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：XLEX-词法自动生成器

指导老师：黄煜廉

开课时间：2023 ～ 2024 年度第 1 学期

专　　业：计算机科学与技术

班　　级：计科 1 班

学　　生：李达良

学　　号：20203231004

华南师范大学教务处

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 **学　号** 20203231004

专　业 计算机科学与技术 **年级、班级** 2021 级 1 班

课程名称 编译原理 **实验项目** XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 **实验评分**

一、实验题目

XLEX-词法自动生成器

二、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

三、实验目的

（1）正则表达式应该支持单个字符，运算符号有： 连接、选择（|）、闭包（*）、括号（）、可选（?）扩充正则表达式的运算符号，如 [ ] 、 正闭包（+） 等。

（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）需要提供窗口以便用户可以查看转换得到的词法分析程序（该分析程序需要用 C/C++语言描述）

（7）用户界面应该是 windows 界面

（9）应该书写完善的软件文档

四、实验文档

（1）程序界面设计

通过 QT 实现 UI 的设计，给予一个正则表达式输入框，输入框旁边有两个按钮，分别是上传 txt 和下载正则表达式，方便使用者的输入和保存。填写正则表达式后，点击开始分析，分析成功后，点击“NFA”、“DFA”、“DFA 最小化”、“C++程序”就可以获得对应的结果，操作简单便捷。

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

（2）程序逻辑设计

对于本程序，我们主要分为五大块：正则表达式前置处理、NFA 生成、DFA 生成、DFA 最小化和 C++程序的生成。

2.1 正则表达式前置处理

当我们从文件中读出正则表达式或者输入正则表达式到输入框中后，我们需要先对正则表达式进行一些前置处理。

因为题目要求我们可以输出多行正则表达式，所以我们对每一行正则表达式进行**或运算**的拼接，具体算法是用 std 中的 getline 函数，放入 vector 中，在放入的同时我们进行非空判断，以防用户输入了空字符串，同时取出来的时候，加上括号和或运算符：

```
// 多行处理
QString handleMoreLine(QString regex)
{
    string regexStd = regex.toStdString();
    vector<std::string> lines;
    istringstream iss(regexStd);
    string line;

    // 防止中间有换行符
    while (std::getline(iss, line)) {
```

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
    if (!line.empty()) {
        lines.push_back(line);
    }
}

std::string output;

for (size_t i = 0; i < lines.size(); ++i) {
    output += "(" + lines[i] + ")";
    if (i < lines.size() - 1) {
        output += "|";
    }
}

return QString::fromStdString(output);

}
```

多行处理完毕后，由于题目有额外要求，要求我们支持[]运算符和正闭包（+），所以我们可以先对正则表达式进行处理，转换成基本的符号，比如[]其实就是把里面的元素用或运算符进行拼接，正闭包就是转换成形如 ss*的形式。同时我们对连接符号定义成”.”的形式，因此我们可以得到下面的处理算法：

```
// 判断是不是字符
bool isChar(char c)
{
    if ((c >= 'a' && c <= 'z') || (c >= 'A' && c <= 'Z'))
        return true;
    return false;
}

// 处理正则表达式（加.符号，方便后续操作）
QString handleRegex(QString regex)
{
    string regexStd = regex.toStdString();
    qDebug() << "enter handleRegex: " << regex;

    // 处理中括号
    string result;
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
bool insideBrackets = false;
string currentString;

for (char c : regexStd) {
    if (c == '[') {
        insideBrackets = true;
        currentString.push_back('(');
    }
    else if (c == ']') {
        insideBrackets = false;
        currentString.push_back(')');
        result += currentString;
        currentString.clear();
    }
    else if (insideBrackets) {
        if (currentString.length() > 1) {
            currentString.push_back('|');
        }
        currentString.push_back(c);
    }
    else {
        result.push_back(c);
    }
}

regexStd = result;

