第二章部分习题答案

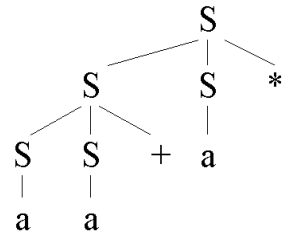
2.1 考虑文法

S→ S S + | S S * | a

a) 证明文法可生成符号串 a a + a *

解：S→ S S * → S S + S * →a S + S * → a a + S *→ a a + a *

b) 为此符号串构造语法树



解：

c) 文法生成什么样的语言？证明结论

解：将 a 看作运算数，文法生成语言 L={支持加法、乘法的表达式的后缀表示形式}

证明类似 2.2 题 b)

═══════════════════════════════════

2.2 下列文法生成什么样的语言？证明你的结论。是否有二义性？

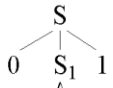
a) S → 0 S 1 | 0 1

解：生成语言 L={0^n1^n | n>=1}

证明：1) 证文法推导出的符号串都在 L 中

i) 考虑最小语法树  ，推导出的符号串 01 显然∈L

ii) 假定结点数<n 的语法树对应的符号串都∈L，



考虑结点数=n 的语法树 S，其结构必为 ，

子树 S_1 结点数<n，因此对应符号串 t_1∈L，

S 对应符号串为 t=0 t_1 1，因此 t∈L

综合 i)、ii)，1)得证

2) 证 L 中符号串都可由文法推导出

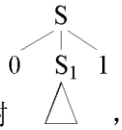
i) L 中最短符号串 01，显然可由文法推导出

ii) 假定 L 中长度<2n 的符号串都可由文法推导出，

考虑长度=2n 的符号串 t＝0^n1^n，它可表示为 0 t_1 1，

$t_1 \in L$ 且长度<2n，因此它可被文法推导出，对应语法树 

，

构造语法树 

，显然，它的输出为 t，即 t 可被文法推导出

综合 i)、ii)，2)得证

综合 1)、2)，文法生成的语言即为 L

另外，文法没有二义性

==========

b) $S \rightarrow +SS \mid -SS \mid a$

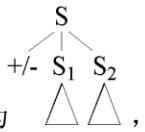
解：生成语言 L={支持加法、减法的表达式的前缀表示形式}

证明：1) 证文法推导出的符号串都在 L 中

i) 考虑最小语法树

，推导出的符号串 a 显然∈L

ii) 假定结点数<n 的语法树对应的符号串都∈L，

考虑结点数=n 的语法树 S，其结构必为 

，

子树 S_1、S_2 结点数<n，因此对应符号串 E_1、$E_2 \in L$（前缀表达式），

S 对应符号串为 $E=+/- E_1 E_2$，E 也是前缀表达式。

综合 i)、ii)，1)得证

2) 证 L 中符号串都可由文法推导出

i) L 中最短符号串 a，显然可由文法推导出

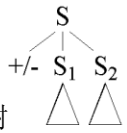
ii) 假定 L 中长度<n 的符号串都可由文法推导出，

考虑长度=n 的符号串 E，它的形式必为$+/- E_1 E_2$，

E_1、$E_2 \in L$ 且长度<n，因此可被文法推导出，对应语法树 

、 

，

构造语法树 

，显然，它的输出为 E，即 E 可被文法推导出

综合 i)、ii)，2)得证

综合 1)、2)，文法生成的语言即为 L

另外，文法没有二义性

==========

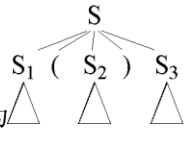
c) $S \rightarrow S(S)S \mid \varepsilon$

