作业5

T1.[习题 4.9]

4.9 用 S 的综合属性 val 给出下面文法中 S 产生的二进制数的值。例如，输入 101.101 时，$S.val = 5.625$。

$S \rightarrow L.L \mid L$

$L \rightarrow LB \mid B$

$B \rightarrow 0 \mid 1$

(a) 仅用综合属性决定 $S.val$。

(b) 用 L 属性定义决定 $S.val$。在该定义中，B 的唯一综合属性是 c（还需要继承属性），它给出由 B 产生的位对最终值的贡献。例如，101.101 的最前一位和最后一位对值 5.625 的贡献分别是 4 和 0.125。

(a) 给文法符号 B、L、S 综合属性 val，再给 L 一个表示其长度的综合属性 length.

所求语法制导定义为：

$S \rightarrow L_1.L_2$	$S.val = L_1.val + L_2.val / 2^{L_2.length}$
$S \rightarrow L$	$S.val = L.val$
$L \rightarrow L_1 B$	$L.val = L_1.val \times 2 + B.val$; $L.length = L_1.length + 1$
$L \rightarrow B$	$L.val = B.val$; $L.length = 1$
$B \rightarrow 0$	$B.val = 0$
$B \rightarrow 1$	$B.val = 1$

(b) 将文法改成

$S \rightarrow L.R \mid L$

$L \rightarrow BL \mid B$

$R \rightarrow RB \mid B$

$B \rightarrow 0 \mid 1$

所求语法制导定义如下，i 为 B 的继承属性，val 和 c 是综合属性：

$S \to L.R$ $\quad S.val = L.val + R.val$

$S \to L$ $\quad S.val = L.val$

$L \to BL_1$ $\quad B.i = L_1.c \times 2$; $L.c = L_1.c \times 2$; $L.val = L_1.val + B.c$

$L \to B$ $\quad B.i = 1$; $L.c = 1$; $L.val = B.c$

$R \to R_1 B$ $\quad B.i = R_1.c / 2$; $R.c = R_1.c / 2$; $R.val = R_1.val + B.c$

$R \to B$ $\quad B.i = 0.5$; $R.c = 0.5$; $R.val = B.c$

$B \to 0$ $\quad B.c = 0$

$B \to 1$ $\quad B.c = B.i$

(c) 给出(a)(b)2个语法制导定义的属性栈代码实现：

```
def calculate_S_val(S):
    if S is of type L.L:
        return calculate_L_val(S.L1) + calculate_L_val(S.L2) * pow(2, -len(S.L2))
    else:  # S is of type L
        return calculate_L_val(S.L)

def calculate_L_val(L):
    if L is of type LB:
        return calculate_L_val(L.L1) * 2 + L.B.val
    else:  # L is of type B
        return L.B.val

def calculate_B_val(B):
    return int(B)  # B is "0" or "1", so convert to int
```

```
def calculate_S_val(S):
    if S is of type L.L:
        L1_pos = calculate_length(S.L1) - 1  # Position for L1 starts from left to right
        L2_pos = -1  # Position for L2 starts from right to left
        return calculate_L_val(S.L1, L1_pos) + calculate_L_val(S.L2, L2_pos)
    else:  # S is of type L
        L_pos = calculate_length(S.L) - 1
        return calculate_L_val(S.L, L_pos)

def calculate_L_val(L, pos):
    if L is of type LB:
        return calculate_L_val(L.L1, pos - 1) + calculate_B_val(L.B, pos)
    else:  # L is of type B
        return calculate_B_val(L.B, pos)

def calculate_B_val(B, pos):
    return B.c * pow(2, pos)

def calculate_length(L):
    if L is of type LB:
        return 1 + calculate_length(L.L1)
    else:  # L is of type B
        return 1
```

T2. [习题 4.12]

4.12 文法如下：

$S \rightarrow (L) \mid a$

$L \rightarrow L,S \mid S$

(a) 写一个翻译方案，它输出每个 a 的嵌套深度。例如，对于句子 $(a,(a,a))$，输出的结果是 1 2 2。

(b) 写一个翻译方案，它打印出每个 a 在句子中是第几个字符。例如，当句子是 $(a,(a,(a,a),(a)))$ 时，打印的结果是 2 5 8 10 14。

(a) 用继承属性 depth 表示嵌套深度.

$S' \rightarrow \{S.depth = 0;\}\ S$

$S \rightarrow \{L.depth = S.depth + 1;\}\ (L)$

$S \rightarrow a\{print(S.depth)\}$

$L \rightarrow \{L_1.depth = L.depth;\}\ L_1,\ \{S.depth = L.depth;\}\ S$

$L \rightarrow \{S.depth = L.depth;\}\ S$

(b) 给文法符号 S 和 L 一个继承属性 in 和一个综合属性 out

表示在句子中该文法符号推出的字符序列前面有多少字符及该文法符号推出的字符序列的最后一个字符在句子中是第 n 个字符.

