**第五章 自底向上语法分析**

**5.1 算符优先分析**

1、已知文法**G[S]**为：

**S→a|**∧**|(T)**

**T→T,S|S**

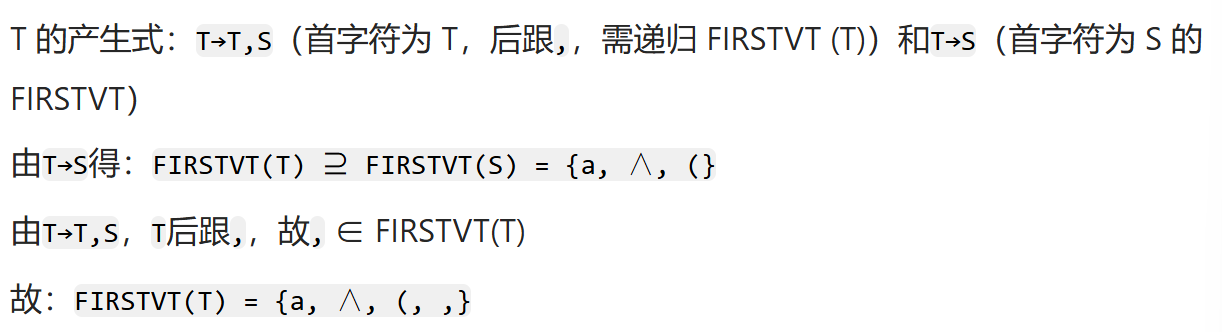
1. 计算**G[S]**的**FIRSTVT** 和**LASTVT**。

答：文法 G[S]: S → a | ∧ | (T)

