



**实验二：PDA、CFG相关实验**

**学院：计算机学院（国家示范性软件学院）**

**班级： 2023211303**

**组长： 计子毅2023212872**

**组员： 陈子容2023210710**

**江宝金2023212219**

**张贺维2023212185**

**分工： 陈子容：设计核心算法并编程实现**

**计子毅：参与编程实现并编写文档**

**江宝金：进行案例测试并编写文档**

**张贺维：解决程序问题并进行优化**

# 实验一 CFG文法化简与规范化

# 实验环境描述及使用编程语言

操作系统：Windows 11

调试软件：PyCharm、VSCode

代码实现语言：Python

# 程序设计思路及核心算法

## 设计思路

本程序实现了上下文无关文法(CFG)的规范化处理，包括：

1. **消去ε生成式**（算法3）
2. **消去单生成式**（算法4）
3. **消去无用符号**（算法1和算法2）

程序特点：

* 采用Python实现，注重算法清晰性
* 使用集合操作高效处理文法规则
* 分阶段展示文法转换过程
* 提供清晰的格式化输出

## 核心算法

此程序的核心算法是NFA到DFA转换算法（子集构造法）

**2.1 算法原理**

1. 消去ε生成式:识别所有可推导出空串的非终结符，并生成所有可能的非空变体

2. 消除形如A->B的单生成式，直接生成所有可能的推导

3. 消去无用符号:

·算法1:消除不产生终结符串的非终结符

·算法2:消除从开始符号不可达的符号

**2.2 数据结构设计**

·文法表示:使用Grammar类存储四元组(N,T,P,S)

·产生式规则:使用字典存储，键为非终结符，值为产生式右部的集合

·中间结果:使用集合操作处理非终结符和产生式

**2.3 算法步骤**

1.算法3（消去ε生成式）

·找出所有可推导出ε的非终结符N’

·对每个产生式A->α，生成所有可能的非空变体

·若开始符号可推导出ε，添加新的开始符号

2.算法4（消去单生成式）

·对每个非终结符A，计算其可推导出的所有非终结符N\_A

·将N\_A中所有非单生成式添加到A的产生式中

3.算法1（消去不产生终结符的非终结符）

·找出所有直接产生终结符串的非终结符

·迭代扩展直到找到所有可产生终结符串的非终结符

4.算法2（消去不可达符号）

·从开始符号出发，找出所有可达符号

·博保留可达符号及其产生式

**2.4 伪代码**

算法伪代码如下:

## 3. 程序输入描述

**·** 非终结符：空格分隔的符号（如：”S A B”）

· 终结符：空格分隔的符号（如”a b c”）

· 产生式：每行格式为”A->α”,其中α可以是多个符号或”em”表示ε

· 开始符号：单个非终结符

## 4. 程序输出解释

**·** 原始文法：显示输入文法的四元组

· 消去ε生成式后：显示处理后的文法

· 消去单生成式后：显示处理后的文法

· 消去无用符号后：显示最终规范化文法

# 三、样例测试

**输入：**

非终结符：**S, A, B, C, D**

终结符：**a, b, c, d**

产生式：

**S → a bA B ccD**

**A → abB em**

**B → aA**

**C → ddC**

**D → ddd**

开始符号：**S**

**输出：**

文本

AI 生成的内容可能不正确。

**输入**：

非终结符: **S, A, B, C, D, E**

终结符: **a, b, c, d**

产生式:

**S → AB | CD**

**A → aA | b**

**B → bB | c**

**C → cC | dD**

**D → dD | ε**

**E → aE | Ea**

开始符号: **S**

图片包含 背景图案

AI 生成的内容可能不正确。

## 四、改进思路和方法

1. 可以增加文法最小化功能

2. 输出只做到以书面形式进行输出，可以改进为增加状态图输出

# 实验二：PDA到CFG转换

# 实验环境描述及使用编程语言

操作系统：Windows 11

调试软件：PyCharm、VSCode

代码实现语言：Python

# 程序设计思路及核心算法

## 设计思路

将下推自动机（PDA）转换为等价的上下文无关文法（CFG），通过模拟PDA的栈操作生成文法规则

## 核心算法

此程序的核心算法是NFA到DFA转换算法（子集构造法）

**2.1 算法原理**

1. 对PDA的每个可能转移，生成对应的文法规则

2. 使用三元组[q,A,p]表示非终结符，模拟从状态q到p且弹出A的推导过程

**2.2 数据结构设计**

·文法表示:使用Grammar类存储四元组(N,T,P,S)

