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：2023 ～ 2024 年度第 2 学期

专　业：计算机科学与技术

班　级：计科 2 班

学　生：林泽勋

学　号：20212821020

华南师范大学教务处

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 **学 号** 20212821020

专 业 计算机科学与技术 **年级、班级** 22 级计科 2 班

课程名称 编译原理 **实验项目** 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 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)

1. 必做实验内容及要求：（已完成）

（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。

2. 选做实验内容及要求：（已完成）

将最小化得到 DFA 图转换得到的词法分析源程序（该分析程序需要用 C/C++语言描述，而且只能采用讲稿中的转换方法一或转换方法二来生成源程序。），系统需提供窗口以便用户可以查看转换得到的词法分析源程序；

注意事项：如果选做该内容，实验文档以及使用说明书中就一定要有关于这个内容的设计文档及执行操作说明。

二、实验目的

1. 本实验旨在通过设计和实现一个应用软件，深入理解并掌握正则表达式到非确定性有限自动机（NFA）、确定性有限自动机（DFA）的转换过程，以及 DFA 的最小化技术。
2. 此外，实验还包括将最小化的 DFA 转换为词法分析程序的选做内容，这有助于加深对词法分析在编译原理中应用的理解。
3. 实验要求学生使用 C++语言，并在 Windows 界面下完成软件开发，同时提供完善的软件文档和测试报告。

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

实验过程概述

1. 需求分析与设计：首先，对实验要求进行了详细分析，确定了软件的基本功能和界面布局。设计了软件的架构，包括正则表达式的解析模块、NFA 和 DFA 的转换算法、DFA 最小化的算法以及词法分析程序的生成模块。
2. 编码实现：使用 C++语言编写了软件的各个模块。在实现过程中，特别注意了对正则表达式中特殊字符的转义处理，以及对不同运算符的支持。
3. 界面开发：开发了 Windows 界面，提供了源程序编辑、NFA 和 DFA 状态转换表展示、最小化 DFA 展示以及词法分析程序展示的窗口。
4. 测试与调试：编写了测试数据，对软件的各个功能进行了测试，确保了软件的正确性和稳定性。

三、实验文档：

(一)必做内容：

1. 系统总体结构

系统使用 Qt 框架，C++程序设计语言实现。主要包含如下组件：正则表达式输入、分析、展示组件，显示 NFA、DFA、最小化 DFA 的组件，词法分析程序展示组件。利用 Qt 事件 connect 进行绑定实现业务逻辑，完成各类分析与转换并显示的功能。

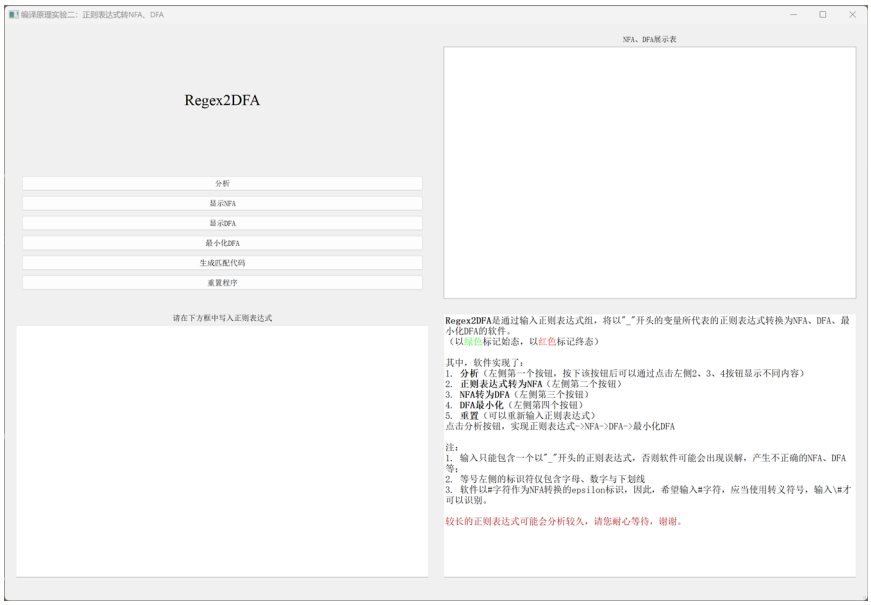

图 1 主窗口完整展示图

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

图 2 按钮展示图

下面对各个按钮功能进行详细介绍：

1) 分析并产生 NFA、DFA、最小化 DFA：当用户点击“分析”按钮时，程序会从文本编辑框中获取输入的正则表达式，然后进行一系列的转换，包括将正则表达式转换为后缀表达式，再转换为 NFA，接着转换为 DFA，最后进行 DFA 最小化。
2) 显示 NFA：当用户点击“显示 NFA”按钮时，程序会清空当前表格，并将 NFA 的状态转换表填充到表格中。同时，它会用不同的颜色标记开始状态和结束状态。
3) 显示 DFA：当用户点击“显示 DFA”按钮时，程序会执行类似的操作，但是这次是填充 DFA 的状态转换表。
4) 显示最小化的 DFA：当用户点击“显示最小化 DFA”按钮时，程序会展示最小化 DFA 的状态转换表。
5) 生成匹配代码：用户点击“生成代码”按钮时，会弹出一个对话框，展示从最小化 DFA 转换得到的词法分析源程序代码。
6) 重置程序：用户点击“重置”按钮时，程序会清除所有已生成的 NFA、DFA、最小化 DFA 和代码，并清空显示表格。

