**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**

## 实验一 文法产生语言

1. **实验目的**
2. 掌握文法的表示方式；
3. 理解文法产生语言的过程，并理解有穷文法产生无穷语言。
4. 进一步理解由问题形式化到问题解决自动化的过程。
5. **实验内容**

在掌握文法形式化的基础上，根据正规文法，产生长度不超过给定整数N的句子。

1. **实验指导**
   1. 文法的表示：

文法的形式化表示G=<V，T，P，S>是一个四元组，在程序设计中，可以使用适当的数据结构来表示四元组中的每一个分量。例如四元式表示：采用字符数组表示字母表和变量表，字符表示开始符号，字符串表示产生式组。（产生式符号向右箭头没有可以用“->”表示）

（2）关于文法的存储

可以使用两种存储方式：程序方式和文件方式。程序方式是指将文法的四元组固定保存在程序中， 即一个程序处理一个文法。 文件方式是指将文法的四元组用文本方式存储，并定义格式，相应程序可以处理任意文法。

（3）句子的产生

如何根据正规文法产生长度不超过N的句子，是本实验的关键点。是递归产生？是通过设置推导次数限制产生？是先枚举生产所有再根据合法性判定取舍？还是采用其它策略？是产生长度不超过N的所有句子，还是产生一部分句子？设计出你认为的合适方法。

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。

由于在形式语言与自动机中，我们用的最多的时正规文法（右线性文法），因此本实验将利用正规文法进行文法产生句子。

**设计思想：**遍历字符串，当遇到非终极符的时候，进行文法推导，直到句子中不含非终极符。在每一次推导产生新的句子时，需要检验该句子是否符合要求（即判断该句子是否含非终极符，该句子的长度是否是我们所想要的），如果是，就将其输出，如果不满足要求则做其他处理。

**主要数据结构：**主要是利用了python中的字典数据结构。字典的键表示为产生式左端的非终极符，字典的值表示为产生式的右部的若干候选式，通过键值对的索引来定位到产生式的语法信息。

**主要程序流程图：**程序流程图如下图1所示。

是终极符

定义法四元组

获取产生式中的语法P

获取开始符S

遍历句子，找到非终极符

进行语法推导，产生新句子

不再进行推导

符合要求，输出

S是非终极符

长度非法

长度、语法符合

S

有非终极符

语法不符合

初始符不能为终极符

无非终结符

图1 程序流程图

**实验代码：**

1. **import** os

4. **def** loop(string):
5. **for** i **in** range(len(string)):  # 遍历当前句子，判断其中是否含有非终极符
6. **if** string[i] **in** convert.keys():
7. **for** j **in** convert[string[i]]:  # 如果当前句子存在非终极符，就进行文法的推导
8. # print(string)
9. new\_string = string[0:i] + j + string[i + 1:]  # 推到后产生的新句子
10. # print(new\_string)
11. **if** len(new\_string) > max\_length:  # 判断句子长度是否合法，如果不合法就不再进行推导（因为继续推导不会更短）
12. **break**  # 如果句子长度合法、且不含非终极符，那么久成功的产生了一个句子
13. **elif** new\_string.islower():
14. f.write(new\_string + '\n')                                             # 写入结果
15. **print**(new\_string)
16. # print('')
17. result.append(new\_string)
18. **else**:  # 如果句子长度合法、但是包含非终极符，那么就递归进行推导
19. # print(new\_string)
20. loop(new\_string)

