HW-第五/六章

产生式	语义规则
1) $L \to E$ **n**	$L.val = E.val$
2) $E \to E_1 + T$	$E.val = E_1.val + T.val$
3) $E \to T$	$E.val = T.val$
4) $T \to T_1 * F$	$T.val = T_1.val \times F.val$
5) $T \to F$	$T.val = F.val$
6) $F \to (E)$	$F.val = E.val$
7) $F \to$ **digit**	$F.val =$ **digit**.lexval

图 1. 一个简单的桌上计算器的语法制导定义

产生式	语义规则
1) $T \to F\,T'$	$T'.inh = F.val$ $T.val = T'.syn$
2) $T' \to *\,F\,T_1'$	$T_1'.inh = T'.inh \times F.val$ $T'.syn = T_1'.syn$
3) $T' \to \epsilon$	$T'.syn = T'.inh$
4) $F \to$ **digit**	$F.val =$ **digit**.*lexval*

图 2. 一个适用于自顶向下语法分析文法的SDD

问题 1. (原书5.1.2，薄书5.1.2)拓展图2中的SDD，使它可以像图1所示的那样处理表达式。

一个更准确的表述为，依照图1，为图2制造一个适用于自顶向下语法分析的SDD，你可能需要先为文法消除左递归。

图1是L属性先序后处理的计算器

拓展图2得：

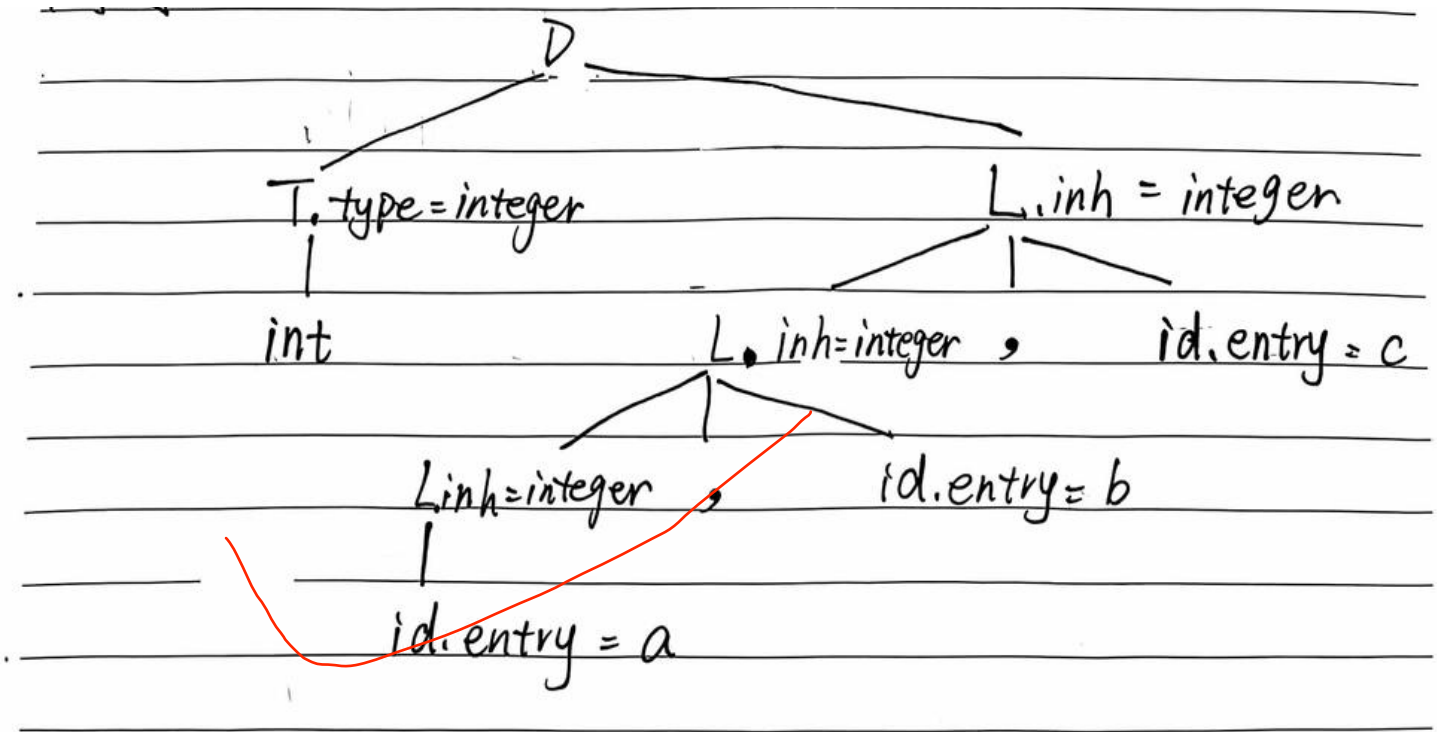
产生式	语义规则
$E \to T\ E'$	$E'.inh = T.val$; $E.val = E'.syn$
$E' \to +\ T\ E'_1$	$E'_1.inh = T.val + E'.inh$; $E'.syn = E'_1.syn$
$E' \to \varepsilon$	$E'.syn = E'.inh$
$T \to F\ T'$	$T'.inh = F.val$; $T.val = T'.syn$
$T' \to *\ F\ T'_1$	$T'_1.inh = T'.inh \times F.val$; $T'.syn = T'_1.syn$
$T' \to \varepsilon$	$T'.syn = T'.inh$
$F \to digit$	$F.val = digit.lexval$
$F \to (\ E\ )$	$F.val = E.val$

