班　级　1504012

学　号　15040120169

西安電子科技大學

# 本科毕业设计论文

**题　　目**　**有限自动机 C++ 工具箱**

**等价性和最小化类软件测试**

**学　　院**　机电工程学院

**专　　业**　机械设计制造及其自动化


**学生姓名**　胡双朴

**导师姓名**　段江涛

# 毕业设计（论文）诚信声明书

本人声明：本人所提交的毕业论文《有限自动机 C++ 工具箱等价性和最小化类软件测试》是本人在指导教师指导下独立研究、写作成果，论文中所引用他人的无论以何种方式发布的文字、研究成果，均在论文中加以说明；有关教师、同学和其他人员对本文本的写作、修订提出过并为我在论文中加以采纳的意见、建议，均已在我的致谢辞中加以说明并深致谢意。

本文和资料若有不实之处，本人承担一切相关责任。

论文作者：______________（签字）　时间：2019 年 06 月 01 日

指导教师已阅：__________（签字）　时间：2019 年 06 月 01 日

# 摘要

本测试样例，供 XDU*thesis* 模板使用者学习之用，会尽量将模板涉及到的命令、环境；以及一些常用的图表、公式排版技巧进行展示，会对参考文献进行进一步阐述。

其余不太重要的部分，会以其他内容进行展示，例如，诗歌、歌词等。

**关键词：XDUthesis　命令　环境　排版技巧**

# Abstract

This paper is just a sample example for the users in learning the XDU*thesis*. I will try my best to use the commands and environments which are involved by the XDU*thesis*. Also, the popular composition skills in figures, tables and equations will be elaborated.

In the part unimportant, I will show something others, such as poems and lyrics.

**Key words: XDUthesis commands environments skills**

# 目录

# 第一章　引言

自动机理论是许多科学的重要理论基础，从硬件电路的简化，到各式各样的编译器构造，到处都有着自动机理论的应用，而确定性有限自动机（Deterministic Finite Automata，DFA）的最小化是自动机理论的一个重要组成部分，研究确定性有限自动机的最小化对自动机理论的健全和发展有重要意义。

## 1.1　FIRE engine

FIRE engine[1] 是一个使用 C++ 语言实现有限自动机和正则表达式的类库。本文中也把“有限自动机 C++ 工具箱”称作“FIRE engine”。有限自动机 C++ 工具箱（FIRE engine）实现了论文[2,3] 中几乎所有的相关自动机理论算法，其中也包括了 Hopcroft、Ullman 等人提出的著名的确定性有限自动机最小化算法，验证这些算法的正确性和有效性对研究其他最小化算法很有帮助。

## 1.2　国内外研究现状

自动机理论的发展已经有很长时间，有限状态自动机、确定性有限状态自动机及相关的理论出现的时间都比较早。

# 第二章　预备知识

在进行本文的叙述之前，需要预先了解一些相关的知识和定义，以便于使用更加简洁的描述方式。

一般来说，我们使用状态转移图来表示一个自动机，如图2.1

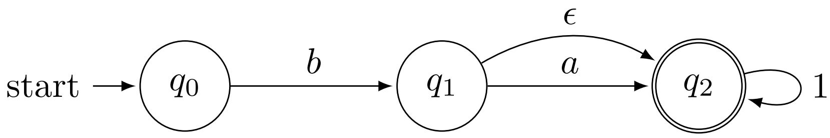


图 2.1　自动机状态转移图

图2.1中，自动机由 $q_0$ 进入，接收字符“b”进入状态 $q_1$，状态 $q_1$ 接收字符“a”进入状态 $q_2$，图中的两个同心圆圈住状态 $q_2$ 表示结束状态或者接受状态，意味自动机处理字符串到达此状态时，自动机就接受当前处理的字符串，状态 $q_2$ 上有一个指向自己的箭头，意为接收字符“1”之后，仍然指向自己。而状态 $q_1$ 经过 $\epsilon$ 转移到状态 $q_2$，意为状态 $q_1$ 可以不接收任何字符即可转移到状态 $q_2$。图2.1中的自动机接受的字符串为 $ba(1)^* \cup b(1)^*$。