//先处理+号
for (int i = 0; i < regexStd.size(); i++)
{
    if (regexStd[i] == '+')
    {
        int kcount = 0;
        int j = i;
        do
        {
            j--;
            if (regexStd[j] == ')')
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
                {
                    kcount++;
                }
                else if (regexStd[j] == '(')
                {
                    kcount--;
                }
            } while (kcount != 0);
            string str1 = regexStd.substr(0, j);
            string kstr = regexStd.substr(j, i - j);
            string str2 = regexStd.substr(i + 1, (regexStd.size() - i));
            regexStd = str1 + kstr + kstr + "*" + str2;
        }
    }

    for (int i = 0; i < regexStd.size() - 1; i++)
    {
        if (isChar(regexStd[i]) && isChar(regexStd[i + 1])
            || isChar(regexStd[i]) && regexStd[i + 1] == '('
            || regexStd[i] == ')' && isChar(regexStd[i + 1])
            || regexStd[i] == ')' && regexStd[i + 1] == '('
            || regexStd[i] == '*' && regexStd[i + 1] != ')' && regexStd[i + 1] != '|'
&& regexStd[i + 1] != '?'
            || regexStd[i] == '?' && regexStd[i + 1] != ')'
            || regexStd[i] == '+' && regexStd[i + 1] != ')')
        {
            string str1 = regexStd.substr(0, i + 1);
            string str2 = regexStd.substr(i + 1, (regexStd.size() - i));
            str1 += ".";
            regexStd = str1 + str2;
        }

    }
    return QString::fromStdString(regexStd);
}
```

2.2 NFA 生成

2.2.1 数据结构

对于 NFA，我们需要一个 NFA 的图结构来方便我们对 NFA 进行遍历进行一个后续的操作，所以我们首先可以定义 NFA 结点和它的边：

```
struct nfaNode; // 声明一下，不然报错

// 定义NFA图的边
struct nfaEdge
{
    char c;
    nfaNode* next;
};

// 定义一个nfa图的结点
struct nfaNode
{
    int id; // 结点唯一编号
    bool isStart;   // 初态标识
    bool isEnd; // 终态标识
    vector<nfaEdge> edges;  // 边，用vector因为有可能一个结点有多条边可走
    nfaNode() {
        id = nodeCount++;
        isStart = false;
        isEnd = false;
    }
};
```

因为一个 NFA 状态可以连接多条边到多个 NFA 状态，我们使用了一个 vector 来存每个 NFA 状态的边，对于这个边，我们用 nfaEdge 这个结构通过哪个字符指向了下一个结点是哪。同时对这个 NFA 状态，我们定义了初态和终态的标识，方便我们之后来识别初态和终态。

2.2.2 核心算法

对于正则表达式转 NFA 图，其实我们可以参考逆波兰表达式的双栈法，对正则表达式进行一个扫描，用两个栈，一个栈存入运算符，一个栈存 NFA 图，定义好正则表达式运算符的优先级。

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

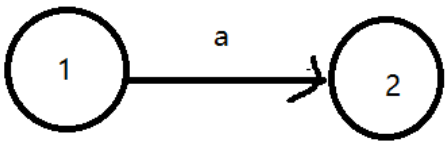
专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

当我们遇到一个字符的时候，我们就要创建一个 NFA 图，如下图，同时初态和终态应该都为 true：




```
// 创建基本字符NFA，即只包含一个字符的NFA图
NFA CreateBasicNFA(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);

    // 存入全局nfa字符set
    nfaCharSet.insert(character);
    // 存入全局dfa字符set
    dfaCharSet.insert(character);

    return nfa;
}
```

然后，假如我们遇到一个运算符，我们得判断优先级来进行处理，以或运算为例子：a|b，我们事先一定是存入了 a 和 b 的 NFA 图到 NFA 栈中，pop 出来后，我们要对他们进行连接，就是要创建一个开始节点和结束节点，并用一个 ε 边连接

华 南 师 范 大 学 实 验 报 告

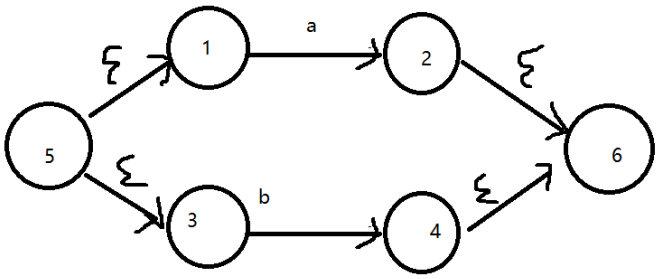
学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分



因此我们就能写出或运算的算法：

