北京郵電大學

形式语言与自动机报告二

班级： 2022211305

组号： 05

组员： 罗星贺，吴林涛，朱远川，叶宇洋

学院： 计算机学院 (国家示范性软件学院)

时间： 2024 年 7 月 3 日

目录

1　实验环境描述

1. OS: win11
2. 编程语言：python 3.11.9
3. IDE：pycharm 2024.1

2　设计上下文无关文法的变换算法

2.1　实验目的

编程实现上下文无关文法的变换算法，用于消除文法中的 ε 产生式、单产生式、以及无用符号。

2.2　实验内容及要求

下文无关文法对产生式的右部没有限制，这种完全自由的形式有时会对文法分析带来不良影响，而对文法的某些限制形式在应用中更方便。通过上下文无关文法的变换，在不改变文法的语言生成能力的前提下，可以消除文法中的 产生式、单产生式、以及无用符号。

要求编程实现消除上下文无关文法中的 产生式、单产生式、以及无用符号的算法。输入是一个上下文无关文法，输出是与该文法等价的没有 产生式、单产生式、无用符号的上下文无关文法。

至少使用如下文法中的产生式进行程序的正确性验证。

$$
\begin{aligned}
\mathrm{S} &\rightarrow \mathrm{a|bA|B|ccD} \\
\mathrm{A} &\rightarrow \mathrm{abB|} \\
\mathrm{B} &\rightarrow \mathrm{aA} \\
\mathrm{C} &\rightarrow \mathrm{ddC} \\
\mathrm{D} &\rightarrow \mathrm{ddd}
\end{aligned}
$$