·产生式规则:使用字典存储，键为非终结符，值为产生式右部的集合

·中间结果:使用集合操作处理非终结符和产生式

**2.3 算法步骤**

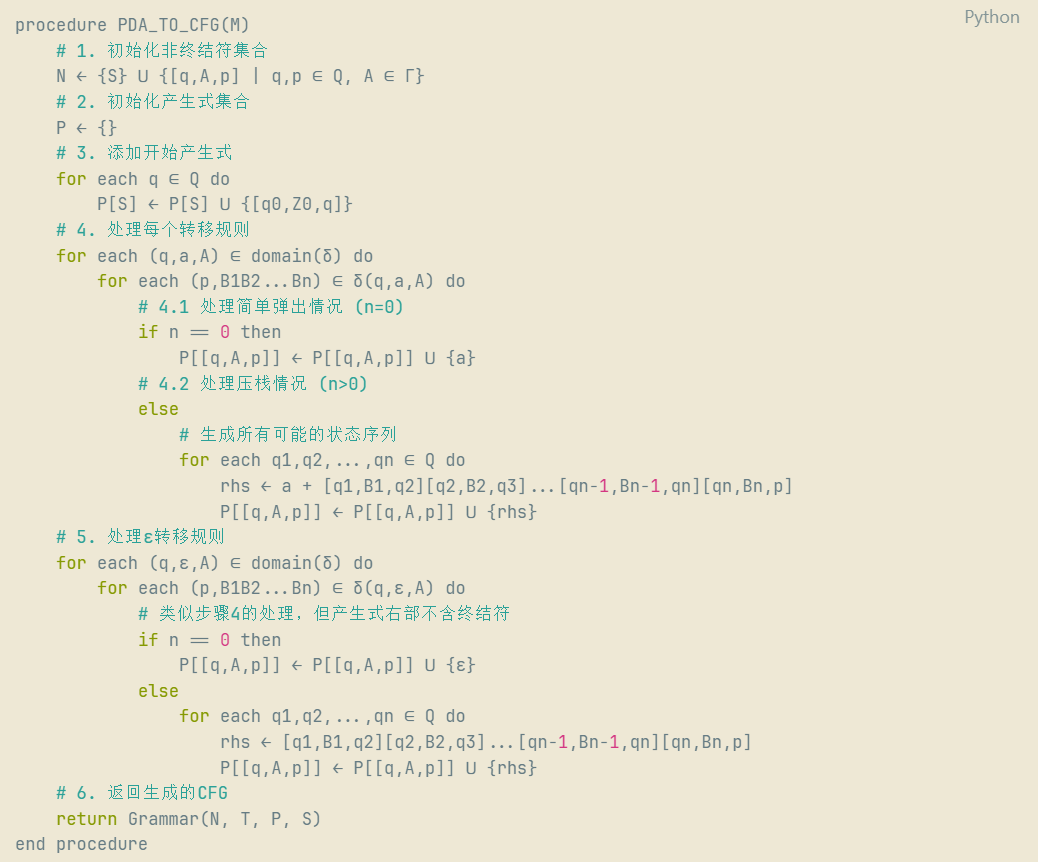
1.为每个栈操作生成对应的产生式

2.处理ε转移的特殊情况

3.应用与实验一相同的规范化步骤

**2.4 伪代码**

算法伪代码如下:



## 3. 程序输入描述

**· 状态：**空格分隔的符号（如：”**q0, q1**”）

**· 终结符：**空格分隔的符号（如”**a, b, c**”）

* **栈符：**空格分隔的符号（如：”**B, Z**”）

**· 产生式：**每行格式为” 格式为 **Q,T,Γ** → **Q,Γ**，**QTΓ** 和 **QΓ\*** 用空格分隔,**em**代表空”,其中α可以是多个符号或”**em**”表示ε

**· 开始符号：**单个非终结符

## 4. 程序输出解释

**· 最终文法四元组形式：**显示输入文法的四元组

# 三、样例测试

**输入**：

状态：**q0, q1**

终结符：**a, b**

栈符：**B, Z**

转移函数：

**q0,b,Z → q0,BZ**

**q0,b,B → q0,BB**

**q0,a,B → q1,ε**

**q1,a,B → q1,ε**

**q1,ε,B → q1,ε**

**q1,ε,Z → q1,ε**

**输出：**

文本

AI 生成的内容可能不正确。

**输入**：

状态: **q0, q1, q2**

输入符号: **a, b**

栈符号: **Z, A, B**

转移函数:

**q0,a,Z → q0,AZ**

**q0,a,A → q0,AA**

**q0,a,B → q0,AB**

**q0,b,Z → q0,BZ**

**q0,b,A → q0,BA**

**q0,b,B → q0,BB**

**q0,ε,Z → q1,Z**

**q0,ε,A → q1,A**

**q0,ε,B → q1,B**

**q1,a,A → q1,ε**

**q1,b,B → q1,ε**

**q1,ε,Z → q2,ε**

初始状态: **q0**

开始栈符号: **Z**

接受状态: **q2**

**输出：**

电脑萤幕画面

AI 生成的内容可能不正确。

## 四、改进思路和方法

​1. 状态爆炸问题​​：PDA状态与栈符号的组合导致生成的非终结符数量呈指数级增长

​​2. 冗余产生式​​：转换过程中会产生大量实际不可达或冗余的产生式

​​3. 规范化效率​​：多步骤规范化处理可能导致中间文法规模过大

​​4. 输入输出友好性​​：PDA输入格式不够直观，输出文法可读性有待提高

​​5. ε转移处理​​：对复杂ε转移场景的支持不够完善