解：生成语言 L={括号匹配的符号串，包括ε}

证明：1) 证文法推导出的符号串都在 L 中

i) 考虑最小语法树，推导出的符号串ε显然∈L

ii) 假定结点数<n 的语法树对应的符号串都∈L，

考虑结点数=n 的语法树 S，其结构必为  ，

子树 S_1、S_2、S_3 结点数<n，因此对应符号串 t_1、t_2、t_3∈L，

S 对应符号串为 $t=t_1 (t_2) t_3$，显然 t∈L。

综合 i)、ii)，1)得证

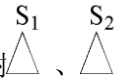
2) 证 L 中符号串都可由文法推导出

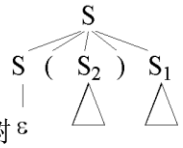
i) L 中最短符号串ε，显然可由文法推导出

ii) 假定 L 中长度<n 的符号串都可由文法推导出，

考虑长度=n 的符号串 t=(…)，寻找它的∈L 的最短前缀 t’，则 $t=t' t_1$，t_1∈L（可能为ε），

t’的形式必为( )或((…))（否则不是最短前缀），将它表示为 $t'=(t_2)$，无论哪种情况，必有 t_2∈L，

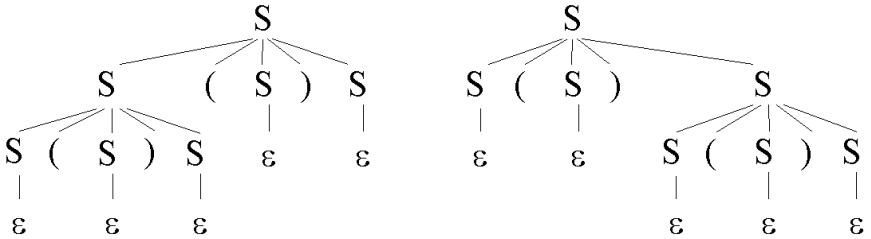
t_1、t_2 长度<n，可被文法推导出，对应语法树  ，

构造语法树  ，显然，它的输出为 t，即 t 可被文法推导出

综合 i)、ii)，2)得证

综合 1)、2)，文法生成的语言即为 L

另外，文法有二义性，符号串( ) ( )可对应两个语法树



==========

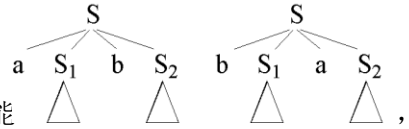
d) S → a S b S | b S a S | ε

解：生成语言 L={相同个数的 a、b 组成的所有符号串，包括ε}

证明：1) 证文法推导出的符号串都在 L 中

i) 考虑最小语法树 S|ε，推导出的符号串ε显然∈L

ii) 假定结点数<n 的语法树对应的符号串都∈L，

考虑结点数=n 的语法树 S，其结构有两种可能 

，

子树 S_1、S_2 结点数<n，因此对应符号串 t_1、t_2∈L，

S 对应符号串 $t=a\,t_1\,b\,t_2$ 或 $b\,t_1\,a\,t_2$，显然 t∈L。

综合 i)、ii)，1)得证

2) 证 L 中符号串都可由文法推导出

i) L 中最短符号串ε，显然可由文法推导出

ii) 假定 L 中长度<n 的符号串都可由文法推导出，

考虑长度=n 的符号串 t，考虑以 a 开头的情况 t=a…（以 b 开头的情况类似），

寻找它的∈L 的最短前缀 t’，t’必有形式 a … b（否则不是最短前缀），

将它表示为 $t'=a\,t_1\,b$，显然 t_1∈L（可能为ε），

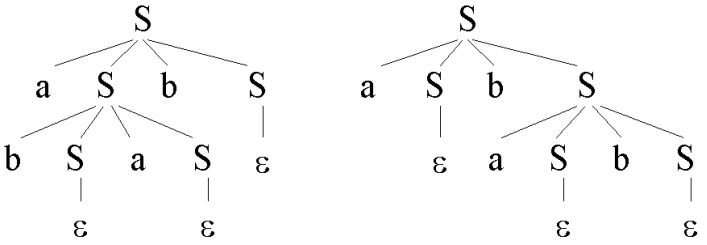
而 $t=t'\,t_2$，t_2∈L，则 $t=a\,t_1\,b\,t_2$。

