華南師範大學

本科学生实验（实践）报告

院　　系：计 算 机 学 院

实验课程：编译原理

实验项目：正则表达式转换为 DFA 图

指导老师：黄煜廉

开课时间：2024 ～ 2025 年度第 1 学期

专　　业：网络工程

班　　级：网工二班

学　　生：肖翔

学　　号：20222132002

华南师范大学教务处

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 **实验评分**

一、实验内容

设计一个应用软件，以实现将正则表达式-->NFA--->DFA-->DFA 最小化-->词法分析程序(选做内容)

二、实验目的

（1）正则表达式应该可以支持命名操作，运算符号有： 转义符号(\)、连接、选择(|)、 闭包(*)、 正闭包（+）、 [ ]、 可选（?）、 括号 ( )。

（2）要提供一个源程序编辑界面，让用户输入一行（一个）或多行（多个）正则表达式（可保存、打开正则表达式文件）。

（3）需要提供窗口以便用户可以查看转换得到的 NFA（用状态转换表呈现即可）。

（4）需要提供窗口以便用户可以查看转换得到的 DFA（用状态转换表呈现即可）。

（5）需要提供窗口以便用户可以查看转换得到的最小化 DFA（用状态转换表呈现即可）。

（6）要求应用程序为 Windows 界面。

（7）书写完善的软件文档。

三、实验文档：

（一）系统概述

1. 系统结构

系统分为 4 个模块：正则表达式预处理模块、正则表达式转 NFA 模块、NFA 转 DFA 模块、DFA 最小化模块。

2. 数据结构的选择

本系统使用了结构体 struct、映射 map、集合 set、向量 vector、栈 stack、双端队列 dequeue 等多种数据结构。

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 **实验评分**

以下是对本系统部分数据结构的详细介绍。

（1） set<char> charSet：用于记录正则表达式中涉及到的基本字符。

（2） unordered_map<string, string> name：赋值映射，将命名语句中的变量名映射成正则表达式。

（3） struct nfaEdge：NFA 图的边，里面存储了基本字符和指向下一个结点的 nfaNode 型指针 next。

（4） struct nfaNode：NFA 图的结点，里面存储了 nfaEdge 型的向量、初态标识（bool）、终态标识（bool）和结点编号，向量用来表示从该结点出发的多条边。

（5） struct nfaStatusNode：它与 nfaNode 一一对应，里面存储了 flag、状态结点编号、字符和集合的映射关系。flag 通过“+”和“-”表示 NFA 图的终态和初态。映射关系用来表示对应字符能够到达的结点集合。

（6） struct NFA：NFA 图，里面存储了 nfaNode 型指针，只需一个起始结点指针和终止结点指针就能标识一个 NFA 图。

（7） struct dfaNode：DFA 图的结点，里面存储了 flag、整型集合、字符和集合的映射关系。flag 通过“+”和“-”表示 DFA 图的终态和初态。映射关系用来表示对应字符能够到达的结点集合。整型集合表示当前 NFA 结点包含的 NFA 结点标识。