T → T,S | S

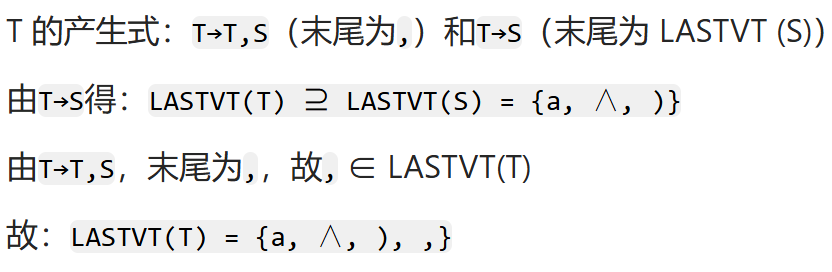
FIRSTVT(S) = {a, ∧, (}

对于 FIRSTVT(T)：



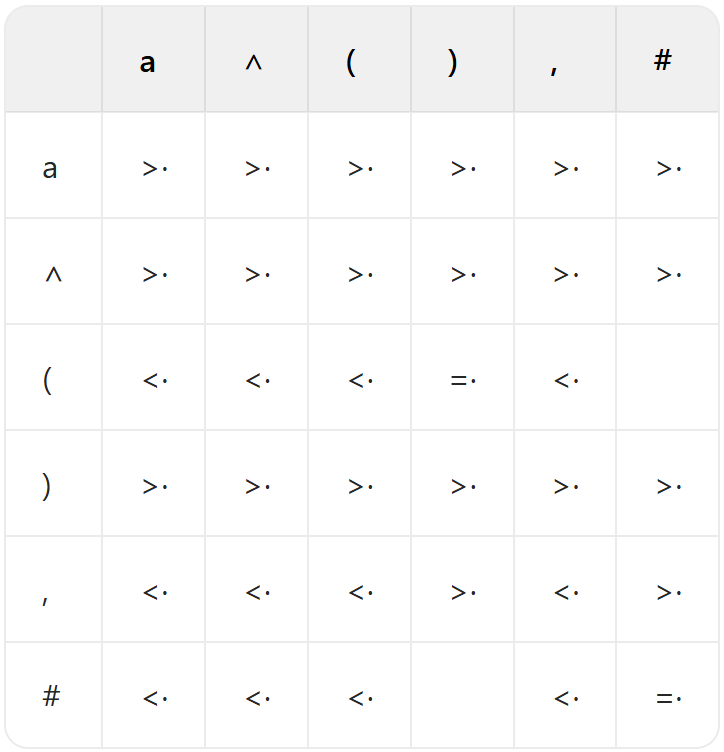
LASTVT(S) = {a, ∧, )}

对于 LASTVT(T)：



1. 构造**G[S]**的算符优先关系表并说明**G[S]**是否为算符优先文法。

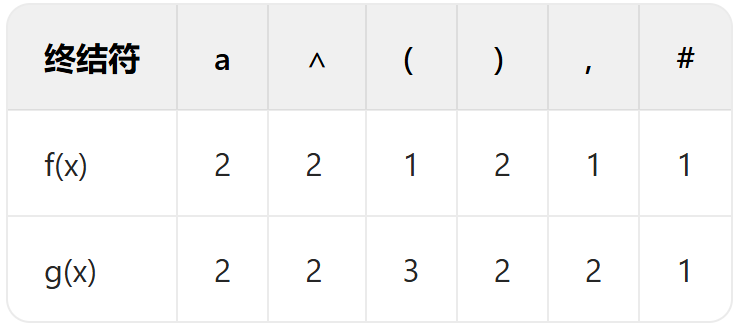
答：



所有终结符对最多有一个优先关系，无冲突，故 G [S] 是算符优先文法

1. 计算**G[S]**的优先函数。

答：



**(4)** 给出输入串**(a,a)#**和**(a,(a,a))#**的算符优先分析过程。

答：

2、已知文法**G[S]**为：

**S→a|**∧**|(T)**

**T→T,S|S**

1. 给出**(a,(a,a))**和**(a,a)**的最右推导，和规范归约过程。

答：S ⇒ (T) ⇒ (T,S) ⇒ (T,(T)) ⇒ (T,(T,S)) ⇒ (T,(T,a)) ⇒ (T,(S,a)) ⇒ (T,(a,a)) ⇒ (S,(a,a)) ⇒ (a,(a,a))

S ⇒ (T) ⇒ (T,S) ⇒ (T,a) ⇒ (S,a) ⇒ (a,a)

**(2)** 将**(1)**和题**1** 中的**(4)**进行比较给出算符优先归约和规范归约的区别。

答：对于归约单位，算符优先归约基于终结符的优先关系，归约素短语（不包含非终结符的最左子串）；规范归约基于句柄，严格遵循产生式的右部。

对于归约顺序，算符优先归约可能提前归约部分子结构。例如，在处理 (a,a) 时，先将 a 归约为 S，再归约为 T，步骤更细化；规范归约则严格按照最右推导的逆序。

