

**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**

|  |
| --- |
| 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。  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。   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     下图是对以上代码的解释。     1. 定义连接运算符     下图是对以上代码的解释。     1. 定义选择运算符     下图是对以上代码的解释。     1. 定义闭包运算符     下图是对以上代码的解释。     1. 定义可选运算符     下图是对以上代码的解释。    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。    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]。    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。  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](https://blog.csdn.net/m0_61843614/article/details/134790617) |