23. **if** \_\_name\_\_ == '\_\_main\_\_':
24. content = []
25. convert = {}
26. """存储文法"""
27. V = "V=" + input('非终极符(大写字母,用英文逗号隔开,并用大括号括起来）：') + '\n'
28. T = "T=" + input('终极符(小写字母,用英文逗号隔开并用大括号括起来）：') + '\n'
29. **print**('文法(一行输入，不同文法之间以"#"开始,"\\n"结尾，用空格隔开，内部用 "|" 和 "->" 隔开）：')
30. P = []
31. num = V.count(',') + 1
32. **for** i **in** range(num):                                                    # 循环输入语法
33. **print**('产生式%d:' % (i+1))
34. P.append(input() + '\n')
35. S = 'S=' + input('开始符：')
36. name = input('存储文件名称：')
37. f = open(name + '.txt', 'w', encoding='utf-8')
38. f.write(V)                                                              # 写入非终极符
39. f.write(T)                                                              # 写入终极符
40. f.write('P\n')                                                          # 声明以下内容是产生式
41. **for** i **in** P:                                                             # 写入产生式
42. f.write(i)
43. f.write(S)                                                              # 写入开始符
44. f.write('\n\n利用上述文法，得到的结果是:\n\n')                            # 写入结果声明
46. content = [V] + [T] + ['P\n'] + P + [S]
47. **for** i **in** content[3:-1]:  # 获取产生式中的语法信息，将其存储在convert字典中
48. key = i[1:-1][0]
49. value = i[1:-1][3:].split('|')
50. convert[key] = value
51. result = []
52. max\_length = int(input('输入句子允许的最大长度：'))  # 设置最大长度（因为有些文法能产生无穷长度的句子）
54. string = content[-1][-1]  # 获取开始符S
55. # print(string)
56. **if** string.islower():  # 如果开始符本身就是终极符，就需要进行异常处理
57. **print**('错误！初始符不能为终结符！！！')
58. **else**:
59. loop(string)  # 递归调用loop函数完成句子的产生
60. **print**("写入文件至%s，此次共产生了%d个句子" % (os.getcwd() + '\\' + name + '.txt', len(set(result))))  # 产生句子的总数

**代码及流程解释：**

在主函数中，首先就是要定义文法四元组的各个部分，并将其放在content列表中。之后获取产生式中的语法信息，将其存储在convert字典中。之后再获取开始符S，这些都是利用python自带的切片属性和分割函数来实现的。同时，要将定义的四元组的各个部分以字符串的形式存储起来，因此要定义存储文件的名称。由于有些文法能够产生无穷长度的句子，在这里要设置一个最大长度。本人亲测，不设置最大长度会无限递归到第998层，结果是{a}\*，这显然不是所需要的。在这之后需要对开始符进行讨论，如果它本身就是一个终极符，一般就不需要经过语法推导，并且在正规文法（右线性文法）中也不会出现开始符是终极符的情况，这里多考虑一步是为了程序的健壮性。如果开始符是非终极符，就进入函数内部。

在函数内部，首先要遍历句子的每一个字符，判断其中是否含有非终极符（这一步没必要，因为在主体函数中已经判断过这里面一定有非终极符），已经存在非终极符了，就要对其进行推导，利用前文获取的语法得到产生的新句子（这里的新句子一定不要和原来传入的句子名称一样，否则会导致可能性的丢失），首先判断句子长度是否合法，因为不管再好的句子，长度不合法就一定不能要，即便能继续推导，有线性文法也不会使句子变得更短。因此考虑长度的优先级是最高的，其次考虑是否含有非终极符·，若不含有非终极符，就得到该情况下的合法句子，本次循环结束，将结果输出出来并存储在新的文本文档里面。若含有非终极符，就需要继续推导，步骤和原先一样，相当与原先问题扩大的子问题（叫父问题可能更合适，因为长度比原来的更长），因此可以用递归来调用。

