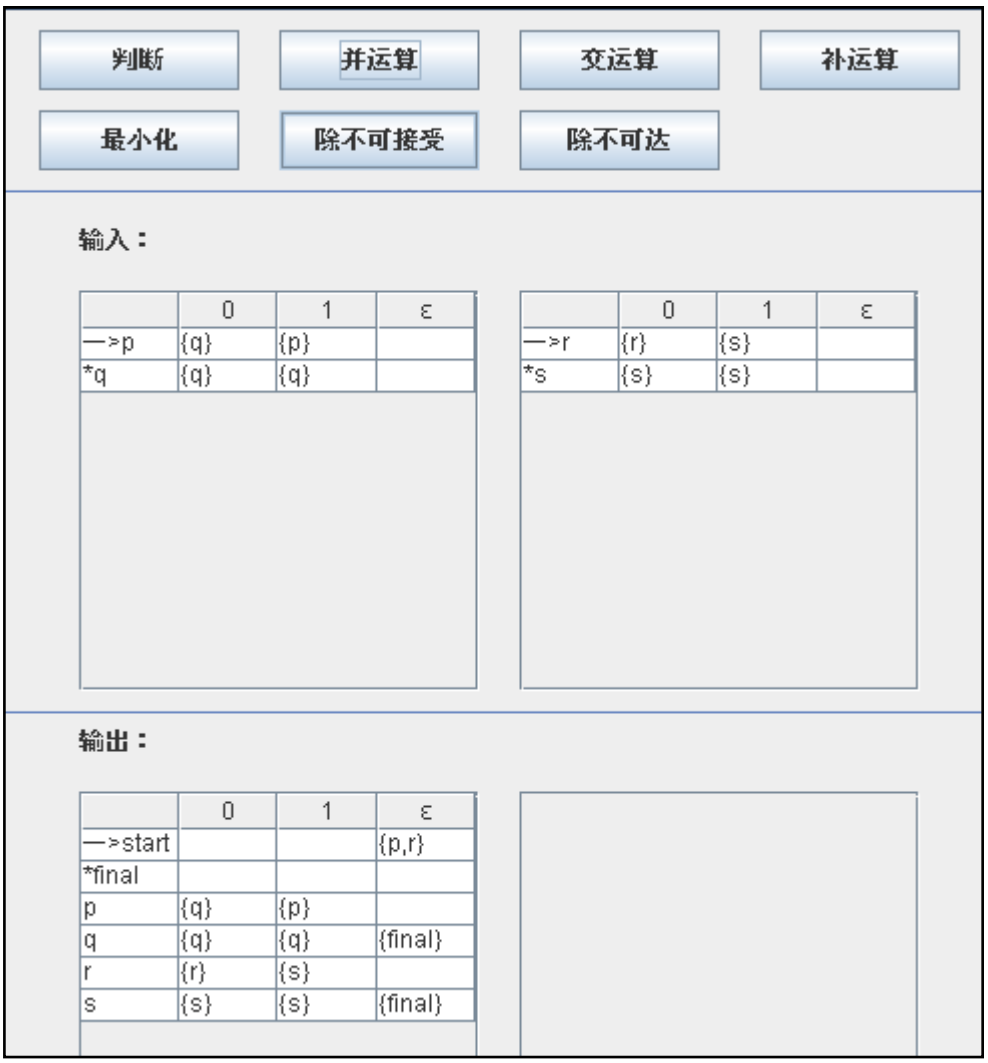
由于没有 DFA 输入方式，测试使用固定的 DFA。

6.1DFA， NFA 判断

	0	1	ε
—>A	{B}	{F}	
B	{G}	{C}	
*C	{A}	{C}	
D	{C}	{G}	
E	{E}	{F}	
F	{C}	{G}	
G	{G}	{E}	
H	{G}	{C}	

	0	1	ε
—>A	{B}	{F}	
B	{G}	{C}	
*C	{A}	{C}	
D	{C}	{G,H}	
E	{E}	{F}	
