**实验报告**

**课程编号：**3132112040 **实践课程名称：形式语言与自动机 学年：2021-2022 学期：春**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **学生姓名** | |  | **学号** |  |
| **指导教师姓名** | |  | **起止时间** |  |
| **项目名称** | | ε-ΝFA ->DFA转换的C++实现 | | |
| **项**  **目**  **内**  **容** | **环境配置：**Visual Studio 2019或其它C语言编译器 输入文件input.txt要与源代码放在同一文件目录下  **设计方案：**  **数据结构**  定义结构体state。内部定义To0,To1,Toe三个set容器分别存放NFA读入0、1、ε后转移到的状态。定义数组eNFA和NFA分别用于存放读入的ε-ΝFA 和转换的NFA  定义结构体DFA\_state。set容器include用于存放DFA的状态（原NFA的状态集合）,To用于存放读入0、1后转移到的状态，isEnd用于标识该状态是否为终止状态（1是，0不是）。定义数组DFA用于存放转换后的DFA  **程序架构**   1. **main()函数：程序的主函数**，顺序调用ToNFA(),ToDFA(),Out()函数完成程序的执行     **2、ToNFA()函数：读入ε-ΝFA并将其转化为NFA。**  用C++中fstream流操作来完成输入文件input.txt的读入（注意：默认状态下标从1开始（即q1),状态数<100)，将起始状态和终止状态存入start和End中，用Read()函数处理输入数据并将其存入数组eNFA中。  先求NFA的终止状态F2。算法为遍历起始状态集合start，用ε\_CLOSURE（）函数求当前状态的ε闭包C。用set\_intersection方法求C与End的交集tmp，若交集不为空，则用set\_union方法求start与End的并集（即FU{q0}）并存入End中。此时End为NFA终止状态F2。  最后遍历eNFA中各状态，用Extended\_transferFunc（）函数求该状态经拓展转移函数后达到的状态，并存入NFA数组对应的下标中。如此便完成了ε-ΝFA到NFA的转换。   1. **ToDFA()函数：用子集构造法将NFA转换为等价的DFA**   定义集合State存放DFA的状态并用于判断转移到的状态是否为新状态。先将NFA的开始状态读入DFA中    遍历现有的DFA状态,设本轮状态为q,用transferFunc()函数将q读入0、1后跳转到的状态写入DFA对应下标index的To中。用transferFunc()函数的返回值来判断本轮转移到的状态集合中是否有NFA的终止状态，用count方法查询State中是否有该状态集合来判断其是否为新状态，是的话写入DFA对应下标n+s中。遍历结束后便完成了ΝFA到DFA的转换。   1. **Out()函数：将构造好的DFA输出**   用C++中fstream流操作来完成输出并存入到output.txt文件中，用C STL中的string完成输出格式的控制   1. **Read()函数：配合ToNFA()函数完成输入文件的读入并保存到eNFA对应下标中** 2. **ε\_CLOSURE()函数和traverse()函数：用递归回溯的方式实现求ε\_CLOSURE(q)**   用全局变量\_tmp记录递归过程中遍历到的状态     1. **Extended\_transferFunc()函数：将ε-NFA转为NFA的扩展转移函数实现**   先求ε\_CLOSURE(q)，再将ε\_CLOSURE(q)中的每个状态读入0/1之后转移到的状态存入tmp中，再对tmp状态集合求一次ε闭包并存入C中，最后将C存入NFA[q].To0/NFA[q].To1中。     1. **transferFunc()函数：将NFA转为DFA的转移函数实现**   遍历NFA[q].To0/NFA[q].To1，将得到的状态集合存入tmp\_中，若遍历过程中有遇到NFA的终止状态，则judge=1，最终返回judge。 | | | |
| **结**  **论** | **测试用例：**  输入:  0 1 epsilon  #q1 {} {q2} {q3}  q2 {q2} {} {q3}  \*q3 {} {} {}  输出：  0 1  #\*{q1} {} {q2,q3}  \*{q2,q3}{q2,q3} {}    推导：  bb7e0b6bebfb0eed3ea0fa7a83ee1fa  输入：  0 1 epsilon  #q1 {q1} {} {q2}  q2 {} {q2} {q3}  q3 {q4} {q3} {}  \*q4 {} {} {}  输出：  0 1  #{q1} {q1,q2,q3,q4} {q2,q3}  \*{q1,q2,q3,q4} {q1,q2,q3,q4} {q2,q3}  {q2,q3} {q4} {q2,q3}  \*{q4} {} {}    推导：  2e27c823cf16543e711242af77d2caa  上述的列举两个测试用例均被正确转换，并且经过多个其它测试用例测试，所得的结果均正确，故可认为本程序已基本实现了ε-ΝFA ->DFA的转换 | | | |