编译原理第四次作业

by qq

（1）习题4.9， 并分别给出（a）和（b）两个语法制导定义的属性栈代码实现（非yacc代码）。

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)

引入L的综合属性length，用来表示L所推出的二进制串长度。除此之外，各文法符号都有一个综合属性val，表示文法符号所推出二进制串的值。写出语法制导定义如下：

产生式	语义规则
$S \rightarrow L_1.L_2$	$S.val = L_1.val + L_2.val/2^{L_2.length}$
$S \rightarrow L$	$S.val = L.val$
$L \rightarrow L_1B$	$L.val = L_1.val \times 2 + B.val; L.length = L_1.length + 1$
$L \rightarrow B$	$L.val = B.val; L.length = 1$
$B \rightarrow 0$	$B.val = 0$
$B \rightarrow 1$	$B.val = 1$

根据语法制导定义，写出属性栈代码如下：

产生式	属性栈代码
$S \rightarrow L_1.L_2$	$Stack[ntop].val = Stack[top-2].val + Stack[top].val/2^{Stack[top].length}$
$S \rightarrow L$	
$L \rightarrow L_1B$	$Stack[ntop].val = Stack[top-1].val \times 2 + Stack[top].val;$ $Stack[ntop].length = Stack[top-1].length + 1$
$L \rightarrow B$	$Stack[ntop].length = 1$
$B \rightarrow 0$	$Stack[ntop].val = 0$
$B \rightarrow 1$	$Stack[ntop].val = 1$

(b)

引入L、R、B的继承属性weight，分别表示L的最右一位权重、R的最左一位权重、B所在位的权重。同时，每个文法符号都有综合属性val（为了方便大家阅读，没有使用题目给的综合属性名称B.c），表示文法符号所推出二进制串的值乘以它们的权重。写出语法制导定义如下：

产生式	语义规则
$S \rightarrow L.R$	$L.weight = 1; R.weight = 0.5; S.val = L.val + R.val$
$S \rightarrow L$	$L.weight = 1; S.val = L.val$
$L \rightarrow L_1B$	$L_1.weight = L.weight \times 2; B.weight = L.weight; L.val = L_1.val + B.val$
$L \rightarrow B$	$B.weight = L.weight; L.val = B.val$
$R \rightarrow BR_1$	$B.weight = R.weight; R_1.weight = R.weight/2; R.val = B.val + R_1.val$
$R \rightarrow B$	$B.weight = R.weight; R.val = B.val$
$B \rightarrow 0$	$B.val = 0$
$B \rightarrow 1$	$B.val = B.weight$

考虑属性计算的先后顺序，写出翻译方案如下：

$$
\begin{aligned}
S \rightarrow & \{L.weight = 1\} \\
& L.\ \{R.weight = 0.5\} \\
& R\ \{S.val = L.val + R.val\} \\
S \rightarrow & \{L.weight = 1\} \\
& L\ \{S.val = L.val\} \\
L \rightarrow & \{L_1.weight = L.weight \times 2\} \\
& L_1\ \{B.weight = L.weight\} \\
& B\ \{L.val = L_1.val + B.val\} \\
L \rightarrow & \{B.weight = L.weight\} \\
& B\ \{L.val = B.val\} \\
R \rightarrow & \{B.weight = R.weight\} \\
& B\ \{R_1.weight = R.weight/2\} \\
& R_1\ \{R.val = B.val + R_1.val\} \\
R \rightarrow & \{B.weight = R.weight\} \\
& B\ \{R.val = B.val\} \\
B \rightarrow & 0\ \{B.val = 0\} \\
B \rightarrow & 1\ \{B.val = B.weight\}
\end{aligned}
$$