F	{C}	{G}	
G	{G}	{E}	
H	{G}	{C}	

6.2DFA 并运算

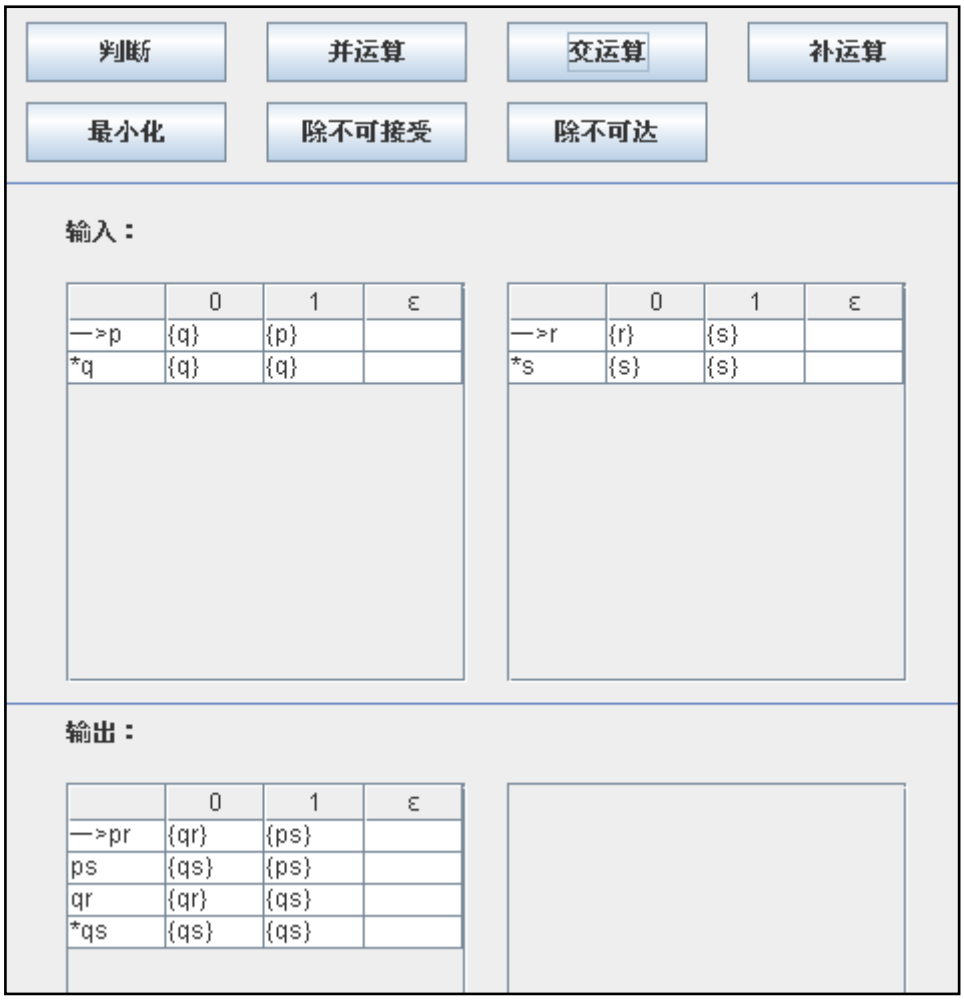


	0	1	ε
—>p	{q}	{p}	
*q	{q}	{q}	

	0	1	ε
—>r	{r}	{s}	
*s	{s}	{s}	

	0	1	ε
—>start			{p,r}
*final			
p	{q}	{p}	
q	{q}	{q}	{final}
r	{r}	{s}	
s	{s}	{s}	{final}