```
// 创建选择（|）运算符的NFA图
NFA CreateUnionNFA(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;    // 初态结束

    nfaEdge edge2;
    edge2.c = EPSILON;
    edge2.next = nfa2.start;
    start->edges.push_back(edge2);
    nfa2.start->isStart = false;    // 初态结束

    // 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;
    }
```

其他运算符也是同理，采用双栈法后（具体代码参考源程序，代码行数较多），我们会得到一个完整的 NFA 图，但我们要将其转换成状态转换表，所以，我们要用 DFS 深搜，得到每个节点在每个字符下的一个状态转换，方便我们后续的处理：

```
struct statusTableNode
{
    string flag;  // 标记初态还是终态
    int id; // 唯一id值
    map<char, set<int>> m;  // 对应字符能到达的状态
    statusTableNode()
    {
        flag = ""; // 默认为空
    }
};

// 全局存储状态转换表
unordered_map<int, statusTableNode> statusTable;
// statusTable插入顺序记录，方便后续输出
vector<int> insertionOrder;
set<int> startNFAstatus;
set<int> endNFAstatus;

// 对NFA图进行DFS，形成状态转换表
void createNFAStatusTable(NFA& nfa)
{
    stack<nfaNode*> nfaStack;
    set<nfaNode*> visitedNodes;
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
// 初态
nfaNode* startNode = nfa.start;
statusTableNode startStatusNode;
startStatusNode.flag = '-'; // -表示初态
startStatusNode.id = startNode->id;
statusTable[startNode->id] = startStatusNode;
insertionOrder.push_back(startNode->id);
startNFAstatus.insert(startNode->id);

nfaStack.push(startNode);

while (!nfaStack.empty()) {
    nfaNode* currentNode = nfaStack.top();
    nfaStack.pop();
    visitedNodes.insert(currentNode);

    for (nfaEdge edge : currentNode->edges) {
        char transitionChar = edge.c;
        nfaNode* nextNode = edge.next;

        // 记录状态转换信息
        statusTable[currentNode->id].m[transitionChar].insert(nextNode->id);

        // 如果下一个状态未被访问，将其加入堆栈
        if (visitedNodes.find(nextNode) == visitedNodes.end()) {
            nfaStack.push(nextNode);

            // 记录状态信息
            statusTableNode nextStatus;
            nextStatus.id = nextNode->id;
            if (nextNode->isStart) {
                nextStatus.flag += '-'; // -表示初态
                startNFAstatus.insert(nextStatus.id);
            }
            else if (nextNode->isEnd) {
                nextStatus.flag += '+'; // +表示终态
                endNFAstatus.insert(nextStatus.id);
            }
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学　号 20203231004

