燕山大学

编译原理课程设计报告

NFA 确定化和最小化程序的设计与实现

学　　院　　信息科学与工程学院

年级专业　　2021 级计算机 3 班

学生姓名　　李梦浩

学生学号　　202111070121

设计日期　　2023 年 12 月 31 日-2024 年 1 月 5 日

NFA 确定化和最小化程序的设计与实现

1 概述

1.1 目的与意义

该项目旨在设计和实现一个应用程序，能够将给定的任意非确定性有限自动机（NFA）转化为确定性有限自动机（DFA），并对得到的 DFA 进行最小化。通过这个项目，学生将能够深入理解和掌握自动机的相关理论和技术方法，包括 NFA 到 DFA 的转化和 DFA 的最小化过程。

1.2 主要完成的任务

1.2.1NFA 的输入与展示

实现通过文件读入或者窗口提示用户输入一个 NFA。以状态转换图或表格形式呈现 NFA，使学生能够直观地了解 NFA 的结构和状态转换关系。

1.2.2NFA 到 DFA 的转化

使用子集法将 NFA 转化为 DFA，演示该计算过程。以状态转换图或表格形式呈现得到的 DFA，突出开始状态和终止状态。通过这一步骤，学生能够理解 NFA 和 DFA 之间的关系，以及 NFA 转化为 DFA 的具体步骤。

1.2.3DFA 的最小化

使用分割法对得到的 DFA 进行最小化，演示该计算过程。以状态转换图或表格形式呈现最小化后的 DFA，同时指出开始状态和终止状态。这一步骤使学生了解如何优化自动机，减少状态的数量，提高自动机的效率。

1.3 使用的开发工具

编译器：Visual Studio 2022。

1.4 解决的主要问题

1.4.1NFA 到 DFA 的转化问题

如何将非确定性有限自动机转化为确定性有限自动机。学生通过实际计算过程理解子集法的应用。

1.4.2DFA 的最小化问题

如何对确定性有限自动机进行最小化，以达到降低状态数量的目的。学生通过实际计算过程理解分割法的应用。

1.4.3 图形化展示问题

如何以直观的方式展示 NFA、转化过程中的 DFA 以及最小化后的 DFA。通过状态转换图或表格形式，使学生能够清晰地观察自动机的结构和状态转换关系。

1.5 课程设计计划

课程设计第一天查找资料并且对选定的课设题目进行总体设计。第二天进行详细设计设计，并进行具体的功能描述。第三天根据详细设计进行编码和调试。第四天进行对课设题目编码、排错、进行联调及测试和撰写课设报告。

2 使用的基本概念和原理

2.1 非确定性有穷自动机(NFA)

NFA 是一种自动机模型，相对于确定性有限自动机(DFA)，它在某个状态和输入符号的组合下，可以有多个可能的转移状态。

NFA 的状态转移可以是非确定性的，即在某一状态和输入下，可以有多个下一个状态。这允许更灵活的自动机设计，但也增加了转化为 DFA 的复杂性。

2.2 确定性有穷自动机(DFA)

DFA 是一种自动机模型，相对于 NFA，它在任意给定的状态和输入符号下，只有唯一的下一个状态。

DFA 的设计更加结构化，每个状态和输入的组合都有唯一的下一个状态，使得自动机行为更加可预测和明确。

2.3 子集法(NFA 到 DFA 的转化)

子集法是将 NFA 转化为 DFA 的一种方法，通过创建 DFA 的状态集合，每个集合表示 NFA 中可能的状态组合。

对于每个 DFA 状态集合，通过 NFA 的非确定性转移，计算可能的下一个状态，逐步构建出等价的 DFA。

2.4 分割法(DFA 最小化)

