**第四章 自顶向下语法分析**

1、对文法**G[S]**

**S→a|**∧**|(T)**

**T→T,S|S**

1. 给出**(a,(a,a))**和**(((a,a),**∧**,(a)),a)**的最左推导。

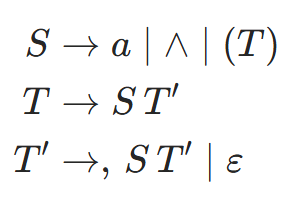
**答：S->(T)->(T,S)->(S,S)->(a,S)**

**(a,(a,a))->(a,(T))->(a,(T,S))->(a,(S,S))->(a,(a,S))->(a,(a,a))**

**S(T)->(T,S)->((T),S)->((T,S),S)->((S,S),S)->((a,S),S)->((a,a),S)->((a,a), ∧,S)->((a,a), ∧,(T))->((a,a), ∧,(S))->((a,a), ∧,(a),S)->((a,a), ∧(a)),a)**

1. 对文法**G**，进行改写，然后对每个非终结符写出不带回溯的递归子程序。

**答：原文法中 T->T,S|S 存在左递归，消除后：**



**递归子程序：**

// 解析 S

def parse\_S():

if lookahead == 'a' or lookahead == '∧' or lookahead == '(':

if lookahead == 'a':

consume('a')

elif lookahead == '∧':

consume('∧')

elif lookahead == '(':

consume('(')

parse\_T()

if lookahead == ')':

consume(')')

else:

error()

else:

error()

// 解析 T

def parse\_T():

parse\_S()

parse\_T\_prime()

// 解析 T'

def parse\_T\_prime():

if lookahead == ',':

consume(',')

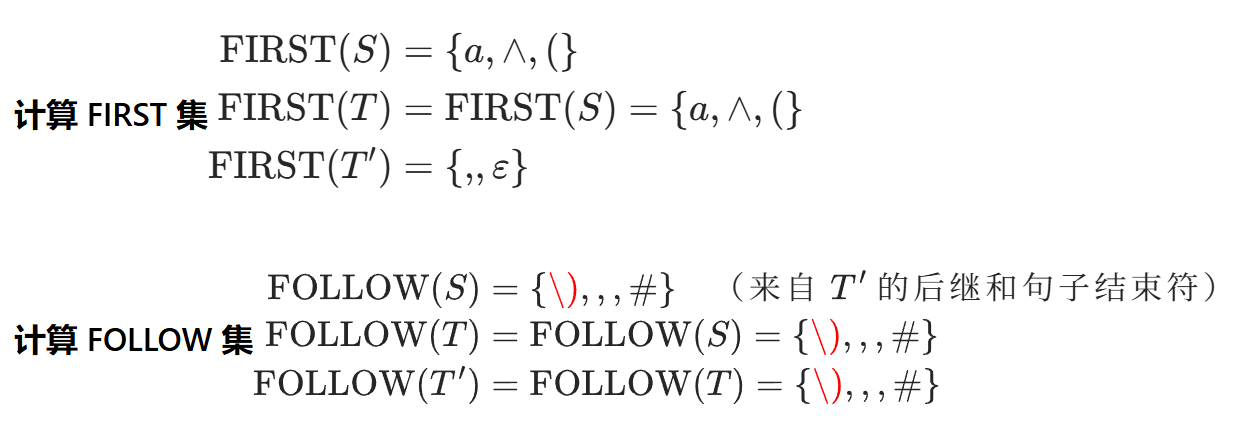
parse\_S()

parse\_T\_prime()

