

课程实验报告

课 程 名 称： 编译原理

实验项目名称： 正则运算表达式的 DFA 构建

专 业 班 级： 软件2003

姓 名： 曾佳宝

学 号： 202026010324

指 导 教 师： 黎文伟

完 成 时 间： 2023 年 3 月 16 日

信息科学与工程学院

|  |  |
| --- | --- |
| 实验题目：正则运算表达式的 DFA 构建 | |
| 实验目的：   1. 学习和掌握正则表达式构造NFA的方法 2. 学习和掌握NFA转换为DFA的方法和步骤 3. 掌握字符集的基本运算（包括范围运算、连接运算、并运算、差运算等等） 4. 掌握NFA和DFA的基本方法和运算 | |
| 实验环境：笔记本电脑、Win10、idea Java环境 | |
| 实验内容及操作步骤：  **一、基本数据结构**  1）字符集     1. 字符集表定义      1. NFA或DFA定义     4）边定义    5）状态定义    6）转换枚举类型    7）词类别枚举类型      8）状态取值枚举类型    9）正则表达式定义    **二、针对字符集的创建，实现如下函数**  1）int range (char fromChar, char toChar)； // 字符的范围运算  函数作用：得到起始字符到结束字符之间的任意字符集  实现方法：新建一个字符集，直接加入字符集表即可。  实现函数：      2）int union(char c1, char c2)； // 字符的并运算  函数作用：进行字符与字符之间的并运算  实现方法：新建一个字符集对象，判断c1和c2是否相等，不相等的话新建一个段，加入字符集表  实现函数：    3）int union(int charSetId, char c)；// 字符集与字符之间的并运算  函数作用：进行字符和字符集之间的并运算  实现方法：先新建一个字符集，获取其stateId，把原字符集的所有段赋值给新建字符集，再给字符新建一个段，放入字符集表中。最后返回新得到的字符集的Id。  实现函数：    4）int union(int charSetId1,int charSetId2)；//字符集与字符集的并运算  函数作用：字符集与字符集的并运算  实现方法：直接将两个字符集的所有段加到新的字符集中，并返回相应Id即可。  实现函数：    5）int difference(int charSetId, char c)； // 字符集与字符之间的差运算  实现方法：判断字符是否在字符集中间，如果不在就将原字符集的所有段赋值给新的字符集，如果在的话就分为两个段，但是在边界条件上只需新建一个段。最后返回新字符集id即可。  实现函数：      **三、基于NFA的数据结构定义，按照最简NFA构造法，实现如下函数。**  1）Graph \* generateBasicNFA(DriverType driverType，int driverId );  函数作用：构造一个最简单的NFA  实现方法：构造两个状态，一个初状态，一个末状态。此处新增了一个category属性便于之后词法分析的识别。  实现函数：    2）Graph \* union(Graph \*pNFA1, Graph \*pNFA2)； // 并运算  函数作用：两个NFA进行并运算。  实现方法：新建一个图和初始状态，对原来的两个NFA进行等价改造，再合并其初始状态和终结状态即可。等价改造规则如下：      实现函数：    其中具体函数实现如下：   1. change函数：若初始状态存在入边，则新增一个初始状态，用ε边连接原初始状态；若终结状态存在出边，则构造一个状态设为终结状态，所有原终结状态连接该状态。     haveInSide函数：判断是否有边到达初始状态    haveOutSide函数：判断是否有边从终结状态出发     1. reNumber函数：对状态和边对应的状态重新编号，确保状态有序。      1. addTable函数：将参数NFA中的所有边、状态、结束状态（均已重新编号）加入到该NFA中。      1. mergeEnd函数：将pNFA1的终结状态合并到pNFA2中，终结状态的序号为最大值，即stateNum1+stateNum2-3      1. mergeStart函数：将pNFA2的初始状态合并到pNFA1中，初始状态的序号为0     3）Graph \* product(Graph \*pNFA1, Graph \*pNFA2); // 连接运算  函数作用：对两个NFA进行连接运算  实现思路：NFA的连接运算分为两种情况，情况之一是前一个图的接收状态有出边，后一个图的初状态有入边，则需要中间添加一个状态来防止倒灌；其余的情况则是前一个的接收状态和后一个的初状态合二为一，然后根据状态Id的变化添加Id和添加边即可。最后返回一个新建的图。    