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

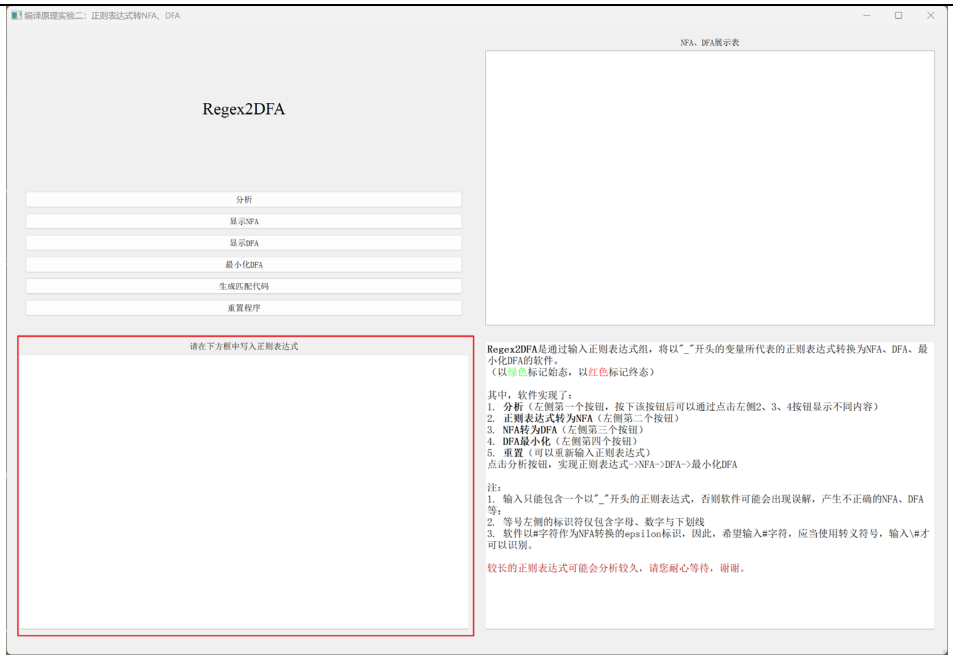

图 3 正则表达式写入位置

在图 3 位置中写入待分析的正则表达式组后，点击分析按钮，并点击需要展示的 NFA、DFA、最小化 DFA、匹配代码，即可完成上述功能。

图 4 NFA、DFA 展示图

在指定位置输入正则表达式组并分析后，点击显示 NFA、显示 DFA、最小化 DFA，就会在图 4 位置产生指定的数据表，以显示对应的数据状态与转

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

换。

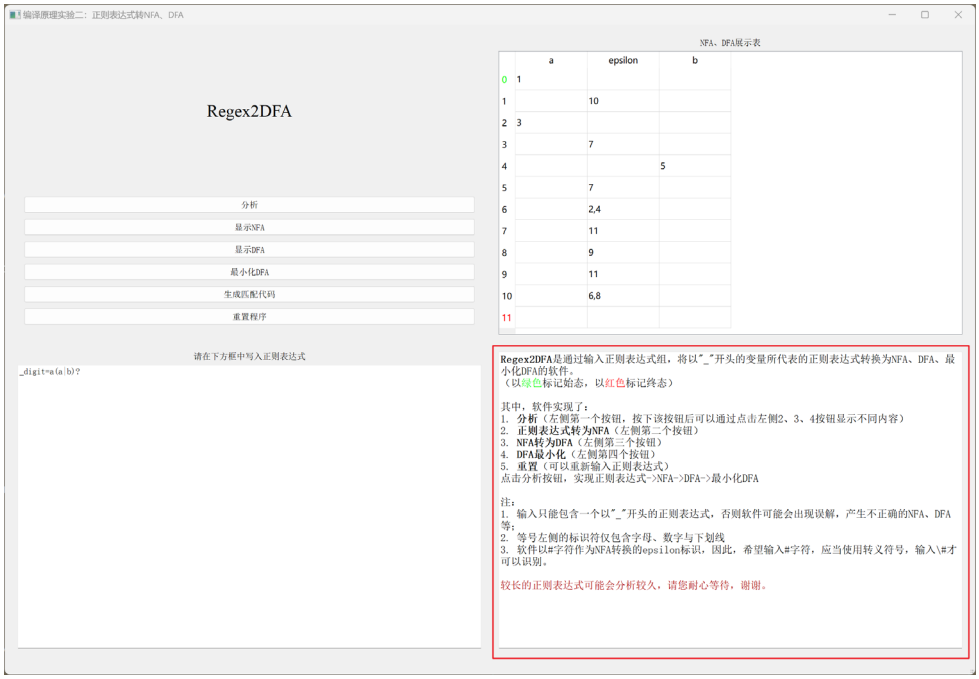

图 5 软件说明图

在图 5 位置展示了该程序的一些注意事项与功能概述，便于使用者了解使用方法，避免错误发生。

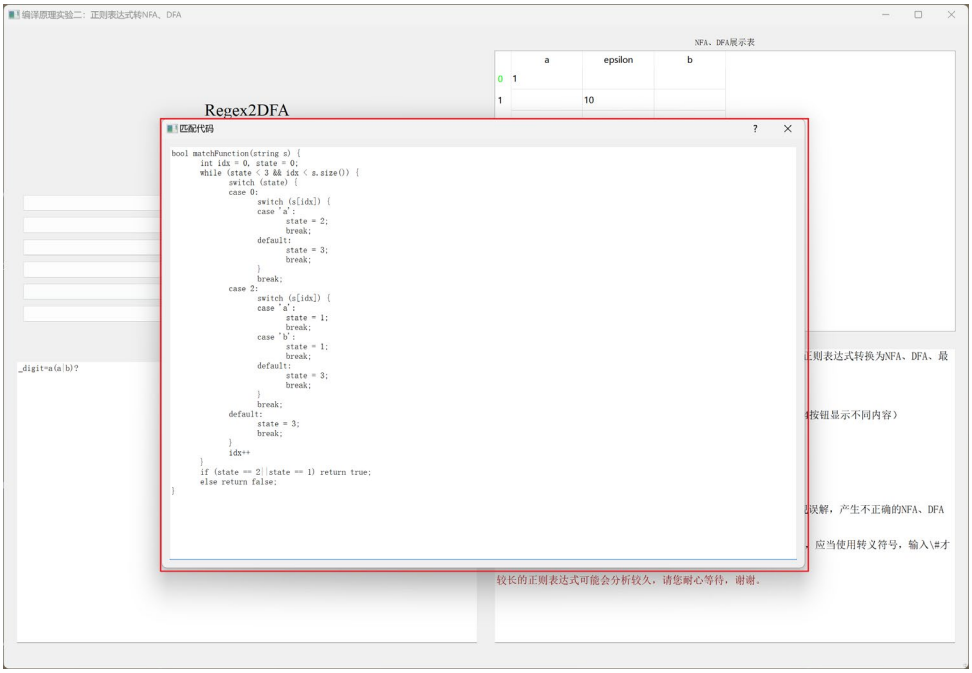

图 6 匹配代码弹窗

在点击生成匹配代码的按钮后，系统会弹出弹窗，产生指定正则表达式组的匹配代码，实现词法分析源程序的生成与展示。

2. 正则表达式预处理

程序设计了一个正则表达式的预处理函数，减少正则表达式的操作类型。regexListPreprocessing 函数的设计思路是处理和转换输入的正则表达式列表，使其适合后续的 NFA 构建过程。函数的主要步骤包括：

1) 去除所有空格：首先，函数遍历输入的正则表达式列表 regexList，移除每个正则表达式中的所有空白字符。
2) 替换字符集[]：函数遍历每个正则表达式，寻找方括号[]。当找到时，它会检查方括号内的字符，并将其转换为使用|（选择操作符）分隔的字符集合。例如，如果方括号表示一个字符范围，如[a-c]，则转换为 a|b|c。
3) 替换正闭包+：在正则表达式中，+ 表示一个字符的正闭包，即该字符可以出现一次或多次。函数会将+替换为*（表示 0 次或多次），并将紧随其后的字符或字符集括起来，以避免影响正则表达式的其他部分。
4) 替换可选?：可选操作符?表示前面的字符可以出现 0 次或 1 次。函数将?替换为|，并在紧随其后的字符或字符集中添加一个空操作，表示字符可以不出现。
5) 获取主正则表达式：函数会从处理后的正则表达式列表中寻找以下划线_开头的正则表达式，这被认为是主正则表达式。同时，它会从等号=左侧获取标识符，并将其与右侧的正则表达式体关联起来，存储在 regexHash 哈希表中。
6) 合并正则表达式：使用 regexHash 中的映射关系，将标识符替换为对应的正则表达式体。这允许函数扩展和替换使用命名操作定义的复杂表达式。
7) 添加连接符：最后，函数遍历主正则表达式，添加必要的连接符.，以确保正则表达式中的各个部分能够正确地串联起来。例如，它在转义字符后面、闭合括号前或在运算符（如*或)）前添加.，以确保正则表达式的结构正确。

下面给出该预处理函数的代码：

```
1.    QString MainWindow::regexListPreprocessing(QStringList regexLi
   st) {
2.        int lineSize = regexList.size();
3.
```

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22级计科2班

课程名称 编译原理 实验项目 正则表达式转换为DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
4.          // 去除所有空格
5.          for (int i = 0; i < lineSize; i++) {
6.              regexList[i].replace(" ", "");
7.          }
8.
9.          // 替换[]
10.         for (int i = 0; i < lineSize; i++) {
11.             int leftBracket = -1;    // 左右括号位置
12.             for (int j = 0; j < regexList[i].size(); j++) {
13.                 if (regexList[i][j] == '\\') {              // 转义字符
14.                     j++;
15.                 } else if (regexList[i][j] == '[') {        // 匹配左括号
16.                     leftBracket = j;
17.                 } else if (regexList[i][j] == ']') {
18.                     if (leftBracket == -1) {            // 出现了右括号，但没有出现左括号
19.                         continue;
20.                     } else {
21.                         if (leftBracket == j - 1) continue; // 特判一下
22.                         QString bracketString = regexList[i].mid(leftBracket + 1, j - leftBracket - 1);
23.                         QString newBracketString = "";
24.                         int bracketLen = bracketString.size();
25.                         int groupNum = bracketLen / 3;
26.                         for (int k = 0; k < groupNum; k++) {
27.                             int beginIndex = 3 * k;
28.                             int endIndex = 3 * k + 2;
29.                             for (char l = bracketString[beginIndex].unicode(); l <= bracketString[endIndex].unicode(); l++) {
30.                                 newBracketString.append(l);
31.                                 newBracketString.append('|');
32.                             }
33.                         }
34.                         if (groupNum != 0) {
35.                             // 替换字符串
36.                             int newBracketStringLen = newBracketString.size();
37.                             newBracketString = newBracketString.mid(0, newBracketStringLen - 1);
38.                             regexList[i] = regexList[i].left(leftBracket) + "(" + newBracketString + ")" + regexList[i].right(regex
```

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22级计科2班

课程名称 编译原理 实验项目 正则表达式转换为DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
   List[i].size() - j - 1);
                            j = leftBracket + newBracketString.siz
   e() + 2 - 1;
                        }
                        leftBracket = -1;               // 后续需要
   重新匹配中括号
                    }
                }
            }
        }

        // 替换正闭包+
        for (int i = 0; i < lineSize; i++) {
            int leftBracket = -1;
            for (int j = 0; j < regexList[i].size(); j++) {
                if (regexList[i][j] == '\\') {              // 转义
   字符
                    j++;
                } else if (regexList[i][j] == '(') {        // 匹配
   左括号
                    leftBracket = j;
                } else if (regexList[i][j] == '+') {        // 遇到
   加号
                    if (j > 0) {
                        regexList[i][j] = '*';
                        if (regexList[i][j - 1] == ')') {
                            regexList[i].insert(leftBracket, regex
   List[i].mid(leftBracket, j - leftBracket));
                            j += j - leftBracket;
                        } else {
                            regexList[i].insert(j -
    1, regexList[i][j - 1]);
                            j += 1;
                        }
                    }
                }
            }
        }

        // 替换可选?
        for (int i = 0; i < lineSize; i++) {
            int leftBracket = -1;
            for (int j = 0; j < regexList[i].size(); j++) {
                if (regexList[i][j] == '\\') {              // 转义
```