对于非终结符的处理，算符优先归约的非终结符仅作为中间标记，不影响归约决策，而规范归约的非终结符是句柄的必要组成部分

3、有文法**G[S]**：

**S→V**

**V→T|ViT**

**T→F|T+F**

**F→)V\*|(**

1. 给出**(+(i(**的规范推导。

答：S ⇒ V ⇒ ViT ⇒ ViF ⇒ Vi( ⇒ T i( ⇒ T+F i( ⇒ T+( i( ⇒ F+( i( ⇒ ( +( i(

1. 指出句型 **F+Fi(**的短语，句柄，素短语。

答：短语：由 S 推导：F+Fi(、由 T 推导：F+F、由 F 推导：F（第一个 F）、F（第二个 F）、(

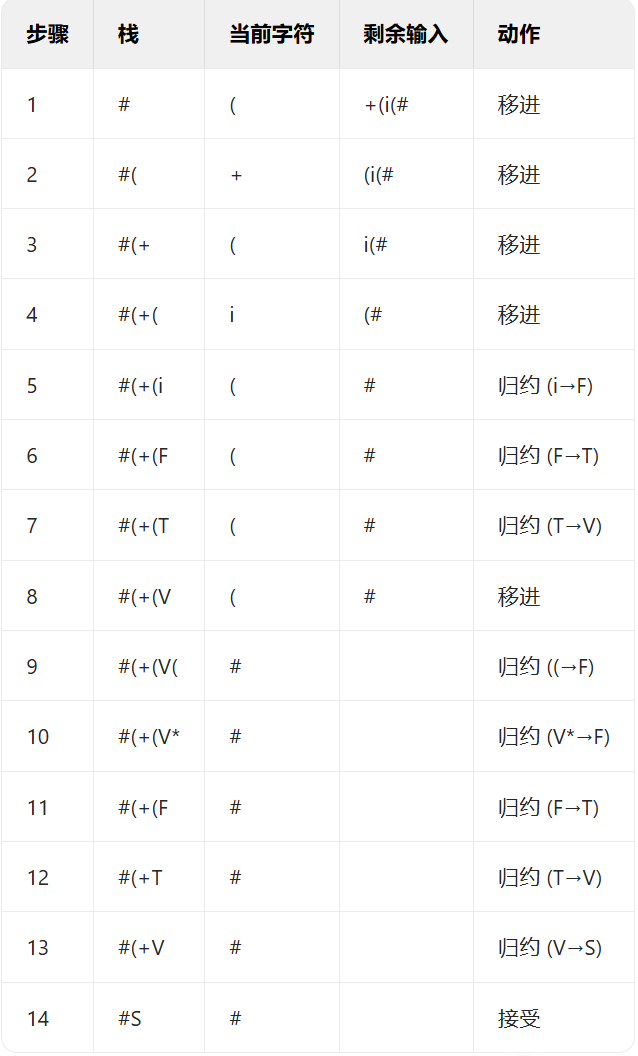
直接短语：由 T→F：F（第一个 F）、F（第二个 F）；由 F→(：(

句柄：最左直接短语：F（第一个 F）

素短语：至少含一个终结符，不含更小素短语：F+F、(

1. **G[S]**是否为**OPG**？若是，给出**(1)**中句子的分析过程。

答：产生式中无 ε 产生式，所有产生式右部中，非终结符间至少有一个终结符，并且算符优先关系唯一，因此是OPG



**5.2 LR分析**

1、已知文法

A→aAd|aAb|ε

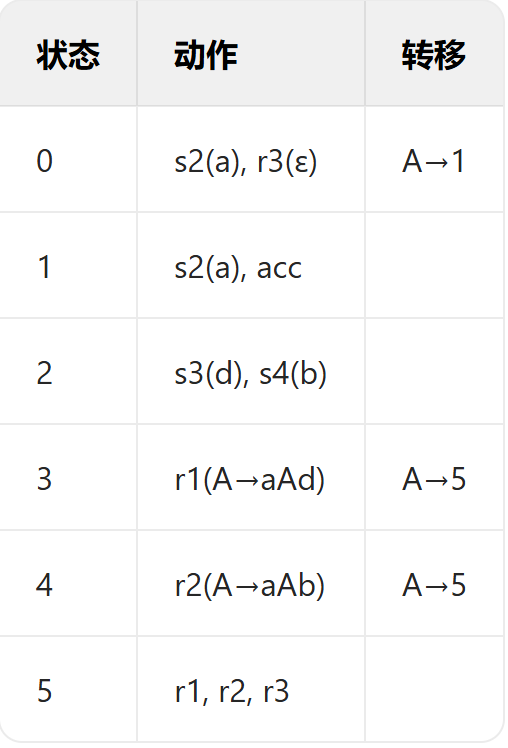
判断该文法是否是SLR(1)文法，若是构造相应分析表，并对输入串ab#给出分析过程。

答：首先构造LR(0)项目集，初始项目A’ →·A, A→·aAd, A→aAb, A→·ε

项目集闭包和传递后，发现冲突，在状态1（项目A→a·Ad和A→a·Ab），遇到d和b时转移明确，无冲突。在状态2（项目A→ε·），面临 FOLLOW (A)={d,b,#}，归约时与移进无冲突（因 FOLLOW (A) 不含其他输入符号）

SLR(1)冲突解决：唯一可能的冲突是归约与移进，但 FOLLOW (A)={d,b,#}，与移进符号不重叠，故可用 SLR (1) 分析表消除冲突，因此该文法是SLR(1)文法