**定义** 2.1： 有限自动机[3]：也称有限状态自动机（Finite automata，FA），有限自动机是一个 6 元组 $(Q, V, T, E, S, F)$，其中

- $Q$ 是有限状态集;
- $V$ 是一个字母表;
- $T \in \mathcal{P}(P \times V \times Q)$ 是一个转移关系;
- $E \in \mathcal{P}(Q \times Q)$ 是一个 $\epsilon$-转移关系（空转移，不需要接收字符即可转移）;
- $S \subseteq Q$ 是开始状态集;
- $F \subseteq Q$ 是结束状态集;

字母表和函数 $\mathcal{P}$ 的定义分别在“定义A.2”和“惯例A.1”。

**例 2.1：** 我们可以用 $M = (\{q_0, q_1, q_2\}, \{b, a, 1, \epsilon\}, T, E, \{q_0\}, \{q_2\})$ 来指代图2.1中的自动机。其中，$T = \{(q_0 \times b \times q_1), (q_1 \times a \times q_2), (q_2 \times 1 \times q_2)\}$，$E = (q_1 \times q_2)$。

为了在某些情况下方便的展示状态之间的关系，也把状态转移图写成表格[4]，如图2.1中的自动机，写成表格2.1：

表 2.1 状态转移函数

| 状态说明 | 状态 | 输入字符 | | | |
|---|---|---|---|---|---|
| | | $a$ | $b$ | $1$ | $\epsilon$ |
| 开始状态 (start) | $q_0$ | - | $q_1$ | - | - |
| | $q_1$ | $q_2$ | - | - | $q_2$ |
| 结束状态 (accept) | $q_2$ | - | - | $q_2$ | - |

下文中将表格2.1简称为转移函数。表格2.1的状态之间的关系与 $T = \{(q_0 \times b \times q_1), (q_1 \times a \times q_2), (q_2 \times 1 \times q_2)\}$，$E = (q_1 \times q_2)$ 一一对应。

**定义 2.2：** 确定性有限自动机：当且仅当

- 无多重初始状态;
- 无 $\epsilon$ 转移;
- 转移函数 $T \in Q \times V \longrightarrow \mathcal{P}(Q)$ 不将 $Q \times V$ 映射至多重状态。

时有限自动机 $M$ 是确定性的。公式形式表达为:

$$Det(M) \equiv (|S| \leq 1 \wedge \epsilon\text{-}free(E) \wedge (\forall q, a : q \in Q \wedge a \in V : |T(q, a)| \leq 1)) \tag{2-1}$$

下文中将确定性有限自动机简称为 $DFA$（Deterministic Finite Automata）。

# 第三章　等价关系

**这是一段黑体字**

rmfamily

`ttfamily 1234567890 ABCDEFGQWE`

普通仿宋字体

加粗仿宋字体

Xidian University

# 第四章　实例化类 DFA 对象

## 4.1　DFA 类

在本文中，我们仅仅对 $DFA$ 应用最小化算法，FIRE engine 中的 DFA 类为应用最小化算法的类。FIRE engine 提供了多种方式构造一个 $DFA$，查看类 $DFA$ 的实现，有如下内容 (文件 $DFA.h$)

```
class DFA : virtual public FAabs
{
public:
    // Default copy constructor, destructor, operator= are okay.
    inline DFA();

    // A special constructor used for subset construction etc.
    inline DFA(const DFA_components& r);
    ......
    StatePool Q;
    // S must be a singleton set, or empty. |S| <= 1
    StateSet S;
    StateSet F; // final states
    DTransRel T;
}
......
inline DFA::DFA(const DFA_components& r) :Q(r.Q), S(r.S), F(r.F), T(r.T)
{
    current = Invalid;
    assert(class_invariant());
}
......
```

代码 4.1　class DFA

由上面的代码4.1可知，类 $DFA$ 默认情况下，提供一个用“DFA_com-ponents”对象的引用来实例化 DFA 对象的方法。

除了使用“DFA_components”来实例化 DFA 对象之外，FIRE engine 还可以通过执行类 $LBFA$ 的成员函数 $LBFA :: determinism()$ 来将 $LBFA$ 对象转化成一个 $DFA$ 对象，该成员函数声明如代码4.2

```
class LBFA : virtual public FAabs
{
public:
……
    virtual DFA determinism() const;
……
}
```

代码 4.2 LBFA::determinism()