分割法是对 DFA 进行最小化的方法，通过对等价状态进行合并，减少自动机的状态数。

将 DFA 的状态分为不同的等价类，然后逐步合并等价类，直到无法合并为止。最终得到的 DFA 是最小化的。

2.5 状态转换图

状态转换图是一种图形化表示自动机状态和转移关系的方式，用于直观展示自动机的结构。

每个节点代表自动机的一个状态，边表示状态之间的转移关系，方便观察和理解自动机行为。

2.6 开始状态和终止状态

自动机的开始状态是自动机在开始时所处的状态，终止状态是自动机在某些输入序列结束时所处的状态。

开始状态和终止状态的明确定义是自动机设计的基础，通过它们可以确定自动机的输入和输出关系。

3 总体设计

3.1 总体结构

3.1.1 初始化模块

用于用户输入 NFA 的相关信息，包括状态个数、开始状态、结束状态、字母表个数、字母表字母以及状态转换弧。

3.1.2E_Closure 模块

计算 NFA 每个状态的 E_Closure，输出相关信息。

3.1.3DFA 构造模块

使用子集法将 NFA 转化为 DFA，输出 DFA 的状态转换图或表格。

3.1.4DFA 最小化模块

使用分割法对 DFA 进行最小化，输出最小化后的 DFA 的状态转换图或表格。

3.1.5 打印函数模块

用于输出状态集合、矩阵等信息。

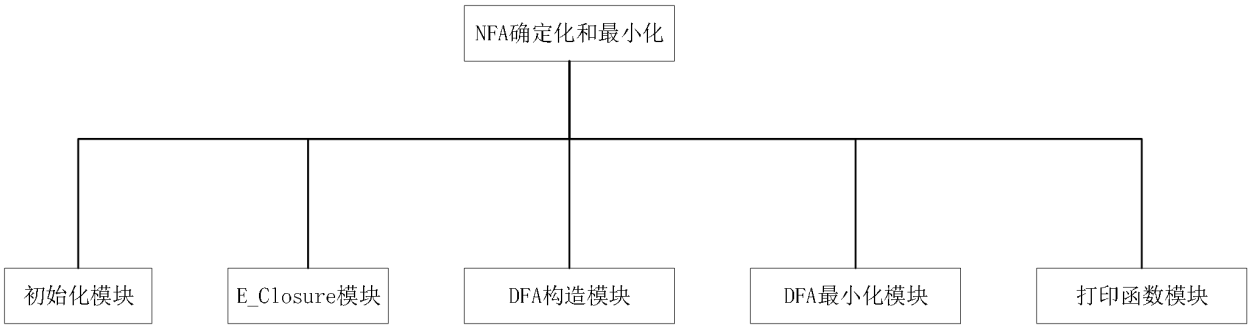

图 3-1 功能模块图

3.2 总体流程

用户输入 NFA 的相关信息。计算每个状态的 E_Closure，输出 E_Closure 信息。使用子集法构造 DFA，输出 DFA 的状态转换图或表格。使用分割法对 DFA 进行最小化，输出最小化后的 DFA 的状态转换图或表格。输出最终的 DFA 的开始状态和结束状态。

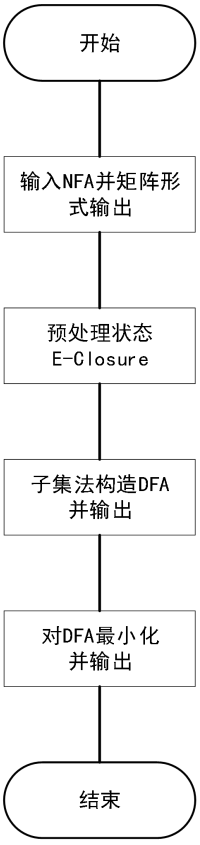

图 3-2 总体流程图

4 详细设计

4.1 初始化模块

4.1.1 流程

用户输入状态个数、开始状态个数、结束状态个数等 NFA 的相关信息。用户输入字母表个数和字母表字母。用户输入状态转换弧，直到输入结束标志。

4.1.2 函数

void init()

4.1.3 实现算法

用户输入，依次获取 NFA 的各项信息，构建状态转换弧。

4.2E_Closure 模块

4.2.1 流程

对每个状态进行广度优先搜索，计算其 E_Closure。输出每个状态的 E_Closure。

4.2.2 函数

void E_Closure()