2.3　方法概述

实现上下文无关文法的简化分为 4 步

1. 去除文法中所有非可终结状态
2. 去除文法中所有非初态可达状态和终止符

3. 去 ε 生成式

4. 去单生成式

2.4 算法描述

2.4.1 去非可终结状态算法

该算法采用了 BFS 的思想，方法如下

1. 建立可终结状态集合，并将终结符加入其中

2. 遍历所有生成式，如果有状态满足其存在一个生成式均由可终结状态组成，说明该状态为可终结状态，加入到可终结状态集合中

3. 重复上一步，直到不再有新的可终结状态产生

4. 去除掉文法中所有非可终结状态及其相关生成式

2.4.2 去非可达状态及终结符算法

该算法也采用了 BFS 的思想，方法如下

1. 建立可达状态集合，并将初态加入其中

2. 遍历所有可达状态的生成式，由可达状态生成得到的内容一定是可达的，所以将生成得到的串中所有新状态以及新终结符添加到可达状态集合中

3. 重复上一步，直到不再有新的可达状态产生

4. 去除掉文法中所有不可达状态及其相关生成式

2.4.3 去 ε 生成式算法

方法如下

1. 利用去非可终结状态算法，将最开始的可终结状态集合设为仅含 ε 元素，得到所有可导空状态集合

2. 遍历所有生成式，对于所有生成式中的所有可导空状态，将生成式分裂为含这些可导空状态任意组合的生成式，并且移除所有生成空的生成式

3. 如果初态再可导空状态集合中，则建立新初态，并添加新初态到老初态和空的生成式，否则将空串字符从终结符中删除

4. 重新运行去非可终结状态算法，去除掉因为移除生成空的生成式而不可终结的状态

重点描述一下分裂算法

对于一个生成式，建立一个映射集合，键为生成式，值为下一个需要更新的位置

一开始只有原生成式在映射集合中，值为 0，代表下一个更新的位置为 0，也就是第一个字符

之后遍历映射集合中所有的映射，对每个生成式而言从映射的值开始向右遍历，直到遍历串的末尾或者遍历到可导空状态

如果遍历到末尾，则将生成式映射的设为串长，代表该生成式已经更新完毕

如果遍历到可导空状态，则将原生成式的映射的值改为可导空状态的下一个位置，并且添加新的映射：键为去掉可导空状态的生成式，值为原可导空状态的位置

如此反复更新映射集合，直到映射集合不再更新，说明所有生成式分裂完毕

举个例子，$aSbS$，其中 S 是可导空状态

最开始映射集合为：$\{\ aSbS:0\ \}$

之后从 0 遍历 $aSbS$，发现位置 1 出现可导空状态，先将原映射的值改为可导空状态出现位置加 1，再添加新的去可导空状态的映射

此时映射集合为 $\{\ aSbS:2, abS:1\ \}$

从新遍历映射集合，先从 2 遍历 $aSbS$，发现位置 3 出现可导空状态，更新映射集合

此时映射集合为 $\{\ aSbS:4, aSb:3, abS:1\ \}$

再从 1 遍历 abS，发现位置 2 出现可导空状态，更新映射集合

此时映射集合为 $\{\ aSbS:4, aSb:3, abS:3, ab:2\ \}$

此时所有映射的值都为键生成式的长度，无法再更新，说明所有生成式分裂完毕，算法结束

2.4.4 去单生成式算法

方法如下

1. 对于每一个状态而言，建立一个仅含该状态的单生成状态集合，
2. 遍历单生成状态的所有生成式，将其中所有单生成式的目标状态添加到单生成集合中
3. 重复上一步，直到没有新单生成状态产生
4. 对于该状态而言，将其生成式改为所有单生成集合中状态的生成式，并去除掉其中的单生成式
5. 对所有状态重复上述操作

2.5 数据结构描述

建立了 Grammar 类，属性如下

属性	描述
N	非终结符集合
T	终结符集合
P	生成式集合，为字典，键为状态，值为集合，包含由该状态生成的生成式，生成式为元组
S	初态

方法如下：

方法	描述
search1	给予终结符集合，返回所有一步可终结的状态集合，algorithm1 和 3 中使用
search2	给予初始状态集合，返回所有一步可达的状态集合，algorithm2 中使用
seperate	根据可导空状态将生成式分裂，algorithm3 中使用
search4	给予一个状态集合，返回所有一步可单生成的状态，algorithm4 中使用
algorithm1	去除文法中所有非可终结状态
algorithm2	去除文法中所有非初态可达状态和终止符
algorithm3	去 ε 生成式
algorithm4	去单生成式
update	根据新非终结集和新终结符集更新文法，algorithm1 和 2 中使用
print	打印文法
generate	根据文法生成自动机

2.6 结果及其分析

代表空串

- 未简化文法：

初态：

S

状态：

A，C，B，S，D，

字符：

a，d，c，#，b，

边：

```
S  --->  B|ccD|a|bA
A  --->  #|abB
B  --->  aA
C  --->  ddC
D  --->  ddd
```

- 去除文法中所有非可终结状态后：

初态：

S

状态：

B，S，D，A，

字符：

a，d，c，#，b，

边：

```
S  --->  B|ccD|a|bA
A  --->  #|abB
B  --->  aA
D  --->  ddd
```

其中 C 状态不可终结状态，将其去除

- 去除文法中所有非初态可达状态和终止符后：

初态：

S

状态：

B，S，A，D，

字符：

c，#，b，a，d，

边：

```
S   --->   B|ccD|a|bA
A   --->   #|abB
B   --->   aA
D   --->   ddd
```

没有不可达状态和终结符

- 去 ε 生成式后：

  初态：

  S

  状态：

  B，S，A，D，

  字符：

  b，a，d，c，

  边：

  ```
  S   --->   B|b|bA|a|ccD
  A   --->   abB
  B   --->   aA|a
  D   --->   ddd
  ```

