任务3.4 LL(1)文法判定与预测分析器设计及实现

一、实验目的

- 理解LL(1)文法的概念及其在语法分析中的应用。
- 掌握判定文法是否为LL(1)的方法。
- 学习设计和实现LL(1)预测分析器的过程。
- 培养运用编程语言实现自顶向下语法分析的能力。

二、实验内容

实现LL(1)文法的判定算法和预测分析器的设计与实现。具体步骤包括：

1. 输入上下文无关文法。
2. 判断文法是否为LL(1)。
3. 构造预测分析表。
4. 实现预测分析器，能够根据输入串进行语法分析。

三、实验要求

1. 输入一个上下文无关文法，包括非终结符、终结符和产生式。
2. 在**任务3.1-3.3**基础上来实现判断
3. 输出文法是否为LL(1)的判断结果。
4. 输出预测分析表。
5. 输入一个字符串，输出语法分析结果（是否成功以及分析过程）。**【可以结合课程的实际例子运行，比如表达式文法】**

四、实验指南

LL(1)文法判定

1. **定义**：
   - 文法为LL(1)的条件是对于每个非终结符A及其产生式A → α | β，满足以下条件：
     - $FIRST(\alpha) \cap FIRST(\beta) = \emptyset$
     - 如果 `ε` 在 $FIRST(\alpha)$ 或 $FIRST(\beta)$ 中，则需满足 $FOLLOW(A) \cap FIRST(\beta) = \emptyset$ 或 $FOLLOW(A) \cap FIRST(\alpha) = \emptyset$。
2. **算法流程**：
   - 计算文法的$FIRST$集和$FOLLOW$集。
   - 对于每个非终结符的所有产生式，检查是否满足LL(1)条件。

预测分析器设计

1. **构造预测分析表**：
   - 对于每个产生式$A \to \alpha$，将其加入分析表$M[A, a]$中，其中a是$FIRST(\alpha)$中的第一个终结符。
   - 如果ε在$FIRST(\alpha)$中，则还需将$A \to \alpha$加入$M[A, b]$中，其中b是$FOLLOW(A)$中的每个终结符。
2. **实现预测分析器**：
   - 使用**栈结构**实现分析器，初始时将文法的**开始符号**压入栈中。
   - 持续进行如下操作，直到**栈为空**：
     - 如果栈顶符号是终结符且与输入符号匹配，弹出栈顶并移动输入符号。
     - 如果栈顶符号是非终结符，从预测分析表中查找对应的产生式，替换栈顶符号为产生式右侧的符号。
     - 如果无法进行匹配，则语法分析失败。

示例

给定文法：

```
1. S → AB
2. A → aA | ε
3. B → b
```

- 计算得到：
  - `FIRST(S) = {a, b}`
  - `FIRST(A) = {a, ε}`
  - `FIRST(B) = {b}`
  - `FOLLOW(S) = {$}`
  - `FOLLOW(A) = {b}`
  - `FOLLOW(B) = {$}`
- 预测分析表如下：

非终结符	a	b	$
S	S → AB	S → AB	
A	A → aA	A → ε	A → ε
B		B → b	

五、难点分析

- 判定文法是否为LL(1)时，需准确计算FIRST集和FOLLOW集，且确保文法的各个产生式满足LL(1)条件。
- 预测分析器的实现需要处理输入串的逐步匹配，确保栈的操作正确无误。

伪代码示例【仅供参考，总体流程参考课件和教材】

以下是LL(1)文法判定和预测分析器的伪代码示例：

```
FUNCTION is_LL1(grammar):
    FIRST_sets = compute_FIRST(grammar)
    FOLLOW_sets = compute_FOLLOW(grammar)
    FOR each non-terminal A in grammar:
        FOR each production A → α | β:
            IF FIRST(α) ∩ FIRST(β) ≠ ∅ THEN
                RETURN false
            IF ε in FIRST(α) THEN
                IF FOLLOW(A) ∩ FIRST(β) ≠ ∅ THEN
                    RETURN false
    RETURN true

FUNCTION LL1_parser(input_string, grammar):
    stack = [grammar.start_symbol]
    input_buffer = input_string + "$"
    WHILE stack is not empty:
        top = stack.pop()
        current_symbol = input_buffer[0]

        IF top is terminal THEN
            IF top == current_symbol THEN
                input_buffer = input_buffer[1:]
            ELSE
                RETURN "Syntax Error"
        ELSE IF top is non-terminal THEN
            production = M[top][current_symbol]  // 从预测分析表中查找
            IF production is not empty THEN
                FOR each symbol in reversed production:
                    stack.push(symbol)
            ELSE
                RETURN "Syntax Error"

    IF current_symbol == "$" THEN
        RETURN "Parsing Successful"
    ELSE
        RETURN "Syntax Error"

```