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22级计科2班

课程名称 编译原理 实验项目 正则表达式转换为DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
    字符
                    j++;
                } else if (regexList[i][j] == '(') {        // 匹配
    左括号
                    leftBracket = j;
                } else if (regexList[i][j] == '?') {         // 遇到
    加号
                    if (j > 0) {
                        regexList[i][j] = '|';
                        if (regexList[i][j - 1] == ')') {
                            regexList[i].insert(leftBracket, '(');
                            regexList[i].insert(j + 2, "#)");
                            j += 3;
                        } else {
                            regexList[i].insert(j - 1, '(');
                            regexList[i].insert(j + 2, "#)");
                            j += 3;
                        }
                    }
                }
            }
        }

        // 获取要转换成为NFA 的主正则表达式
        QString mainRegex;
        QHash<QString, QString> regexHash;
        for (int i = 0; i < lineSize; i++) {
            QString regexItem = regexList[i];
            if (regexItem[0] == '_') {   // 主正则表达式（仅考虑有一
    个主正则表达式的情况）
                mainRegex = regexItem;
                regexList.removeAt(i);
            } else {                    // 非主正则表达式，获取其等号
    左侧的标识符
                int regexLen = regexItem.size();
                for (int j = 0; j < regexLen; j++) {
                    if (regexItem[j] == '=') {
                        QString identifier = regexItem.left(j);
                        QString regexBody = regexItem.right(regexL
    en - j - 1);
                        regexHash.insert(identifier, regexBody);
                        break;          // 只捕获第一个等号(等号前的
    转义符不考虑)
                    }
```

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
112.                 }
113.             }
114.         }
115.
116.         // 合并正则表达式
117.         for (auto key: regexHash.keys()) {
118.             QString replaceString = regexHash[key];
119.             if (replaceString[0] != '(' || replaceString[replaceSt
    ring.size() - 1] != ')') {
120.                 replaceString = "(" + replaceString + ")";
121.             }
122.             for (auto& value: regexHash.values()) {
123.                 value.replace(key, replaceString);
124.             }
125.             mainRegex.replace(key, replaceString);
126.         }
127.
128.         // 主正则表达式添加连接符
129.         mainRegex = mainRegex.right(mainRegex.size() -
     mainRegex.indexOf('=') - 1);
130.         for (int i = 0; i < mainRegex.size() - 1; i++) {
131.             if (mainRegex[i] == '\\') {
132.                 i++;
133.                 if (i < mainRegex.size() - 1) {
134.                     if (mainRegex[i + 1] == '(' || !isOperator(mai
    nRegex[i + 1].unicode())) {
135.                         mainRegex.insert(i + 1, '.');
136.                         i++;
137.                     }
138.                 }
139.             } else if (isOperator(mainRegex[i].unicode())) {
140.                 if (mainRegex[i] == ')' || mainRegex[i] == '*') {
141.                     if (!isOperator(mainRegex[i + 1].unicode()) ||
     mainRegex[i + 1] == '(') {
142.                         mainRegex.insert(i + 1, '.');
143.                         i++;
144.                     }
145.                 }
146.             } else {
147.                 if (!isOperator(mainRegex[i + 1].unicode()) || mai
    nRegex[i + 1] == '(') {
148.                     mainRegex.insert(i + 1, '.');
149.                     i++;
150.                 }
```

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
151.             }
152.         }
153.
154.         return mainRegex;
155.     }
```

3. 正则表达式转后缀表达式

正则表达式转后缀表达式的设计思路是将正则表达式转换为后缀表达式（也称为逆波兰表示法）。后缀表达式是一种不需要括号来标识操作符优先级的数学表达式，它通过操作符的顺序来表达运算的优先级。以下是该函数的详细步骤：

1) 初始化栈：函数使用两个栈 s1 和 s2，其中 s1 用于存储操作符（符号栈），s2 用于存储操作数（操作数栈）。
2) 遍历正则表达式：函数逐个字符遍历输入的正则表达式 re。
3) 处理转义字符：如果当前字符是转义字符\，并且它不是字符串的最后一个字符，那么将转义字符和下一个字符作为一个整体推入操作数栈。
4) 处理括号：
   a) 如果当前字符是左括号(，则直接推入符号栈。
   b) 如果当前字符是右括号)，则从符号栈中弹出操作符并推入操作数栈，直到遇到左括号。在此过程中，所有遇到的操作符都推入操作数栈，左括号被弹出但不推入操作数栈。
5) 处理操作符：如果当前字符是一个操作符（通过 isOperator 函数判断），则根据操作符的优先级（通过 getPriority 函数获取）与符号栈顶部的操作符优先级进行比较：
   a) 如果符号栈为空，或者符号栈顶部的操作符优先级小于或等于当前操作符的优先级，则将当前操作符推入符号栈。
   b) 否则，从符号栈中弹出操作符并推入操作数栈，直到遇到优先级小于当前操作符的操作符或符号栈为空。
   c) 处理操作数：如果当前字符不是操作符、左括号或右括号，它被视为操作数，并直接推入操作数栈。
6) 将符号栈中剩余的操作符推入操作数栈：遍历结束后，将符号栈中剩余的所有操作符依次推入操作数栈。
7) 反转操作数栈：为了使操作数的顺序符合后缀表达式的顺序，将操作数栈 s2 中的内容反转到另一个栈 s3 中。
8) 构建后缀表达式：最后，从栈 s3 中弹出所有元素并拼接成字符串，这个字符串就是转换后的后缀表达式。
9) 返回结果：函数返回构建好的后缀表达式。

通过这个函数，正则表达式中嵌套的括号和操作符被转换为后缀形式，这

使得后续的 NFA 构建过程可以更容易地处理表达式，因为后缀表达式通过操作符的顺序直接表达了运算的优先级，无需额外的括号。

下面展示正则表达式转后缀表达式的代码：

```
1.    QString MainWindow::regexToPostFix(QString re)
2.    {
3.        QStack<QChar> s1;
4.        QStack<QString> s2;
5.        for (int i = 0; i < re.size(); i++) {
6.            if (re[i] == '\\') {
7.                if (i < re.size() - 1) {
8.                    s2.push(re.mid(i, 2));
9.                }
10.               i++;
11.           }
12.           else if (isOperator(re[i].unicode())) {
13.               if (re[i] == '(') {
14.                   s1.push(re[i]);                    // 左括号直接放入符号栈
15.               } else if (re[i] == ')') {
16.                   while (!s1.empty() && s1.top() != '(') {
                          // 右括号不断拿出符号栈内容，放入结果栈，直到遇到结果栈
17.                       s2.push(s1.top());
18.                       s1.pop();
19.                   }
20.                   if (!s1.empty() && s1.top() == '(') s1.pop();
21.               } else {
22.                   int nowPriority = getPriority(re[i].unicode());
23.                   while (!s1.empty() && getPriority(s1.top().unicode()) >= nowPriority) {
24.                       s2.push(s1.top());
25.                       s1.pop();
26.                   }
27.                   if (s1.empty() || getPriority(s1.top().unicode()) < nowPriority) {
28.                       s1.push(re[i]);
29.                   }
30.               }
31.           } else {
32.               s2.push(re.mid(i, 1));
33.           }
34.       }
```