  A 状态为可导空状态，其中 S 和 B 的生成式含有 A，bA 分裂为 b 和 A，aA 分裂为 a 和 aA，去除掉 A 导空串的生成式

  由于初态不为可导空状态，空串字符被移除出终结符集合

- 去单生成式后：

初态：

S

状态：

B，S，A，D，

字符：

b，a，d，c，

边：

```
S  --->  aA|b|bA|a|ccD
A  --->  abB
B  --->  aA|a
D  --->  ddd
```

S 有单生成式 B，将 B 的生成式 aA 和 a 加入到 S 的生成式中，其中生成式 a 重复了，去掉 S 的单生成式 B

3 构造与下推自动机等价的上下文无关文法

3.1 实验目的

编程实现由下推自动机构造等价的上下文无关文法的算法。

3.2 实验内容及要求

下推自动机和上下文无关文法，是用于描述上下文无关语言的两种方式。对于给定的一个下推自动机，必存在一个上下文无关文法，使得该文法产生的语言与下推自动机接受的语言等价。

要求

1. 编程实现由下推自动机构造等价的上下文无关文法的算法。输入为一个下推自动机，输出为与该下推自动机等价的上下文无关文法。

2. 将输出的等价上下文无关文法作为输入，利用所实现的上下文无关文法变换算法，输出与该文法等价的没有　产生式、单产生式、无用符号的上下文无关文法。

至少使用如下的下推自动机进行程序的正确性验证。

设 PDA M＝（{q0,q1},{a,b},{B,z0}, δ ,q0,z0, Φ ）
δ 定义为： δ （q0,b,z0）={(q0,Bz0)}
δ （q0,b,B）={(q0,BB)}
δ （q0,a,B）={(q1, ε)}
δ （q1,a,B）={(q1, ε)}
δ （q1, ε ,B）={(q1, ε)}
δ （q1, ε ,z0）={(q1, ε)}

3.3 方法概述

大致分为 4 步

1. 建立一个初态 S，并将 $S \to (q_0, Z_0, q)$，添加到生成式中，q 为任意状态
2. 对于 $\delta(q, t, X) \to (q', \varepsilon)$ 而言，生成式中添加 $(q, X, q') \to t$
3. 对于 $\delta(q, t, X) \to (q_0, X_1X_2...X_N)$ 而言，生成式中添加
   $(q, X, q_n) \to t(q_0, X_1, q_1)(q_1, X_2, q_2)...(q_{n-1}, X_n, q_n)$
   其中 $q_1, q_2...q_n$ 为任意状态
4. 基于上面的生成式生成文法并简化

3.4 算法描述

该功能算法涉及的比较少，主要描述步骤 3 的算法

主要思想为遍历栈序列，一点一点延长生成式

考虑转移 $\delta(q_a, t, X) \to (q_a, X_1X_2)$，假设只有 q_a, q_b 两种状态

建立映射集合，键为当前的生成式，值为当前生成式最后一元状态的末态，也就是下一个要生成的状态的初态

初始时映射为字符对转移结果的初态，映射集合为 $\{t : q_a\}$，说明下一个要生成的状态的初态为 q_a

之后遍历栈序列，准备延长生成式

遍历到 X_1，遍历所有映射和所有状态，生成延长后的映射集合，键为延长后生成式，值为新末态，集合为 $\{t(q_a, X_1, q_a) : q_a, t(q_a, X_1, q_b) : q_b\}$

遍历到 X_2 重复上一步操作，集合如下

$\{t(q_a, X_1, q_a)(q_a, X_2, q_a) : q_a, t(q_a, X_1, q_a)(q_a, X_2, q_b) : q_b,$
$t(q_a, X_1, q_b)(q_b, X_2, q_a) : q_a, t(q_a, X_1, q_b)(q_b, X_2, q_b) : q_b\}$

最后根据映射的值将这些生成式补齐，比如 $t(q_a, X_1, q_b)(q_b, X_2, q_b)$ 根据末状态 q_b 得出源状态为 (q_a, X_1, q_b)

3.5 数据结构描述

建立了 PA 类，属性如下

属性	描述
Q	状态集合
T	字符集合
Gamma	栈状态集合
delta	转移字典，键为初态三元组，值为集合，包含转移初态和栈序列二元组，栈序列为多元组
q0	初态
Z	栈初态