专　业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

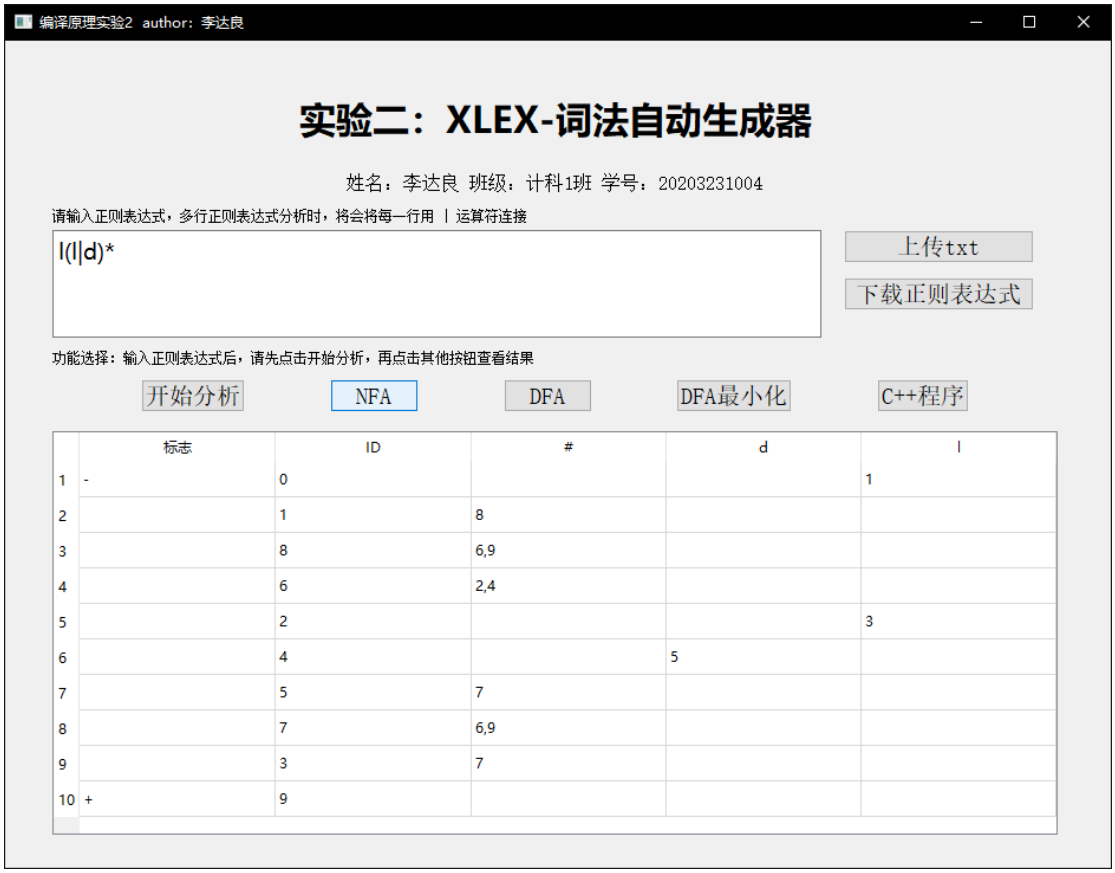
实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
                statusTable[nextNode->id] = nextStatus;
                // 记录插入顺序（排除终态）
                if (!nextNode->isEnd)
                {
                    insertionOrder.push_back(nextNode->id);
                }
            }
        }
    }

    // 顺序表才插入终态
    nfaNode* endNode = nfa.end;
    insertionOrder.push_back(endNode->id);
}
```

最后可以得到 statusTable，可以对其进行展示：



	标志	ID	#	d	l
1	-	0			1
2		1	8		
3		8	6,9		
4		6	2,4		
5		2			3
6		4		5	
7		5	7		
8		7	6,9		
9		3	7		
10	+	9			

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

2.3 DFA 生成

2.3.1 数据结构

对于 DFA 图，数据结构其实和 NFA 是差不多的，但是 DFA 需要检测有无新状态产生，所以我们要有一个 set 进行存储，通过对 set 大小的判断来知道是否有新状态的产生。

```
// dfa节点
struct dfaNode
{
    string flag; // 是否包含终态（+）或初态（-）
    set<int> nfaStates; // 该DFA状态包含的NFA状态的集合
    map<char, set<int>> transitions; // 字符到下一状态的映射
    dfaNode() {
        flag = "";
    }
};
// dfa状态去重集
set<set<int>> dfaStatusSet;

// dfa最终结果
vector<dfaNode> dfaTable;

//下面用于DFA最小化
// dfa终态集合
set<int> dfaEndStatusSet;
// dfa非终态集合
set<int> dfaNotEndStatusSet;
// set对应序号MAP
map<set<int>, int> dfa2numberMap;
int startStaus;
```

2.3.2 核心算法

对于 DFA 图，我们已经得到了 NFA 的状态表，我们其实可以很容易得到一个状态的 ε 闭包和字符闭包。从 NFA 的起始状态开始，计算其 ε 闭包，并将这个状态作为 DFA 的初始状态。然后，对每个字符集合（DFA 字符集）进行遍历，计算字符闭包，生成新的状态集合，并建立字符到下一状态的映射。如果生成的状态集合是新的，则将其添加到 DFA 状态集中，同时判断是否包含终态，然后将该状态继续作为下一个状态进行处理，直到没有新状态生成。

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

下面函数是计算 ε 闭包：