实现函数：    4）Graph \* plusClosure(Graph \*pNFA) //正闭包运算  函数作用：实现除了0个以外的图重复  实现思路：因为没有0到结束状态的干扰，可以直接添加一条边，从接收状态到初状态，转换条件为空。  实现函数：    5）Graph \* closure(Graph \*pNFA) // 闭包运算  函数作用：包含0次和很多次的图的重复  实现思路：在4的基础上增加一个从初始状态到接收状态的边，此处此时需要考虑初状态是否有入边，接受状态是否有出边，即首先进行规范化。最后返回新建的图。      实现函数：    6）Graph \* zeroOrOne(Graph \*pNFA)； // 0 或者 1 个运算。  函数作用：进行图的一次或者0次运算  实现思路：在实现之前先进行规范化，、再添加一条初状态到接受状态的边。    实现函数：    三、基于NFA数据结构定义，实现如下函数。  1）子集构造法  list<int> move(Graph\* pNFA, list<int> stateIdTable, int driverId)  函数作用：找到从一个状态集合通过某个转换条件可以跳转到的下一个状态集合  实现思路：循环该表的边集合，如果出现开始状态是存在对应集合中，并且是该引导条件Id，则将该状态id存入set（因为set可以消除重复元素）中，再将状态集合从set中放到list中并返回。  实现函数：    list<int> ε\_closure(Graph\* pNFA, list<int> stateIdTable)  函数作用：得到状态集合中的所有空转换的状态集合  实现思路：将传入参数中的状态集合在此图上能够通过空转移转换到的状态都存到set中，最后再将状态Id从set中转移到list中并进行返回。因为可能会出现连续的多个空转移，故可在外面进行使用的时候对该函数进行循环，直到找全其空转换状态集合为止。  实现函数：    list<int> DTran(Graph\* pNFA, list<int> stateIdTable,int driverId)  函数作用：将前面两个函数功能集合  实现思路：直接调用前面实现的函数并且对空转换进行多次循环  实现函数：    2）Graph \* NFA\_to\_DFA(Graph \*pNFA)  函数作用：将NFA转换为DFA  实现思路：保存整个图的驱动id，并且计算出初状态的空转换状态集合，然后通过此状态集合，对驱动id进行循环，调用DTran函数，得到可达的状态集合，并将这些状态集合都存入set中。接着从set中读取这些状态集合并且为其标号，向新建的DFA添加这些状态。接着通过这些状态再次对驱动id进行循环并且得到相应的状态集合，找到这些状态集合的对应的状态id，最后则得到了边，并将向DFA中添加这些边，最后返回DFA。  实现函数：    四、请以正则表达式(a|b)\*abb 来测试，检查实现代码的正确性  实现思路：依次构建正则表达式的NFA图，再将其转换为DFA图  实现代码：    生成结果:    代码输出如下：   |  |  |  | | --- | --- | --- | | a |  |  | | b |  |  | | a|b |  |  | | (a|b)\* |  |  | | (a|b)\*a |  |  | | (a|b)\*ab |  |  | | (a|b)\*abb |  |  | | (a|b)\*abb |  |  |   以 TINY 语言的词法来验证程序代码的正确性。   1. 构建字符集表      1. 构建关键字的NFA 2. if      1. then      1. else      1. end      1. repeat      1. until      1. read      1. write      1. 构建专用符号 2. +      1. -      1. \*      1. /      1. =      1. <      1. (      1. )      1. ;      1. :=      1. ID      1. NUM      1. 空格      1. 注释      1. 总结      1. 转换为DFA     收获与体会：   1. 通过此处实验对Java语言进行了一次回忆，并且对链表等数据结构的使用有了进一步的心得。 2. 通过此处练习，对最简生成NFA的方法有了进一步的理解，并且在编码实现的过程中可以逐渐熟练掌握最简生成NFA的方法。 3. 对于子集构造法将NFA转换为DFA也有了进一步的了解，在实现空转换函数和跳转函数的时候，对子集构造法有了进一步的学习和记忆。 4. 对于NFA和DFA的作用有了进一步的了解 5. 对字符集的构造有了一定的了解，进一步熟练了字符和字符，字符集和字符集，字符和字符集之间的运算。 6. 对NFA和NFA之间的运算也进一步的了解。 7. 掌握了构造一个基础的词法分享器的能力。 | |
| 实  验成绩 |  |