

编译原理作业 2

20152100121 林伟业

1 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

2 实验要求

1. 要提供一个源程序编辑界面，让用户输入正则表达式（可保存、打开源程序）
2. 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）
3. 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）
4. 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）
5. 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 语言
6. 应该书写完善的软件文档

3 程序说明

实验使用编程语言：Python。主要文件 NFA.py、DFA.py。程序运行：
python DFA.py -s "(a|b)*abb"。程序输出 NFA，DFA，最小化的 DFA，词法分析程序。

4 类和函数说明

NFA.py 文件

4.1 类 Condition

保存 NFA 一个节点开始和结束状态的类

4.2 类 Node

保存一个 NFA 机器的类，包含有开始结束状态和转移字符

4.3 类 Reg_To_NFA

实现正则表达式转 NFA。

思路：对正则表达式做开始前的处理，加入 & 连接运算符，把读到的一个字符 a 到 z，作为一个 NFA 机器添加到队列里，读取到的运算符保存到 opt_list 队列里，运算符队列不为空的时候，取出运算符做相应的操作。连接运算符：在 NFA 机器队列里拿出 2 个，合并为一台机器后再加入到 NFA 的队列。闭包运算符：在状态队列里保存着最新添加到 NFA 队列里机器的开始和结束状态，读取到闭包运算符就从状态队列里拿出最新的 2 个状态，做闭包运算。或运算：在 NFA 队列里拿出两太机器，加入新的开始和结束状态，还有 4 条边，把合并后的机器重新加入和 NFA 队列。括号运算：把括号里的表达式拿出来，递归处理。

主要函数：

1. def return_NFA_condition(self) 返回 NFA 的开始和结束状态
2. def pre_string(self): 正则表达式预处理函数，主要在两个字符之间加入 & 连接运算符，方便处理
3. def print_NFA(self) 输出 NFA
4. def add_to_NFA(self, start_condition, ch, end_condition) 添加一个 NFA 机器
5. def add_to_condition_list(self, start_condition, end_condition) 添加一个 NFA 的开始和结束状态

6. def check_oper(self) 检测运算符 &| ○ *, 不同运算符分别处理。
7. def get_one_char(self) 获取正则表达式的一个字符, 并返回
8. def to_NFA(self, string) 主要函数, 正则表达式转 NFA

DFA.py 文件

4.4 类 Node

保存 DFA 机器状态, 有开始, 结束状态和转移字符

4.5 类 NFA_To_DFA

实现 NFA 转 DFA。实现思路: 将保存 NFA 机器的列表转变存储结构, 方便 DFA 的状态转移遍历。使用 2 个列表 start_map 和 end_map。0->a->1, 保存为 start_map[0]=a, end_map[0]=1, 2->b->3, 保存为 start_map[2]=b, end_map[2]=3, 如此类推。首先是找到包含开始状态和经过 * 转换的所有状态, 作为 DFA 的第一个状态 A。从 A 出发遍历正则表达式的字母表, 转换出其他状态 BCD...

主要函数:

1. def pre_oper(self) 将 NFA 转化成两个列表
2. def get_alp_from_string(self) 获取正则表达式的字母表
3. def move(self, start, ch) 从开始状态 start 经过字符 ch 转移, 返回结束状态列表
4. def add_DFA_list(self, start, ch, end) 添加一个 DFA 机器到列表, 开始状态, 结束状态, 转移字符
5. def print_DFA(self) 输出 DFA
6. def make_condition_set(self) 主要算法, 子集构造法

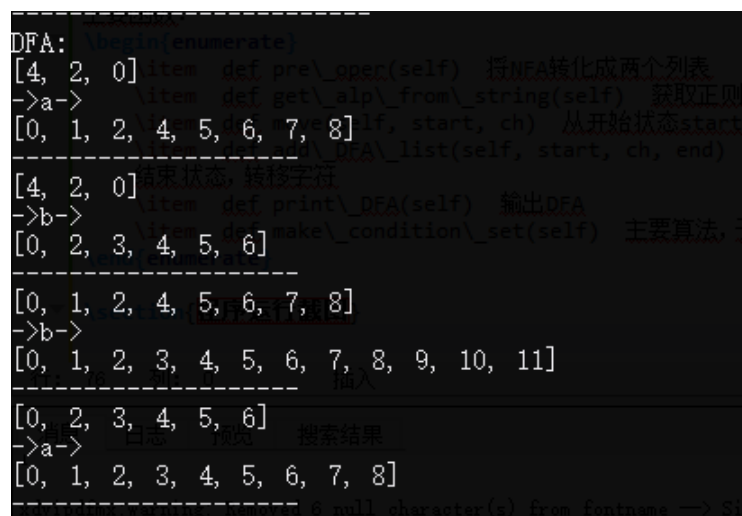
5 程序运行截图

测试正则表达式 $(a|b)^*abb$



```
PS C:\Users\sunn\Desktop\New folder\编译原理作业2> python .\DFA.py -s "(a|b)*abb"
NFA:
0->a->1
1->b->3
3->a->2
2->b->4
4->a->5
5->b->6
6->a->7
7->b->8
8->a->9
9->b->10
10->a->11
11->b->10
DFA:
0->a->1
1->b->3
3->a->2
2->b->4
4->a->5
5->b->6
6->a->7
7->b->8
8->a->9
9->b->10
10->a->11
11->b->10
```

图 1: NFA



```
DFA:
[4, 2, 0] item def pre_oper(self) 将NFA转化成两个列表
->a-> item def get_alp_from_string(self) 获取正则
[0, 1, 2, 4, 5, 6, 7, 8] if, start, ch) 从开始状态start
DFA_list(self, start, ch, end)
[4, 2, 0] 结束状态, 转移字符
->b-> item def print_DFA(self) 输出DFA
[0, 2, 3, 4, 5, 6] item def make_condition_set(self) 主要算法, 3
[0, 1, 2, 4, 5, 6, 7, 8]
->b->
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11]
[0, 2, 3, 4, 5, 6]
->a->
[0, 1, 2, 3, 4, 5, 6, 7, 8]
removed 6 null character(s) from fontname => Si
```

图 2: DFA