（8） struct dfaMinNode：最小化 DFA 图的结点，数据结构与 nfaStatusNode 类似。

（9） vector<set<int>> dividedSetVec：用于记录在最小化过程中，分割得到的集合。

（10）map<int, int> dfaSameMinSet：DFA 的 dfaStatusCount 只要被划分到同一个集合就会被映射成同样的值。

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

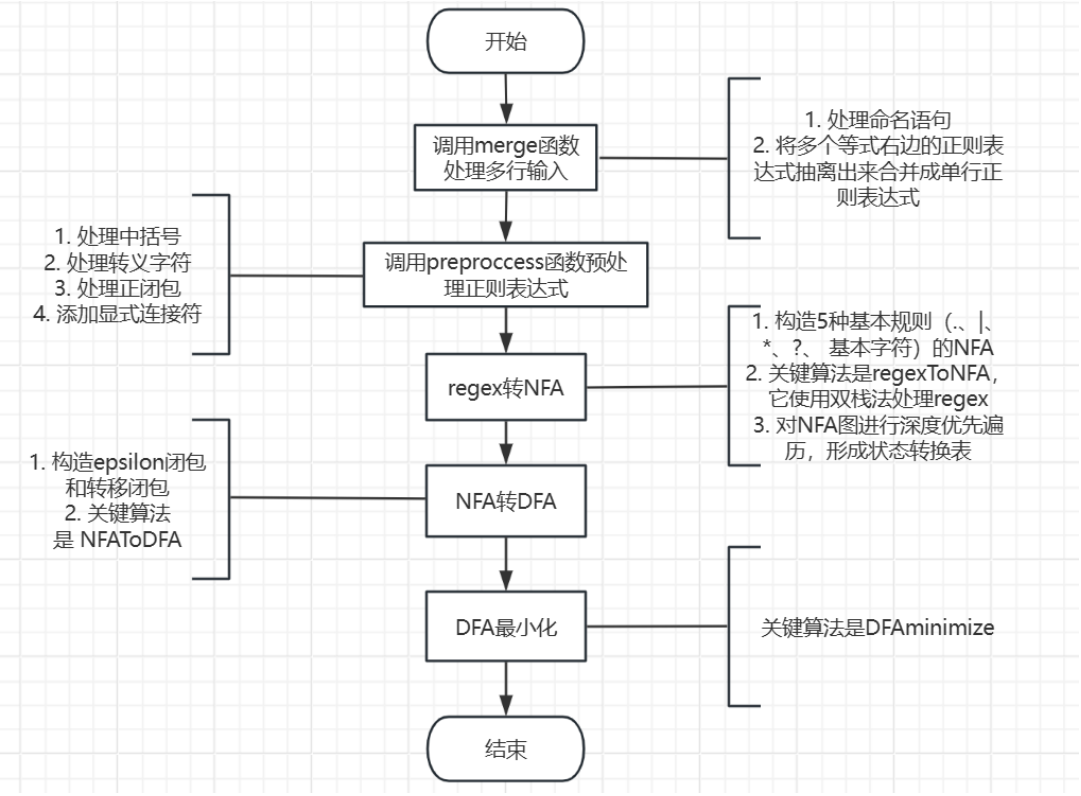
专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 实验评分

3. 程序整体大致流程的流程图如下。



（二）系统的详细介绍。

1. 正则表达式预处理

系统将用户输入的字符串划分为第一个等号的左右两部分，同时通过字符串开头是否拥有“_”判断此语句是命名语句还是 DFA 生成语句。命名语句等号左边是别名，右边是正则表达式，它们之间会形成一种映射关系，具体来说就是从别名映射到正则表达式。DFA 生成语句等号右边是正则表达式，如果遇到别名时，会调用映射关系，将别名改为对应的正则表达式。对于输入的多行

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 **实验评分**

DFA 生成语句，系统会将等号右边的正则表达式抽取出来，并将各个表达式通过或运算符号连接起来，合并成一行正则表达式。[1]

预处理部分会接着对这一行表达式进行中括号处理、转义字符处理、正闭包处理和添加显式连接符处理。如将[a-z]转换为(a|b|c...y|z)、将 abc|d 转换为 a.b.c|d、将(abc)+转换为(abc)(abc)*、将*转换为.*。转义字符多出的连接运算符“.”在后面的 regexToNFA 函数中视情况处理。

2. regex 转 NFA

将 regex 转为 NFA 的关键算法是 regexToNFA 函数，该函数需使用双栈法[2]，即“操作数”栈（存放 NFA 类型）和操作符栈，类似处理四则运算表达式，将中缀表达式转换为后缀表达式，再计算后缀表达式。运算符的优先级设置是：’*’=’?’>’.’>’|’>基本字符。不过在转换 regex 前，我们还需要定义正则表达式中的基本规则[1]。

（1） 定义基本字符

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　　业 网络工程 年级、班级 2022 级网工二班

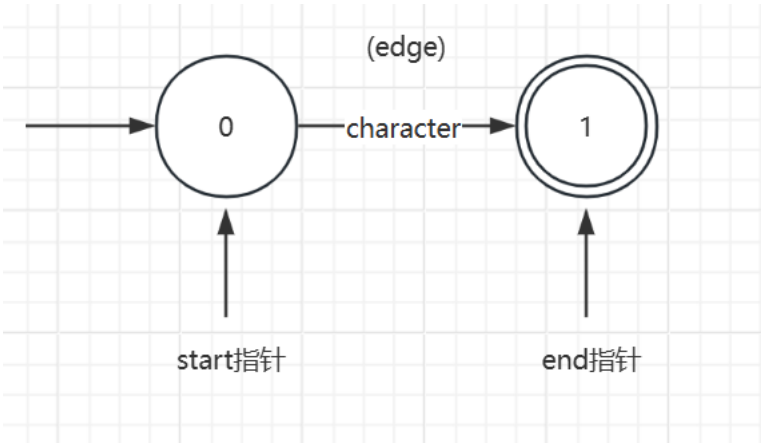
课程名称 编译原理 实验项目 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 实验评分

```
// 基本字符NFA
NFA basicCharNFA(char character) {
    nfaNode* start = new nfaNode();
    nfaNode* end = new nfaNode();

    start->isStart = true;
    end->isEnd = true;

    nfaEdge edge;
    edge.c = character;
    edge.next = end;
    start->edges.push_back(edge);

    NFA nfa(start, end);

    nfaCharSet.insert(character);
    dfaCharSet.insert(character);

    return nfa;
}
```