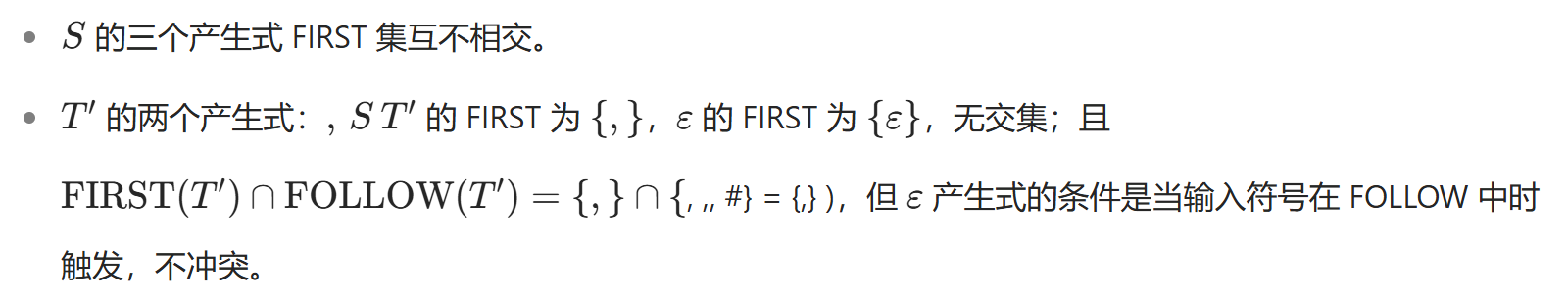
// ε 产生式，无需操作

1. 经改写后的文法是否是**LL(1)**的**?**给出它的预测分析表。

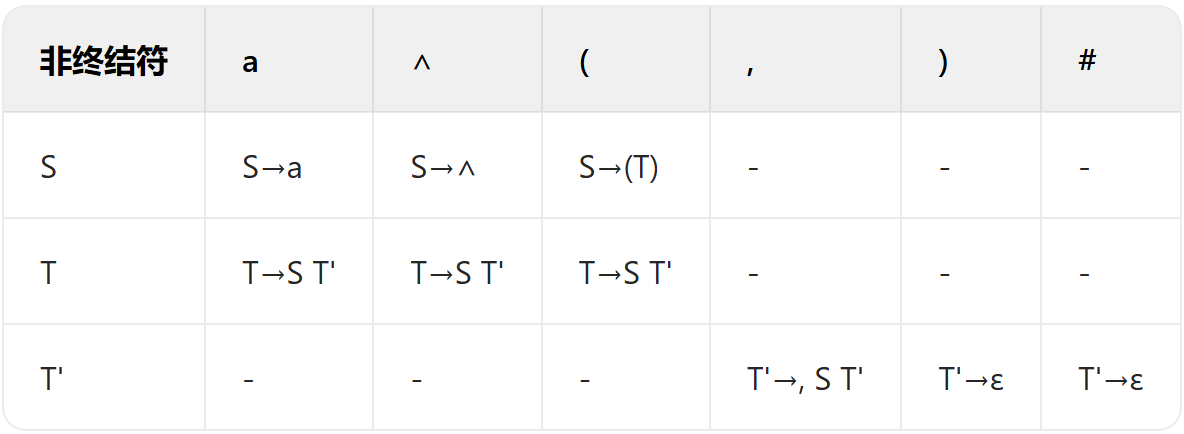
**答：**



LL(1)条件验证



预测分析表：



1. 给出输入串**(a,a)#**的分析过程，并说明该串是否为**G** 的句子。

答：



2、已知文法**G[S]**：

**S→MH|a**

**H→LSo|ε**

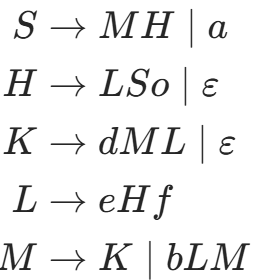