```
35.         while (!s1.empty()) {
36.             s2.push(s1.top());
37.             s1.pop();
38.         }
39.         QStack<QString> s3;
40.         QString postFixRegex = "";
41.         while (!s2.empty()) {
42.             s3.push(s2.top());
43.             s2.pop();
44.         }
45.         while (!s3.empty()) {
46.             postFixRegex += s3.top();
47.             s3.pop();
48.         }
49.         return postFixRegex;
50.     }
```

4. 后缀表达式转换 NFA

后缀表达式转 NFA 的设计思路是将后缀表达式转换为非确定性有限自动机（NFA）。后缀表达式是一种不需要括号来表示的表达式形式，它通过操作符的顺序直接表达了运算的优先级。NFA 是一种计算模型，用于识别正则表达式所描述的语言。以下是该函数的详细步骤：

1) 初始化栈：使用一个栈 s 来存储状态对 QPair<int, int>，这些状态对表示 NFA 中的状态转换。
2) 遍历后缀表达式：函数逐个字符遍历输入的后缀表达式 re。
3) 处理转义字符：如果当前字符是转义字符\，并且它不是字符串的最后一个字符，那么将接下来的字符作为一个新的字符状态添加到 NFA 中，并将其对应的两个状态（当前状态和下一个状态）入栈。
4) 处理运算符：
   a) 对于.（连接符），如果栈中至少有两个元素，表示可以连接这两个状态。在这两个状态之间添加一个标记为"epsilon"的连接，表示这两个状态可以通过空字符串 ε 直接转移。
   b) 对于|（选择操作符），如果栈中至少有两个元素，表示可以选择这两个状态之一。创建一个新状态作为这两个状态的公共起点，并将它们通过"epsilon"连接到一个新状态，表示可以选择进入任一状态。
   c) 对于*（闭包操作符），如果栈非空，表示闭包操作应用于栈顶的状态。创建一个新状态，并通过"epsilon"将这个新状态与栈顶状态的两个端点连接起来，形成闭环，表示可以重复任意次（包括零次）。
5) 处理 epsilon（ε）：如果当前字符是#，表示 epsilon（ε），即空字符串。

创建一个新的 epsilon 状态对，并将其入栈。

6) 处理普通字符：如果当前字符既不是运算符也不是 epsilon 标记，它被视为一个普通字符。为这个字符创建一个新的状态对，并将其入栈。

7) 状态数量更新：在添加新状态对后，更新 stateNum（状态数量）并添加相应的字符到 stateSet（状态集合）中。

8) 确定开始和结束状态：在遍历结束后，栈顶的状态对 statePair 表示 NFA 的开始状态和结束状态。

9) 构建 NFA 的转移函数：在整个过程中，G 矩阵被用来表示 NFA 的状态之间的转移。每个状态对之间的转移字符被存储在 G 矩阵的对应位置。

通过这个函数，后缀表达式被转换为 NFA 的表示，其中 NFA 的状态转换由 G 矩阵定义，状态集合 stateSet 包含所有可能的字符和 epsilon 状态。开始状态和结束状态分别由 startState 和 endState 变量表示。这样，NFA 就可以用于识别与给定后缀表达式匹配的字符串。

下面给出后缀表达式转换 NFA 的代码：

```
1.    void NFA::fromRegex(QString re)     // 后缀表达式转NFA
2.    {
3.        QStack<QPair<int, int>> s;
4.        for (int i = 0; i < re.size(); i++) {
5.            if (re[i] == '\\') {                          // 转义字符
6.                if (i < re.size() - 1) {
7.                    allocateMemory(2);
8.                    G[stateNum][stateNum + 1] = re[i + 1];  // 转义
   符不再出现
9.                    s.append({stateNum, stateNum + 1}); // 入栈
10.                   stateNum += 2;                        // 状态数
   量 + 2
11.                   stateSet.insert(QString(re[i + 1]));
12.               }
13.               i++;
14.           } else if (re[i] == '*' || re[i] == '|' || re[i] == '.
   ') {   // 运算符
15.               if (re[i] == '.') {
16.                   if (s.size() >= 2) {                  // 取出两个
   元素相连重新入栈
17.                       QPair<int, int> tail = s.top();
18.                       s.pop();
19.                       QPair<int, int> head = s.top();
20.                       s.pop();
21.                       G[head.second][tail.first] = "epsilon";
    // 连接操作
22.                       s.append({head.first, tail.second});
    // 重新入栈
```

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22级计科2班

课程名称 编译原理 实验项目 正则表达式转换为DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
                        stateSet.insert("epsilon");
                    }
                } else if (re[i] == '|') {
                    allocateMemory(2);
                    if (s.size() >= 2) {                // 取出两个元素相连重新入栈
                        QPair<int, int> left = s.top();
                        s.pop();
                        QPair<int, int> right = s.top();
                        s.pop();
                        G[stateNum][left.first] = "epsilon";
                        G[stateNum][right.first] = "epsilon";
                        G[left.second][stateNum + 1] = "epsilon";
                        G[right.second][stateNum + 1] = "epsilon";
                        s.append({stateNum, stateNum + 1}); // 入栈
                        stateNum += 2;                      // 状态数量 + 2
                        stateSet.insert("epsilon");
                    }
                } else {
                    allocateMemory(2);
                    if (!s.empty()) {                   // 闭包为单元运算符，因此只需要取出一个元素
                        QPair<int, int> item = s.top();
                        s.pop();
                        G[stateNum][stateNum + 1] = "epsilon";
                        G[stateNum][item.first] = "epsilon";
                        G[item.second][stateNum + 1] = "epsilon";
                        G[item.second][item.first] = "epsilon";
                        s.append({stateNum, stateNum + 1}); // 入栈
                        stateNum += 2;                      // 状态数量 + 2
                        stateSet.insert("epsilon");
                    }
                }
            } else {
                if (re[i] == '#') {                     // epsilon
                    allocateMemory(2);
                    G[stateNum][stateNum + 1] = "epsilon";
                    s.append({stateNum, stateNum + 1}); // 入栈
                    stateNum += 2;                      // 状态数量 + 2
                    stateSet.insert("epsilon");
                } else {                                // 其他非运
```

```
    算符
                    allocateMemory(2);
                    G[stateNum][stateNum + 1] = re[i];
                    s.append({stateNum, stateNum + 1}); // 入栈
                    stateNum += 2;                      // 状态数
    量 + 2
                    stateSet.insert(QString(re[i]));
                }
            }
        }
        if (!s.empty()){
            QPair<int, int> statePair = s.top();
            startState = statePair.first;
            endState = statePair.second;
        }
    }
```

5. NFA 转换 DFA

NFA 转换 DFA 的设计思路是将一个非确定性有限自动机（NFA）转换为一个确定性有限自动机（DFA）。DFA 是 NFA 的一种简化形式，它在任何给定的状态下对于每个可能的输入字符只有一个转移，而 NFA 可以有多个转移。以下是该函数的详细步骤：