1. **实验结果及分析**

通过四中的实验设计和代码，本实验的结果如下图2-3所示。

**结果1：**当定义初始符误定义为终极符时，运行上述代码得到的结果如下图2所示.。

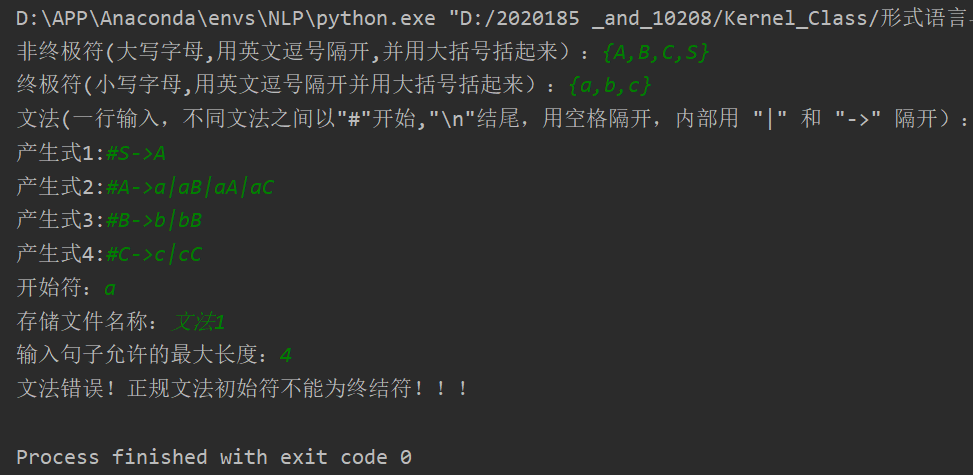


图2初始符为终极符的文法输入

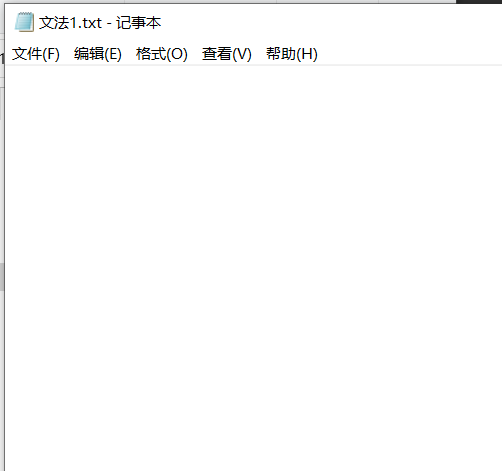


图3初始符为终极符的结果

**结果2：**当定义的初始符为非终极符时，运行上述代码得到的结果如下图5所示.。

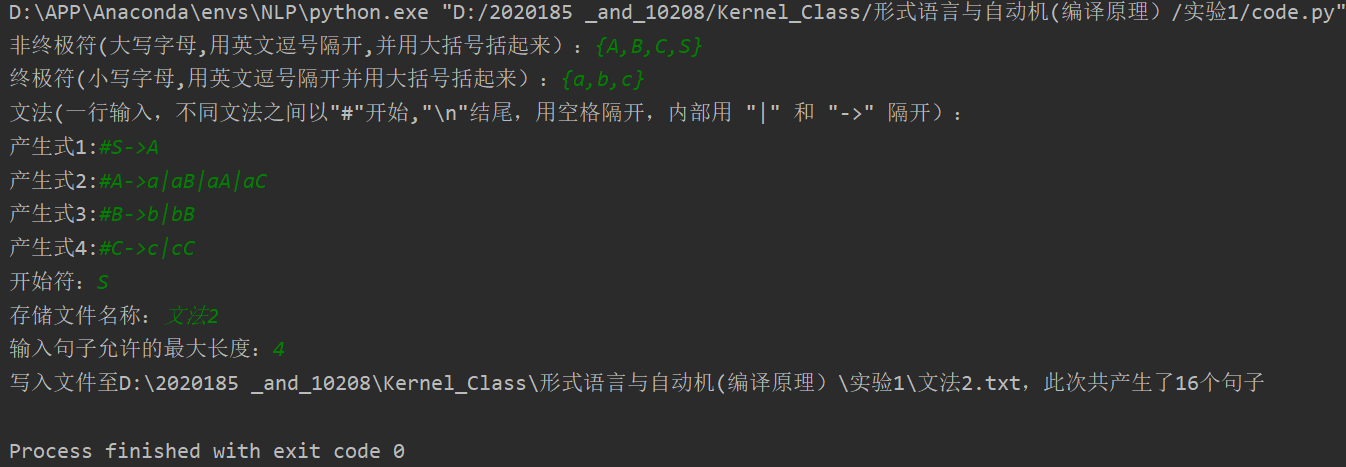


图4 初始符为非终极符的文法输入

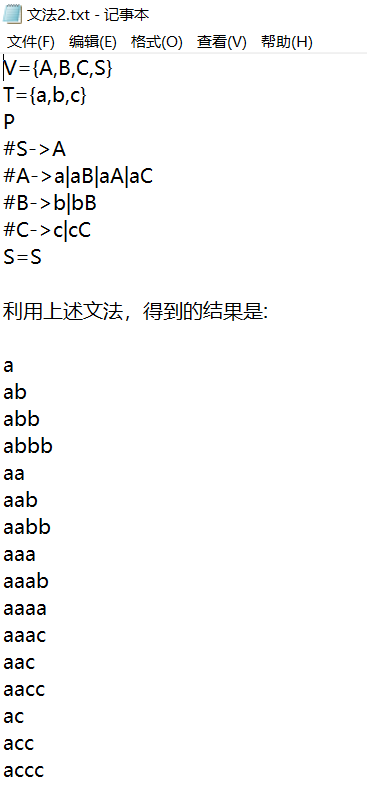


图5初始符为终极符的结果

**实验结果分析：**

对第1个结果，利用右线性文法推导得：

S ->A

->a

对第2个结果，利用右线性文法推导得：

S ->A

->aB

->ab

对第3个结果，利用右线性文法推导得：

S ->A

->aB

->abB

->abb

对第4个结果，利用右线性文法推导得：

S ->A

->aB

->abB

->abbB

->abbb

对第5个结果，利用右线性文法推导得：

S ->A

->aA

->aa

对第6个结果，利用右线性文法推导得：

S ->A

->aA

->aaB

->aab

对第7个结果，利用右线性文法推导得：

S ->A

->aA

->aaB

->aabB

->aabb

对第8个结果，利用右线性文法推导得：

S ->A

->aA