方案：

$S' \rightarrow \{S.in = 0;\}\ S$

$S \rightarrow \{L.in = S.in + 1;\}\ (L)\ \{S.out = L.out + 1;\}$

$S \rightarrow a\{S.out = S.in + 1;\ print(S.out);\}$

$L \rightarrow \{L_1.in = L.in;\}\ L_1,\ \{S.in = L_1.out + 1;\}\ S\ \{L.out = S.out;\}$

$L \rightarrow \{S.in = L.in;\}\ S\ \{L.out = S.out;\}$

(c)

```c
def parse_S_prime(tokens):
    # 初始深度设置为0
    depth = 0
    parse_S(tokens, depth)

def parse_S(tokens, depth):
    token = next(tokens)
    if token == '(':
        # 进入更深一层的括号，增加深度
        parse_L(tokens, depth + 1)
        # 忽略闭括号 ')'
        next(tokens)
    elif token == 'a':
        # 遇到 'a'，打印当前深度
        print(depth)

def parse_L(tokens, depth):
    while True:
        parse_S(tokens, depth)
        token = next(tokens, None)
        if not token or token != ',':
            # 如果没有更多的token或下一个token不是','，则结束这个L的解析
            break
```

```c
def parse_S_prime(tokens):
    # 初始位置设置为0
    in_position = 0
    parse_S(tokens, in_position)

def parse_S(tokens, in_position):
    token = next(tokens)
    if token == '(':
        # 记录括号后的字符位置
        L_out = parse_L(tokens, in_position + 1)
        # 更新S的out属性为L的out加上右括号的长度
        S_out = L_out + 1
        # 忽略闭括号 ')'
        next(tokens)
        return S_out
    elif token == 'a':
        # 记录'a'的位置，并打印它
        S_out = in_position + 1
        print(S_out)
        return S_out

def parse_L(tokens, in_position):
    L_out = in_position
    while True:
        # 更新S的in属性为当前L的out属性
        S_out = parse_S(tokens, L_out)
        L_out = S_out
        token = next(tokens, None)
        if not token or token != ',':
            # 如果没有更多的token或下一个token不是','，则结束这个L的解析
            break
    return L_out
```

T3. 第七讲 语法制导翻译第34页的翻译方案，在输入串是 (id+id)*id 时的输出结果。

E->E+T { print(“1”) }
E->T { print(“2”) }
T->T*F { print(“3”) }
T->F { print(“4”) }
F->(E) { print(“5”) }
F->id { print(“6”) }

(id + id) * id
⇒ 5 6 4 1 2 6 3 2 4

T4. 针对习题4.3或4.12中的文法：

（4.1）参考第五讲 自顶向下分析的第57-58页内容，给出相应的递归下降语法分析函数；

（4.2）在（4.1）基础上，分别给出习题4.3(a)和习题4.12(a)的（递归下降）预测翻译器。

(4.1)

4.3、4.12的文法：$S \to (L) \mid a$

$L \to L,S \mid S$

```c
void parse_S() {
    if (token == 'a') {
        match('a'); // 假设我们有一个match函数来匹配并消耗当前的输入记号
    } else if (token == '(') {
        match('(');
        parse_L();
        match(')');
    } else {
        // 出错处理
        error();
    }
}

void parse_L() {
    parse_S(); // 匹配S部分
    parse_L_prime(); // 匹配L'部分
}

void parse_L_prime() {
    while (token == ',') {
        match(','); // 匹配逗号
        parse_S(); // 匹配后面的S
    }
    // 如果不是逗号，则L' -> ε，这里不做任何事情，自然退出函数
}
```

(4.2) [4.3 a] 请给出制导定义输出括号对数：

$S' \to S$	print (S.num)
$S \to (L)$	S.num = L.num + 1
$S \to a$	S.num = 0
$L \to L_1, S$	L.num = L_1.num + S.num
$L \to S$	L.num = S.num

递归下降预测翻译器：

```
#include <stdio.h>

// 定义全局变量来跟踪括号对数
int numBrackets = 0;

void parse_S();
void parse_L();

void parse_S_prime() {
    parse_S();
    printf("%d\n", numBrackets);  // 输出最终的括号对数
}

void parse_S() {
    if (token == '(') {
        match('(');
        parse_L();
        match(')');
        numBrackets++;  // 遇到新的括号对，增加计数
    } else if (token == 'a') {
        match('a');
    } else {
        error();
    }
}

void parse_L() {
    if (lookahead == '(' || lookahead == 'a') {
        parse_S();
        while (lookahead == ',') {
            match(',');
            parse_S();
        }
    } else {
        error();
    }
}
```

[4.12a] 翻译方案输出每个a嵌套深度:

$S' \to \{ S.depth = 0; \}\ S$

$S \to \{ L.depth = S.depth + 1; \}\ (L)$

$S \to a \{ print(S.depth) \}$

$L \to \{ L_1.depth = L.depth; \}\ L_1, \{ S.depth = L.depth; \}\ S$

$L \to \{ S.depth = L.depth; \}\ S$

递归下降预测翻译器:

```
#include <stdio.h>

void parse_S(int depth);
void parse_L(int depth);

void parse_S_prime() {
    parse_S(0);
}

void parse_S(int depth) {
    if (token == '(') {
        match('(');
        parse_L(depth + 1);  // 进入新的括号层，增加深度
        match(')');
    } else if (token == 'a') {
        match('a');
        printf("%d ", depth);  // 输出当前字母a的嵌套深度
    } else {
        error();
    }
}

void parse_L(int depth) {
    if (lookahead == '(' || lookahead == 'a') {
        parse_S(depth);
        while (lookahead == ',') {
            match(',');
            parse_S(depth);  // 同一列表示同一深度
        }
    } else {
        error();
    }
}
```