1) 初始化：创建一个反向映射 revMapping，它将 NFA 的状态集合映射到 DFA 的状态。初始化 DFA 的开始状态，通过 NFA 的 ε -闭包（epsilon closure）找到与之对应的 DFA 状态，并将其加入到映射和反向映射中。同时，如果这个状态集合包含 NFA 的结束状态，则将其加入到 DFA 的结束状态集合中。
2) 状态编号：更新 DFA 的状态数量 stateNum。
3) 初始化访问集合和队列：使用一个集合 vis 来记录已经找到的 NFA 状态集合，避免重复处理。使用一个队列 q 来实现广度优先搜索（BFS），开始时将 DFA 的开始状态加入队列。
4) 广度优先搜索：当队列非空时，执行循环：
   a) 从队列中取出一个 DFA 状态（称为 stateItem），并获取其对应的 NFA 状态集合 nfaStateSet。
   b) 对于 NFA 状态集合中的每一个字符转移 changeItem（不是 ε ），执行以下操作：
      i. 计算在当前 DFA 状态下，对应字符转移的 ε -闭包 changeClosure。
      ii. 确定这个 ε -闭包的 ε -闭包 changeEpsilon。
5) 处理转移：

a) 如果 changeEpsilon 非空，检查是否已经存在对应的 DFA 状态：

i. 如果存在，通过反向映射找到对应的 DFA 状态（称为 nextItem），更新 DFA 的转移函数 G，将当前状态 stateItem 和字符 changeItem 的转移指向 nextItem。

ii. 如果这个状态集合包含 NFA 的结束状态，则将 nextItem 加入到 DFA 的结束状态集合中。

iii. 如果这个 ε -闭包在访问集合 vis 中尚未记录，则将其加入队列，以便后续处理。

b) 如果不存在，创建新的 DFA 状态，更新映射和反向映射，并将新状态的相关信息加入到 DFA 的转移函数和结束状态集合中。同样，如果这个状态集合尚未记录在访问集合中，则将其加入队列。

6) 标记已访问：无论是否找到对应的 DFA 状态，都将当前的 NFA 状态集合 nfaStateSet 标记为已访问，并从队列中移除当前处理的 DFA 状态。

通过这个函数，NFA 中的每一个状态集合都转换为 DFA 中的一个状态，并且通过 ε -闭包和 ε -转移的概念来确定 DFA 的状态转移。最终得到的 DFA 具有更简单的结构，同时仍然能够识别与原始 NFA 相同的语言。

下面给出 NFA 转 DFA 的代码：

```
1.    void DFA::fromNFA(NFA nfa)
2.    {
3.        QHash<QSet<int>, int> revMapping;
4.        // 获取始态
5.        QSet<int> startSet;
6.        startSet.insert(nfa.startState);
7.        startSet = nfa.epsilonClosure(startSet);
8.        mapping[startState] = startSet;
9.        revMapping[startSet] = startState;
10.       if (startSet.contains(nfa.endState)) {
11.           endStates.insert(startState);
12.       }
13.       stateNum++;
14.
15.       QSet<QSet<int>> vis;          // 判断某个状态是否被找到
16.       QQueue<int> q;                // 循环队列
17.       q.push_back(startState);
18.       while (!q.empty()) {
19.
20.           // 取出队头
21.           int stateItem = q.front();
22.           q.pop_front();
23.           QSet<int> nfaStateSet = mapping[stateItem];
```

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

```
24.
25.             // 未曾出现过的状态集合才需要查找
26.             if (!vis.contains(nfaStateSet)) {
27.
28.                 // 查询当前状态集合的转移
29.                 for (QString changeItem: nfa.stateSet) {
30.                     if (changeItem != "epsilon") {
31.
32.                         // 获取对应转移到的 epsilon 闭包
33.                         QSet<int> changeClosure = nfa.valueClosure
   (nfaStateSet, changeItem);
34.                         QSet<int> changeEpsilon = nfa.epsilonClosu
   re(changeClosure);
35.
36.                         // 找不到集合
37.                         if (changeEpsilon.empty()) continue;
38.
39.                         // 之前找到过这个状态
40.                         if (revMapping.contains(changeEpsilon)) {
41.                             int nextItem = revMapping[changeEpsilo
   n];
42.
43.                             G[stateItem][changeItem] = nextItem;
44.                             if (changeEpsilon.contains(nfa.endStat
   e)) {
45.                                 endStates.insert(nextItem);
46.                             }
47.
48.                             if (!vis.contains(changeEpsilon)) {
49.                                 q.push_back(nextItem);
50.                             }
51.                         } else {
52.                             int nextItem = stateNum;
53.                             mapping[nextItem] = changeEpsilon;
54.                             revMapping[changeEpsilon] = nextItem;
55.
56.                             G[stateItem][changeItem] = nextItem;
57.                             if (changeEpsilon.contains(nfa.endStat
   e)) {
58.                                 endStates.insert(nextItem);
59.                             }
60.                             stateNum++;
61.
62.                             if (!vis.contains(changeEpsilon)) {
```

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

```
                                q.push_back(nextItem);
                            }
                        }
                    }
                }
                vis.insert(nfaStateSet);
            }
        }
    }
```

6. DFA 最小化

DFA 最小化设计思路是将一个确定性有限自动机（DFA）转换为一个等价的最小化 DFA。最小化的 DFA 具有最小的状态数，同时仍然识别相同的语言。以下是该函数的详细步骤：

1) 初始化：创建一个集合 notEndStates 来存储原始 DFA 中所有非结束状态的集合。
2) 收集转移字符：通过遍历 DFA 的状态转移函数 G，收集所有可能的转移字符并存储在集合 stateSet 中。
3) 初始化队列：使用一个队列 q 来存储将要处理的状态集合。开始时，将所有非结束状态和所有结束状态分别加入队列。
4) 执行状态划分：使用一个循环来迭代地划分状态集合，直到无法进一步划分：
   a) 对于队列 q 中的每个状态集合，对每个转移字符 changeItem 进行处理。
   b) 对于当前状态集合中的每个状态，检查在该状态下通过转移字符转移后到达的状态是否已经在队列 q 中。如果是，则将当前状态添加到对应的状态集合中。
   c) 如果在某个转移字符下，当前状态集合可以被划分为多于一个的子集合（即存在两个或以上的不同状态集合），则将这些子集合加入队列 q 中，并标记 flag 为 false，表示当前状态集合已被成功划分。
   d) 如果在所有转移字符下，当前状态集合都不能被进一步划分，则将当前状态集合重新加入队列 q 中，并标记 flag 为 true。
5) 处理队列结束条件：如果队列 q 中的所有元素都已经被检查过且没有发生改变（cnt 等于 lastQueLen），并且队列的大小没有变化，则结束循环。
6) 设置新的状态编号：为队列 q 中的每个状态集合分配新的编号，更新映射 mapping 和反向映射 revMapping，并且更新状态数量 stateNum。
7) 找到新的开始状态和结束状态：遍历队列 q 中的每个状态集合，根据原始 DFA 的开始状态和结束状态集合，确定最小化 DFA 的开始状态和结束状态。

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

8) 构建新的转移函数：对于队列 q 中的每个状态集合，根据原始 DFA 的转移函数 G，构建最小化 DFA 的转移函数。对于每个状态集合中的每个状态，查找其在每个转移字符下的转移目标状态，并将其映射到最小化 DFA 中对应的状态集合。

通过这个过程，原始 DFA 被转换为一个最小化的 DFA，它具有更少的状态数，同时保持了识别相同语言的能力。最小化的 DFA 在某些应用中更为高效，因为它简化了状态机的结构。

下面展示 DFA 最小化的代码：

```
1.    void DFA::fromDFA(DFA dfa)
2.    {
3.        QSet<int> notEndStates;
4.        for (int i = 0; i < dfa.stateNum; i++) {
5.            if (!dfa.endStates.contains(i)) {
6.                notEndStates.insert(i);
7.            }
8.        }
9.
10.       // 找到原始DFA 的转移类型
11.       QSet<QString> stateSet;
12.       for (int i = 0; i < dfa.stateNum; i++) {
13.           for (QString changeItem: dfa.G[i].keys()) {
14.               stateSet.insert(changeItem);
15.           }
16.       }
17.
18.       QQueue<QSet<int>> q;
19.       if (!notEndStates.empty()) q.push_back(notEndStates);
20.       if (!dfa.endStates.empty()) q.push_back(dfa.endStates);
21.
22.       int lastQueLen = 0, cnt = 0;
23.       if (!notEndStates.empty()) lastQueLen++;
24.       if (!dfa.endStates.empty()) lastQueLen++;
25.
26.       while (true) {
27.
28.           QSet<int> stateItem = q.front();
29.
30.           // 划分集合
31.           bool flag = true;
32.           for (QString changeItem: stateSet) {
33.               QSet<QSet<int>> vis;
34.               QHash<QSet<int>, QSet<int>> setHash;        // 当前
   选出
```

