**华北水利水电大学 形式语言与自动机 实验报告**

**2022-2023 学年 第**  一 **学期 2020 级 人工智能 专业**

**指导老师 张瑞霞 学号 202018526 姓名 高树林**

## 实验三 由右线性文法构造与之等价的FA的程序实现

1. 实验目的
   1. 理解右线性文法与FA之间的等价性；
   2. 理解根据右线性文法构造FA的过程；
   3. 进一步理解由问题形式化到解决问题自动化的过程。
2. 实验内容

输入任意给定的右线性文法，转换为等价的有限自动机，并以状态转换图方式输出。

1. 实验指导

（1）关于右线性文法的输入，可以自设输入格式，可以设立自检程序自动检测用户输入的合法性；

（2）根据FA的形式化定义，自行设计FA的数据结构；

（3）FA的转换过程，可以看做是文法的解析过程，如何实现文法中非终结符到FA中状态的构造，如何根据产生式构造FA的状态转移是设计程序的关键；

（4）把自动机输出到屏幕或文件，以五元组形式输出或以状态转换图方式输出FA，要求状态结点大小位置布局合理，具有通用性，可以查找相关资料，使用相关包或插件，也可自己设计实现。

1. 实验设计与实现

包括主要设计思想、主要数据结构、主要程序设计流程图或伪代码、主要程序代码。

**设计思想：**

利用右线性文法转FA，最重要的就是转移函数的确定。相比于文法要先转到NFA，此外，文法转NFA只需要读取文法，对于候选式只有终极符和非终极符两种情况分类讨论，最后整理成一个双重字典嵌套的结构就能实现文法到NFA的等价转换。

**数据结构：**

在形式语言与自动机或者编译原理课程里面，更多关系的就是映射的关系。这时候就可以考虑使用python自带的数据结构——字典。在一一映射的情况下，一般采用单键值对的字典结构较为合适。在一对多的映射上（比如文法的产生式）一般采用列表来做字典的值。在像这种涉及转移函数，读取一个值能从一个状态转到另一个状态的情况下。使用字典的嵌套较为合适。

**流程图：**

输入文法

文法处理

遍历文法

合并字典

输出转移函数

图1 正规文法转NFA流程图

**程序代码：**

1. big = list(input('开始符(用","隔开):'))
2. small = list(input('字母表(用","隔开):'))
3. **print**('文法(格式为A -> a|aB|....),按"#"退出:')
4. point1 = {}
5. **while** 1:
6. ipt = input('')
7. **if** ipt == '#':
8. **break**
9. key,value = ipt.split(' -> ')[0],ipt.split(' -> ')[-1].split('|')
10. point1[key] = value
12. point = {'S': ['a', 'aA'], 'A': ['a', 'aA', 'cA', 'bB'], 'B': ['a', 'b', 'c', 'aB', 'bB', 'cB']}  # 文法
13. **print**(point)
14. **print**(point1)
15. big = list(set([i **for** i **in** str(point) **if** 'A' <= i <= 'Z']))+['Y']
16. **print**(big)
17. small = list(set([i **for** i **in** str(point) **if** 'a' <= i <= 'z']))+['Y']
18. """文法转NFA"""
19. NFA\_result = []
20. **for** key, value **in** point.items():
21. dic = []  # 所有读取一个值到另外状态的字典的列表
22. **for** i **in** value:  # 遍历point的一个value
23. **if** len(i) == 1:
24. dic.append({i[0]: 'Y'})  # 将转移函数加到列表里
25. **else**:
26. dic.append({i[0]: i[-1]})  # 将转移函数加到列表里
27. di = {}
28. **for** \_ **in** dic:  # 值相同的字典合并
29. **for** k, v **in** \_.items():
30. di.setdefault(k, []).append(v)
31. NFA\_result.append({key: di})  # 格式为{key1:{key2:value2}},其中key1是一个状态，key2是读入的字符,value2是转移状态的集合
32. # print(NFA\_result)
34. **print**('五元组形式输出结果为：')
35. **print**('M = (%s,%s,$,S,Y)' % (str(big).replace('[', '(').replace(']', ')'),str(small).replace('[', '(').replace(']', ')')))
36. **for** i **in** NFA\_result:
37. **for** key, value **in** i.items():
38. **for** j, k **in** value.items():
39. **print**('$(' + key + ',' + j + ') = ', str(k).replace('[', '{').replace(']', '}'))