4.2.3 实现算法

使用 queue 进行广度优先搜索，对每个状态计算 E_Closure。输出每个状态的 E_Closure 集合。

4.3DFA 构造模块

4.3.1 流程

使用子集法构造 DFA，遍历每个状态的输入符号，计算 Move 和 E_Closure。输出 DFA 的状态转换图或表格。

4.3.2 函数

void Solve()

4.3.3.实现算法

使用循环遍历每个状态，对每个输入符号计算 Move 和 E_Closure，构建 DFA 的状态转换图或表格。输出 DFA 的状态转换图或表格。

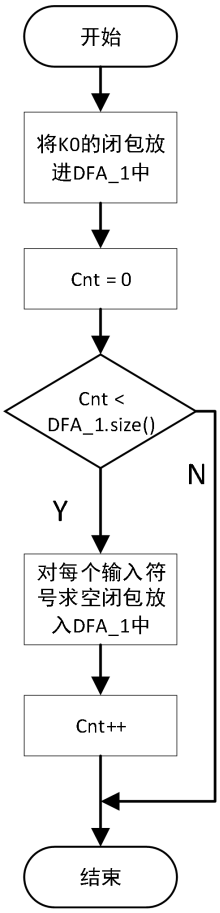

图 4-1 DFA 确定化流程图

4.4DFA 最小化模块

4.4.1 流程

使用分割法对 DFA 进行最小化，合并等价状态。输出最小化后的 DFA 的状态转换图或表格。

4.4.2 函数

void SolveMin()

bool Segment(pair<set<int>, int> se)

void Simplify(vector<pair<set<int>, int>> &segment)

4.4.3 参数

se：表示当前要分割的子集。

&segment：表示分割完后需要消除多余状态的集合

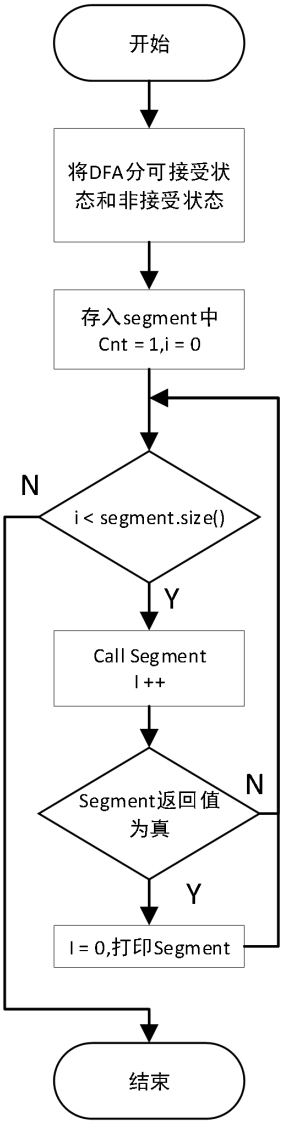

图 4-2 DFA 最小化流程图

4.4.4 实现算法

使用循环进行多次分割，直至不再能分割为止。合并等价状态，简化 DFA 的状态集合。输出最小化后的 DFA 的状态转换图或表格。

4.5 打印函数模块

4.5.1 流程

打印 NFA 的状态转换图或表格，包括开始状态、结束状态等信息。

打印 DFA 或最小化后的 DFA 的状态转换图或表格，包括开始状态、结束状态等信息。

4.5.2 函数

void MyPrint(set<int> T)

void MyPrint()

void MyPrint(vector<set<int>> DFA)

void MyPrint_Temp(vector<pair<set<int>, int>> segment)