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22级计科2班

课程名称 编译原理 实验项目 正则表达式转换为DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

```
35.                 for (int beginItem: stateItem) {
36.                     if (!dfa.G.contains(beginItem)) continue;
37.                     if (!dfa.G[beginItem].contains(changeItem)) {
38.                         vis.insert(QSet<int>());
39.                         setHash[QSet<int>()].insert(beginItem);
40.                         continue;
41.                     }
42.                     for (QSet<int> queueItem: q) {
43.                         if (queueItem.contains(dfa.G[beginItem][ch
   angeItem])){
44.                             vis.insert(queueItem);
45.                             setHash[queueItem].insert(beginItem);
46.                             break;
47.                         }
48.                     }
49.                 }
50.                 if (vis.size() <= 1) {
51.                     continue;
52.                 } else {
53.                     for (QSet<int> visItem: vis) {
54.                         q.push_back(setHash[visItem]);
55.                     }
56.                     flag = false;
57.                     break;
58.                 }
59.             }
60.
61.             q.pop_front();
62.             if (flag) q.push_back(stateItem);
63.
64.             // 判断“上一个”队列是否全被扫过一次
65.             cnt++;
66.             if (lastQueLen == cnt) {
67.                 if (lastQueLen == q.size()) {
68.                     break;
69.                 }
70.                 lastQueLen = q.size();
71.                 cnt = 0;
72.             }
73.         }
74.
75.         int idx = 0;
76.         // 设置新编号
77.         QHash<QSet<int>, int> revMapping;
```

```
78.         for (QSet<int> queueItem: q) {
79.             if (queueItem.empty()) continue;
80.             mapping[idx] = queueItem;
81.             revMapping[queueItem] = idx;
82.             idx++;
83.         }
84.         stateNum = idx;
85.
86.         // 找到始态和终态
87.         for (QSet<int> queueItem: q) {
88.             if (queueItem.empty()) continue;
89.             // 找始态
90.             if (queueItem.contains(dfa.startState)) {
91.                 startState = revMapping[queueItem];
92.             }
93.             // 找终态
94.             for (int endState: dfa.endStates) {
95.                 if (queueItem.contains(endState)) {
96.                     endStates.insert(revMapping[queueItem]);
97.                     break;
98.                 }
99.             }
100.        }
101.
102.        // 构建新的G
103.        for (QSet<int> queueItem: q) {
104.            if (queueItem.empty()) continue;
105.            int stateItem = *queueItem.begin();
106.            for (QString changeItem: stateSet) {
107.                if (!dfa.G[stateItem].contains(changeItem)) contin
    ue;
108.                int endItem = dfa.G[stateItem][changeItem];
109.                for (QSet<int> endQueueItem: q) {
110.                    if (endQueueItem.contains(endItem)) {
111.                        G[revMapping[queueItem]][changeItem] = rev
    Mapping[endQueueItem];
112.                        break;
113.                    }
114.                }
115.            }
116.        }
117.    }
```

7. 最小化 DFA 生成词法分析程序

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

DFA 转换词法分析程序的设计思路是将确定性有限自动机（DFA）转换成一个用于词法分析的 C++函数。这个函数 matchFunction 将接受一个字符串输入，并返回一个布尔值，指示该字符串是否被 DFA 所识别。以下是该函数的详细步骤：

1) 函数定义：开始编写一个名为 matchFunction 的 C++函数，它接受一个字符串 s 作为参数，并返回一个布尔值。
2) 初始化状态和索引：在函数内部，初始化两个变量 idx 和 state。idx 用于跟踪字符串中当前处理到的位置，而 state 用于表示 DFA 的当前状态，初始状态由 DFA 的开始状态决定。
3) 主循环：使用一个 while 循环来处理输入字符串。循环条件是 state 小于状态总数且 idx 小于字符串长度。这意味着循环将继续，直到在 DFA 中到达终止状态或字符串结束。
4) 状态切换逻辑：
   a) 使用一个 switch 语句来根据当前的 state 执行不同的代码块。
   b) 对于 DFA 中的每个状态，如果该状态有有效的转移，则使用另一个 switch 语句来根据输入字符串中的当前字符进行转移。
5) 字符转移：对于每个状态和每个可能的输入字符，检查 DFA 的状态转移函数 G 来确定转移到的新状态，并更新 state 变量。
6) 默认转移：如果在当前状态下没有找到匹配的字符转移，或者在状态转移函数中没有找到该字符的转移，那么将 state 设置为一个非法状态（通常是状态总数，表示不接受的状态）。
7) 循环结束：在每次循环迭代的末尾，增加 idx 的值以移动到输入字符串的下一个字符。
8) 最终状态检查：在主循环结束后，检查 state 是否为 DFA 的任何一个结束状态。如果是，则函数返回 true，表示字符串被接受；否则返回 false。
9) 函数结束：完成 matchFunction 函数的定义。

通过这个过程，DFA 被转换成一个简单的词法分析程序，它可以用于模式匹配或词法分析任务。这种转换方法的优点是生成的代码结构清晰，易于理解和实现。此外，由于 DFA 的设计，这种转换方法能够高效地处理词法分析任务，因为 DFA 的状态转移是确定性的，没有回溯。

下面展示 DFA 转换词法分析程序的代码：

```
1.    QString DFA::toCode()
2.    {
3.        QString code = "bool matchFunction(string s) {\n";
```

```
        code += "\tint idx = 0, state = ";
        code += QString::number(startState) + ";\n";
        code += "\twhile (state < " + QString::number(stateNum) +
    " && idx < s.size()) {\n";
        code += "\t\tswitch (state) {\n";
        for (int i = 0; i < stateNum; i++) {
            if (G[i].keys().empty()) continue;
            code += "\t\tcase " + QString::number(i) + ":\n";
            code += "\t\t\tswitch (s[idx]) {\n";
            for (QString changeItem: G[i].keys()) {
                code += "\t\t\tcase \'" + changeItem + "\':\n";
                code += "\t\t\t\tstate = " + QString::number(G[i][
    changeItem]) + ";\n";
                code += "\t\t\t\tbreak;\n";
            }
            code += "\t\t\tdefault:\n";
            code += "\t\t\t\tstate = " + QString::number(stateNum)
     + ";\n";
            code += "\t\t\t\tbreak;\n";
            code += "\t\t\t}\n";
            code += "\t\t\tbreak;\n";
        }
        code += "\t\tdefault:\n";
        code += "\t\t\tstate = " + QString::number(stateNum) + ";\
    n";
        code += "\t\t\tbreak;\n";
        code += "\t\t}\n";
        code += "\t\tidx++\n";
        code += "\t}\n";
        code += "\tif (";
        int cnt = 0;
        for (int i: endStates) {
            code += "state == " + QString::number(i);
            cnt++;
            if (cnt != endStates.size()) code += "||";
        }
        code += ") return true;\n";
        code += "\telse return false;\n";
        code += "}";
        return code;
    }
```

8. 系统测试

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

测试样例 1：

```
nat=[0-9]+
signedNat=(\+|-)?nat
_number=signedNat(\.nat)?(E signedNat)?
```