**代码及流程说明：**

首先要输入文法。在输入文法之前要做相应的准备工作，比如输入开始符和字母表。这些内容是需要严格控制器输入与输出的，因为后需要对其输入的结构进行读取，因此在这部分不能有任何差错，否则正则文法无法被识别，就不会转换为相对应的NFA了。在输入文法的时候写一个死循环，当输入特地的符号的时候才能退出循环，这一点我的灵感来自于Linux的VIM文本编译器。当文法输入完毕之后，键入#键就能退出输入。接下来就是对干刚刚输入的符号进行提取，用split函数将文法分隔开，并放在一个列表中，最后将其作为字典的值加到字典里面。当上述操作结束后，代码来到了对上面得到的字典操作的部分。在这里，遍历字典的键值，键作为最后转移函数的开始符。该键对应的值是能达到的状态，涉及到一次达到和多次的达到，因此要对值列表里面的元素进行遍历，遍历到当前位置时，如果当前字母的长度为1，说明直接能够到达终态，此时引入一个状态Y表示终态，结果，这个过程被记录下来，其含义为Key读取了a(长度为1)到达了Y状态。当当前字母的长度大于1的时候，就说明通过一个函数到不了终态，因此将这个状态记录下来，其含义为Key读取了a(长度为1)到达了Key2状态。当所有的键被遍历一遍之后，得到的结果为{key1:{key2:value2}},其中key1是一个状态，key2是读入的字符,value2是转移状态的集合。之后利用循环语句将最终得到的状态函数输出在屏幕上。

1. 实验结果及分析

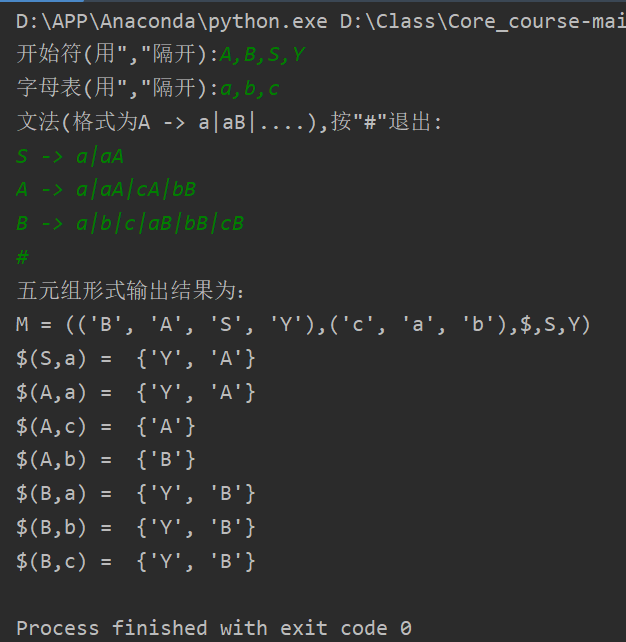


图2 正规文法转化为NFA的结果

如上图输出结果所示，当S读取a的时候，能转移到Y(终止状态)和A状态，这与S -> a|aA这个产生式是等价的。当A读取a的时候，能转移到Y(终止状态)和A状态，这与A -> a|aA这个产生式是等价的。当A读取b的时候，能转移到B状态，这与A -> bB这个产生式是等价的。当A读取c的时候，能转移到A状态，这与A -> cA这个产生式是等价的。当B读取a的时候，能转移到Y(终止状态)和B状态，这与B -> a|aB这个产生式是等价的。当B读取b的时候，能转移到Y(终止状态)和B状态，这与B -> b|bB这个产生式是等价的。当B读取c的时候，能转移到Y(终止状态)和B状态，这与B -> c|cB这个产生式是等价的。因此该NFA和对应的正则文法是等价的。