4.5.3 参数

T：表示状态集合。

DFA：表示 DFA 的状态集合。

segment：表示最小化后的 DFA 的状态集合。

4.5.4 实现算法

使用循环和条件判断，输出相应的状态转换图或表格。

5 编码实现

5.1 开发环境的设置和建立

使用 Visual Studio 2022 创建项目 compile，添加源文件名称为 deter.cpp。

5.2 主要程序的代码设计及注释

5.2.1DFA 构造模块

```
void Solve()//子集法构造 DFA
{

    DFA_1.push_back(E_Closure(St));//将 K0 的 e-closure 放入集合 DFA

    int cnt = 0;
    while (cnt < DFA_1.size())
    {
        auto temp = DFA_1[cnt];
        qTemp.push_back(temp); //  便于以后输出矩阵
        for (auto a : xiTa)
        {
            set<int> U;
            U = Move(temp, a);
            U = E_Closure(U);
            qTemp.push_back(U); //  便于以后输出;

            if (find(DFA_1, U) == -1)
            {
                DFA_1.push_back(U);
            }
            int location = find(DFA_1, U);//  先放再找 location 否则可能为  -1;
            DFA_2.push_back({ { cnt, location }, a });
        }
        cnt++;
    }
}
```

5.2.2DFA 最小化模块

```
void SolveMin()//  分割法
{
    set<int> dfaSt;//  非终态
    for (int i = 0; i < DFA_1.size(); i++)
    {
        if (dfaEd.end() == dfaEd.find(i))dfaSt.insert(i);
    }
    segment.push_back({ dfaSt, segment.size() });//放入可接受状态
    segment.push_back({ dfaEd, segment.size() });//放入非接受状态

    int cnt = 1;
    cout << endl << "分割法最小化过程" << endl << "----------------------" << endl;
    cout << "第" << cnt++ << "次    ";
    MyPrint_Temp(segment);//每次过程打印
    for (int i = 0; i < segment.size();)
    {
        if (Segment(segment[i]))
        {
            i = 0;
            cout << "第" << cnt++ << "次    ";
            MyPrint_Temp(segment);
        }
        else i++;
    }
}
bool Segment(pair<set<int>, int> se)//分割函数
{
    for (auto a : xiTa)
    {
```

```
            unordered_map<int, set<int>> temp;
            for (auto S : se.first)
            {
                int v = Serach(S,a);//
                int V = Serach(segment, v);// 每个状态的转移状态所在的子集号
                temp[V].insert(S);
            }

            if (temp.size() > 1)// 只有分割了才放入
            {
                //删除原来的子集并且更新子集号
                segment.erase(segment.begin() + i);
                for (int i = 0; i < segment.size(); i ++)
                    segment[i].second = i;

                // 将分割的子集加入
                for (auto it = temp.begin(); it != temp.end(); it++)
                {
                    segment.push_back({ (*it).second , segment.size() });
                }
                return true;
            }
        }
        return false;
}
void Simplify(vector<pair<set<int>, int>> &segment)//合并状态
{
        for (auto &K : segment) // 把多余要合并的状态删了
            if (K.first.end() != K.first.find(0))
            {
                K.first.clear(); K.first.insert(0);
```

```
            }
            else
            {
                int t = *K.first.begin();//只留一个状态即可
                K.first.clear(); K.first.insert(t);
            }
    }
```