->aaA

->aaa

对第9个结果，利用右线性文法推导得：

S ->A

->aA

->aaA

->aaaB

->aaab

对第10个结果，利用右线性文法推导得：

S ->A

->aA

->aaA

->aaaA

->aaaa

对第11个结果，利用右线性文法推导得：

S ->A

->aA

->aaA

->aaaC

->aaac

对第12个结果，利用右线性文法推导得：

S ->A

->aA

->aaC

->aac

对第13个结果，利用右线性文法推导得：

S ->A

->aA

->aaC

->aacC

->aacc

对第14个结果，利用右线性文法推导得：

S ->A

->aC

->ac

对第15个结果，利用右线性文法推导得：

S ->A

->aC

->acc

对第16个结果，利用右线性文法推导得：

S ->A

->aC

->acC

->accC

->accc

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：**在调试代码的过程中，我遇到了如下报错，错误截图如下图6所示。

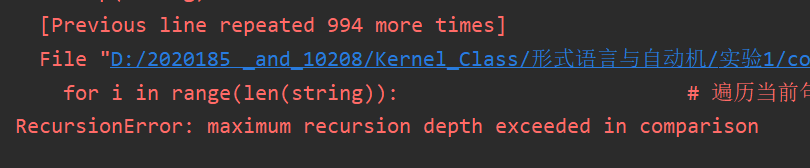


图6 问题1报错截图

**解决1：**报错显示的是递归已达到上限，但是在题目中我原本是设置了长度阈值的，后来我用调试器调试代码的时候发现，原来是我一直用string来作为长度的判断，在这个过程中，string的值一直没有变化，长度恒为1，因此就会导致一直递归的问题。后来我将新生成的字符串赋值给了new\_string后问题得到解决。