下图是对以上代码的解释。



（2） 定义连接运算符

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

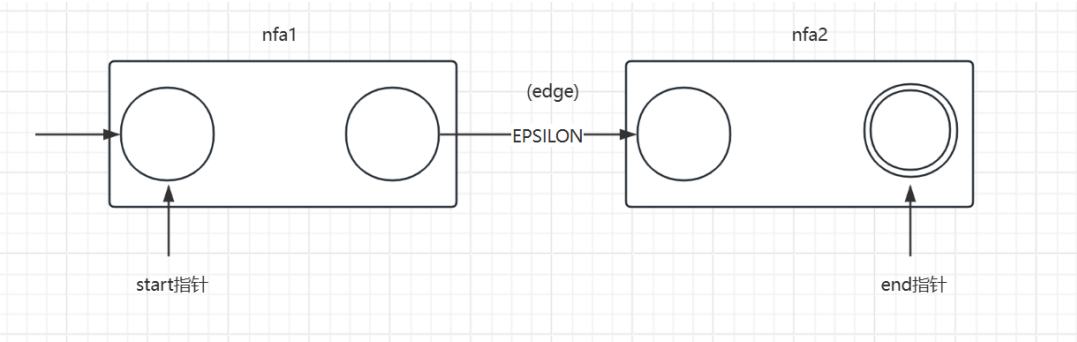
课程名称 编译原理 实验项目 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 实验评分

```
// 连接运算符的NFA图
NFA concatNFA(NFA nfa1, NFA nfa2) {
    // 把nfa1的终止状态与nfa2的起始状态连接起来
    nfa1.end->isEnd = false;
    nfa2.start->isStart = false;

    nfaEdge edge;
    edge.c = EPSILON;
    edge.next = nfa2.start;
    nfa1.end->edges.push_back(edge);

    NFA nfa;
    nfa.start = nfa1.start;
    nfa.end = nfa2.end;

    return nfa;
}
```

下图是对以上代码的解释。



（3） 定义选择运算符

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 实验评分

```
NFA orNFA(NFA nfa1, NFA nfa2) {
    nfaNode* start = new nfaNode();
    nfaNode* end = new nfaNode();

    start->isStart = true;
    end->isEnd = true;

    // 把新的初态与nfa1和nfa2的初态连接起来
    nfaEdge edge1;
    edge1.c = EPSILON;
    edge1.next = nfa1.start;
    start->edges.push_back(edge1);
    nfa1.start->isStart = false;    // 将nfa1的原初态修改为普通状态

    nfaEdge edge2;
    edge2.c = EPSILON;
    edge2.next = nfa2.start;
    start->edges.push_back(edge2);
    nfa2.start->isStart = false;    // 将nfa2的原初态修改为普通状态

    // 把nfa1和nfa2的终止状态与新的终止状态连接起来
    nfa1.end->isEnd = false;
    nfa2.end->isEnd = false;

    nfaEdge edge3;
    edge3.c = EPSILON;
    edge3.next = end;
    nfa1.end->edges.push_back(edge3);

    nfaEdge edge4;
    edge4.c = EPSILON;
    edge4.next = end;
    nfa2.end->edges.push_back(edge4);

    NFA nfa(start, end);

    return nfa;
}
```