拥有同样功能的类还有类 $FA$、类 $RFA$ 和类 $RBFA$，这四个类与类 $DFA$ 都派生自类 $FAabs$。

类 $FAabs$ 及其子类关系如图4.1

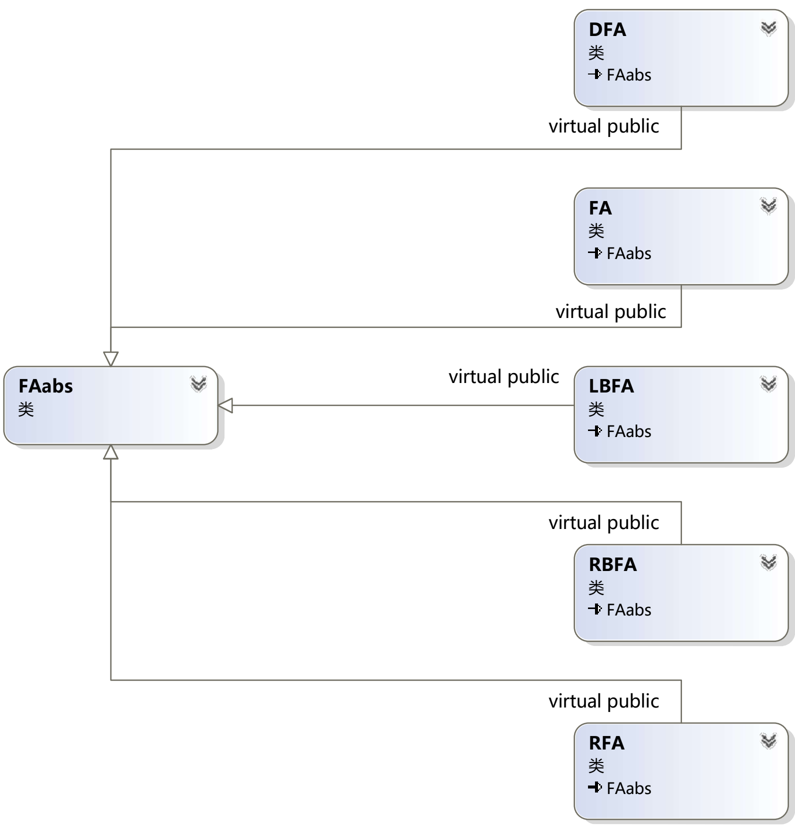


图 4.1 FAabs 及其派生类

## 4.2　DFA_components 结构体

从“DFA_components”构造一个 $DFA$ 对象是最简单直接的方法，观察“DFA_components”的实现，如下（文件 $DFA\_components.h$）：

```
#include "StatePool.h"
#include "StateSet.h"
#include "DTransRel.h"

struct DFA_components
{
    StatePool Q;
    StateSet S;
    DTransRel T;
    StateSet F;
};
```

代码 4.3　DFA_components

由代码4.3可知，结构体 $DFA\_components$ 内包含 $StatePool$ 变量 $Q$，$StateSet$ 变量 $S$，$DTransRel$ 变量 $T$，$StateSet$ 变量 $F$。若需要声明一个 $DFA\_components$ 变量，则需要分别实例化 $StatePool$、$StateSet$、$DTransRel$ 对象。这几个类还分别继承自其他的类，如图4.2

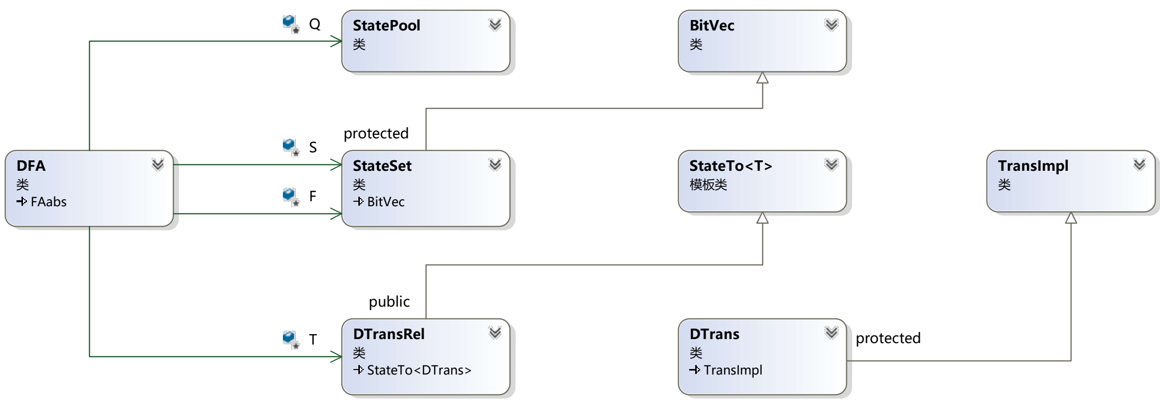


图 4.2　DFA 与其成员类及成员类的基类

类 $DFA$ 的成员变量“$T$”为一个 $DTransRel$ 对象，公有继承自模板类 $StateTo<T>$，模板参数“$T$”为保护继承自类 $TRansImpl$ 的类 $DTrans$。