由 i)图，可为 t_1、t_2 构造语法树，则可为 t 构造语法树，即 t 可被文法推导出

综合 i)、ii)，2)得证

综合 1)、2)，文法生成的语言即为 L

另外，文法有二义性，符号串 abab 可对应两个语法树



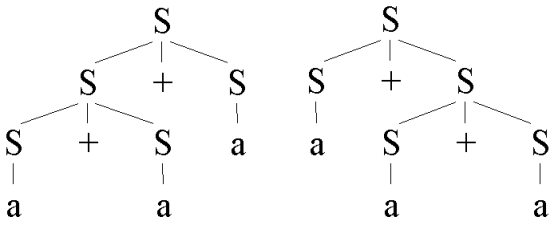
＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝＝

e) S → a | S+S | SS | S* | (S)

解：生成语言 L={支持加法等四种运算的表达式}

证明：类似 b)

文法是二义性文法，符号串 a + a + a 可构造两个语法树



==========

2.4 为下列语言构造非二义性的上下文无关文法（任选三个）

a) 后缀表示的算术表达式

解：E→+ E E | - E E | * E E | / E E | **num**

证明参照 2.3

b) 以’,’间隔的左结合的标识符列表

解：list→ list , **id** | **id**

c) 以’,’间隔的右结合的标识符列表

解：list→ **id** , list | **id**

d) 包含整数和标识符，支持+、-、*、/的算术表达式

解：

expr→expr + term | expr – term | term

term→term * factor | term / factor | factor

factor→ **id** | **num** | (expr)

e) 在 d)的基础上，添加一元+、-运算符

解：

expr→expr + term | expr – term | term

term→term * unary | term / unary | unary

unary→ + factor | - factor

factor→ **id** | **num** | (expr)

==========

2.5

a) 证明文法

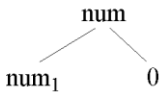
num→ 11 | 1001 | num 0 | num num

生成的二进制串所表示的数值均可被 3 整除

证明：1）考虑规模最小的语法树，生成二进制串为 11、110、1001、1100、1111，表示数 3、6、9、12、15，均可被 3 整除

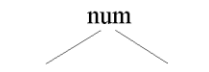
2）假定结点数<n 的语法树生成的二进制串均可被 3 整除，考虑结点数=n 的语法树