测试效果如下：

图 7 NFA 展示图

图 8 DFA 展示图

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

图 9 最小化 DFA 展示图

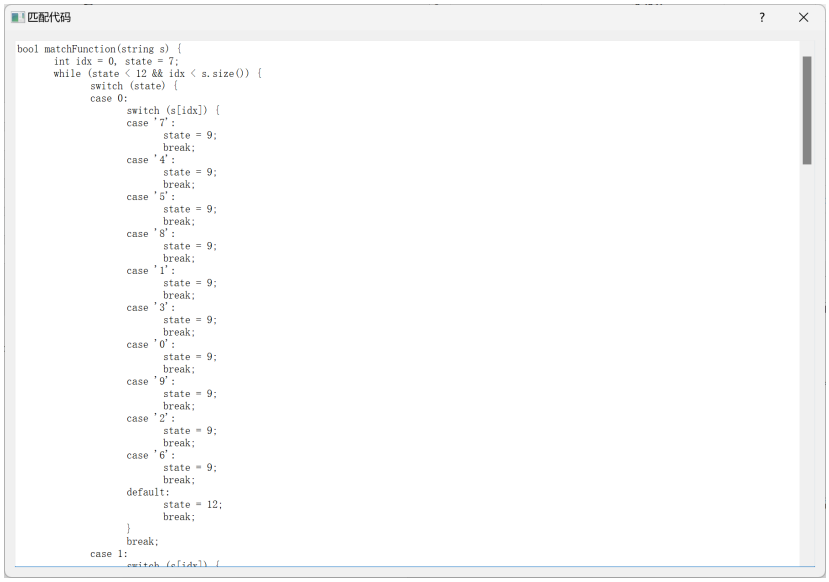

图 10 匹配代码展示图

测试样例 2：

```
_test=a(a|b)?
```

测试结果：（同样按照上述顺序展示：NFA->DFA->最小化 DFA->匹配代码）

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

NFA、DFA展示表

	b	a	epsilon
0		1	
1			10
2		3	
3			7
4	5		
5			7
6			2,4
7			11
8			9
9			11
10			6,8
11			

NFA、DFA展示表

	b	a
0		1:...
1	2:{5,7,11}	3:{3,7,11}
2		
3		

NFA、DFA展示表

	b	a
0		2:{1}
1		
2	1:{3,2}	1:{3,2}

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 **学 号** 20212821020

专 业 计算机科学与技术 **年级、班级** 22 级计科 2 班

课程名称 编译原理 **实验项目** 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 **实验评分** _____

匹配代码

```
bool matchFunction(string s) {
	int idx = 0, state = 0;
	while (state < 3 && idx < s.size()) {
		switch (state) {
		case 0:
			switch (s[idx]) {
			case 'a':
				state = 2;
				break;
			default:
				state = 3;
				break;
			}
			break;
		case 2:
			switch (s[idx]) {
			case 'b':
				state = 1;
				break;
			case 'a':
				state = 1;
				break;
			default:
				state = 3;
				break;
			}
			break;
		default:
			state = 3;
			break;
		}
		idx++
	}
	if (state == 1||state == 2) return true;
	else return false;
}
```

图 11-14 测试样例 2 结果展示

四、实验总结（心得体会）

实验成果

1) 正则表达式解析：成功实现了对包含命名操作和多种运算符的正则表达式的解析。
2) NFA 和 DFA 转换：实现了从正则表达式到 NFA，再到 DFA 的转换算法，并能够展示转换过程中的状态转换表。
3) DFA 最小化：开发了 DFA 最小化的算法，能够有效地将 DFA 转换为最小化的 DFA，并展示其状态转换表。
4) 词法分析程序生成（选做）：对于选择完成此部分的同学，实现了将最小化 DFA 转换为词法分析程序的功能，提供了两种转换方法供用户选择。
5) 软件文档：提供了完整的软件设计文档，详细说明了软件的使用方法和设计细节。

遇到的问题及解决方案

问题 1：在处理正则表达式中的转义字符时，如何正确区分用户意图表示的转义和正则表达式本身的转义。

解决方案：设计了一套明确的转义规则，并通过界面提示用户输入正确的转义字符。

问题 2：在 DFA 最小化过程中，状态的等价性判断较为复杂，且容易出错。

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

解决方案：采用了高效的算法来识别等价状态，并进行了多次测试以确保算法的准确性。

问题 3：在开发 Windows 界面时，如何合理布局界面以展示复杂的状态转换表。

解决方案：设计了可滚动的表格控件，允许用户查看完整的状态转换信息。

后续改进方向

1) 用户界面优化：当前的界面设计较为简单，后续可以进一步优化用户界面，使其更加友好和直观。
2) 性能提升：对于复杂的正则表达式，转换过程可能需要优化以提高效率。
3) 功能扩展：可以考虑增加更多的功能，如正则表达式的修改和保存、DFA 可视化，以及更丰富的词法分析程序生成选项等。
4) 错误处理：增强软件的错误处理能力，提供更清晰的错误提示，帮助用户快速定位问题。
5) 文档完善：继续完善软件文档，增加更多的示例和使用场景说明。

实验心得

通过本次实验，我深入理解了正则表达式、NFA、DFA 以及 DFA 最小化的理论基础和实际应用。同时，我也提高了使用 C++进行软件开发的能力，特别是在处理复杂数据结构和算法实现方面。此外，实验还锻炼了我的软件设计、文档编写和用户界面设计的能力。

五、参考文献：

[1] 正则表达式->NFA->DFA(C++实现)_c++ 存储 dfa-CSDN 博客

[2] 编译原理：NFA 转 DFA（原理+完整代码+可视化实现）_nfa 转化为 dfa-CSDN 博客

六、附录

NFA 类结构：

```
#ifndef NFA_H
#define NFA_H

#include <QString>
#include <QVector>
#include <QSet>
```

```
7.
8.      class NFA
9.      {
10.     public:
11.         // 构造函数
12.         NFA(int begin = -1, int end = -1);
13.         void clear();                           // 清空NFA
14.         void fromRegex(QString re);             // 从后缀表达式构造NFA
15.         void allocateMemory(int num);           // 分配G内存的函数
16.         QSet<int> epsilonClosure(QSet<int> state);  // 计算epsilon
    闭包
17.         QSet<int> valueClosure(QSet<int> state, QString value); //
     计算某个集合状态能通过value转移到的状态集合
18.
19.         // 成员变量
20.         QVector<QVector<QString>> G;            // 邻接矩阵
21.         QSet<QString> stateSet;                 // 存储所有状态类型
22.         int startState;                         // 表示起始状态
23.         int endState;                           // 表示终止状态
24.         int stateNum;                           // 表示状态数量
25.         int maxStateNum;                        // 表示当前最多存储的状
    态数量
26.     };
27.
28.     #endif // NFA_H
```

DFA 类结构：

```
1.      #ifndef DFA_H
2.      #define DFA_H
3.
4.      #include <QHash>
5.      #include <QSet>
6.      #include <QString>
7.      #include <QVector>
8.
9.      #include "nfa.h"
10.
11.     class DFA
12.     {
13.     public:
14.         DFA();
15.         void fromNFA(NFA nfa);
16.         void fromDFA(DFA dfa);
```

```
17.       void clear();
18.       QString toCode();
19.
20.       QHash<int, QSet<int>> mapping;
21.       QHash<int, QHash<QString, int>> G;
22.       int startState;
23.       QSet<int> endStates;
24.       int stateNum;                          // 表示状态数量
25.
26.   };
27.
28.   #endif // DFA_H
```

主窗口构造函数：

```
1.    MainWindow::MainWindow(QWidget *parent)
2.        : QMainWindow(parent)
3.        , ui(new Ui::MainWindow)
4.        , nfa()
5.    {
6.        ui->setupUi(this);
7.
8.        // 页面基本布局
9.        this->setWindowTitle("编译原理实验二：正则表达式转NFA、DFA");
10.       QScreen *screen = QGuiApplication::primaryScreen();
11.       QRect screenGeometry = screen->geometry();
12.       int screenWidth = screenGeometry.width();
13.       int screenHeight = screenGeometry.height();
14.       int newWidth = screenWidth * 0.9;
15.       int newHeight = screenHeight * 0.9;
16.       this->resize(newWidth, newHeight);
17.
18.       // 分析并产生NFA、DFA、最小化DFA
19.       connect(ui-
   >analysisPushButton, &QPushButton::clicked, this, [=]() {
20.           this->nfa.clear();
21.           this->dfa.clear();
22.           this->miniDFA.clear();
23.           this->code = "";
24.           tableRemoveAll();
25.
26.           QString allText = ui->textEdit->toPlainText();
27.           if (allText != "") {
```