6.3DFA 交运算

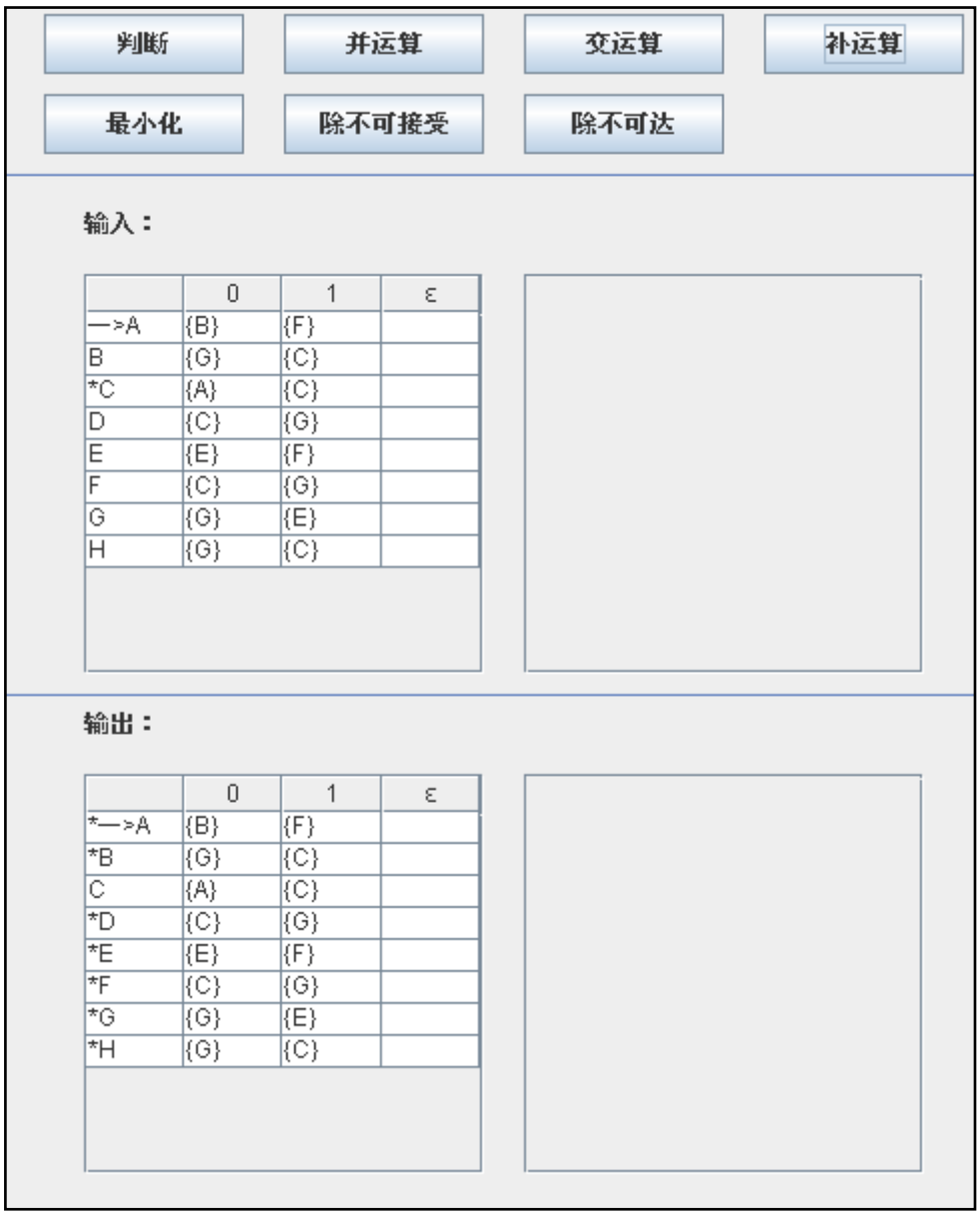


	0	1	ε
—>p	{q}	{p}	
*q	{q}	{q}	

	0	1	ε
—>r	{r}	{s}	
*s	{s}	{s}	

	0	1	ε
—>pr	{qr}	{ps}	
ps	{qs}	{ps}	
qr	{qr}	{qs}	
*qs	{qs}	{qs}	

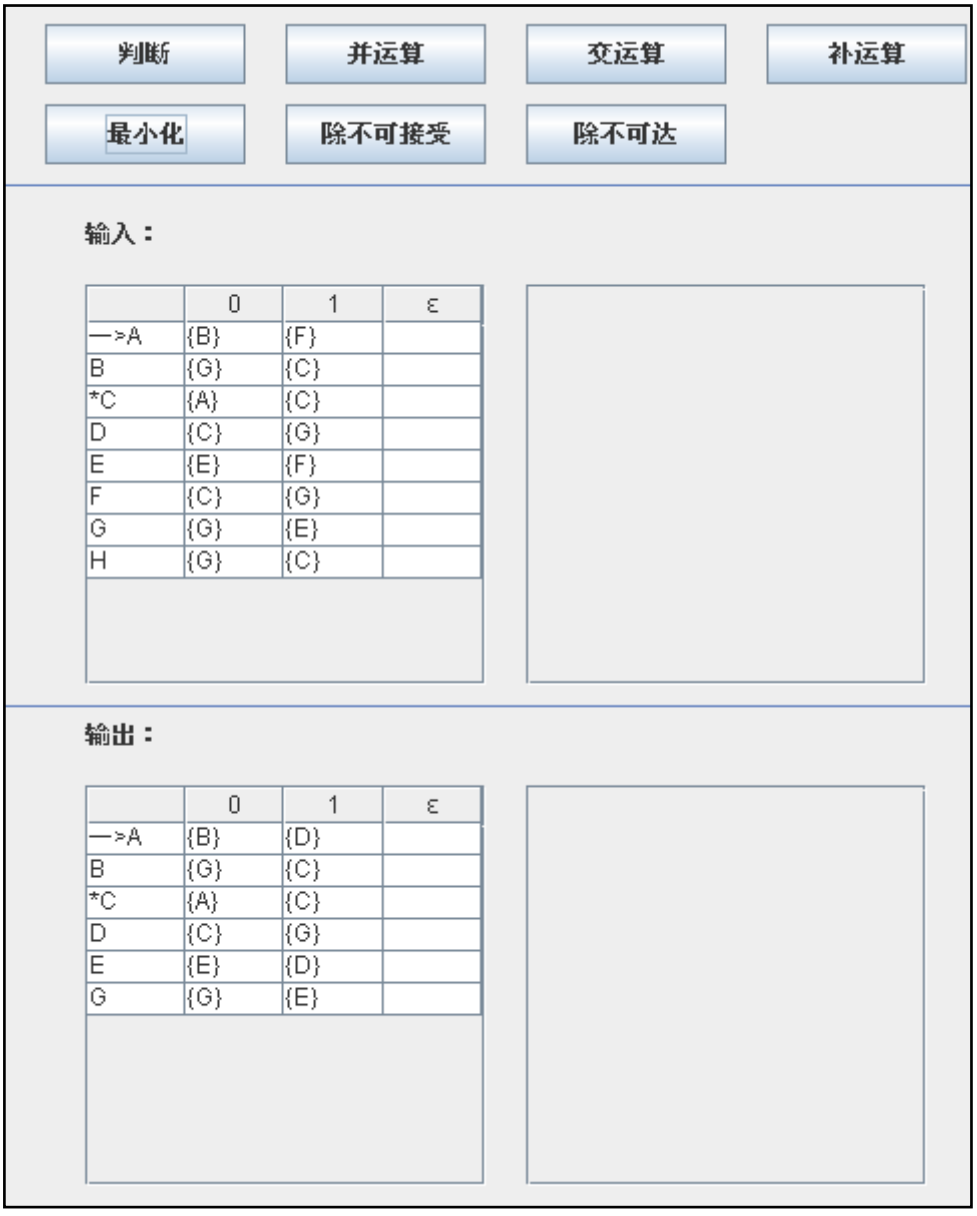
6.4DFA 补运算



	0	1	ε
—>A	{B}	{F}	
B	{G}	{C}	
*C	{A}	{C}	
D	{C}	{G}	
E	{E}	{F}	