产生式	语义规则
1) $D \to T\ L$	$L.inh = T.type$
2) $T \to \mathbf{int}$	$T.type = \text{integer}$
3) $T \to \mathbf{float}$	$T.type = \text{float}$
4) $L \to L_1\ ,\ \mathbf{id}$	$L_1.inh = L.inh$ $addType(\mathbf{id}.entry, L.inh)$
5) $L \to \mathbf{id}$	$addType(\mathbf{id}.entry, L.inh)$

图 3. 简单类型声明的语法制导定义

问题 2. (原书5.2.2，薄书5.2.2)考虑图3中的SDD，给出下列表达式对应的标注语法分析树

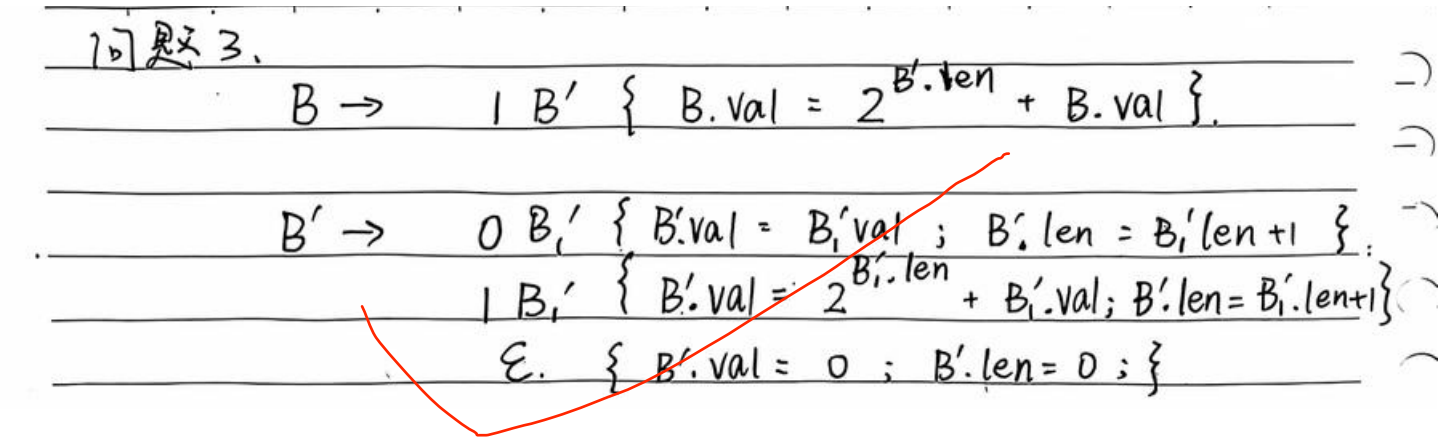
int a,b,c



问题 3. (原书5.4.3，薄书5.4.3)下面的SDT计算了一个由0和1组成的串的值。它把输入的符号串当作正二进制数来解释

$$
\begin{aligned}
B &\rightarrow B_1 0 \{B.\text{val} = 2 \times B_1.\text{val}\} \\
&\mid B_1 1 \{B.\text{val} = 2 \times B_1.\text{val} + 1\} \\
&\mid 1 \{B.\text{val} = 1\}
\end{aligned}
$$

