



**实验一：NFA到DFA的转化实验报告**

**学院：计算机学院（国家示范性软件学院）**

**班级： 2023211303**

**组长： 计子毅2023212872**

**组员： 陈子容2023210710**

**江宝金2023212219**

**张贺维2023212185**

**分工： 陈子容：设计核心算法并编程实现**

**计子毅：参与编程实现并编写文档**

**江宝金：进行案例测试并编写文档**

**张贺维：解决程序问题并进行优化**

**目录**

[一、实验环境描述及使用编程语言 1](#_Toc14460)

[二、程序设计思路及核心算法 1](#_Toc28744)

[1. 设计思路 1](#_Toc9408)

[2. 核心算法 1](#_Toc20209)

[2.1 算法原理 1](#_Toc10243)

[2.2 数据结构设计 1](#_Toc19083)

[2.3 算法步骤 2](#_Toc31652)

[1.初始化 2](#_Toc30839)

[2.状态扩展 2](#_Toc789)

[3.终止条件 2](#_Toc2982)

[4.确定接受状态 2](#_Toc13731)

[2.4 伪代码 2](#_Toc30155)

[3. 程序输入描述 3](#_Toc8041)

[1. NFA基本信息输入： 3](#_Toc14081)

[2. 转移函数输入： 3](#_Toc31444)

[3. 特殊状态输入： 3](#_Toc30647)

[4. 程序输出解释 3](#_Toc12141)

[1. NFA描述输出： 3](#_Toc9950)

[2. DFA描述输出： 4](#_Toc10174)

[3. 转换规则格式： 4](#_Toc26290)

[4. 特殊表示： 5](#_Toc16268)

[三、样例测试 5](#_Toc9358)

[四、改进思路和方法 9](#_Toc21696)

# 实验环境描述及使用编程语言

操作系统：Windows 11

调试软件：PyCharm

代码实现语言：Python

# 程序设计思路及核心算法

## 设计思路

本程序采用分阶段处理的方式实现NFA到DFA的转换：

1. **输入阶段：**通过交互式界面获取NFA的完整定义
2. **转换阶段：**使用子集构造法将NFA转换为DFA
3. **输出阶段：**展示转换结果

程序特点：

* 采用Python实现，注重算法，忽略计算机底层细节
* 使用字典结构高效存储状态转移关系
* 通过深度优先搜索（DFS）实现状态集遍历
* 提供清晰的表格化输出

## 核心算法

此程序的核心算法是NFA到DFA转换算法（子集构造法）

**2.1 算法原理**

将NFA状态集的幂集作为DFA的状态集，模拟NFA在所有可能的输入字符下的转移路径

**2.2 数据结构设计**

·状态表示:使用集合存储NFA状态集

·转移函数:字典结构，键为元组(状态，输入字符)，值为目标状态集

·状态集:使用列表存储生成的所有DFA状态

**2.3 算法步骤**

1.初始化

·将NFA初始状态的单元素集合作为DFA的初始状态

·初始化待处理状态栈和状态集

2.状态扩展

·从中弹出状态

·对每个输入字符

·计算

当且仅当

·记录转移函数

·若未被处理过，则将其插入和

3.终止条件

当为空时算法终止

4.确定接受状态

包含NFA任何接受状态的DFA状态均为接受状态

**2.4 伪代码**

算法伪代码如下:

## 3. 程序输入描述

1. **NFA基本信息输入**：
   * 状态数：输入一个整数表示NFA的状态总数（如4）
   * 字母表：输入用空格分隔的符号（如"a b"）
2. **转移函数输入**：
   * 格式：源状态 符号 目标状态列表
   * 示例："0 a 0 1"表示
   * 空行结束输入
3. **特殊状态输入**：
   * 初始状态：输入一个整数（如0）
   * 终止状态：输入用空格分隔的整数（如"3"）

## 4. 程序输出解释

**4.1 NFA描述输出：**

·状态集合：Q = {q0, q1, ..., qN}

·字母表：T = {a, b, ...}

·初始状态：q0 = qX

·接受状态：F = {qX, ...}

·状态转换表（表格形式）

**4.2 DFA描述输出：**

·状态集合：Q = {[q0], [q0,q1], ...}

·字母表：T = {a, b, ...}

·初始状态：q0 = [qX]