F	{C}	{G}	
G	{G}	{E}	
H	{G}	{C}	

	0	1	ε
*—>A	{B}	{F}	
*B	{G}	{C}	
C	{A}	{C}	
*D	{C}	{G}	
*E	{E}	{F}	
*F	{C}	{G}	
*G	{G}	{E}	
*H	{G}	{C}	

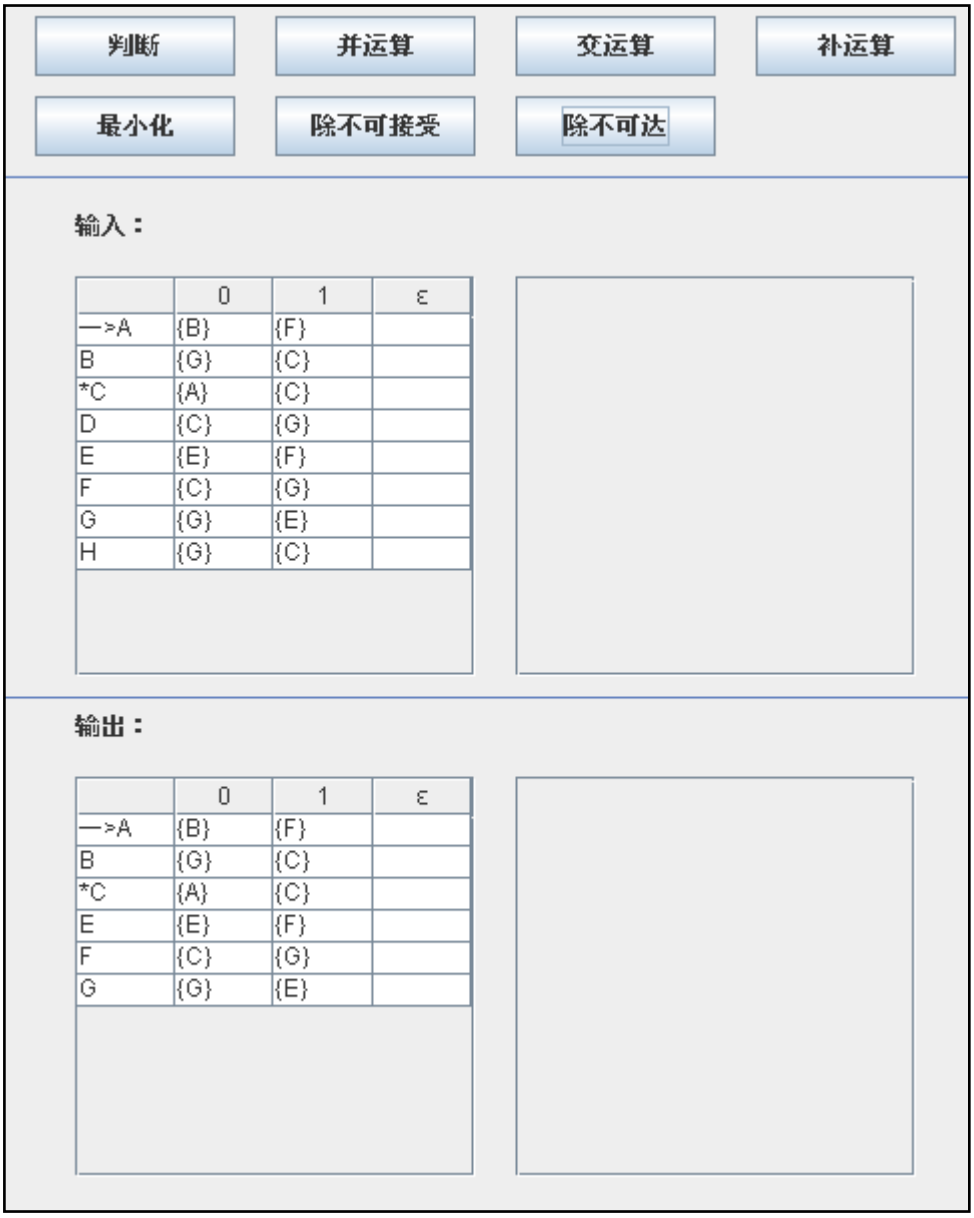
6.5DFA 最小化



	0	1	ε
—>A	{B}	{F}	
B	{G}	{C}	
*C	{A}	{C}	
D	{C}	{G}	
E	{E}	{F}	
F	{C}	{G}	
G	{G}	{E}	
H	{G}	{C}	

	0	1	ε