引入标记非终结符M, N, P, Q, T，改写翻译方案，并写出相应的属性栈代码：

翻译方案	属性栈代码
$S \rightarrow M\ \{L.weight = M.s\}$	
$\quad L.N\ \{R.weight = N.s\}$	
$\quad R\ \{S.val = L.val + R.val\}$	$Stack[ntop] = Stack[top-3] + Stack[top]$
$M \rightarrow \epsilon\ \{M.s = 1\}$	$Stack[ntop] = 1$
$N \rightarrow \epsilon\ \{N.s = 0.5\}$	$Stack[ntop] = 0.5$
$S \rightarrow M\ \{L.weight = M.s\}$	
$\quad L\ \{S.val = L.val\}$	$Stack[ntop] = Stack[top]$
$L \rightarrow \{P.i = L.weight\}$	
$\quad P\ \{L_1.weight = P.s\}$	
$\quad L_1\ \{Q.i = L.weight\}$	
$\quad Q\ \{B.weight = Q.s\}$	
$\quad B\ \{L.val = L_1.val + B.val\}$	$Stack[ntop] = Stack[top-2] + Stack[top]$
$P \rightarrow \epsilon\ \{P.s = P.i \times 2\}$	$Stack[ntop] = Stack[top] \times 2$
$Q \rightarrow \epsilon\ \{Q.s = Q.i\}$	$Stack[ntop] = Stack[top-2]$
$L \rightarrow \{B.weight = L.weight\}$	
$\quad B\ \{L.val = B.val\}$	
$R \rightarrow \{B.weight = R.weight\}$	
$\quad B\ \{T.i = R.weight\}$	
$\quad T\ \{R_1.weight = T.s\}$	
$\quad R_1\ \{R.val = B.val + R_1.val\}$	$Stack[ntop] = Stack[top-2] + Stack[top]$
$T \rightarrow \epsilon\ \{T.s = T.i/2\}$	$Stack[ntop] = Stack[top-1]/2$
$R \rightarrow \{B.weight = R.weight\}$	
$\quad B\ \{R.val = B.val\}$	
$B \rightarrow 0\ \{B.val = 0\}$	$Stack[ntop] = 0$
$B \rightarrow 1\ \{B.val = B.weight\}$	$Stack[ntop] = Stack[top-1]$

> 解析：所有文法符号的继承属性weight，均存放在归约这个文法符号时的句柄下面，即属性栈上对应标记非终结符的位置。（见郑老师讲义lec7 39~48页）
>
> 什么时候需要引入标记非终结符?
>
> 1. 当同一个文法符号继承属性存放的位置不固定时，可引入标记非终结符，在标记非终结符的位置存放继承属性。比如这道题文法符号的继承属性weight，都存放在属性栈上对应前一个文法符号的位置。
> 2. 当继承属性的值不是综合属性的复写拷贝，而是综合属性的函数时。比如这道题的“ $L \rightarrow L_1 B$ ”产生式，由于 $L_1.weight = L.weight \times 2$ ，需要引入标记非终结符P。
>
> 当上述两种情况都不存在时，就可以不引入标记非终结符。比如这道题的" $R \rightarrow B$ "产生式。

L.weight存放在属性栈上对应前一个文法符号的位置(标红的位置)

$S \to ML.NR$

$S \to ML$

$L \to PL_1QB$

R.weight存放在属性栈上对应前一个文法符号的位置

$S \to ML.NR$

$R \to BTR_1$

B.weight也存放在属性栈上对应前一个文法符号的位置

$L \to PL_1QB$

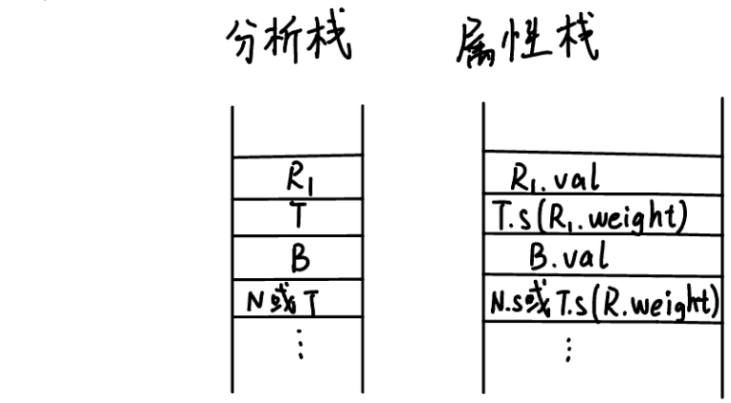
$L \to B$

$R \to BTR_1$

$R \to B$

B.weight = L.weight (或 R.weight),
因为L(或R)前一个文法符号的属性栈已经存放了L.weight(或R.weight),所以无需引入标记非终结符.