·接受状态：F = {[qX], [qY,qZ], ...}

·状态转换表（表格形式）

**4.3 转换规则格式：**

·表格形式展示每个状态在不同输入下的转移

·示例：

状态 | a | b

-----------------------------------------------

q0 | q0,q1 | ∅

q1 | ∅ | q2

q2 | ∅ | q3

q3 | ∅ | ∅

**4. 4 特殊表示：**

·空集与DFA中空状态用"∅"表示

·DFA状态用方括号表示（如[q0,q1]）

# 三、样例测试

### 测试集1:最简单的NFA

**输入：**

状态数(例如4): 1

字母表(用空格分隔，例如'a b'): a

请输入转移函数(格式：源状态 符号 目标状态列表)：

例如: 0 a 0 1 表示 δ(q0,a)={q0,q1}

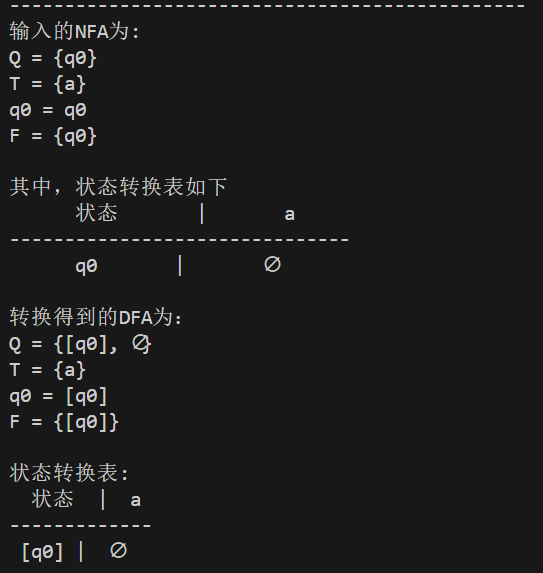
输入空行结束

>

初始状态(例如0): 0

终止状态(用空格分隔，例如'3'): 0

**输出：**



### 测试集2:简单确定性NFA

**输入：**

状态数(例如4): 2

字母表(用空格分隔，例如'a b'): a b

请输入转移函数(格式：源状态 符号 目标状态列表)：

例如: 0 a 0 1 表示 δ(q0,a)={q0,q1}

输入空行结束

> 0 a 1

> 0 b 0

> 1 a 1

> 1 b 1

>

初始状态(例如0): 0

终止状态(用空格分隔，例如'3'):

**输出：**

文本

AI 生成的内容可能不正确。

### 测试集3:含非确定性转移的NFA

**输入：**

状态数(例如4): 3

字母表(用空格分隔，例如'a b'): a b

请输入转移函数(格式：源状态 符号 目标状态列表)：

例如: 0 a 0 1 表示 δ(q0,a)={q0,q1}

输入空行结束

> 0 a 0 1

> 0 b 0

> 1 a 2

> 1 b 2

> 2 a 2

> 2 b 2

>

初始状态(例如0): 0

终止状态(用空格分隔，例如'3'): 2

**输出：**

文本

AI 生成的内容可能不正确。

### 测试集4:复杂NFA

**输入：**

状态数(例如4): 5

字母表(用空格分隔，例如'a b'): a b c

请输入转移函数(格式：源状态 符号 目标状态列表)：

例如: 0 a 0 1 表示 δ(q0,a)={q0,q1}

输入空行结束

> 0 a 0 1

> 0 b 0 2

> 1 a 3

> 1 b 3

> 2 a 4

> 2 b 4

> 3 a 3

> 3 b 3

> 4 a 4

> 4 b 4

>

初始状态(例如0): 0

终止状态(用空格分隔，例如'3'): 3 4

**输出：**

文本

AI 生成的内容可能不正确。

## 四、改进思路和方法

1. 输入时可能使用户不够明确，输入形式较简单，考虑优化改进

2. 输出只做到以书面形式进行输出，可以改进为增加状态图输出

3. 当前代码未处理 ε 转移，可以增加 ε 闭包的预处理

4. 转换后的DFA可能不是最小化的，可以使用新学习的最小化算法对DFA进行最小化