### 4.2.1 StatePool 类

StatePool 类的部分实现如下（文件 $StatePool.h$）:

```
class StatePool
{
public:
    // How many states are already allocated (one more than that last allocated one,
    // since it begins at 0).
    inline int size() const;
    ......
    // Allocate a new state.
    inline State allocate();
    ......
private:
    // The next one to be allocated.
    int next;
};
......
inline int StatePool::size() const
{
    return(next);
}
......
inline State StatePool::allocate()
{
    return(next++);
}
```

代码 4.4 StatePool

如代码4.4所示，其中 $StatePool :: size()$ 和 $StatePool :: allocate()$ 为类 $StatePool$ 两个比较重要的函数，前者为自动机 $M$ 的大小，也即 $|M| = |Q|$。$StatePool :: allocate()$ 的作用稍后解释。

### 4.2.2 StateSet 类

类 $StateSet$ 的部分声明如下（文件 $StateSet.h$）:

```
class StateSet :protected BitVec
{
```

```
public:
……
    // inserts a State. r = [0,domain())
    inline StateSet& add(const State r);

    // set How many States can this set contain.
    // [O, r) can be contained in *this.
    inline void set_domain(const int r);
……
}
```

代码 4.5 StateSet

由于在实例化过程 $DFA$ 对象的过程中，只需要用到类 $StateSet$ 的成员函数 $StateSet::set\_domain()$ 和 $StateSet::add()$，所以这里仅仅列出这两个成员函数。类 $StateSet$ 中实现了集合的交、并、差、补、判断是否为空集等功能。类 $StateSet$ 的大部分功能建立在类 $BitVec$ 上，为了避免本文篇幅过于冗长，这里不再赘述类 $BitVec$ 的内容。

### 4.2.3 DTransRel 类

类 $DTransRel$ 是 $DFA$ 的一个重要成员，它存储了 $DFA$ 的状态转移关系，后面将会提到的等价类分割，等价状态合并等都与这个类息息相关。类 $DTransRel$ 的部分实现如下（文件 $DTransRel.h$）:

```
// Implement a deterministic transition relation, as a function from States time
// char to State. This is used for transition relations in DFA's.
class DTransRel :public StateTo<DTrans>
{
public:
……
    // Some functions updating *this:
    inline DTransRel& add_transition(const State p, const CharRange a, const State q);
……
    // Change the domain of this relation.
    inline void set_domain(const int r);
……
}
```

代码 4.6 DTransRel

在实例化类 $DFA$ 的过程中，类 $DTransRel$ 需要用到的成员函数只有 $DTransRel::set\_domain()$ 和 $DTransRel::add\_transition()$。

## 4.3 实例化类 DFA 对象

在文件 $DFA.cpp$ 中有如下实现：

```
inline int DFA::class_invariant() const
{
    return(Q.size() == S.domain()
        && Q.size() == F.domain()
        && Q.size() == T.domain()
        && current < Q.size()
        && S.size() <= 1);
}
```

代码 4.7 DFA.cpp

而文件 $DFA.h$ 中的构造函数如下：

```
inline DFA::DFA(const DFA_components& r) :Q(r.Q), S(r.S), F(r.F), T(r.T)
{
    current = Invalid;
    assert(class_invariant());
}
```

代码 4.8 DFA.h

由代码4.7和代码4.8可知，在语句“assert(class_invariant());”处，若函数“class_invariant()”返回值为“false”，那么程序将在此处中止[5]。由此可以知道，类 $DFA$ 要求其成员变量满足

$$(Q.size() \equiv S.domain() \equiv T.domain) \wedge (S.size() \leq 1) \tag{4-1}$$

对于 $current \leq Q.size()$，“current”变量的值被初始化为“Invalid”，并且之后没有对其进行更改，所以不列入式4-1中。

在文件 $State.h$ 中有如下定义

```
// Encode automata states as integers.
typedef signed int State;

```

```
// Invalid states mean something bad is about to happen.
const State Invalid = −1;
```