5.3 解决的技术难点、经常犯的错误

使用 set 类模版解决状态重复的问题，使用邻接表存储状态图。

经常犯的错误，没有认真构想算法的具体逻辑，调试太长时间。

6 测试和试运行

6.1 测试

本次测试采用黑盒测试。

6.1.1 样例一

测试样例 1（单开始、单终止状态）：

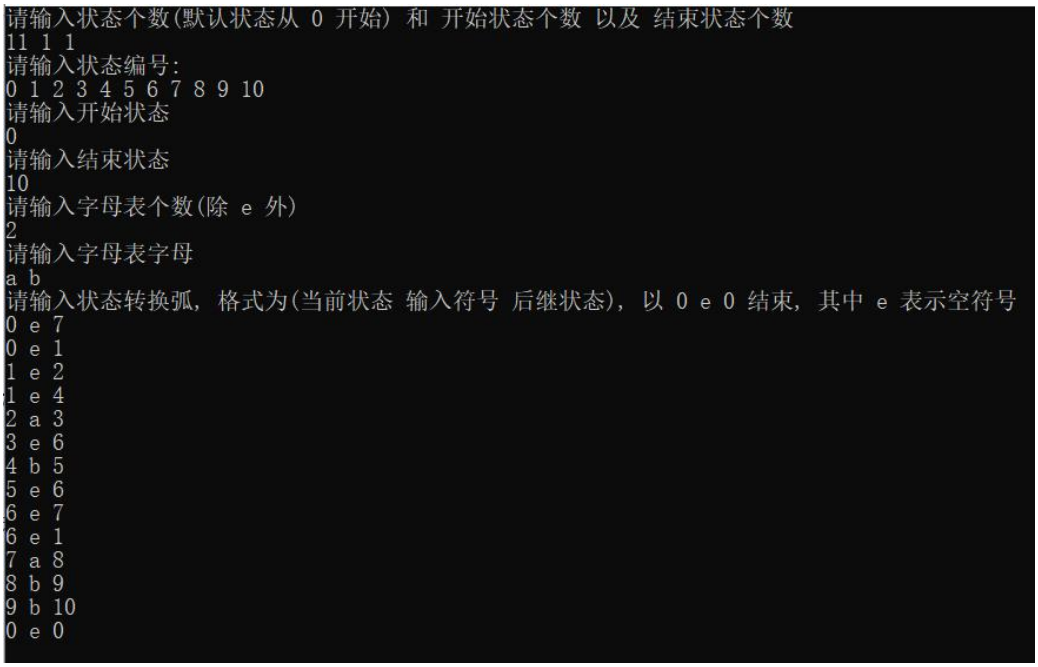

图 6-1 测试样例 1 输入

```
NFA表示为:
--------------------
          a          b          e(空字符)
0                              1 7
1                              4 2
2           3
3                              6
4                    5
5                              6
6                              1 7
7           8
8                    9
9                    10
10
NFA开始状态为:0
结束状态为:10

--------------------
NFA状态的E_Closure如下:
0 的E_Closure为: {0,1,2,4,7}
1 的E_Closure为: {1,2,4}
2 的E_Closure为: {2}
3 的E_Closure为: {1,2,3,4,6,7}
4 的E_Closure为: {4}
5 的E_Closure为: {1,2,4,5,6,7}
6 的E_Closure为: {1,2,4,6,7}
7 的E_Closure为: {7}
8 的E_Closure为: {8}
9 的E_Closure为: {9}
10 的E_Closure为: {10}

DFA矩阵表示
----------------------------
                    a                    b
T0: {0,1,2,4,7}          T1 {1,2,3,4,6,7,8}          T2 {1,2,4,5,6,7}
T1: {1,2,3,4,6,7,8}          T1 {1,2,3,4,6,7,8}          T3 {1,2,4,5,6,7,9}
T2: {1,2,4,5,6,7}          T1 {1,2,3,4,6,7,8}          T2 {1,2,4,5,6,7}
T3: {1,2,4,5,6,7,9}          T1 {1,2,3,4,6,7,8}          T4 {1,2,4,5,6,7,10}
T4: {1,2,4,5,6,7,10}          T1 {1,2,3,4,6,7,8}          T2 {1,2,4,5,6,7}

DFA的开始状态为: T0
DFA的终止状态为: T4
```

图 6-2 样例 1NFA 确定化

```
分割法最小化过程
----------------------
第1次   {0,1,2,3}        {4}
第2次   {4}        {0,1,2}        {3}
第3次   {4}        {3}        {0,2}        {1}

DFA最小化的矩阵表示
----------------------------
          a          b
T4        T1         T2
T3        T1         T4
T0        T1         T2
T1        T1         T3

DFA最小化的开始状态为:T0

DFA最小化的终止状态为: T4
```