```
set<int> epsilonClosure(int id)
{
    set<int> eResult{ id };
    stack<int> stack;
    stack.push(id);

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

        set<int> eClosure = statusTable[current].m[EPSILON];
        for (auto t : eClosure)
        {
            if (eResult.find(t) == eResult.end())
            {
                eResult.insert(t);
                stack.push(t);
            }
        }
    }

    return eResult;
}
```

下面函数是计算字符闭包：

```
set<int> otherCharClosure(int id, char ch)
{
    set<int> otherResult{};
    set<int> processed;
    stack<int> stack;
    stack.push(id);

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

```
        if (processed.find(current) != processed.end())
            continue;

        processed.insert(current);

        set<int> otherClosure = statusTable[current].m[ch];
        for (auto o : otherClosure)
        {
            auto tmp = epsilonClosure(o);
            otherResult.insert(tmp.begin(), tmp.end());
            stack.push(o);
        }
    }

    return otherResult;
}
```

在判断有没有新状态生成，其实我们将所有状态放入一个 set 中，去重，看看这个 size 有没有发生变化，如果没有发生变化，就是没有新状态产生。

整个 NFA 到 DFA 的转换过程遵循子集构造法，它通过计算 ε 闭包和字符闭包来确定 DFA 的状态转移。最终，得到了一个表示等价 DFA 的 dfaTable，其中包含了 DFA 节点的信息。

核心算法如下：

```
void NFA2DFA(NFA& nfa)
{
    int dfaStatusCount = 1;
    auto start = nfa.start; // 获得NFA图的起始位置
    auto startId = start->id;   // 获得起始编号
    dfaNode startDFANode;
    startDFANode.nfaStates = epsilonClosure(startId); // 初始闭包
    startDFANode.flag = setHasStartOrEnd(startDFANode.nfaStates); // 判断初态终态
    deque<set<int>> newStatus{};
    dfa2numberMap[startDFANode.nfaStates] = dfaStatusCount;
    startStaus = dfaStatusCount;
    if (setHasStartOrEnd(startDFANode.nfaStates).find("+") != string::npos) {
        dfaEndStatusSet.insert(dfaStatusCount++);
    }
    else
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
    {
        dfaNotEndStatusSet.insert(dfaStatusCount++);
    }
    // 对每个字符进行遍历
    for (auto ch : dfaCharSet)
    {
        set<int> thisChClosure{};
        for (auto c : startDFANode.nfaStates)
        {
            set<int> tmp = otherCharClosure(c, ch);
            thisChClosure.insert(tmp.begin(), tmp.end());
        }
        if (thisChClosure.empty())  // 如果这个闭包是空集没必要继续下去了
        {
            continue;
        }
        int presize = dfaStatusSet.size();
        dfaStatusSet.insert(thisChClosure);
        int lastsize = dfaStatusSet.size();
        // 不管一不一样都是该节点这个字符的状态
        startDFANode.transitions[ch] = thisChClosure;
        // 如果大小不一样，证明是新状态
        if (lastsize > presize)
        {
            dfa2numberMap[thisChClosure] = dfaStatusCount;
            newStatus.push_back(thisChClosure);
            if (setHasStartOrEnd(thisChClosure).find("+") != string::npos) {
                dfaEndStatusSet.insert(dfaStatusCount++);
            }
            else
            {
                dfaNotEndStatusSet.insert(dfaStatusCount++);
            }

        }

    }
    dfaTable.push_back(startDFANode);
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
    // 对后面的新状态进行不停遍历
    while (!newStatus.empty())
    {
        // 拿出一个新状态
        set<int> ns = newStatus.front();
        newStatus.pop_front();
        dfaNode DFANode;
        DFANode.nfaStates = ns;  // 该节点状态集合
        DFANode.flag = setHasStartOrEnd(ns);

        for (auto ch : dfaCharSet)
        {

            set<int> thisChClosure{};
            for (auto c : ns)
            {
                set<int> tmp = otherCharClosure(c, ch);
                thisChClosure.insert(tmp.begin(), tmp.end());
            }
            if (thisChClosure.empty())  // 如果这个闭包是空集没必要继续下去了
            {
                continue;
            }
            int presize = dfaStatusSet.size();
            dfaStatusSet.insert(thisChClosure);
            int lastsize = dfaStatusSet.size();
            // 不管一不一样都是该节点这个字符的状态
            DFANode.transitions[ch] = thisChClosure;
            // 如果大小不一样，证明是新状态
            if (lastsize > presize)
            {
                dfa2numberMap[thisChClosure] = dfaStatusCount;
                newStatus.push_back(thisChClosure);
                if (setHasStartOrEnd(thisChClosure).find("+") != string::npos) {
                    dfaEndStatusSet.insert(dfaStatusCount++);
                }
                else
```

```
                {
                    dfaNotEndStatusSet.insert(dfaStatusCount++);
                }

            }

        }
        dfaTable.push_back(DFANode);

    }
}
```