代码 4.9　State.h

由代码4.9可知，*State* 类型实际上是整型，而“Invalid”为“-1”。

根据本节以上内容以及4.2节内容所说，可以实例化如图4.3的自动机。

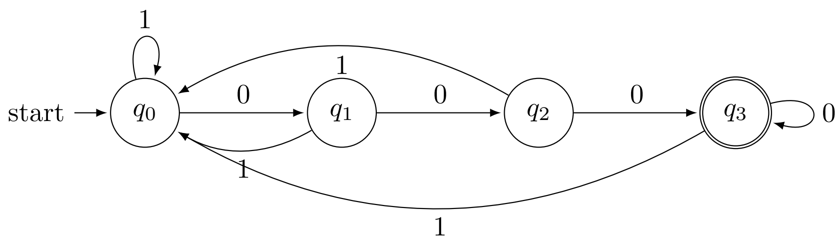


图 4.3　DFA 示例

图4.3中的自动机的转移函数如表4.1

表 4.1　图4.3状态转移函数

| 状态说明 | 状态 | 输入字符 | |
|---|---|---|---|
| | | 0 | 1 |
| 开始状态 (start) | $q_0$ | $q_1$ | $q_0$ |
| | $q_1$ | $q_2$ | $q_0$ |
| | $q_2$ | $q_3$ | $q_0$ |
| 结束状态 (accept) | $q_3$ | $q_3$ | $q_0$ |

实例化 *DFA* 类如代码4.10：

```
#include"DFA.h"
#include<iostream>
int main()
{
    DFA_components dfa_com1;

    // StateSet S 开始状态集
    dfa_com1.S.set_domain(10);
    dfa_com1.S.add(0);

    // StateSet F 结束状态集
    dfa_com1.F.set_domain(10);
    dfa_com1.F.add(3);
```

```

    // StatePool Q
    int i = 10;
    while (i--)
    {
        dfa_com1.Q.allocate();
    }

    // DTransRel T transition
    dfa_com1.T.set_domain(10);
    dfa_com1.T.add_transition(0, '0', 1);
    dfa_com1.T.add_transition(1, '0', 2);
    dfa_com1.T.add_transition(2, '0', 3);
    dfa_com1.T.add_transition(3, '0', 3);
    dfa_com1.T.add_transition(0, '1', 0);
    dfa_com1.T.add_transition(1, '1', 0);
    dfa_com1.T.add_transition(2, '1', 0);
    dfa_com1.T.add_transition(3, '1', 0);

    DFA dfa1(dfa_com1);
    dfa1.usefulf();

    std::cout<<dfa1<<std::endl;
    // 代码中的中文测试
    return 0;
}
```

代码 4.10　实例化 DFA 示例

代码4.10将在控制台输出如下信息：

```
DFA
Q = [0,4)
S = { 0 }
F = { 3 }
Transitions =
0->{ '0'->1 '1'->0 }
1->{ '0'->2 '1'->0 }
2->{ '0'->3 '1'->0 }
3->{ '0'->3 '1'->0 }

```

```
current = −1
```

代码 4.11　图4.3中自动机在 FIRE engine 中的表现形式

下文中将以表4.1的形式来描述一个自动机，以代码4.11的形式来表示自动机在 FIRE engine 中的展现形式。

# 第五章　测试内容

本章内容为测试过程中发现的问题，以及如何修复（部分）这些问题。

**注释** 5.1**：** 因为 $DFA$ 的最小化建立在状态等价性的基础之上，所以本文并未专门针对等价性进行测试。

# 致谢

虽为致谢环境，其实就是一个 Chapter，为啥这么费事？因为，致谢一章没有编号。直接使用 `\chapter*{}`的话，页眉又不符合工作手册要求，而且要往目录中添加该章节，还需要添加两行代码；为了简单快捷的设计出符合要求，又方便用户使用，只能借 `\backmatter` 模式和本模板自定义的 `\comtinuematter` 模式配合环境来做了。

# 参考文献

[1] Watson B. The design and implementation of the fire engine: a c++ toolkit for finite automata and regular expressions[J]. 1994.

[2] Watson B W. A taxonomy of finite automata construction algorithms[J]. 1993.

[3] Watson B W. A taxonomy of finite automata minimization algorithms[J]. 1993.

[4] 蒋宗礼, 姜守旭. 形式语言与自动机理论[M]. 三. 清华大学出版社, 2013.