改写这个SDT，使得基础文法不再是左递归的，但仍然可以计算出整个输入串的相同的$B.\text{val}$的值



问题 4. (原书5.4.4，薄书5.4.4)仿照书中例5.19，为下面的产生式写出一个L属性的SDD并转换为SDT

$$S \rightarrow \textbf{do}\ S_1\ \textbf{while}\ (C)$$

SDD:

$S \to$ do S_1 while (C) $\quad L_1 = new()$; S_1 语句代码开头

$L_2 = new()$; while 语句代码开头

$S_1.next = L_2$

$C.false = S.next$

$C.true = L_1$

$S.code = label \| L_1 \| S_1.code \| label \| L_2 \| C.code$

SDT:

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

S_1

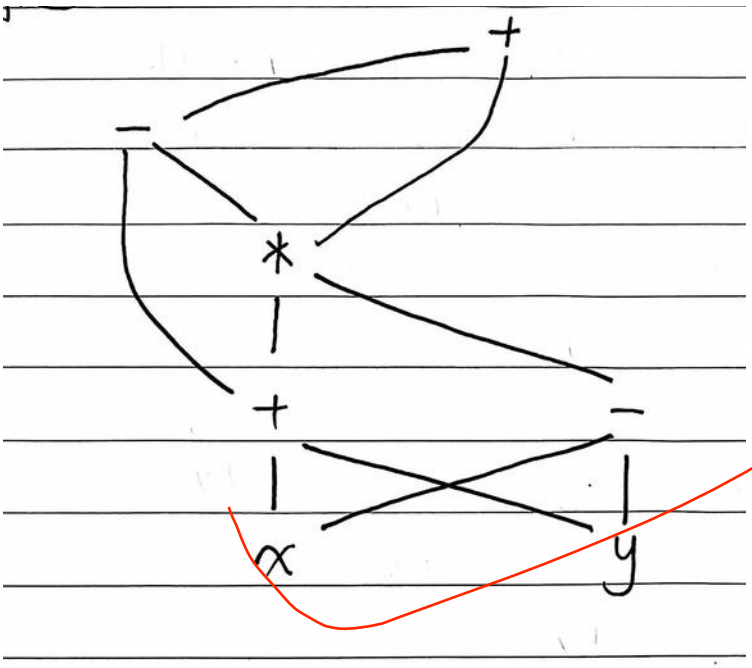
while $(\{C.false = S.next;\ C.true = L_1\}$

