Assignment9

ID	2019K8009929039
name	周鹏宇

- **练习5.2.1：考虑文法**

$$\mathbf{E \rightarrow E + T \mid T}$$
$$\mathbf{T \rightarrow T * F \mid F}$$
$$\mathbf{F \rightarrow (E) \mid num \mid var}$$

其中 num 表示数字字面量，如123；var 表示变量，如 x。

1. **如果一个项或表达式中不存在变量，它可以在编译期进行求值，例如 1+2*3。设计一个 SDD 来判断各个项 T 和表达式 E 能否在编译期确定它的值。（备注：可以用 E.isconst = true 表示它可以在编译期求值）**

给出SDD如下：

产生式	语义规则
$E \rightarrow E_1 + T$	$E.iscounst = E_1.iscounst\ and\ T.iscounst$ $E.val = (E.iscounst == true)?(E_1.val + T.val) : error$
$E \rightarrow T$	$E.iscounst = T.iscounst$ $E.val = (E.iscounst == true)?T.val : error$
$T \rightarrow T_1 * F$	$T.iscounst = T_1.iscounst\ and\ F.iscounst$ $T.val = (T.iscounst == true)?(T_1.val * F.val) : error$
$T \rightarrow F$	$T.iscounst = F.iscounst$ $T.val = (T.iscounst == true)?F.val : error$
$F \rightarrow (E)$	$F.iscounst = E.iscounst$ $F.val = (F.iscounst == true)?E.val : error$
$F \rightarrow num$	$F.iscounst = true$ $F.val = num.lexval$
$F \rightarrow var$	$F.iscounst = false$ $f.val = error$

练习5.2.2：改写下面的 SDT，消除左递归。其中a、b、c、d为语义动作，不涉及属性计算。0和1是终结符。

$$\mathbf{A \rightarrow A\ \{a\}\ B \mid A\ B\ \{b\} \mid 0}$$
$$\mathbf{B \rightarrow B\ \{c\}\ A \mid B\ A\ \{d\} \mid 1}$$

由于a,b,c,d不涉及属性计算，其消除左递归方式和第四章中没有区别：

$A \to 0A'$
$A' \to \{a\}BA'|B\{b\}A'|\epsilon$
$B \to 1B'$
$B' \to \{c\}AB'|A\{d\}B'|\epsilon$

练习5.2.3：考虑下图中排版语言的 SDD 和 SDT

1. 修改 SDD，使其包含一个综合属性 B.ch，表示 Box 中字符的个数。用 getCh(text.lexval) 表示 text 中字符的个数
2. 将新规则加入 SDT 的合适位置

产生式	语义规则
1) $S \to B$	$B.ps = 10$
2) $B \to B_1\ B_2$	$B_1.ps = B.ps$ $B_2.ps = B.ps$ $B.ht = \max(B_1.ht, B_2.ht)$ $B.dp = \max(B_1.dp, B_2.dp)$
3) $B \to B_1$ **sub** B_2	$B_1.ps = B.ps$ $B_2.ps = 0.7 \times B.ps$ $B.ht = \max(B_1.ht, B_2.ht - 0.25 \times B.ps)$ $B.dp = \max(B_1.dp, B_2.dp + 0.25 \times B.ps)$
4) $B \to (\ B_1\ )$	$B_1.ps = B.ps$ $B.ht = B_1.ht$ $B.dp = B_1.dp$
5) $B \to$ **text**	$B.ht = getHt(B.ps, \mathbf{text}.lexval)$ $B.dp = getDp(B.ps, \mathbf{text}.lexval)$

产生式	语义动作
1) $S \to B$	{ $B.ps = 10$; }
2) $B \to B_1\ B_2$	{ $B_1.ps = B.ps$; } { $B_2.ps = B.ps$; } { $B.ht = \max(B_1.ht, B_2.ht)$; $B.dp = \max(B_1.dp, B_2.dp)$; }
3) $B \to B_1$ **sub** B_2	{ $B_1.ps = B.ps$; } { $B_2.ps = 0.7 \times B.ps$; } { $B.ht = \max(B_1.ht, B_2.ht - 0.25 \times B.ps)$; $B.dp = \max(B_1.dp, B_2.dp + 0.25 \times B.ps)$; }
4) $B \to (\ B_1\ )$	{ $B_1.ps = B.ps$; } { $B.ht = B_1.ht$; $B.dp = B_1.dp$; }
5) $B \to$ **text**	{ $B.ht = getHt(B.ps, \mathbf{text}.lexval)$; $B.dp = getDp(B.ps, \mathbf{text}.lexval)$; }