例如$R \to BTR_1$归约前:



> 例如在输入串为10.01的时候，分析栈和属性栈的变化如下：

输入串	分析栈	属性栈	归约产生式	属性栈代码
10.01				
10.01	M	1	$M \to \epsilon$	$Stack[ntop] = 1$
10.01	MP	1 2	$P \to \epsilon$	$Stack[ntop] = Stack[top] \times 2$
0.01	$MP1$	1 2 _		
0.01	MPB	1 2 2	$B \to 1$	$Stack[ntop] = Stack[top-1]$
0.01	MPL	1 2 2	$L \to B$	
0.01	$MPLQ$	1 2 2 1	$Q \to \epsilon$	$Stack[ntop] = Stack[top-2]$
.01	$MPLQ0$	1 2 2 1 _		
.01	$MPLQB$	1 2 2 1 0	$B \to 0$	$Stack[ntop] = 0$
.01	ML	1 2	$L \to PL_1QB$	$Stack[ntop] = Stack[top-2] + Stack[top]$
01	$ML.$	1 2 _		
01	$ML.N$	1 2 _ $\frac{1}{2}$	$N \to \epsilon$	$Stack[ntop] = 0.5$
1	$ML.N0$	1 2 _ $\frac{1}{2}$ _		
1	$ML.NB$	1 2 _ $\frac{1}{2}$ 0	$B \to 0$	$Stack[ntop] = 0$
1	$ML.NBT$	1 2 _ $\frac{1}{2}$ 0 $\frac{1}{4}$	$T \to \epsilon$	$Stack[ntop] = Stack[top-1]/2$
	$ML.NBT1$	1 2 _ $\frac{1}{2}$ 0 $\frac{1}{4}$ _		
	$ML.NBTB$	1 2 _ $\frac{1}{2}$ 0 $\frac{1}{4}$ $\frac{1}{4}$	$B \to 1$	$Stack[ntop] = Stack[top-1]$
	$ML.NBTR$	1 2 _ $\frac{1}{2}$ 0 $\frac{1}{4}$ $\frac{1}{4}$	$R \to B$	
	$ML.NR$	1 2 _ $\frac{1}{2}$ $\frac{1}{4}$	$R \to BTR_1$	$Stack[ntop] = Stack[top-2] + Stack[top]$
	S	$\frac{9}{4}$	$S \to ML.NR$	$Stack[ntop] = Stack[top-3] + Stack[top]$

（2）习题4.12，并分别给出（a）和（b）两个翻译方案的属性栈代码实现（非yacc代码）。

4.12 文法如下：

$S \to (L) \mid a$

$L \to L, S \mid S$

（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)

引入继承属性depth，写出翻译方案：

$$
\begin{aligned}
S' &\to \{S.depth = 0\}S \\
S &\to (\{L.depth = S.depth + 1\}L) \\
S &\to a\{print(S.depth)\} \\
L &\to \{L_1.depth = L.depth\}L_1, \{S.depth = L.depth\}S \\
L &\to \{S.depth = L.depth\}S
\end{aligned}
$$

引入标记非终结符M, N, O，写出属性栈代码：

翻译方案	属性栈代码
$S' \to M\{S.depth = M.s\}S$	
$M \to \epsilon\{M.s = 0\}$	$Stack[ntop] = 0$
$S \to (\{N.i = S.depth\}$	
$N\{L.depth = N.s\}L)$	
$N \to \epsilon\{N.s = N.i + 1\}$	$Stack[ntop] = Stack[top-1] + 1$
$S \to a\{print(S.depth)\}$	$print(Stack[top-1])$
$L \to \{L_1.depth = L.depth\}$	
$L_1, \{O.i = L.depth\}$	
$O\{S.depth = O.s\}S$	
$O \to \epsilon\{O.s = O.i\}$	$Stack[ntop] = Stack[top-2]$
$L \to \{S.depth = L.depth\}S$	

(b)