$C)$

$\{S.code = label \| L_1 \| S_1.code \| label \| L_2 \| C.code$

问题 5. (原书6.1.1，薄书6.1.1)为下列表达式构造DAG

$$((x+y)-((x+y)*(x-y)))+((x+y)*(x-y))$$

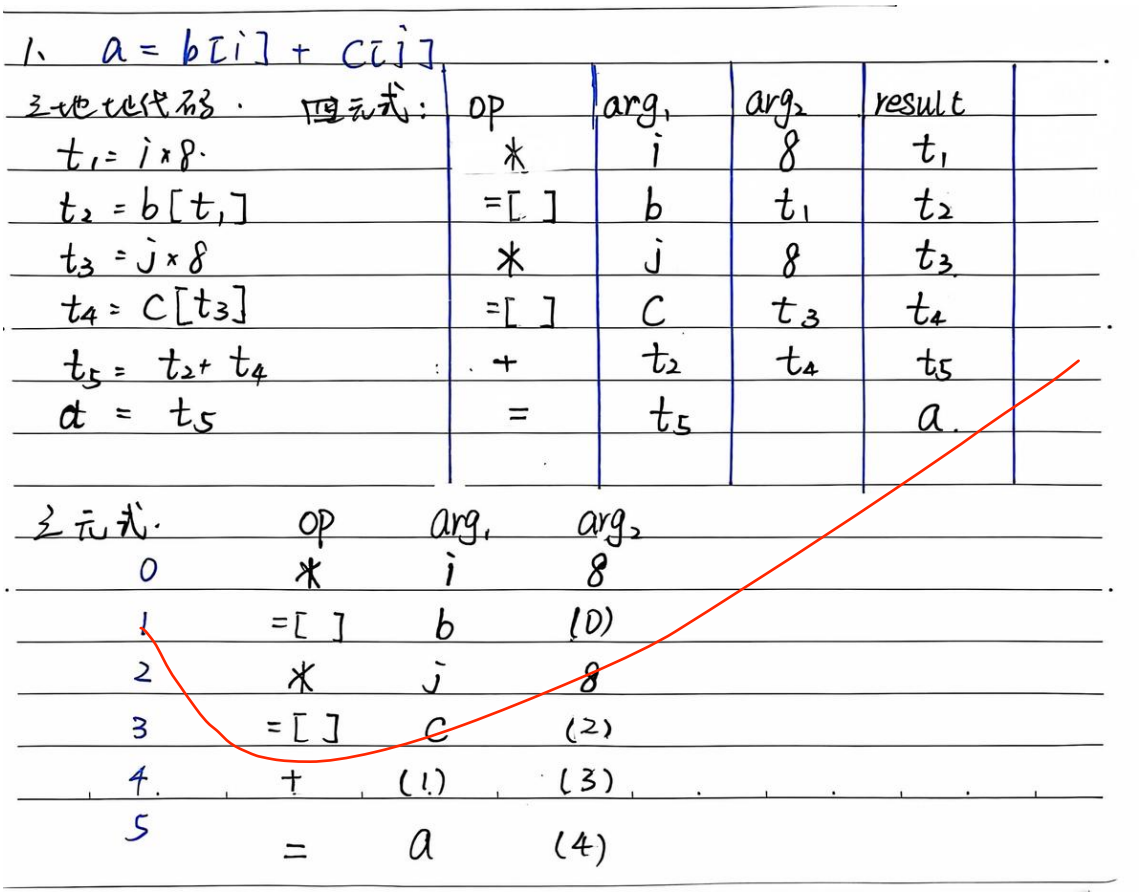


问题 6. (原书6.2.2, 薄书6.2.2)考虑下列赋值语句

1. $a = b[i] + c[j]$
2. $a[i] = b * c - b * d$

假定每个数组元素占八个存储单元，将赋值语句翻译成

1. 四元式序列
2. 三元式序列



1、 a = b[i] + c[j]

三地址代码：　四元式：

	op	arg1	arg2	result
$t_1 = i \times 8$	*	i	8	t_1
$t_2 = b[t_1]$	=[ ]	b	t_1	t_2
$t_3 = j \times 8$	*	j	8	t_3
$t_4 = c[t_3]$	=[ ]	c	t_3	t_4
$t_5 = t_2 + t_4$	+	t_2	t_4	t_5
$a = t_5$	=	t_5		a

三元式：

	op	arg1	arg2
0	*	i	8
1	=[ ]	b	(0)
2	*	j	8
3	=[ ]	c	(2)
4	+	(1)	(3)
5	=	a	(4)

2. a[i] = b * c - b * d

三地址	四元式	op	arg1	arg2	result
$t_1 = b \times c$		*	b	c	t_1
$t_2 = b \times d$		*	b	d	t_2
$t_3 = t_1 - t_2$		-	t_1	t_2	t_3
$t_4 = i \times 8$		*	i	8	t_4
$t_5 = \&a[t_4]$		&[]	a	t_4	t_5
$t_5 = t_3$		=	t_3		t_5

三元式

	op	arg1	arg2
0	*	b	c
1	*	b	d
2	-	(0)	(1)
3	*	i	8
4	&[]	a	(3)
5.	=	(4)	(2)