[5] Microsoft Docs. assert 函数输出的诊断[EB/OL]. https://docs.microsoft.com/zh-cn/cpp/c-runtime-library/reference/assert-macro-assert-wassert?view=vs-2019.

# 附录 A　一些基本定义

**惯例 A.1 (幂集)：** 对于任意集合 $A$，我们使用 $\mathcal{P}(A)$ 代表 $A$ 的所有子集。$\mathcal{P}(A)$ 也叫做 $A$ 的幂集。有时也写作 $2^A$。

**惯例 A.2 (函数集)：** 对于集合 $A$ 和 $B$ ，$A \longrightarrow B$ 代表所有从 $A$ 到 $B$ 的函数的集合。而 $A \not\longrightarrow B$ 代表所有从 $A$ 到 $B$ 的“partial functions”。

**注释 A.1：** 对于集合 $A, B$, 关系 $\mathcal{C} \subseteq A \times B$ , 我们可以把 $\mathcal{C}$ 理解为函数 $\mathcal{C} \in A \longrightarrow \mathcal{P}(B)$。

**惯例 A.3 (元组投影)：** 对于 $n$ 元组 $t = (x_1, x_2, \cdots, x_n)$，我们使用符号 $\pi_i(t)$ $(1 \leq i \leq n)$ 代表元组元素 $x_i$；我们使用符号 $\bar{\pi}_i(1 \leq i \leq n)$ 代表 ($n$-1) 元组 $(x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_n)$。$\pi$ 和 $\bar{\pi}$ 自然扩展到元组集。

**惯例 A.4 (组合关系)：** 给出集合 $A, B, C$ 和两个关系 $E \subseteq A \times B$ 和 $F \subseteq B \times C$，定义组合关系（插入操作符 $\circ$）:

$$E \circ F = \{(a, c) : (\exists b : b \in B : (a, b) \in E \wedge (b, c) \in F)\}$$

**惯例 A.5 (等价关系的等价类)：** 对任何集合 $A$ 上的等价关系 $E$，我们使用 $[A]_E$ 代表等价类集合，即:

$$[A]_E = \{[a]_E : a \in A\}$$

集合 $[A]_E$ 也叫做 $A$ 的由 $E$ 引出的“划分（partition)”。

**定义 A.1 (等价类的指数)：** 对于集合 $A$ 上的等价关系 $E$，定义 $\sharp E = |[A]_E|$。$\sharp E$ 也叫做 $E$ 的“指数”。

**定义 A.2 (字母表)：** 字母表是有限大小的非空集合。

**定义 A.3 (等价关系的细化)：** 对于等价关系 $E$ 和 $E'$（在集合 $A$ 上)，当且仅当 $E \subseteq E'$，$E$ 是 $E'$ 的“细化”。

**定义** A.4 (**划分的细化关系** $\sqsubseteq$)**：** 对于等价关系 $E$ 和 $E'$ (在集合 $A$ 上), 当且仅当 $E \subseteq E'$, $[A]_E$ 也被称为 $[A]_{E'}$ 的细化（写作 $[A]_E \sqsubseteq [A]_{E'}$)。当且仅当 $E$ 下的每一个等价类完全包含在 $E'$ 下的某些等价类时，等价命题是 $[A]_E \sqsubseteq [A]_{E'}$ 。

**定义** A.5 (**元组和关系反转**)**：** 对一个 $n$ 元组 $(x_1, x_2, \cdots, x_n)$，定义反转为函数 $R$（后缀和上标)

$$(x_1, x_2, \cdots, x_n)^R = (x_n, \cdots, x_2, x_1)$$

给出一个集合元组 $A$，定义 $A^R = \{x^R : x \in A\}$。

# 附录 B　有限自动机

本节中我们定义有限自动机、其性质及其一些变化。大部分定义直接取自

**定义** B.1 (**有限自动机** ($Finiteautomata, FA$))**：** 自动机是一个 6 元组 $(Q, V, T, E, S, F)$，其中:

- $Q$ 是有限状态集;
- $V$ 是一个字母表;
- $T \in \mathcal{P}(P \times V \times Q)$ 是一个转换关系;
- $E \in \mathcal{P}(Q \times Q)$ 是一个 $\epsilon$-转换关系（空转换）;
- $S \subseteq Q$ 是开始状态集;
- $F \subseteq Q$ 是结束状态集;