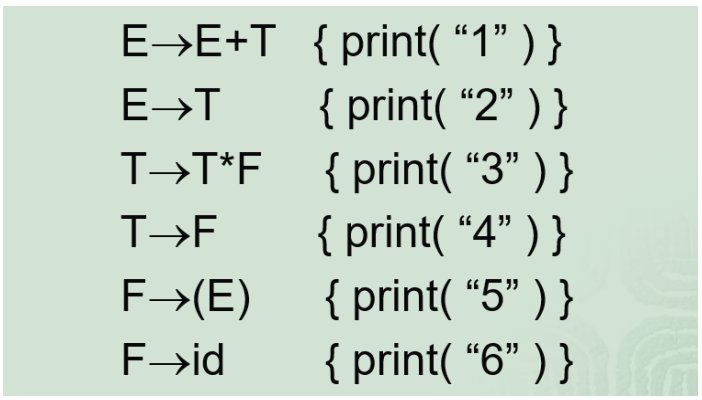
before是继承属性，表示文法符号前的字符数；out是综合属性，表示这个文法符号推出的字符总数。写出翻译方案：

$$
\begin{aligned}
S' &\to \{S.before = 0\}S \\
S &\to (\{L.before = S.before + 1\} \\
&\quad L)\{S.out = L.out + 2\} \\
S &\to a\{print(S.before + 1); S.out = 1\} \\
L &\to \{L_1.before = L.before\} \\
&\quad L_1, \{S.before = L_1.before + L_1.out + 1\} \\
&\quad S\{L.out = L_1.out + 1 + S.out\} \\
L &\to \{S.before = L.before\}S\{L.out = S.out\}
\end{aligned}
$$

引入标记非终结符M, N, O，写出属性栈代码：

翻译方案	属性栈代码
$S' \to M\{S.before = M.s\}S$	
$M \to \epsilon\{M.s = 0\}$	$Stack[ntop] = 0$
$S \to (\{N.i = S.before\}$	
$N\{L.before = N.s\}$	
$L)\{S.out = L.out + 2\}$	$Stack[ntop] = Stack[top-1] + 2$
$N \to \epsilon\{N.s = N.i + 1\}$	$Stack[ntop] = Stack[top-1] + 1$
$S \to a\{print(S.before + 1);$	$print(Stack[top-1] + 1)$
$S.out = 1\}$	$Stack[ntop] = 1$
$L \to \{L_1.before = L.before\}$	
$L_1, \{O.i = L_1.before + L_1.out + 1\}$	
$O\{S.before = O.s\}$	
$S\{L.out = L_1.out + 1 + S.out\}$	$Stack[ntop] = Stack[top-3] + 1 + Stack[top]$
$O \to \epsilon\{O.s = O.i\}$	$Stack[ntop] = Stack[top-2] + Stack[top-1] + 1$
$L \to \{S.before = L.before\}$	
$S\{L.out = S.out\}$	

(3) 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。



输出结果为64264154632，分析过程如下：

输入串	分析栈	归约产生式	输出
$(id+id)*id$			
$id+id)*id$	$($		
$+id)*id$	$(id$		
$+id)*id$	$(F$	$F \to id$	6
$+id)*id$	$(T$	$T \to F$	4
$+id)*id$	$(E$	$E \to T$	2
$id)*id$	$(E+$		
$)*id$	$(E+id$		
$)*id$	$(E+F$	$F \to id$	6
$)*id$	$(E+T$	$T \to F$	4
$)*id$	$(E$	$E \to E+T$	1
$*id$	(E)		
$*id$	F	$F \to (E)$	5
$*id$	T	$T \to F$	4
id	$T*$		
	$T*id$		
	$T*F$	$F \to id$	6
	T	$T \to T*F$	3
	E	$E \to T$	2