首先将产生式的语义展示如下：

$$B \to B_1 B_2 \mid B_1 \mathbf{sub} B_2 \mid (B_1) \mid \mathbf{text}$$

对应于这四个产生式，一个方框可以是下列之一：

1）两个并列的方框，其中第一个方框 B_1 在另一个方框 B_2 的左边。

2）一个方框和一个下标方框。第二个方框的尺寸较小且位置较低，位于第一个方框的右边。

3）一个用括号括起来的方框，用于方框和下标的分组。Eqn 和 Tex 都使用花括号进行分组，但是我们将使用通常的圆括号来分组，以避免和 SDT 动作两边的括号混淆。

4）一个文本串，也就是任何字符串。

这个文法是二义性的，但是如果我们令下标和并列关系都是右结合的，并且令 **sub** 的优先级高于并列，那么我们仍然可以使用它来完成自底向上的语法分析。

那么

1. 在此只给出每个产生式需要新增的语义规则：
   a. $S \to B$ 无需增加
   b. $B \to B_1 B_2$ 新增 $B.ch = B_1.ch + B_2.ch$
   c. $B \to B_1 \mathbf{sub} B_2$ 新增 $B.ch = B_1.ch + B_2.ch$
   d. $B \to (B_1)$ 新增 $B.ch = B_1.ch$
   e. $B \to \mathtt{text}$ 新增 $B.ch = getCh(text.lexval)$

2. 因为新增的规则都是关于B的综合属性ch的规则，因此直接新增在**SDT**的dp规则之后即可，例如对于$B \to B_1 B_2$而言，其规则变为

$$\{B_1.ps = B.ps\}$$
$$\{B_2.ps = B.ps\}$$
$$\{B.ht = max(B_1.ht, B_2.ht)\}$$
$$\{B_1.dp = max(B_1.dp, B_2.dp)\}$$
$$\{B.ch = B_1.ch + B_2.ch\}$$

练习5.2.4：为下面的产生式写出一个和例5.19类似的 L 属性 SDT。这里的每个产生式表示一个常见的 C 语言中那样的控制流结构。你可能需要生成一个三地址语句来跳转到某个标号 L，此时你可以生成语句 goto L

1. **S → if (C) S_1 else S_2**
2. **S → do S_1 while (C)**
3. **S → { L } 和 L → L S | ε**

注1：列表中的任何语句都可能包含一条从它的内部跳转到下一条语句的跳转指令，因此简单地为各个语句按顺序生成代码是不够的。
注2：可以先写出SDD，然后按照5.4.5节方法转换为SDT。

1.

$$L_1 = newlable()$$
$$L_2 = newlable()$$
$$C.true = L_1$$
$$C.false = L_2$$
$$S_1.next = S.next$$
$$S_2.next = S.next$$
$$S.code = C.code||lable||L_1||S_1.code||gotoS.next||lable||L_2||S_2.code$$

转换成SDT如下：

$$S \to if(\quad \{L_1 = newlable(); L_2 = newlable(); C.ture = L_1; C.false = L_2\}$$
$$C) \quad \{S_1.next = S.next\}$$
$$S_1\ else \quad \{S_2.next = S.next\}$$
$$S_2 \quad \{S.code = C.code||lable||L_1||S_1.code||gotoS.next||lable||L_2||S_2.code\}$$

2.

$$L_1 = newlable()$$
$$L_2 = newlable()$$
$$C.true = L_1$$
$$C.false = S.next$$
$$S_1.next = L_2$$
$$S.code = lable||L_1||S_1.code||lable||L_2||C.code$$

转换为SDT如下：

$S \to do($ $\{L_1 = newlable(); L_2 = newlable(); S_1.next = L_2\}$

$S_1)while($ $\{C.true = L_1; C.false = S.next\}$

$C)$ $\{S.code = lable||L_1||S_1.code||lable||L_2||C.code\}$

3. 步骤完全类上，直接给出SDT

$S \to \{$ $\{L.next = S.next\}$

$L\}$ $\{S.code = L.code\}$

$L \to$ $\{M = newlable(); L_1.next = M\}$

L_1 $\{S.next = L.next\}$

S $\{L.code = L_1.code||lable||M||S.code\}$

$L \to \epsilon$ $\{L.code = \textbf{goto}||L.next\}$