$$
\begin{array}{lll}
S \rightarrow \mathbf{id} = E\ ; & \{\ gen(\ top.get(\mathbf{id}.lexeme)\ '='\ E.addr);\ \} \\
\quad \mid\ L = E\ ; & \{\ gen(L.array\ .base\ '['\ L.addr\ ']'\ '='\ E.addr);\ \} \\
E \rightarrow E_1 + E_2 & \{\ E.addr = \mathbf{new}\ Temp\ ();\ gen(E.addr\ '='\ E_1.addr\ '+'\ E_2.addr);\ \} \\
\quad \mid\ \mathbf{id} & \{\ E.addr = top.get(\mathbf{id}.lexeme);\ \} \\
\quad \mid\ L & \{\ E.addr = \mathbf{new}\ Temp\ ();\ gen(E.addr\ '='\ L.array.base\ '['\ L.addr\ ']');\ \} \\
L \rightarrow \mathbf{id}\ [\ E\ ] & \{\ L.array = top.get(\mathbf{id}.lexeme);\ L.type = L.array.type.elem;\ L.addr = \mathbf{new}\ Temp\ ();\ gen(L.addr\ '='\ E.addr\ '*'\ L.type.width);\ \} \\
\quad \mid\ L_1\ [\ E\ ] & \{\ L.array = L_1.array;\ L.type = L_1.type.elem;\ t = \mathbf{new}\ Temp\ ();\ L.addr = \mathbf{new}\ Temp\ ();\ gen(t\ '='\ E.addr\ '*'\ L.type.width);\ gen(L.addr\ '='\ L_1.addr\ '+'\ t);\ \}
\end{array}
$$