最后进行输出：

2.4 DFA 最小化

2.4.1 数据结构

数据结构和 dfa 差不多，多了一个用于分割集合和存下标的数据结构。

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
// dfa最小化节点
    struct dfaMinNode
    {
        string flag; // 是否包含终态（+）或初态（-）
        int id;
        map<char, int> transitions; // 字符到下一状态的映射
        dfaMinNode() {
            flag = "";
        }
    };

    vector<dfaMinNode> dfaMinTable;

    // 用于分割集合
    vector<set<int>> divideVector;
    // 存下标
    map<int, int> dfaMinMap;
```

2.4.2 核心算法

设置一个分割函数，该函数用于根据字符 ch 将状态集合 node 分成两个子集合。它通过遍历状态集合中的状态，根据字符 ch 找到下一个状态，然后根据下一个状态的映射来决定是否分割成新的状态集合。分割后，删除需要删除的元素，将新的状态集合加入到 vector 的末尾中，实现 DFA 状态的最小化，同时在 dfaMinMap 中更新状态到下标的映射。

```
// 根据字符 ch 将状态集合 node 分成两个子集合
void splitSet(int i, char ch)
{
    set<int> result;
    auto& node = divideVector[i];
    int s = -2;

    for (auto state : node)
    {
        int thisNum;
        if (dfaTable[state - 1].transitions.find(ch) == dfaTable[state -
1].transitions.end())
        {
```

```
            thisNum = -1; // 空集
        }
        else
        {
            // 根据字符 ch 找到下一个状态
            int next_state = dfa2numberMap[dfaTable[state - 1].transitions[ch]];
            thisNum = dfaMinMap[next_state];    // 这个状态的下标是多少
        }

        if (s == -2)    // 初始下标
        {
            s = thisNum;
        }
        else if (thisNum != s)   // 如果下标不同，就是有问题，需要分出来
        {
            result.insert(state);
        }
    }

    // 删除要删除的元素
    for (int state : result) {
        node.erase(state);
    }

    // 都遍历完了，如果result不是空，证明有新的，加入vector中
    if (!result.empty())
    {
        divideVector.push_back(result);
        // 同时更新下标
        for (auto a : result)
        {
            dfaMinMap[a] = divideVector.size() - 1;
        }
    }

}
```

在调用这个分割函数的核心算法中，执行以下步骤：

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

a. 初始化 divideVector 和 dfaMinMap，并将非终态和终态集合添加到 divideVector 中，初始化状态到下标的映射。

b. 进入循环，直到不再有新的状态分割。在每次循环中，遍历 divideVector 中的每个状态集合，然后逐个字符尝试分割状态集合。分割时使用 splitSet 函数。

c. 最小化过程中，会不断地合并相同状态，直到不再有新的状态分割。这个过程确保了生成最小化的 DFA。

d. 创建最小化的 DFA 节点 dfaMinNode，并添加到 dfaMinTable 中。

```
void DFAminimize()
{
    divideVector.clear();
    dfaMinMap.clear();

    // 存入非终态、终态集合
    if (dfaNotEndStatusSet.size() != 0)
    {
        divideVector.push_back(dfaNotEndStatusSet);
    }
    // 初始化map
    for (auto t : dfaNotEndStatusSet)
    {
        dfaMinMap[t] = divideVector.size() - 1;
    }

    divideVector.push_back(dfaEndStatusSet);

    for (auto t : dfaEndStatusSet)
    {
        dfaMinMap[t] = divideVector.size() - 1;
    }