字母表和函数 $\mathcal{P}$ 的定义分别在“定义A.2”和“惯例A.1”。

**注释** B.1**：** 我们也会在状态转换关系的表示上采取一定的自由。例如，我们也把转移关系写成 $T \in V \longrightarrow \mathcal{P}(Q \times Q), T \in Q \times Q \longrightarrow \mathcal{P}(V), T \in Q \times V \longrightarrow \mathcal{P}(Q), T \in Q \longrightarrow \mathcal{P}(V \times Q), E \in Q \longrightarrow \mathcal{P}(Q)$。每种情况下，$Q$ 的从左到右的顺序会是“preserved”；例如，函数 $T \in Q \longrightarrow \mathcal{P}(V \times Q)$ 定义为 $T(p) = \{(a, q) : (p, a, q) \in T\}$。所使用的签名将从上下文中清除。详见备注 A.3。“$\longrightarrow$ ”的定义出现在惯例 A.2。

由于本文中我们只考虑有限自动机，所以我们将会频繁的使用简化术语“自动机”。

## B.1　有限自动机的性质

本小节将会定义一些有限自动机（下称 $FA$）的性质。为了使定义更加简洁明了，我们引进三个特殊的 $FA$: $M = (Q, V, T, E, S, F)$，$M_0 = (Q_0, V_0, T_0, E_0, S_0, F_0)$，$M_1 = (Q_1, V_1, T_1, E_1, S_1, F_1)$。

**定义 B.2 ($FA$ 的大小)：** 定义一个 FA 的大小为 $|M| = |Q|$。

**定义 B.3 (FA 的同构 $\cong$)：** 我们把同构定义为 FA 的等价关系。当且仅当 $V_0 = V_1$，并且存在双射 $g \in Q_0 \longrightarrow Q_1$，使得

- $T_1 = \{(g(p,q), a, g(q)) : (p,a,q) \in T_0\}$
- $E_1 = \{(g(p,q), a, g(q)) : (p,q) \in E_0\}$
- $S_1 = \{g(s) : s \in S_0\}$
- $F_1 = \{g(f) : f \in F_0\}$

时 $M_0$ 和 $M_1$ 是同构的（写作 $M_0 \cong M_1$）。

**定义 B.4 (转移关系 $T$ 的扩展)：** 我们把 $T \in V \longrightarrow \mathcal{P}(Q \times Q)$ 到 $T^* \in V^* \longrightarrow \mathcal{P}(Q \times Q)$ 的转换关系以如下方式扩展:

$$T^*(\epsilon) = E^*$$

且对于 $(a \in V, w \in V^*)$ 有

$$T^*(aw) = E^* \circ T(a) \circ T^*(w)$$

操作符 $\circ$ 在惯例 A.5 中定义。这个定义也可以对称的表示。

**注释 B.2：** 有时候我们也使用把转移关系写成：$T^* \in Q \times Q \longrightarrow \mathcal{P}(V^*)$。

**定义 B.5 (左语言和右语言)：** 状态（$M$ 中）的左语言由函数 $\overleftarrow{\mathcal{L}}_M \in Q \longrightarrow \mathcal{P}(V^*)$ 给出，其中:

$$\overleftarrow{\mathcal{L}}_M(q) = (\cup s : s \in S : T^*(s,q))$$

状态（$M$ 中）的右语言由函数 $\overrightarrow{\mathcal{L}}_M \in Q \longrightarrow \mathcal{P}(V^*)$ 给出，其中

$$\overrightarrow{\mathcal{L}}_M(q) = (\cup f : f \in F : T^*(q,f))$$

通常在没有歧义的时候移除下标 $M$。

**定义 B.6 ($FA$ 的语言)：** 有限自动机的语言由函数 $\mathcal{L}_{FA} \in FA \longrightarrow \mathcal{P}(V^*)$ 给出，该函数的定义为:

$$\mathcal{L}_{FA}(M) = (\cup s, f : s \in S \wedge f \in F : T^*(s,f))$$

**定义 B.7 (完全自动机 ($Complete$))：** 一个完全有限自动机满足:

$$Complete(M) \equiv (\forall q, a : q \in Q \wedge a \in V : T(q,a) \neq \emptyset)$$

**定义 B.8 ($\epsilon$-$free$)：** 当且仅当 $E = \emptyset$ 时，$M$ 是 $\epsilon$-$free$ 的。

**定义 B.9 (Start-useful 自动机)：** 一个 $Useful_s$ 有限自动机定义如下:

$$Useful_s(M) \equiv (\forall q : q \in Q : \overleftarrow{\mathcal{L}}(q) \neq \emptyset)$$