下图是对以上代码的解释。

华 南 师 范 大 学 实 验 报 告

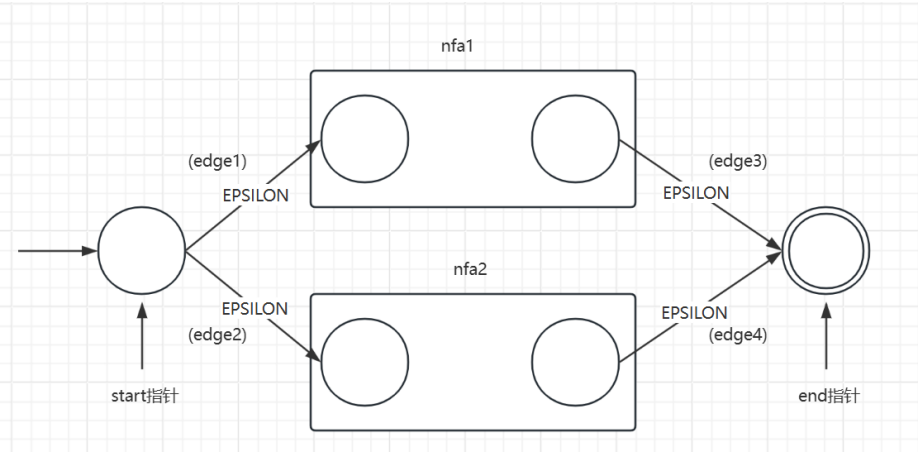
学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 实验评分



（4） 定义闭包运算符

```
NFA closureNFA(NFA nfa1) {
    nfaNode* start = new nfaNode();
    nfaNode* end = new nfaNode();

    start->isStart = true;
    end->isEnd = true;

    // 令nfa1的新初态指向nfa1的原初态
    nfaEdge edge1;
    edge1.c = EPSILON;
    edge1.next = nfa1.start;
    start->edges.push_back(edge1);
    nfa1.start->isStart = false;    // 将原初态修改为普通状态

    // 令nfa1的新初态指向nfa1的新终态
    nfaEdge edge2;
    edge2.c = EPSILON;
    edge2.next = end;
    start->edges.push_back(edge2);

    // 令nfa1的原终止状态指向nfa1的原初始状态
    nfa1.end->isEnd = false;

    nfaEdge edge3;
    edge3.c = EPSILON;
    edge3.next = nfa1.start;
    nfa1.end->edges.push_back(edge3);

    // 令nfa1的原终态指向nfa1的新终态
    nfaEdge edge4;
    edge4.c = EPSILON;
    edge4.next = end;
    nfa1.end->edges.push_back(edge4);

    NFA nfa(start,end);

    return nfa;
}
```

下图是对以上代码的解释。

华 南 师 范 大 学 实 验 报 告

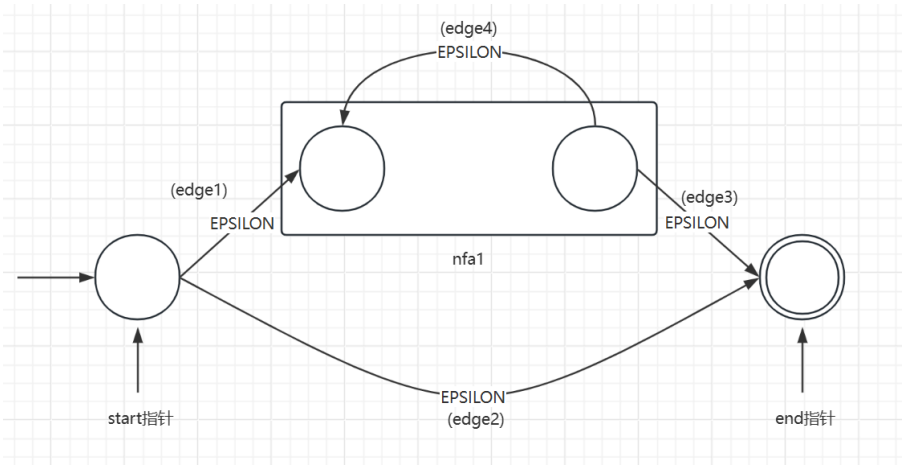
学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 实验评分