其结构有两种可能



，子树 num_1 结点数<n，生成的二进制串 y 可被 3 整除，而 num 生成

的二进制串 x=y*2，因此也可被 3 整除



，子树 num_1、num_2 结点数<n，生成的二进制串 y、z 可被 3 整除，而 num 生成的二进制串 $x=y*2^n+z$，因此也可被 3 整除

综合 1)、2)，命题得证

b) 文法是否生成所有可被 3 整除的二进制串？

解：不是。二进制串 10101，数值为 21，可被 3 整除，但无法由文法推导出

==========

2.6 为罗马数字构造一个上下文无关文法

RomanNumeral → *Thousand Hundreds Tens Ones*

Ones → *LowOnes* | IV | V *LowOnes* | IX

LowOnes → ε | I | II | III

Tens → *LowTens* | XL | L *LowTens* | XC

LowTens → ε | X | XX | XXX

Hundreds → *LowHundreds* | CD | D *LowHundreds* | CM

LowHundreds → ε | C | CC | CCC

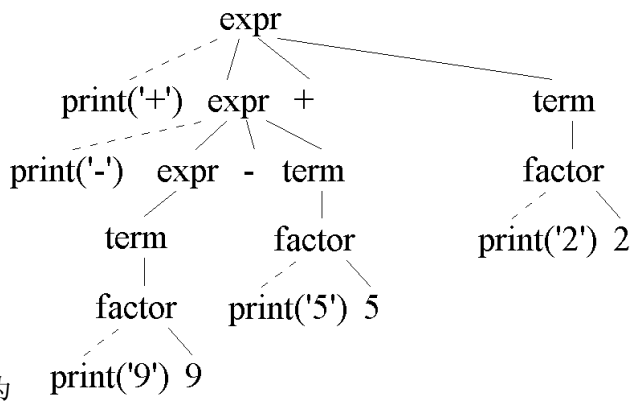
Thousands → M *Thousands* | ε

==========

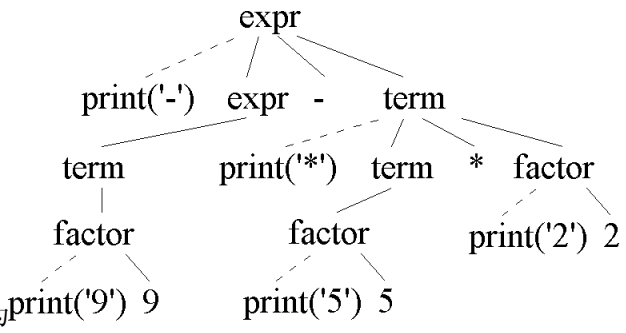
2.7 构造翻译模式，实现表达式中缀表示形式到前缀表示形式的转换。构造 9-5+2 和 9-5*2 的注释语法分析树

解：

```
expr→ { print('+'); } expr + term
    | { print('-'); } expr – term
    | term
term→ { print('*'); } term * factor
    | { print('/'); } term / factor
    | factor
factor→ { print(num.value); } num
    | ( expr )
```



9-5+2 的语法树为 ，输出+ - 9 5 2



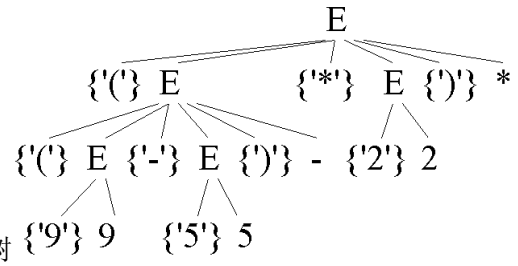
9-5*2 的语法树为 ，输出- 9 * 5 2

==========

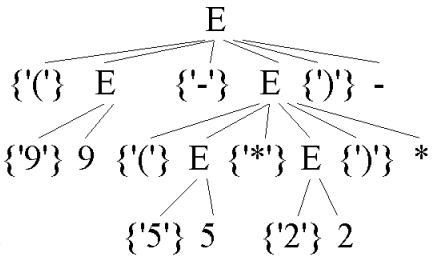
2.8 构造翻译模式，实现后缀表达式到中缀表达式的翻译，构造 95-2*和 952*-的注释语法树

解：

E→ { print(‘(’); }E { print(‘+’); } E { print(‘)’}; } +
| { print(‘(’); } E { print(‘-’); } E { print(‘)’}; } -
| { print(‘(’); } E { print(‘*’); } E { print(‘)’}; } *
| { print(‘(’); } E { print(‘/’); } E { print(‘)’}; } /
| **id** { print(**id**.lexeme); }
| **num** { print(**num**.value); }



95-2*的语法树 ，输出为((9-5)*2)



952*-的语法树 ，输出为(9-(5*2))

显然，此解法括号有冗余情况，但结果是正确的

==========

2.9　构建一个语法制导翻译模式，将整数翻译成罗马数字

解：NT　Ones、Tens、Hundreds 表示整数的个、十、百位，Thousands 表示千以上的所有位，Num 表示完整的整数，每个 NT 设定一个属性 roman，为字符串类型，代表罗马数字的表示形式。假定存在 chrrep(char c, int n)函数，它将字符 c 重复 n 次，形成一个字符串，作为返回结果，若 n＝0，则返回一个空串。

Num → *Thousands Hundreds Tens Ones*
　　{　*Num.roman* = *Thousands.roman* || *Hundreds.roman* || *Tens.roman* || *Ones.roman*; }