图 6-3 样例 1DFA 最小化

6.1.2 样例二

测试样例 2（多个终止状态）：

```
请输入状态个数 和 开始状态个数 以及 结束状态个数
7 1 3
请输入状态的编号:
1 2 3 4 5 6 7
请输入开始状态
1
请输入结束状态
5 6 7
请输入字母表个数(除 e 外)
2
请输入字母表字母
a b
请输入状态转换弧，格式为(当前状态 输入符号 后继状态)，以 0 e 0 结束，其中 e 表示空符号
1 a 6
1 b 3
2 a 7
2 b 3
3 a 1
3 b 5
4 a 4
4 b 6
5 a 7
5 b 3
6 a 4
6 b 1
7 a 4
7 b 2
0 e 0
```

图 6-4 测试样例 2 输入

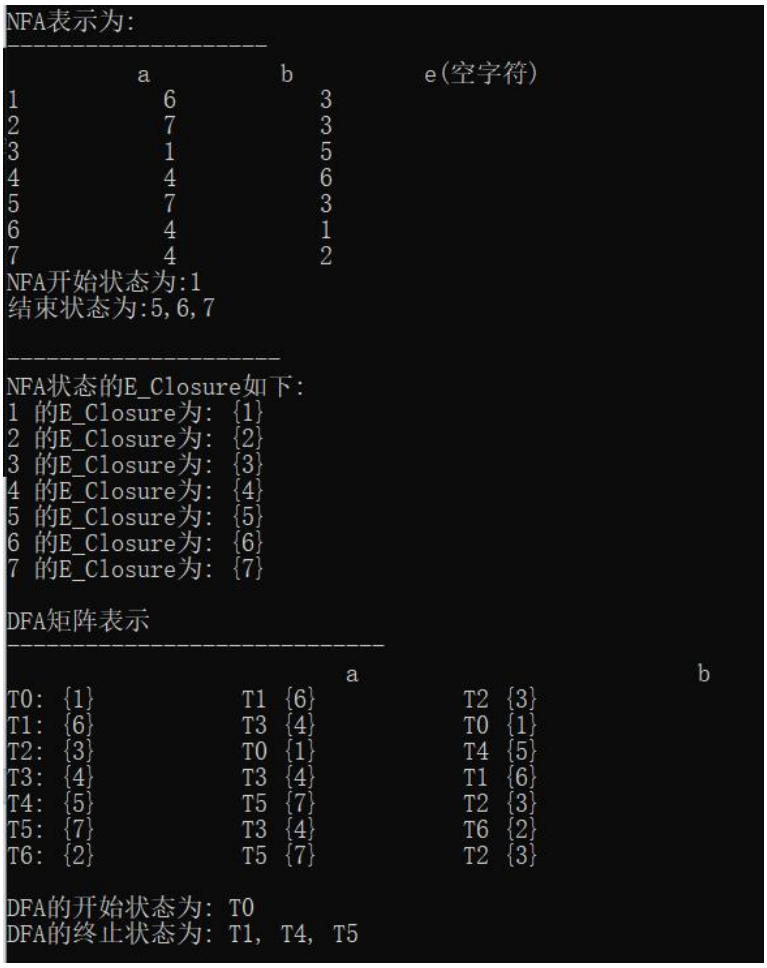

图 6-5 样例 2NFA 确定化

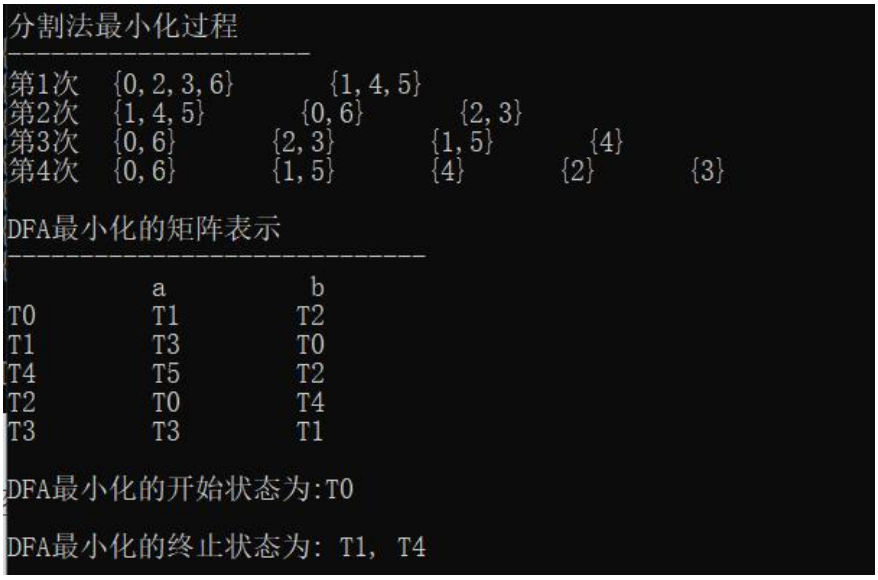

图 6-6 样例 2DFA 最小化

6.2 测试时出现过的问题及其解决方法

在编写 NFA 确定化代码的时候，刚开始没采用预处理每个状态的空字符闭包，导致编写函数时比较麻烦，后来查阅资料后，将每个状态闭包预处理后编码逻辑清晰。

DFA 最小化时候，状态删除的时候，一开始使用的是状态重新排号，测试结果出来发现很乱，而且编码逻辑也很混乱。这一过程就完全按照课本的解决方法，将多余的状态直接删除而不是，对每个状态集合重新编号。

7. 总结

本次课程设计要求都完全完成，具体要求功能如下：

(1)通过文件读入或者窗口提示输入一个 NFA，以状态转换图或者表格形式呈现 NFA；

(2)给出采用子集法将 NFA 转为 DFA 的计算过程，以状态转换图或者表格形式呈现得到的 DFA；

(3)给出采用分割法将 DFA 最小化的计算过程，以状态转换图或者表格形式呈现化简之后的 DFA；

(4)呈现的 NFA 或者 DFA 需要指出开始状态和终止状态。