（5） 定义可选运算符

```
// 可选运算符的NFA图
// ?的NFA图其实就是无需自环的闭包NFA图
NFA optionalNFA(NFA nfa1) {
    nfaNode* start = new nfaNode();
    nfaNode* end = new nfaNode();

    start->isStart = true;
    end->isEnd = true;

    // 令nfa1的新初态指向nfa1原初态
    nfaEdge edge1;
    edge1.c = EPSILON;
    edge1.next = nfa1.start;
    start->edges.push_back(edge1);
    nfa1.start->isStart = false;    // 初态结束

    // 令nfa1的新初态指向nfa1新终态
    nfaEdge edge2;
    edge2.c = EPSILON;
    edge2.next = end;
    start->edges.push_back(edge2);

    // 令nfa1的原终态指向nfa1原终态
    nfa1.end->isEnd = false;

    nfaEdge edge3;
    edge3.c = EPSILON;
    edge3.next = end;
    nfa1.end->edges.push_back(edge3);

    NFA nfa(start, end);

    return nfa;
}
```

下图是对以上代码的解释。

华 南 师 范 大 学 实 验 报 告

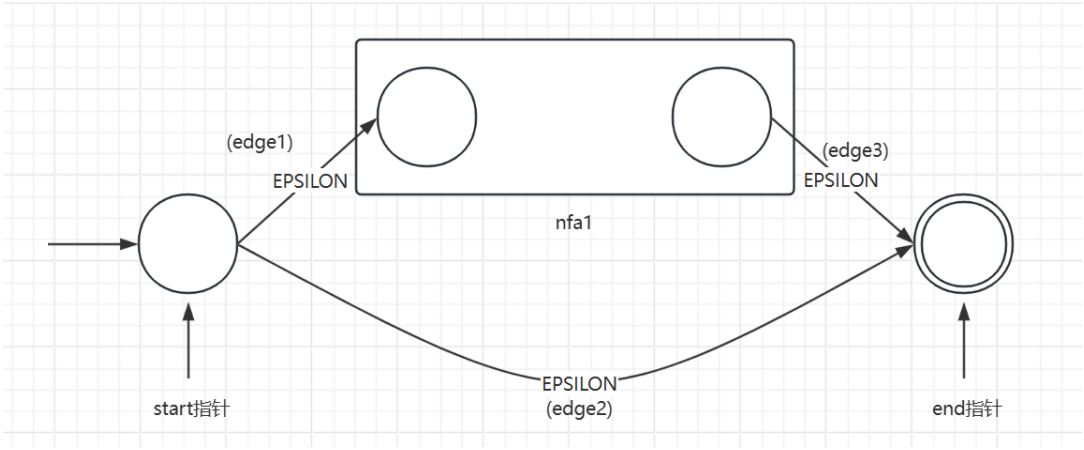
学生姓名 肖翔 学　号 20222132002

专　业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 实验评分



使用 regexToNFA 后会的到一个 NFA 图。我们还需要调用 formNFAStatus 函数，对这个 NFA 进行深度优先遍历，以便记录 NFA 中各个结点的状态转换表。

3. NFA 转 DFA

将 NFA 转换成 DFA 的第一步就是要构建 NFA 初态的 epsilon 闭包。系统使用 epsilonClosure 函数构建 epsilon 闭包。在构建过程中需要用到栈和集合。将 NFA 起始结点压入栈，栈不为空，然后获取可以从起始结点通过 epsilon 到达的结点集合，遍历这个结点集合，将结点都压入栈中并存入结果集合。然后再从栈中取出结点，获取可以从该结点通过 epsilon 到达的结点集合，遍历这个结点集合，将结点都压入栈中并存入结果集合。重复以上过程，直至栈空。

```
set<int> epsilonClosure(int id)
{
    set<int> eResult{ id };     // 结点本身也属于epsilon闭包
    stack<int> stack;
    stack.push(id);

    while (!stack.empty())
    {
        int current = stack.top();
        stack.pop();

        set<int> eClosure = statusTable[current].transition[EPSILON];    // nfa中current结点的id仅通过一次epsilon
        for (auto t : eClosure)
        {
            // 防止遇到值全是epsilon的边组成的圈而导致陷入死循环
            if (eResult.find(t) == eResult.end())
            {
                eResult.insert(t);
                stack.push(t);          // 通过epsilon到达当前结点后，还需查看当前结点是否能通过epsilon到达其它结点
            }
        }
    }

    return eResult;     // epsilon闭包
}
```