方法如下

方法	描述
extend	输入映射集合，将集合中生成式根据其值延长
generate	根据自动机生成文法
print	打印自动机

3.6 结果及其分析

1. 自动机如下

设 PDA M＝（{q0,q1},{a,b},{B,z0}, δ ,q0,z0, Φ ）

δ 定义为： δ （q0,b,z0）={(q0,Bz0)}

δ （q0,b,B）={(q0,BB)}

δ （q0,a,B）={(q1, ε)}

δ （q1,a,B）={(q1, ε)}

δ （q1, ε ,B）={(q1, ε)}

δ （q1, ε ,z0）={(q1, ε)}

2. 生成的文法如下

初态：

@S

状态：

@S, ('q1', 'Z0', 'q1'), ('q1', 'B', 'q1'), ('q0', 'Z0', 'q1'), ('q0', 'Z0', 'q0'), ('q0', 'B', 'q1'),

字符：

b, #, a,

边：

@S ---> ('q0', 'Z0', 'q1')|('q0', 'Z0', 'q0')

('q0', 'Z0', 'q1') ---> b('q0', 'B', 'q0')('q0', 'Z0', 'q1')|b('q0', 'B', 'q1')('q1', 'Z0', 'q1')

('q0', 'Z0', 'q0') ---> b('q0', 'B', 'q1')('q1', 'Z0', 'q0')|b('q0', 'B', 'q0')('q0', 'Z0', 'q0')

('q0', 'B', 'q1') ---> b('q0', 'B', 'q1')('q1', 'B', 'q1')|b('q0', 'B', 'q0')('q0', 'B', 'q1')|a

('q0', 'B', 'q0') ---> b('q0', 'B', 'q0')('q0', 'B', 'q0')|b('q0', 'B', 'q1')('q1', 'B', 'q0')

('q1', 'B', 'q1') ---> #|a

('q1', 'Z0', 'q1') ---> #

3. 去除文法中所有非可终结状态后：

初态：

@S

状态：

@S, ('q1', 'Z0', 'q1'), ('q1', 'B', 'q1'), ('q0', 'Z0', 'q1'), ('q0', 'B', 'q1'),

字符：

b, #, a,

边：

@S ---> ('q0', 'Z0', 'q1')

('q0', 'Z0', 'q1') ---> b('q0', 'B', 'q1')('q1', 'Z0', 'q1')

('q0', 'B', 'q1') ---> b('q0', 'B', 'q1')('q1', 'B', 'q1')|a

('q1', 'B', 'q1') ---> #|a

('q1', 'Z0', 'q1') ---> #

4. 去除文法中所有非初态可达状态和终止符后：

初态：

@S

状态：

@S, ('q1', 'Z0', 'q1'), ('q1', 'B', 'q1'), ('q0', 'Z0', 'q1'), ('q0', 'B', 'q1'),

字符：

b, #, a,

边：

@S ---> ('q0', 'Z0', 'q1')

('q0', 'Z0', 'q1') ---> b('q0', 'B', 'q1')('q1', 'Z0', 'q1')

('q0', 'B', 'q1') ---> b('q0', 'B', 'q1')('q1', 'B', 'q1')|a

('q1', 'B', 'q1') ---> #|a

('q1', 'Z0', 'q1') ---> #

5. 去 ε 生成式后：

```
初态：
@S
状态：
('q1', 'B', 'q1'), ('q0', 'B', 'q1'), @S, ('q0', 'Z0', 'q1'),
字符：
b, #, a,
边：
@S  --->  ('q0', 'Z0', 'q1')
('q0', 'Z0', 'q1')  --->  b('q0', 'B', 'q1')
('q0', 'B', 'q1')  --->  b('q0', 'B', 'q1')('q1', 'B', 'q1')|b('q0', 'B', 'q1')|a
('q1', 'B', 'q1')  --->  a
```

注意状态 $(q1, Z0, q1)$ 仅能导出空串，去 ε 生成式后，该状态没有生成式，所以该状态应当被去除。由于该状态为非可终结状态，再次调用 algorithm1 后，可将其去除