接下来构造SLR分析表



输入串ab#的分析过程



2、若有定义二进制数的文法如下：

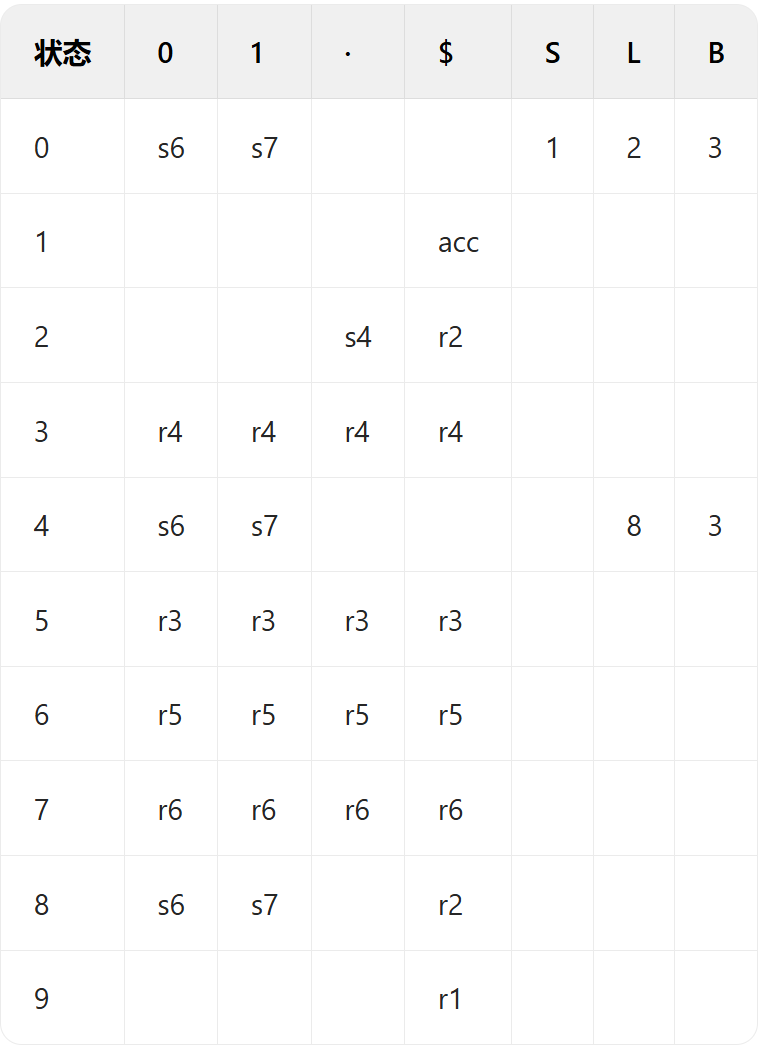
S→L·L|L

L→LB|B

B→0|1

1. 试为该文法构造LR 分析表，并说明属哪类LR 分析表。

答：



属于SLR(1)文法，因为SLR(1)分析表无冲突

1. 给出输入串101.110 的分析过程。

答：



3、文法G=({U,T,S},{a,b,c,d,e},P,S)

其中P 为：

S→UTa|Tb

T→S|Sc|d

U→US|e

1. 判断G 是LR(0)，SLR(1)，LALR(1)还是LR(1)，说明理由。

首先尝试构造LR(0)项目集规范族

初始项目集I0：

S' → ·S

S → ·UTa

S → ·Tb

T → ·S

T → ·Sc

T → ·d

U → ·US

U → ·e

闭包后：I₀ = { S' → ·S, S → ·UTa, S → ·Tb, T → ·S, T → ·Sc, T → ·d, U → ·US, U → ·e }

通过转移函数构造其他项目集，发现关键项目集存在冲突，因此不是LR(0)

尝试构造SLR(1)分析表

计算FOLLOW集：

FOLLOW(S') = { $ }

FOLLOW(S) = { a, b, c, $ }

FOLLOW(T) = { a, b }

FOLLOW(U) = { T }

归约 U → e 的 FOLLOW (U) = { T }，而当前面临的输入符号是 a 或 S，均不在 FOLLOW (U) 中，因此冲突无法解决，该文法不是SLR(1)

尝试构造LR(1)项目集规范族

在 LR (1) 项目集中，项目的展望符（lookahead）会限制归约条件，例如在I2的LR(1)版本中：

U → U·S, [T]

S → ·UTa, [a]

S → ·Tb, [a]

T → ·S, [a, c]

...

此时，归约 U → e 的展望符为 [T]，而移进 S 的展望符为 [a, c]，冲突消除

因此该文法是LR(1)

(2) 构造相应的分析表。