图 4. 处理数组引用的语义动作

问题 7. (原书6.4.3, 薄书6.4.3)使用图4所示的翻译方案翻译下列赋值语句

$$x = a[i][j] + b[i][j]$$

假定:

1. 一个整数的宽度是4
2. 假定a和b均为$n \times 3$的整数数组，即$a[i]$与$b[i]$的宽度均为$3 \times 4 = 12$，注意到n的值不重要

三地址代码：

$t_1 = i \times 12$

$t_2 = j \times 4$

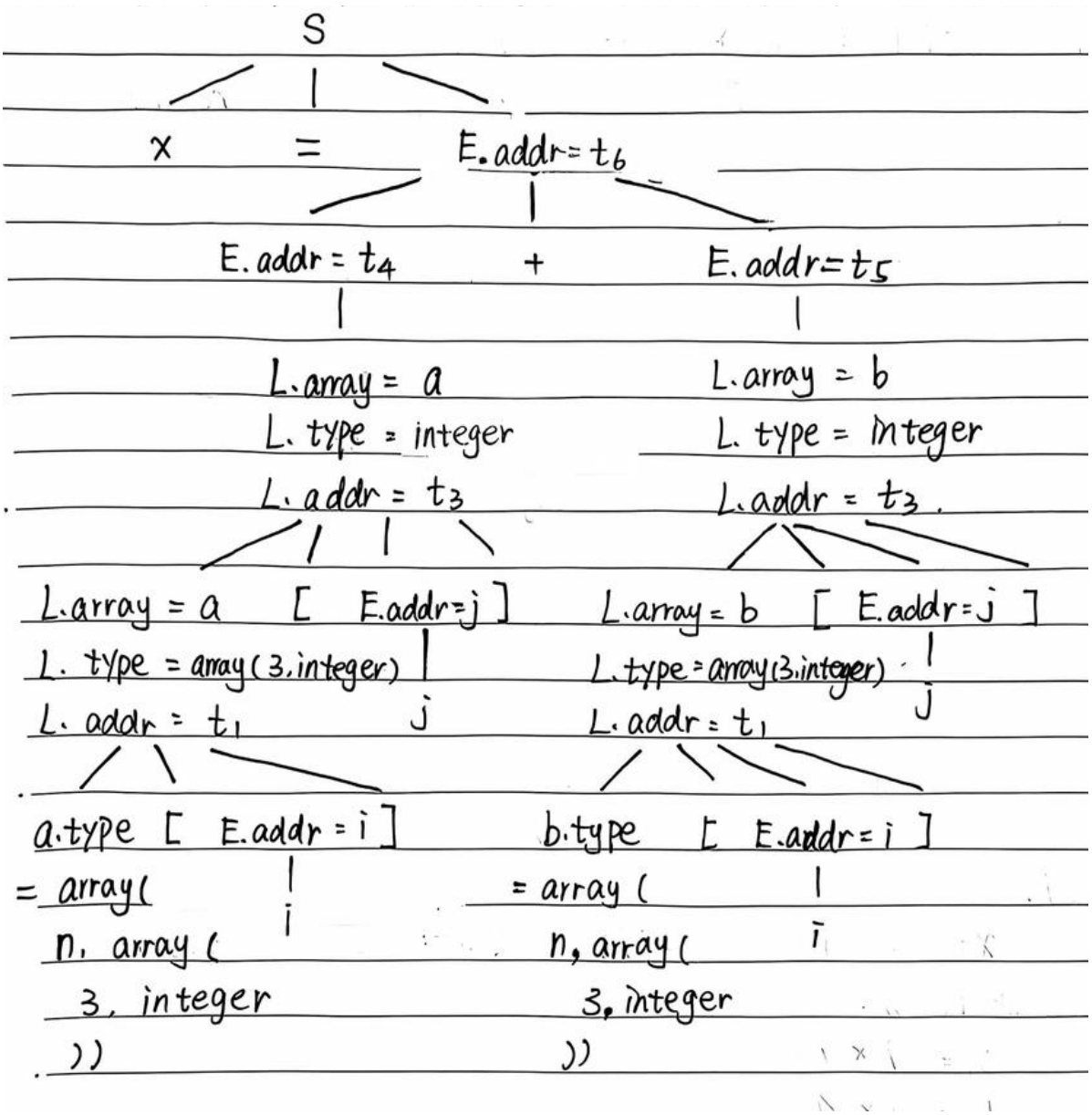
$t_3 = t_1 + t_2$

$t_4 = a[t_3]$

$t_5 = b[t_3]$

$t_6 = t_4 + t_5$

$x = t_6$



问题 8. (原书6.4.8, 薄书6.4.8)一个实数型数组$A[i,j,k]$的下标范围为$1 \le i \le 4, 0 \le j \le 4, 5 \le k \le 10$。假定每个实数占8个字节并且数组$A$从第0字节开始存放，计算下列元素的位置

1. $A[3,4,5]$
2. $A[1,2,7]$
3. $A[4,3,9]$

计算公式：$((i-1)*5*6+j*6+(k-5))*8$

1. $((3-1)*5*6+4*6+(5-5))*8=672$
2. $((1-1)*5*6+2*6+(7-5))*8=112$
3. $((4-1)*5*6+3*6+(9-5))*8=896$

1) $B \to B_1 \ \texttt{||}\ M\ B_2$ { $backpatch(B_1.falselist, M.instr)$;
$B.truelist = merge(B_1.truelist, B_2.truelist)$;
$B.falselist = B_2.falselist$; }

2) $B \to B_1 \ \texttt{\&\&}\ M\ B_2$ { $backpatch(B_1.truelist, M.instr)$;
$B.truelist = B_2.truelist$;
$B.falselist = merge(B_1.falselist, B_2.falselist)$; }

3) $B \to \ \texttt{!}\ B_1$ { $B.truelist = B_1.falselist$;
$B.falselist = B_1.truelist$; }

4) $B \to (\ B_1\ )$ { $B.truelist = B_1.truelist$;
$B.falselist\ =\ B_1.falselist$; }

5) $B \to E_1\ \textbf{rel}\ E_2$ { $B.truelist = makelist(nextinstr)$;
$B.falselist = makelist(nextinstr + 1)$;
$emit('\texttt{if}'\ E_1.addr\ \textbf{rel}.op\ E_2.addr\ '\texttt{goto}\ \_')$;
$emit('\texttt{goto}\ \_')$; }

6) $B \to \textbf{true}$ { $B.truelist = makelist(nextinstr)$;
$emit('\texttt{goto}\ \_')$; }

7) $B \to \textbf{false}$ { $B.falselist = makelist(nextinstr)$;
$emit('\texttt{goto}\ \_')$; }