    // 当flag为1时，一直循环
    int continueFlag = 1;

    while (continueFlag)
    {
        continueFlag = 0;
        int size1 = divideVector.size();
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
    for (int i = 0; i < size1; i++)
    {

        // 逐个字符尝试分割状态集合
        for (char ch : dfaCharSet)
        {
            splitSet(i, ch);
        }
    }
    int size2 = divideVector.size();
    if (size2 > size1)
    {
        continueFlag = 1;
    }
}

for (int dfaMinCount = 0; dfaMinCount < divideVector.size(); dfaMinCount++)
{
    auto& v = divideVector[dfaMinCount];
    dfaMinNode d;
    d.flag = minSetHasStartOrEnd(v);
    d.id = dfaMinCount;
    // 逐个字符
    for (char ch : dfaCharSet)
    {
        if (v.size() == 0)
        {
            d.transitions[ch] = -1;   // 空集特殊判断
            continue;
        }
        int i = *(v.begin()); // 拿一个出来
        if (dfaTable[i - 1].transitions.find(ch) == dfaTable[i -
1].transitions.end())
        {
            d.transitions[ch] = -1;   // 空集特殊判断
            continue;
        }
        int next_state = dfa2numberMap[dfaTable[i - 1].transitions[ch]];
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
            int thisNum = dfaMinMap[next_state];    // 这个状态下标
            d.transitions[ch] = thisNum;
        }
        dfaMinTable.push_back(d);
    }

    // 输出 dfaMinTable
    for (const dfaMinNode& node : dfaMinTable) {
        qDebug() << "State " << node.id << ":";
        qDebug() << "Flag: " << QString::fromStdString(node.flag);

        for (const auto& entry : node.transitions) {
            qDebug() << entry.first << " -> " << entry.second;
        }
    }

    qDebug() << "DFA最小化完成！";
}
```

至此，DFA 最小化完成，我们可以对其进行展示：

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

2.5 C++代码生成

主要函数使用 ostringstream 类创建一个字符串流（codeStream），然后将代码逐步添加到流中。最终，将流的内容保存为字符串，即 resultCode。

包含头文件和命名空间：开始部分包含了 C++的必要头文件和使用了 using namespace std; 以使代码能够使用标准 C++库。

main 函数：生成的 C++代码的主入口是 main 函数，它接受用户输入的字符串，然后执行词法分析。以下是 main 函数的主要组成部分：

a. 创建 input 变量来存储用户输入的字符串。

b. 提示用户输入字符串，然后使用 cin 从标准输入获取用户输入。

c. 初始化 currentState 变量为 0，表示开始状态。

d. 获取输入字符串的长度，存储在 length 变量中。

e. 使用 for 循环遍历输入字符串中的每个字符，从左到右进行词法分析。

状态转移逻辑：在 for 循环内，通过 switch (currentState) 语句，针对当前状态执行状态转移逻辑。对于每个 DFA 状态 node，生成一个 case 分支，其中包括 switch (c) 语句，对当前字符 c 进行状态转移判断。

a. 对于每个输入字符，生成一个 case 分支，例如 case 'a':，并根据状态转移表更新 currentState。如果遇到无效输入字符，输出错误信息并返回。

b. 默认分支 default: 处理无效输入字符的情况，输出错误信息并返回。

接受状态判断：在循环结束后，再次使用 switch (currentState) 判断最终状态。

a.对于每个 DFA 状态 node，如果包含终态（node.flag 中包含 + 标志），生成一个 case 分支，例如 case 2:，并输出 "Accepted" 表示词法分析通过。

b. 默认分支 default: 处理不包含终态的情况，输出 "Not Accepted" 表示词法分析不通过。

最后，main 函数返回 0，表示程序正常退出。所有生成的代码存储在 resultCode 字符串中。

```
string resultCode;