**定义 B.10 (Final-useful 自动机)：** 一个 $Useful_f$ 有限自动机定义如下:

$$Useful_f(M) \equiv (\forall q : q \in Q : \overrightarrow{\mathcal{L}}(q) \neq \emptyset)$$

**注释 B.3：** $Useful_s$ 和 $Useful_f$ 与 $FA$ 的反转密切相关（见变换 B.22），对所有的 $M \in FA$，有 $Useful_f(M) \equiv Useful_s(M^R)$。

**定义 B.11 (Useful 自动机)：** $Useful$ 有限自动机是一个只有可达状态的有限自动机:

$$Useful(M) \equiv Useful_s(M) \wedge Useful_f(M)$$

**性质 B.1 (确定性有限自动机 ($DFA$))：** 当且仅当

- 无多重初始状态;
- 无 $\epsilon$ 转移;
- 转移函数 $T \in Q \times V \longrightarrow \mathcal{P}(Q)$ 不将 $Q \times V$ 映射至多重状态。

时有限自动机 $M$ 是确定性的。形式表达为:

$$Det(M) \equiv (|S| \leq 1 \wedge \epsilon - free(E) \wedge (\forall q, a : q \in Q \wedge a \in V : |T(q, a)| \leq 1))$$

**定义 B.12 ($FA$ 的确定性)：** $DFA$ 代表所有确定性的有限自动机的集合。我们把 $FA \setminus DFA$ 称为非确定性有限自动机 ($NDFA$,*nondeterministic $finite$ automata*) 的集合。

**惯例 B.1 (DFA 的转换函数)：** 对于 $(Q, V, T, \emptyset, S, F) \in DFA$，我们考虑把转换函数记为 $T \in Q \times V \not\rightarrow Q$（$\not\rightarrow$ 的定义可以查看惯例 A.2）。当且仅当 $DFA$ 是完全自动机的时候，转换函数是全函数。

**性质 B.2 (弱确定性自动机)：** 一些作者用比 $Det$ 弱的确定性自动机的定义；使用左语言，定义如下:

$$Det'(M) \equiv (\forall q_0, q_1 : q_0 \in Q \wedge q_1 \in Q \wedge q_0 \neq q_1 : \overleftarrow{\mathcal{L}}(q_0) \cap \overleftarrow{\mathcal{L}}(q_1) = \emptyset)$$

很容易证明 $Det(M) \Rightarrow Det'(M)$。

**定义 B.13 ($DFA$ 的最小化)：** 满足以下条件时，$M \in DFA$ 是最小化的：

$$Min(M) \equiv (\forall M' : M' \in DFA \wedge Complete(M') \wedge \mathcal{L}(M) = \mathcal{L}_{FA}(M') : |M| \leq |M'|)$$

$Min$ 仅定义在 $DFA$ 上。如果我们定义一个最小的但是仍然完全的 $DFA$，那么一些定义将会更加简单。它的定义如下：

$$Min_{\mathcal{C}}(M) \equiv (\forall M' : M' \in DFA \wedge Complete(M') \wedge \mathcal{L}_{FA}(M) = \mathcal{L}_{FA}(M') : |M| \leq |M'|)$$

$Min_{\mathcal{C}}$ 仅定义在完全 $DFA$ 上。

**定义 B.14 ($DFA$ 的最小化)：** 根据 Myhill-Nerode 定理，An M, such that Min(M)，是唯一的最小化 $DFA$，定理的相关介绍在[3]。

**性质 B.3 ($DFA$ 最小化的一个替代定义)：** 为了最小化 $DFA$，使用定义（仅定义在 $DFA$ 上）：

$$Minimal(Q, V, T, \emptyset, S, F) \equiv (\forall q_0, q_1 : q_0 \in Q \wedge q_1 \in Q \wedge q_0 \neq q_1 : \overrightarrow{\mathcal{L}}(q_0) \neq \overrightarrow{\mathcal{L}}(q_1)) \wedge Useful(Q, V, T, \emptyset, S, F)$$

有 $Minimal(M) \equiv Min(M)$ (对所有 $M \in DFA$) 。很容易证明 $Min(M) \Rightarrow Minimal(M)$。The reverse direction follows from the Myhill-Nerode 定理。