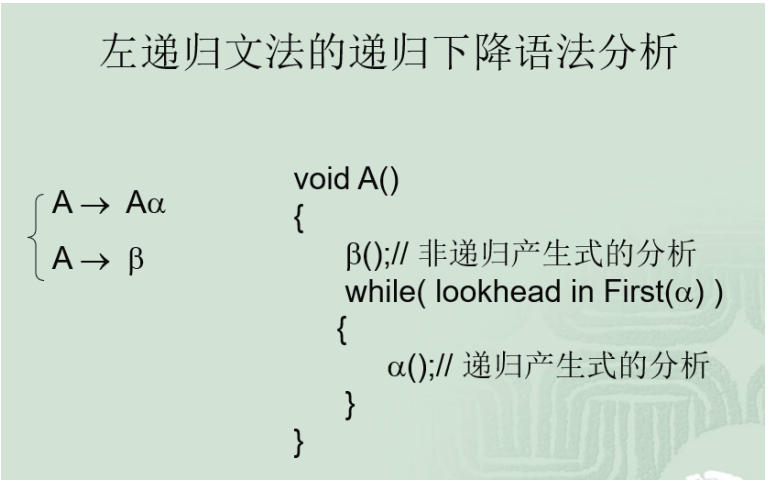
即最右推导次序的逆序：

$$E \Rightarrow_{rm} T \Rightarrow_{rm} T*F \Rightarrow_{rm} T*id \Rightarrow_{rm} F*id \Rightarrow_{rm} (E)*id \Rightarrow_{rm} (E+T)*id \Rightarrow_{rm} (E+F)*id$$
$$\Rightarrow_{rm} (E+id)*id \Rightarrow_{rm} (T+id)*id \Rightarrow_{rm} (F+id)*id \Rightarrow_{rm} (id+id)*id$$

（4）针对习题4.12中的文法：

$$S \to (L) \mid a$$
$$L \to L, S \mid S$$

（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；




```
void S()
{
    if(lookahead == '(')
    {
        match('(');
        L();
        match(')');
    }
    else if(lookahead == 'a')
        match('a');
    else
```

```
        error();
}

void L()
{
    S(); //非递归产生式的分析
    while(lookahead == ',')
    {
        match(',');
        S(); //递归产生式的分析
    }
}
```

（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。

(4.2)-4.3(a)

引入综合属性val，表示文法符号所推出字符串的括号对数，写出翻译方案：

$$
\begin{aligned}
S' &\to S\{print(S.val)\} \\
S &\to (L)\{S.val = L.val + 1\} \\
S &\to a\{S.val = 0\} \\
L &\to L_1, S\{L.val = L_1.val + S.val\} \\
L &\to S\{L.val = S.val\}
\end{aligned}
$$

将综合属性val作为函数的返回值，设计递归下降预测翻译器：

```
void S'()
{
    print(S());
}

int S()
{
    int val;
    if(lookahead == '(')
    {
        match('(');
        val = L() + 1;
        match(')');
    }
    else if(lookahead == 'a')
    {
        match('a');
        val = 0;
    }
    else
        error();
    return val;
}

int L()
{
    int val;
    val = S(); //非递归产生式的分析
    while(lookahead == ',')
    {
        match(',');
        val += S(); //递归产生式的分析
    }
    return val;
```

```
}
```

(4.2)-4.12(a)

可直接使用第(2)小题的翻译方案：

$$
\begin{aligned}
S' \rightarrow & \{S.depth = 0\}S \\
S \rightarrow & (\{L.depth = S.depth + 1\}L) \\
S \rightarrow & a\{print(S.depth)\} \\
L \rightarrow & \{L_1.depth = L.depth\}L_1, \{S.depth = L.depth\}S \\
L \rightarrow & \{S.depth = L.depth\}S
\end{aligned}
$$

将继承属性depth作为函数的参数，设计递归下降预测翻译器：

```
void S'()
{
    S(0);
}

void S(int depth)
{
    if(lookahead == '(')
    {
        match('(');
        L(depth + 1);
        match(')');
    }
    else if(lookahead == 'a')
    {
        match('a');
        print(depth);
    }
    else
        error();
}

void L(int depth)
{
    S(depth); //非递归产生式的分析
    while(lookahead == ',')
    {
        match(',');
        S(depth); //递归产生式的分析
    }
}
```

> 参考资料：
>
> 1. 2023秋季学期课程习题答案
> 2. https://blog.csdn.net/weixin_56462041/article/details/129015814
>
> 尽管我反复核查，笔误可能也难以避免。如果大家有疑惑，请向我反馈，我会尽快答复。