**K→dML|ε**

**L→eHf**

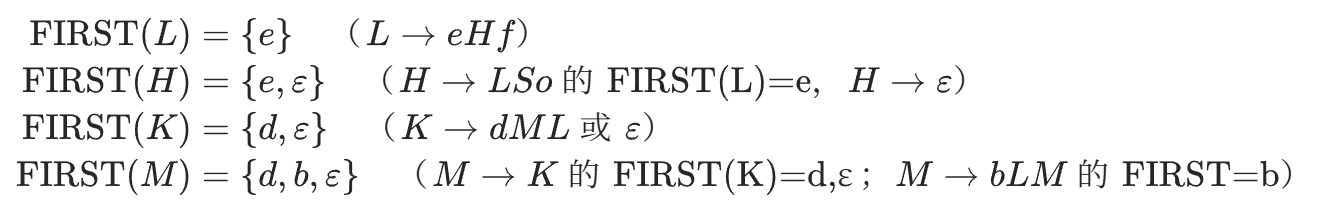
**M→K|bLM**

判断**G** 是否是**LL(1)**文法，如果是，构造**LL(1)**分析表。

**答：原文法：**

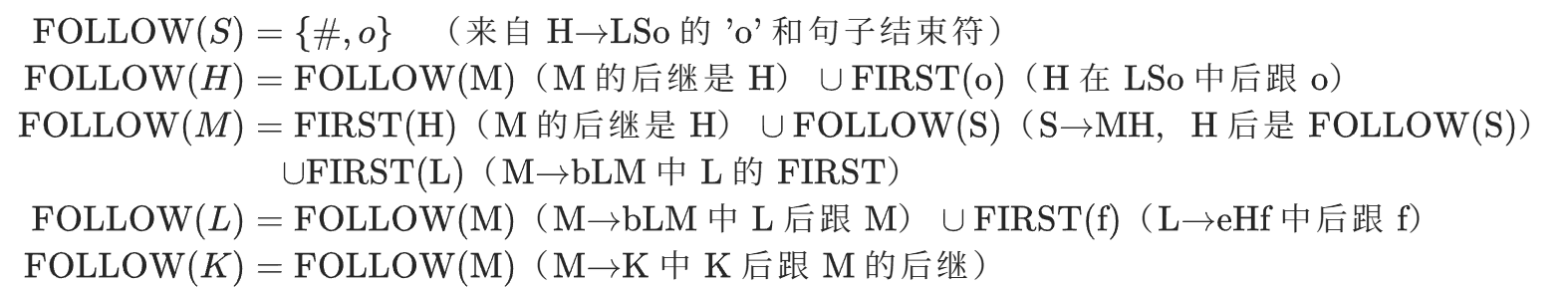


**计算 FIRST 集**

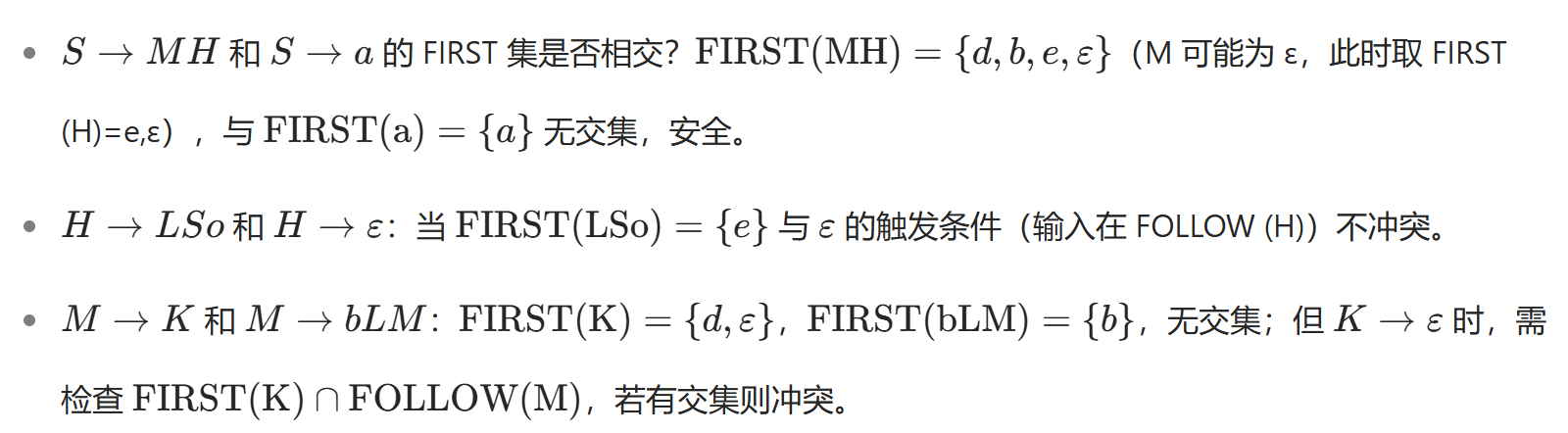




**计算 FOLLOW 集**



**冲突检查**



**结论：文法是 LL (1)，预测分析表需详细计算每个非终结符的产生式对应输入符号的映射**