6. 去单生成式后：

```
初态：
@S
状态：
('q1', 'B', 'q1'), ('q0', 'B', 'q1'), @S, ('q0', 'Z0', 'q1'),
字符：
b, #, a,
边：
@S  --->  b('q0', 'B', 'q1')
('q0', 'Z0', 'q1')  --->  b('q0', 'B', 'q1')
('q0', 'B', 'q1')  --->  b('q0', 'B', 'q1')('q1', 'B', 'q1')|b('q0', 'B', 'q1')|a
('q1', 'B', 'q1')  --->  a
```

4 额外内容：文法生成自动机

4.1 方法概述

1. 将 q_0 添加到自动机初态和初态集合中
2. 将文法初态添加到自动机栈初态
3. 将文法终结符添加到自动机字符集中
4. 将文法终结和非终结符添加到自动机栈字符中
5. 遍历每一个生成式 $S \to abc$，自动机中添加 $(q_0, \varepsilon, S) \to (q_0, abc)$
6. 遍历每一个终结符 a，自动机中添加 $(q_0, a, a) \to (q_0, \varepsilon)$

4.2 结果及其分析

1. 文法如下；

初态：

S

状态：

B，S，A，D，

字符：

b，a，d，c，

边：

```
S  --->  aA|b|bA|a|ccD
A  --->  abB
B  --->  aA|a
D  --->  ddd
```

2. 生成的自动机如下：

```
初态：
q0
栈初态：
S
状态：
q0,
字符：
b,a,d,c,
栈字符：
B,S,D,A,c,b,a,d,
d('q0', '#', 'S')=('q0', ('b',))('q0', ('c', 'c', 'D'))('q0', ('a', 'A'))('q0', ('a',))('q0', ('b', 'A'))
d('q0', '#', 'A')=('q0', ('a', 'b', 'B'))
d('q0', '#', 'B')=('q0', ('a',))('q0', ('a', 'A'))
d('q0', '#', 'D')=('q0', ('d', 'd', 'd'))
d('q0', 'b', 'b')=('q0', '#')
d('q0', 'a', 'a')=('q0', '#')
d('q0', 'd', 'd')=('q0', '#')
d('q0', 'c', 'c')=('q0', '#')
```

3. 根据生成的自动机反生成的文法如下：

```
初态：
@S
状态：
('q0', 'd', 'q0'), ('q0', 'c', 'q0'), ('q0', 'a', 'q0'), ('q0', 'b', 'q0'), ('q0', 'S', 'q0'), @S,
字符：
b, a, d, c,
边：
@S  --->  ('q0', 'S', 'q0')
('q0', 'S', 'q0')  --->  ('q0', 'a', 'q0')('q0', 'A', 'q0')|('q0', 'b', 'q0')('q0', 'A', 'q0')|('q0', 'c', 'q0')('q0', 'c', 'q0')('q0', 'D', 'q0')|
('q0', 'A', 'q0')  --->  ('q0', 'a', 'q0')('q0', 'b', 'q0')('q0', 'B', 'q0')
('q0', 'B', 'q0')  --->  ('q0', 'a', 'q0')('q0', 'A', 'q0')|('q0', 'a', 'q0')
('q0', 'D', 'q0')  --->  ('q0', 'd', 'q0')('q0', 'd', 'q0')('q0', 'd', 'q0')
('q0', 'b', 'q0')  --->  b
('q0', 'a', 'q0')  --->  a
('q0', 'd', 'q0')  --->  d
('q0', 'c', 'q0')  --->  c
```

可以发现，反生成的文法与原来的文法一模一样（图截不全）

5 小组分工

学号	姓名	主要负责板块
2022211480	罗星贺	算法设计及优化，数据结构设计，实验代码编写，附加内容算法设计及编写, 撰写实验报告
2022212036	吴林涛	实验二的算法分析，优化和方法编写
2022212139	朱远川	实验一的算法分析，优化和方法编写
2022210792	叶宇洋	数据结构设计，算法鲁棒性分析，算法优化