—>A	{B}	{D}	
B	{G}	{C}	
*C	{A}	{C}	
D	{C}	{G}	
E	{E}	{D}	
G	{G}	{E}	

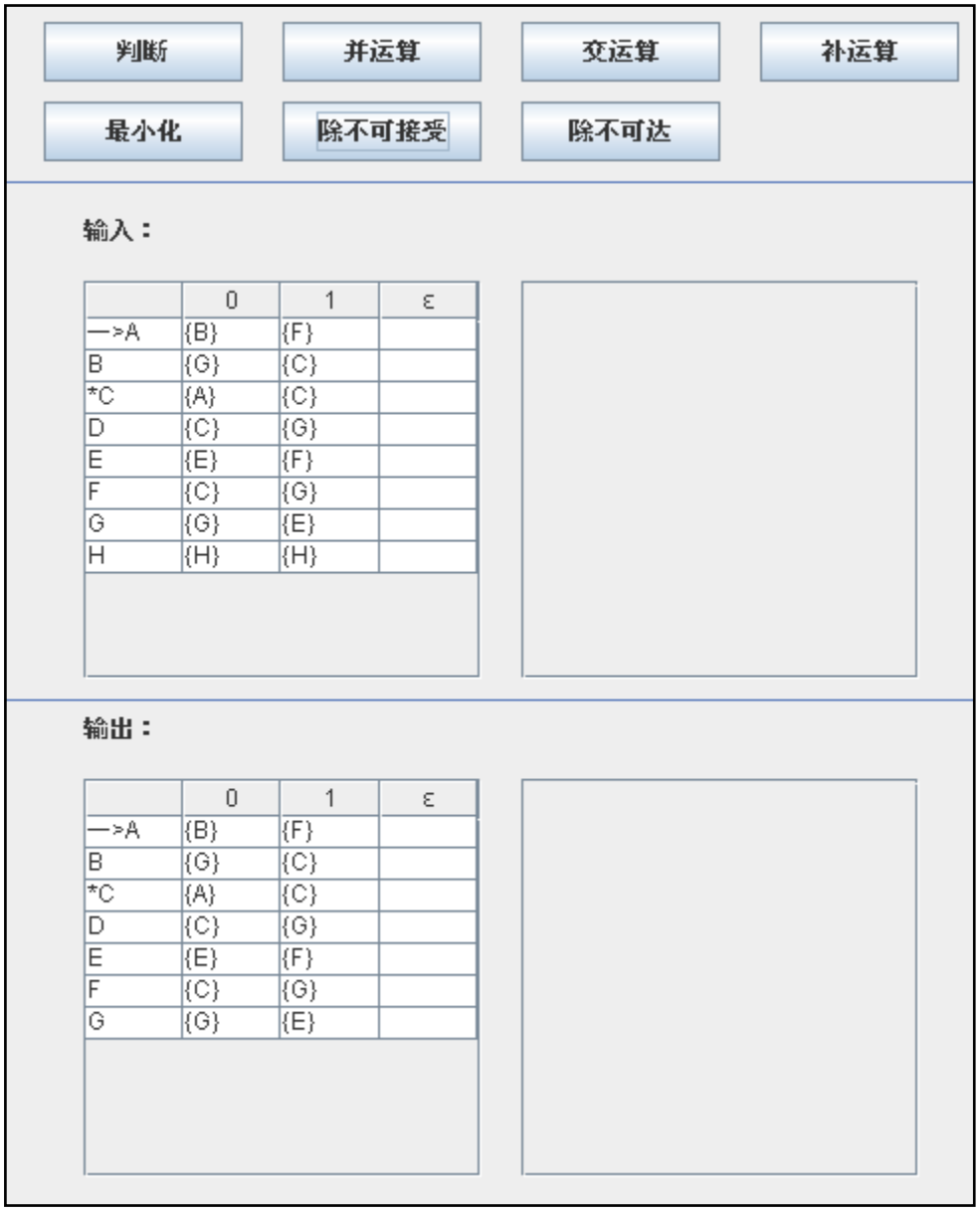
6.6 删除不可到达状态



	0	1	ε
—>A	{B}	{F}	
B	{G}	{C}	
*C	{A}	{C}	
D	{C}	{G}	
E	{E}	{F}	
F	{C}	{G}	
G	{G}	{E}	
H	{G}	{C}	

	0	1	ε
—>A	{B}	{F}	
B	{G}	{C}	
*C	{A}	{C}	
E	{E}	{F}	
F	{C}	{G}	
G	{G}	{E}	

6.7 删除不可接受状态



	0	1	ε
—>A	{B}	{F}	
B	{G}	{C}	
*C	{A}	{C}	
D	{C}	{G}	
E	{E}	{F}	
F	{C}	{G}	
G	{G}	{E}	
H	{H}	{H}	

	0	1	ε
—>A	{B}	{F}	
B	{G}	{C}	
*C	{A}	{C}	
D	{C}	{G}	
E	{E}	{F}	
F	{C}	{G}	
G	{G}	{E}	

7. 完成情况说明

完成基本算法：

- 判断 DFA 和 NFA
- DFA 的交并补运算
- DFA 的最小化算法

完成扩展算法：

- 删除 DFA 中的不可到达状态
- 删除 DFA 中的不可接受状态