// 生成词法分析器代码并返回为字符串
void generateLexerCode() {
    ostringstream codeStream; // Create a stringstream to capture the output

    codeStream << "#include <iostream>" << endl;
    codeStream << "#include <string>" << endl;
    codeStream << endl;
    codeStream << "using namespace std;" << endl;
    codeStream << endl;
    codeStream << "int main() {" << endl;
```

华 南 师 范 大 学 实 验 报 告

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
    codeStream << "    string input;" << endl;
    codeStream << "    cout << \"Enter input string: \";" << endl;
    codeStream << "    cin >> input;" << endl;
    codeStream << "    int currentState = 0;" << endl;
    codeStream << "    int length = input.length();" << endl;
    codeStream << "    for (int i = 0; i < length; i++) {" << endl;
    codeStream << "        char c = input[i];" << endl;
    codeStream << "        switch (currentState) {" << endl;

    for (const dfaMinNode& node : dfaMinTable) {
        codeStream << "            case " << node.id << ":" << endl;
        codeStream << "                switch (c) {" << endl;
        for (const auto& transition : node.transitions) {
            codeStream << "                    case '" << transition.first << "':"
<< endl;
            if (transition.second == -1) {
                codeStream << "                        cout << \"Error: Invalid
input character '\" << c << \"'\" << endl;";
                codeStream << "                        return 1;" << endl;
            }
            else {
                codeStream << "                        currentState = " <<
transition.second << ";" << endl;
            }
            codeStream << "                    break;" << endl;
        }
        codeStream << "                default:" << endl;
        codeStream << "                    cout << \"Error: Invalid input
character '\" << c << \"'\" << endl;" << endl;
        codeStream << "                    return 1;" << endl;
        codeStream << "                }" << endl;
        codeStream << "                break;" << endl;
    }

    codeStream << "        }" << endl;
    codeStream << "    }" << endl;
    codeStream << "    switch (currentState) {" << endl;
```

学生姓名 李达良 学 号 20203231004

专 业 计算机科学与技术 年级、班级 2021 级 1 班

课程名称 编译原理 实验项目 XLEX-词法自动生成器

实验时间 2023 年 10 月 21 日

实验指导老师 黄煜廉 实验评分

```
    for (const dfaMinNode& node : dfaMinTable) {
        if (node.flag.find("+") != string::npos) {
            codeStream << "        case " << node.id << ":" << endl;
            codeStream << "            cout << \"Accepted\" << endl;" << endl;
            codeStream << "            break;" << endl;
        }
    }

    codeStream << "        default:" << endl;
    codeStream << "            cout << \"Not Accepted\" << endl;" << endl;
    codeStream << "    }" << endl;
    codeStream << "    return 0;" << endl;
    codeStream << "}" << endl;

    resultCode = codeStream.str();
}
```

五、实验测试

详见《测试报告》。

六、小结

这个实验涉及了从 NFA 到 DFA 的转换，然后再对 DFA 进行最小化，并最后将其生成为 C++代码以用作简单的词法分析器。首先将一个给定的 NFA 转换为一个等效的 DFA。这一过程中使用了 NFA 的状态迁移、ε -闭包等核心概念。在转换的过程中，通过逐步构建 DFA 状态和状态迁移表，将一个可能非确定性的自动机转化为确定性的自动机。一旦获得 DFA，实验继续进行 DFA 的最小化。最小化的目的是简化 DFA，去除不必要的状态，并保留其等效性。实验使用了等价关系分割状态空间的方法，以减少 DFA 的状态数量，从而减小内存和计算要求。生成 C++代码是实验的最后一步，这段代码可用于构建一个基本的词法分析器。生成的代码使用 switch 语句来实现 DFA 的状态转移和词法分析，通过逐个字符处理输入字符串来识别词法符号。生成的代码包含了错误处理逻辑和最终状态的判断。

这个实验有助于理解自动机理论和如何将其应用于编写词法分析器。同时，通过将自动机转换和最小化的步骤应用于实际问题，可以学到如何将理论知识转化为实际应用，生成可以处理输入字符串的 C++代码。为词法分析提供了一种简单而有效的解决方案，可以用于从输入字符串中识别和提取特定的符号。

总的来说，这个实验提供了一个完整的学习过程，涵盖了自动机理论、状态转移、最小化以及代码生成，为理解和应用自动机在编程中的重要性提供了实际示例。