1. 实验总结

主要总结所遇到的问题及其解决方法（可以是成功解决的方法或部分解决方法或无法解决的原因）。

**问题1：**在实验中，我最开始使用一级字典对文法进行状态的提取，最后得到的提取状态结果如下图3所示。

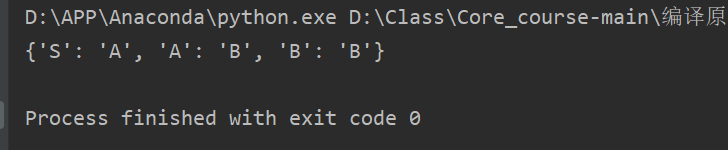


图3 问题1的错误结果

经过观察与分析，我发现这里面的元素都是对应文法中的最后一个候选式。出现这种问题的最大的可能性就是我没有保存遍历过程中得到的结果，在迭代的过程中把对应的状态覆盖掉了，因此我回过头来新建列表，把每一次得到的结果都加到列表里面，最后把列表赋值给字典的值，这样就不会有覆盖的现象。最后得到的正确结果如下图4所示。

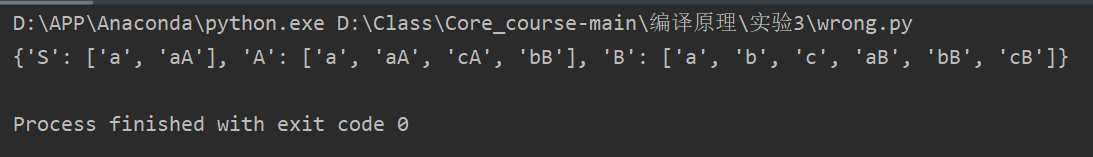


图4 问题1的正确结果

**问题2：**在实验中我在提取Q和字母表的时候出现了大量重复，重复的效果如下图5所示。

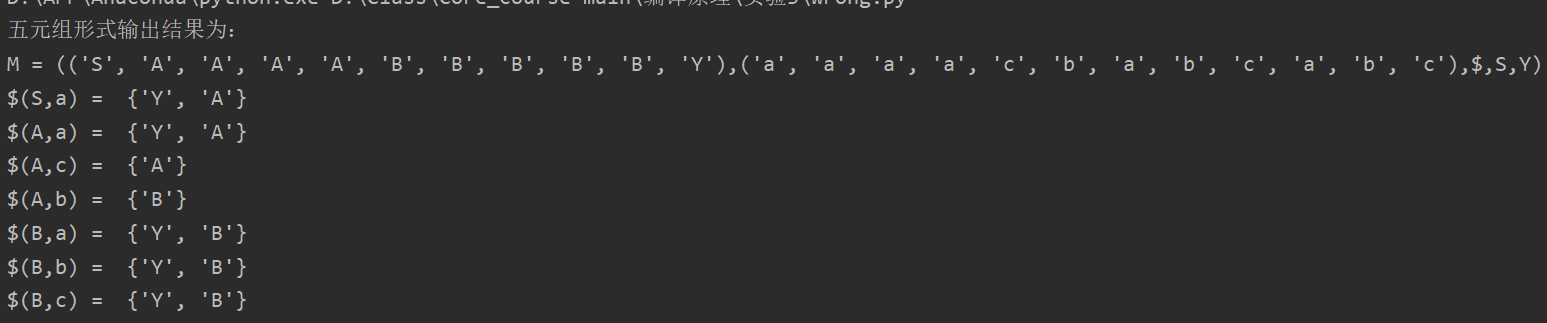


图5 问题2的错误结果

经过分析，这是由于我在提取这些终极符和非终极符的时候，没有去重。我想到python的另外一个数据结构——集合（set），能够实现去重的效果。因此我修改代码。在原列表的基础上强制转化为集合，后又强制转化为列表，实现了去重功能。最后得到的正确结果如图6所示。肉眼可见的短了很多。

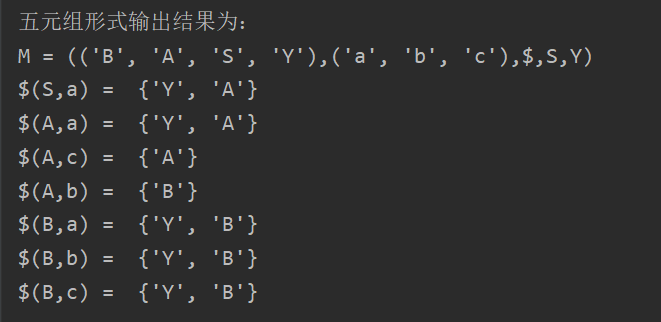


图6 问题2的正确结果