Thousands → *digits*　{ *Thousands.roman* = chrrep('M', *digits.val*); }
digits → *digits digit*　{ *digits.val* = *digits.val* * *10* + *digit.val*; }
　　| ε　{ *digits.val* = *0*; }
digit → *small*　{ *digit.val* = *small.val*; }
　　| *big*　{ *digit.val* = *big.val*; }
　　| 4　{ *digit.val* = 4; }
　　| 9　{ *digit.val* = 9; }
small → 0　{ *small.val* = 0; }
　　| 1　{ *small.val* = 1; }
　　| 2　{ *small.val* = 2; }
　　| 3　{ *small.val* = 3; }
big → 5　{ *small.val* = 5; }
　　| 6　{ *small.val* = 6; }
　　| 7　{ *small.val* = 7; }
　　| 8　{ *small.val* = 8; }
Hundreds→ *small*　{ *Hundreds.roman* = chrrep('C', *small.val*); }
　　| *big*　{ *Hundreds.roman* = "D" || chrrep('C', *small.val*); }
　　| 4　{ *Hundreds.roman* = "CD"; }
　　| 9　{ *Hundreds.roman* = "CM"; }
Tens→ *small*　{ *Tens.roman* = chrrep('X', *small.val*); }
　　| *big*　{ *Tens.roman* = "L" || chrrep('X', *small.val*); }
　　| 4　{ *Tens.roman* = "XL"; }
　　| 9　{ *Tens.roman* = "XC"; }
Ones→ *small*　{ *Ones.roman* = chrrep('I', *small.val*); }
　　| *big*　{ *Ones.roman* = "V" || chrrep('I', *small.val*); }
　　| 4　{ *Ones.roman* = "IV"; }
　　| 9　{ *Ones.roman* = "IX"; }

==========

2.10　构建一个语法制导翻译模式，将罗马数字翻译成整数

解：每个 NT 设定一个属性 val，表示对应整数值

RomanNumeral → *Thousands Hundreds Tens Ones*
{ *RomanNumeral.val* = *Thousands.val* + *Hundreds.val*
+ *Tens.val* + *Ones.val*;
printf("%d\n", *RomanNumeral.val*;) }

Ones → *LowOnes* { *Ones.val* = *LowOnes.val*; }
| IV { *Ones.val* = 4; }
| V *LowOnes* { *Ones.val* = *LowOnes.val* + 5; }
| IX { *Ones.val* = 9; }

LowOnes → ε { *LowOnes.val* = 0; }
| I { *LowOnes.val* = 1; }
| II { *LowOnes.val* = 2; }
| III { *LowOnes.val* = 3; }

Tens → *LowTens* { *Tens.val* = *LowTens.val*; }
| XL { *Tens.val* = 40; }
| L *LowTens* { *Tens.val* = *LowTens.val* + 50; }
| XC { *Tens.val* = 90; }

LowTens → ε { *LowTens.val* = 0; }
| X { *LowTens.val* = 10; }
| XX { *LowTens.val* = 20; }
| XXX { *LowTens.val* = 30; }

Hundreds → *LowHundreds* { *Hundreds.val* = *LowHundreds.val*; }
| CD { *Hundreds.val* = 400; }
| D *LowHundreds* { *Hundreds.val* = *LowHundreds.val* + 500; }
| CM { *Hundreds.val* = 900; }

LowHundreds → ε { *LowHundreds.val* = 0; }
| C { *LowHundreds.val* = 100; }
| CC { *LowHundreds.val* = 200; }
| CCC { *LowHundreds.val* = 300; }

Thousands → M $Thousands_1$ { *Thousands.val* = $Thousands_1$.*val* + 1000; }
| ε { *Thousands.val* = 0; }

==========

2.14

解：for 语句对应堆栈机代码段如下

$expr_1$ 的代码段
label test
$expr_2$ 的代码段
gofalse out
stmt 的代码段
$expr_3$ 的代码段
goto test
label out

stmt → **for** ($expr_1$; $expr_2$; $expr_3$) $stmt_1$ { test = newlabel(); out = newlabel();

stmt.t = $expr_1$.t ||

‘label’ test ||

$expr_2$.t ||

‘gofalse’ out ||

$stmt_1$.t ||

$expr_3$.t ||

‘goto’ test ||

‘label’ out; }