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
28.
29.                 QStringList lines = allText.split('\n', QString::S
    kipEmptyParts);
30.
31.                 // 预处理正则表达式组
32.                 QString mainRegex = regexListPreprocessing(lines);
33.
34.                 // 正则表达式转后缀表达式
35.                 QString postFixRegex = regexToPostFix(mainRegex);
36.
37.                 // 后缀表达式转NFA
38.                 this->nfa.fromRegex(postFixRegex);
39.
40.                 // NFA 转DFA
41.                 this->dfa.fromNFA(this->nfa);
42.
43.                 // DFA 最小化
44.                 this->miniDFA.fromDFA(this->dfa);
45.
46.                 // 转换为代码
47.                 this->code = this->miniDFA.toCode();
48.             }
49.
50.         });
51.
52.         // 显示NFA
53.         connect(ui-
    >nfaPushButton, &QPushButton::clicked, this, [=]() {
54.             // 清空原来的表格
55.             tableRemoveAll();
56.
57.             ui->tableWidget->setRowCount(nfa.stateNum);
58.             ui->tableWidget->setColumnCount(nfa.stateSet.size());
59.             QStringList strListColumnHander;
60.             for (QString item : nfa.stateSet) {
61.                 strListColumnHander << tr(item.toStdString().c_str
    ());
62.             }
63.             ui->tableWidget-
    >setHorizontalHeaderLabels(strListColumnHander);
64.             QStringList strListRowHander;
65.             for (int i = 0; i < nfa.stateNum; i++) {
66.                 strListRowHander << tr(QString::number(i).toStdStr
    ing().c_str());
```

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22级计科2班

课程名称 编译原理 实验项目 正则表达式转换为DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
67.             }
68.             ui->tableWidget-
    >setVerticalHeaderLabels(strListRowHander);
69.             for (int i = 0; i < nfa.stateNum; i++) {
70.                 int j = 0;
71.                 for (QString stateChange: nfa.stateSet) {
72.                     QString itemString = "";
73.                     for (int k = 0; k < nfa.stateNum; k++) {
74.                         if (nfa.G[i][k] == stateChange) {
75.                             itemString += QString::number(k);
76.                             itemString += ",";
77.                         }
78.                     }
79.                     ui->tableWidget-
    >setItem(i, j, new QTableWidgetItem(itemString.left(itemString.si
    ze() - 1)));
80.                     j++;
81.                 }
82.             }
83.
84.             // 添加始态、终态颜色
85.             QTableWidgetItem *beginItem = ui->tableWidget-
    >verticalHeaderItem(nfa.startState);
86.             QTableWidgetItem *endItem = ui->tableWidget-
    >verticalHeaderItem(nfa.endState);
87.             beginItem->setTextColor(QColor(0, 255, 0));
88.             endItem->setTextColor(QColor(255, 0, 0));
89.         });
90.
91.         // 显示DFA
92.         connect(ui-
    >dfaPushButton, &QPushButton::clicked, this, [=]() {
93.             // 清空原来的表格
94.             tableRemoveAll();
95.
96.             ui->tableWidget->setRowCount(dfa.stateNum);
97.             ui->tableWidget->setColumnCount(nfa.stateSet.size() -
     (nfa.stateSet.contains("epsilon") ? 1 : 0));
98.
99.             QStringList strListColumnHander;
100.            for (QString item : nfa.stateSet) {
101.                if (item == "epsilon") continue;
102.                strListColumnHander << tr(item.toStdString().c_str
    ());
```

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分 _____

```
103.          }
104.          ui->tableWidget-
   >setHorizontalHeaderLabels(strListColumnHander);
105.
106.          QStringList strListRowHander;
107.          for (int i = 0; i < dfa.stateNum; i++) {
108.              strListRowHander << tr(QString::number(i).toStdStr
   ing().c_str());
109.          }
110.          ui->tableWidget-
   >setVerticalHeaderLabels(strListRowHander);
111.
112.          for (int i = 0; i < dfa.stateNum; i++) {
113.              if (!dfa.G.contains(i)) continue;
114.              int j = 0;
115.              for (QString stateChange: nfa.stateSet) {
116.                  if (stateChange == "epsilon") continue;
117.                  if (dfa.G[i].contains(stateChange)) {
118.
119.                      int k = dfa.G[i][stateChange];
120.                      QString titleTr = QString::number(k) + ":{
   ";
121.                      for (int item: dfa.mapping[k]) {
122.                          titleTr += QString::number(item);
123.                          titleTr += ",";
124.                      }
125.                      titleTr = titleTr.left(titleTr.size() -
    1);
126.                      titleTr += "}";
127.
128.                      ui->tableWidget-
   >setItem(i, j, new QTableWidgetItem(titleTr));
129.                  }
130.                  j++;
131.              }
132.          }
133.
134.          // 添加始态、终态颜色
135.          QTableWidgetItem *beginItem = ui->tableWidget-
   >verticalHeaderItem(dfa.startState);
136.          beginItem->setTextColor(QColor(0, 255, 0));
137.          for (int endState: dfa.endStates) {
138.              QTableWidgetItem *endItem = ui->tableWidget-
   >verticalHeaderItem(endState);
```

华南师范大学实验报告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22级计科2班

课程名称 编译原理 实验项目 正则表达式转换为DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

```
139.                endItem->setTextColor(QColor(255, 0, 0));
140.            }
141.
142.        });
143.
144.        // 显示最小化的DFA
145.        connect(ui->minimumDfaPushButton, &QPushButton::clicked, this, [=]() {
146.            // 清空原来的表格
147.            tableRemoveAll();
148.
149.            ui->tableWidget->setRowCount(miniDFA.stateNum);
150.            ui->tableWidget->setColumnCount(nfa.stateSet.size() - (nfa.stateSet.contains("epsilon") ? 1 : 0));
151.
152.            QStringList strListColumnHander;
153.            for (QString item : nfa.stateSet) {
154.                if (item == "epsilon") continue;
155.                strListColumnHander << tr(item.toStdString().c_str());
156.            }
157.            ui->tableWidget->setHorizontalHeaderLabels(strListColumnHander);
158.
159.            QStringList strListRowHander;
160.            for (int i = 0; i < miniDFA.stateNum; i++) {
161.                strListRowHander << tr(QString::number(i).toStdString().c_str());
162.            }
163.            ui->tableWidget->setVerticalHeaderLabels(strListRowHander);
164.
165.            for (int i = 0; i < miniDFA.stateNum; i++) {
166.                if (!miniDFA.G.contains(i)) continue;
167.                int j = 0;
168.                for (QString stateChange: nfa.stateSet) {
169.                    if (stateChange == "epsilon") continue;
170.                    if (miniDFA.G[i].contains(stateChange)) {
171.
172.                        int k = miniDFA.G[i][stateChange];
173.                        QString titleTr = QString::number(k) + ":{";
174.                        for (int item: miniDFA.mapping[k]) {
175.                            titleTr += QString::number(item);
```

华 南 师 范 大 学 实 验 报 告

学生姓名 林泽勋 学 号 20212821020

专 业 计算机科学与技术 年级、班级 22 级计科 2 班

课程名称 编译原理 实验项目 正则表达式转换为 DFA

实验时间 2024 年 4 月 18 日

实验指导老师 黄煜廉 实验评分

```
176.                            titleTr += ",";
177.                        }
178.                        titleTr = titleTr.left(titleTr.size() -
    1);
179.                        titleTr += "}";
180.
181.                        ui->tableWidget-
   >setItem(i, j, new QTableWidgetItem(titleTr));
182.                    }
183.                    j++;
184.                }
185.            }
186.
187.            // 添加始态、终态颜色
188.            QTableWidgetItem *beginItem = ui->tableWidget-
   >verticalHeaderItem(miniDFA.startState);
189.            beginItem->setTextColor(QColor(0, 255, 0));
190.            for (int endState: miniDFA.endStates) {
191.                QTableWidgetItem *endItem = ui->tableWidget-
   >verticalHeaderItem(endState);
192.                endItem->setTextColor(QColor(255, 0, 0));
193.            }
194.        });
195.
196.        // 生成匹配代码
197.        connect(ui-
   >codePushButton, &QPushButton::clicked, this, [=]() {
198.            CodeDialog* codeDialog = new CodeDialog(this, code);
199.            codeDialog->exec();
200.        });
201.
202.        // 重置程序
203.        connect(ui-
   >resetPushButton, &QPushButton::clicked, this, [=]() {
204.            this->nfa.clear();
205.            this->dfa.clear();
206.            this->miniDFA.clear();
207.            this->code = "";
208.            tableRemoveAll();
209.        });
210.
211.    }
```