# 编译原理实践报告

目录

[编译原理实践报告 1](#_Toc534219824)

[一 总体说明 2](#_Toc534219825)

[二 正则表达式到最小化DFA 2](#_Toc534219826)

[2.1具体内容 2](#_Toc534219827)

[2.2算法描述 2](#_Toc534219828)

[2.3算法实现 2](#_Toc534219829)

[2.4 运行结果 3](#_Toc534219830)

[三 LL1文法 3](#_Toc534219831)

[3.1具体内容 3](#_Toc534219832)

[3.2算法描述 3](#_Toc534219833)

[3.3算法实现 3](#_Toc534219834)

[3.4 运行结果 3](#_Toc534219835)

[四 LR0文法 3](#_Toc534219836)

[4.1具体内容 3](#_Toc534219837)

[4.2算法描述 3](#_Toc534219838)

[4.3算法实现 3](#_Toc534219839)

[4.4 运行结果 3](#_Toc534219840)

[五 SLR1文法 3](#_Toc534219841)

[5.1具体内容 3](#_Toc534219842)

[5.2算法描述 3](#_Toc534219843)

[5.3算法实现 3](#_Toc534219844)

[5.4 运行结果 3](#_Toc534219845)

## 一 总体说明

所有程序均用python3编写，总共完成四个任务，分别为正则表达式到最小化dfa，LL1文法，LR0文法、SLR1文法，其中在当场测试中，正确的题目有：1.3 2.1 3.1 4.1 4.2 5.1 5.2，后续未对程序做出修改。

## 二 正则表达式到最小化DFA

### 2.1具体内容

输入一个正则表达式，输出其最小化dfa，输出结果使用graphviz以图片的形式展示

### 2.2算法描述

将输入的字符串转换成逆波兰表达式，然后遍历该字符串，与栈进行配合，得到该字符串所对应的nfa，去除去除空边合并状态得到dfa，然后再消除多余状态和合并等价状态得到最小dfa

### 2.3算法实现

#### 2.3.1 nfa的表示

对于nfa中的每一个结点，构造一个class来表示：

class Node(object): # nfa的每个状态

"""docstring for Node"""

def \_\_init\_\_(self):

self.name = -1

self.next = []

self.val = []

其中name为结点的编号，next中存储的是结点指向下一个结点的边，存储的是对应边上的值

对于整个nfa，同样构造一个class来表示：

class Nfa(object): # nfa

"""docstring for nfa"""

def \_\_init\_\_(self):

self.\_\_num = 0 # nfa的状态数

self.\_\_stack = [] # 构造nfa的栈

self.\_\_re = '' # 正则表达式

self.\_\_symbol = ('|', '·', '\*', '(', ')') # 符号元组

self.flag = [] # 判断nfa的图是否遍历完全

最后，栈中剩下的最后一个元素即为对应字符串的nfa

#### 2.3.2 nfa的构建

def create\_nfa(self):

for i in range(len(self.\_\_re)):

if(self.\_\_symbol.count(self.\_\_re[i]) == 0):

self.init(self.\_\_re[i])

else:

self.operation(self.\_\_re[i]) # 构建nfa

遍历逆波兰化的字符串，如果当前字符不为’|’、’\*’等符号，则为其构建一个nfa并压栈，如果当前输入为符号，则根据不同的符号对栈中的nfa进行不同的操作，最后得到最终的nfa

#### 2.3.3 nfa到dfa

### 2.4 运行结果

## 三 LL1文法

### 3.1具体内容

### 3.2算法描述

### 3.3算法实现

### 3.4 运行结果

## 四 LR0文法

### 4.1具体内容

### 4.2算法描述

### 4.3算法实现

### 4.4 运行结果

## 五 SLR1文法

### 5.1具体内容

### 5.2算法描述

### 5.3算法实现

### 5.4 运行结果