通过本次课程设计，我深入理解了 NFA 自动转化为 DFA 和最小化的相关理论，包括子集法、分割法等算法。掌握了自动机的相关概念和技术方法。在实现课程设计项目的过程中，得到了编程实践的机会，提高了对 C++编程语言的熟练程度。通过实际项目的编写，加深了对数据结构和算法的应用能力。

在设计和实现过程中，我也遇到了一些挑战和问题，通过查阅资料、思考和调试，培养了问题解决的能力。学会了在编程中灵活运用各种数据结构和算法。

课程设计软件采用模块化设计的思想，将整个课程设计划分为不同的模块，每个模块负责不同的功能。这种设计方式使得代码结构清晰，易于理解和维护。选择合适的数据结构对于算法的实现至关重要。在本次课程设计中，使用了 set、vector、map 等数据结构，使得对状态集合、状态转换等的操作更为方便和高效。在用户输入阶段，通过良好的异常处理，可以增加程序的鲁棒性。及时给出友好的提示信息，提高用户体验。

课程设计需要合理规划时间，充分考虑到理论学习、设计、编码、调试等多个环节。在实际编码过程中，更好地预估所需时间，避免时间不足的情况。

在设计中要充分考虑测试用例，确保程序在各种情况下都能正常运行。及时进行调试，发现并修复潜在问题，提高程序的稳定性。

本次课程设计是一个较为综合的项目，既涉及理论知识的深入理解，也涉及编程实践的技能提升。通过这个过程，我不仅学到了自动机的相关理论知识，还提高了对 C++编程语言的熟练程度。在解决问题的过程中，锻炼了逻辑思维和问题解决能力。总体而言，这次课程设计让我更深刻地理解了编译原理和自动机理论，为我未来的学习和工作奠定了坚实的基础。

8. 参考文献

[1] 王生原 董源.编译原理（第 3 版）..北京:清华大学出版社,2015 年:47-54 页

编译原理课程设计成绩单

姓名	李梦浩
班级	2021 计算机 3 班
学号	202111070121
分析设计 （满分 30 分）	
程序开发 （满分 40 分）	
汇报答辩 （满分 30 分）	
总成绩	

评阅教师签字：

验收日期：　　年　　月　　日