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 学 号 20222132002

专 业 网络工程 年级、班级 2022 级网工二班

课程名称 编译原理 实验项目 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 实验评分

得到 NFA 起始结点的 epsilon 闭包后，就要构建 epsilon 闭包中每个结点对应基本字符的转移闭包。系统使用 transitionClosure 函数构建转移闭包。转移闭包非常容易构建，求某个结点针对某个基本字符的转移闭包就是要求结点针对基本字符的转移结果以及转移结果的 epsilon 闭包[1]。

```
// 转换闭包
set<int> transitionClosure(int source, char ch)
{
    set<int> result{};

    set<int> chClosure = statusTable[source].transition[ch];
    for (auto o : chClosure)
    {
        result.insert(o);
        auto tmp = epsilonClosure(o);
        result.insert(tmp.begin(), tmp.end());      // 当前结点的字符闭包就是当前结点通过字符指向的下一个结
    }

    return result;
}
```

在起始结点的 epsilon 闭包中，针对同一基本字符的各个结点的转移闭包都放在同一个集合中，这个集合就是 DFA 结点。系统会对相同的集合去重。去重后的集合会被压入双端队列的头部。假设起始结点的 epsilon 闭包是{1, 2, 3}，闭包中的 NFA 结点针对基本字符 a 的转移闭包构成的集合是{4, 5, 6}，针对基本字符 b 的转移闭包构成的集合是{7, 8, 9}，那么{4, 5, 6}和{7, 8, 9}都会被压入队列中，这两个结合是 DFA 的结点。若针对 a、b 构成的集合都是{4, 5, 6}，由于系统的去重机制（set<set<int>>），仅有一个{4, 5, 6}能够压入队列。

从队列尾部取出集合，求这个集合中每个 NFA 结点针对同一基本字符的转移闭包，每个 NFA 结点的转移闭包构成了一个集合，这个集合也是 DFA 结点。对新构成的集合去重，然后将集合压入队列头部。重复以上步骤，直至队列为空，此时 NFA 已经转成 DFA。

4. DFA 最小化

首先，将 DFA 的状态分为终态和非终态，存入 dividedSetVec（一个 set 型 vector，用于存放分割过程中产生的集合）中。

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　号** 20222132002

专　　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 **实验评分**

通过 divideSet 函数，针对每个字符，遍历 dividedSetVec 中的状态集合，检查集合中每个状态针对某个基本字符的转移关系。如果发现某个状态集合 A 内的某个状态在基本字符下转移到的状态集合 B，与状态集合 A 内其他状态通过同一基本字符转移到的状态集合不相同，就将这个状态分割出来，形成新的状态集合。比如，状态集合 A={1, 2, 3}中的状态 1 通过 a 转移到 A 本身，而状态 2 和状态 3 通过 a 无法转移到任何一个状态集合，那么应该将状态 1 分割出来，形成新的状态集合。而判断某几个状态是否同属于一个状态集合则需要用到 map<int, int> dfaSameMinSet，状态只要被划分到同一个集合就会被映射成同样的值。

判断此过程在 while 循环中重复进行，直到无法再细分为止。无法再细分的标准是 dividedSetVec 经过分割函数（divideSet）处理前后的大小没有发生变化。遍历所有划分后的状态集合，构建最小化后的 DFA 节点并将其存入 dfaMinVector，同时更新状态转移映射。

（三）测试

测试结果详见作业文件夹。

四、实验总结（心得体会）

此次实验加深了我对 DFA 和 NFA 的理解，同时让我对正则表达式转 NFA 的过程、NFA 转 DFA 的过程和 DFA 最小化的过程更加熟悉。

正则表达式转换为 NFA 的过程需要用到 Thompson 构造法，只需通过固定步骤就能轻松地将一个正则表达式转换为NFA。同时还需要对构造出的NFA图进行深度遍历，描述各个 NFA 结点的状态转换表。

NFA 转 DFA 的过程的关键是需要理解 epsilon 闭包和转移闭包的概念。同

华 南 师 范 大 学 实 验 报 告

学生姓名 肖翔 **学　　号** 20222132002

专　　业 网络工程 **年级、班级** 2022 级网工二班

课程名称 编译原理 **实验项目** 正则表达式转换为 DFA 图

实验时间 2024 年 10 月 17 日

实验指导老师 黄煜廉 **实验评分**

时还需要对得到的集合进行去重。明白没有新状态集合产生就需要停止添加集合也是重要的。

DFA 最小化实际上就是模拟了基本字符分割集合的过程。

总而言之，此次实验让我将理论和实践相结合，加深了课堂上传授的知识，充分理解了确定型有穷自动机和非确定性有穷自动机。

五、参考文献：

[1] 《编译原理复习提纲》

[2] 编译原理：正则表达式/正规式转 NFA-CSDN 博客