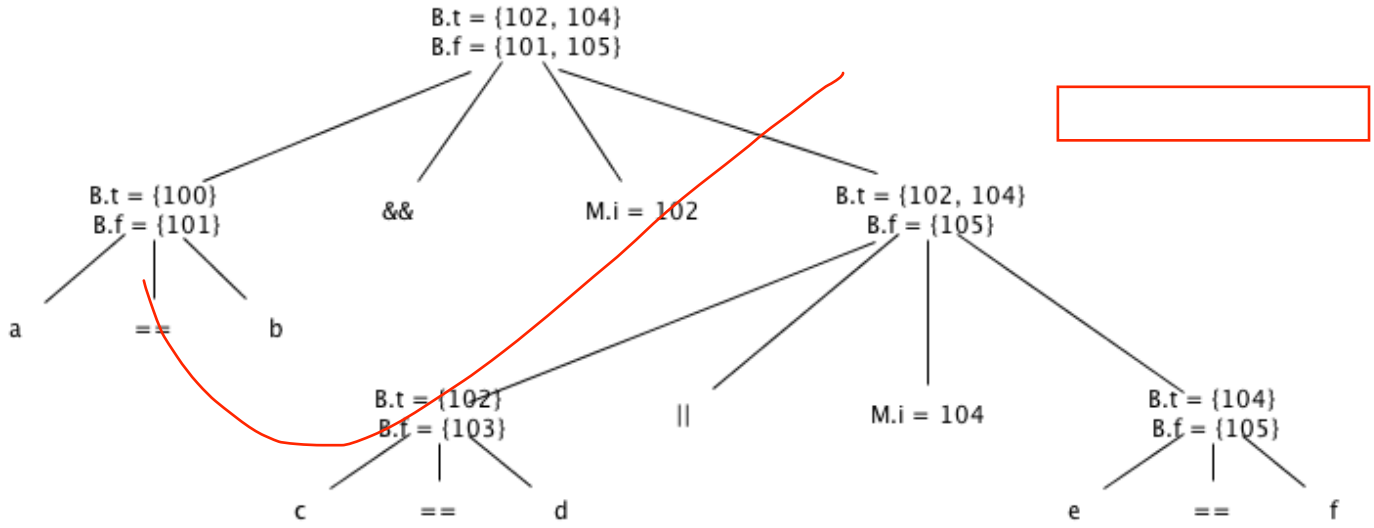
8) $M \to \epsilon$ { $M.instr = nextinstr$; }

图 5. 布尔表达式的翻译方案

问题 9. (原书6.7.1, 薄书6.7.1)使用图5的翻译方案翻译下列表达式并给出每个子表达式的真假值列表。假设第一条被生成的指令的地址为100

$$a==b\&\&(c==d||e==f)$$

> 为了简便，truelist -> t, falselis -> f, instr -> i.



1) $S \rightarrow \textbf{if}\,(\,B\,)\,M\,S_1$ { $backpatch(B.truelist,\ M.instr)$;
$S.nextlist\ =\ merge(B.falselist,\ S_1.nextlist)$; }

2) $S \rightarrow \textbf{if}\,(\,B\,)\,M_1\,S_1\,N\ \textbf{else}\ M_2\,S_2$
{ $backpatch(B.truelist,\ M_1.instr)$;
$backpatch(B.falselist,\ M_2.instr)$;
$temp\ =\ merge(S_1.nextlist,\ N.nextlist)$;
$S.nextlist\ =\ merge(temp,\ S_2.nextlist)$; }

3) $S \rightarrow \textbf{while}\ M_1\,(\,B\,)\,M_2\,S_1$
{ $backpatch(S_1.nextlist,\ M_1.instr)$;
$backpatch(B.truelist,\ M_2.instr)$;
$S.nextlist\ =\ B.falselist$;
$emit(\text{'goto'}\ M_1.instr)$; }

4) $S \rightarrow \{\,L\,\}$ { $S.nextlist\ =\ L.nextlist$; }

5) $S \rightarrow A\ ;$ { $S.nextlist\ =\ \textbf{null}$; }

6) $M \rightarrow \epsilon$ { $M.instr\ =\ nextinstr$; }

7) $N \rightarrow \epsilon$ { $N.nextlist\ =\ makelist(nextinstr)$;
$emit(\text{'goto\ \_'})$; }

8) $L \rightarrow L_1\ M\ S$ { $backpatch(L_1.nextlist,\ M.instr)$;
$L.nextlist\ =\ S.nextlist$; }

9) $L \rightarrow S$ { $L.nextlist\ =\ S.nextlist$; }

图 6. 语句的翻译方案

书-图 6-40

```
while (E1) {
        if (E2)
                while (E3)
                        S1;
        else {
                if (E4)
                        S2;
                S3
        }
}
```

图 7. 一个程序的控制流结构

问题 10. (原书6.7.3,薄书6.7.3) 当我们使用图6的规则翻译图7中的程序时，我们为每个语句S生成S.nextlist。除了图中说明的语句S_1, S_2, S_3之外，我们还有另外五个语句结构：

***S*4. while** (E_3) S_1

***S*5. if** (E_4) S_2

***S*6.** 由S_5和S_3组合的块

***S*7. if** (E_2) S_4 N **else** S_6

***S*8.** 整个程序

为每个块S_i构造S_i.next，你可以使用子语句S_j的nextlist、以及程序中任意表达式E_k的E_k.true和E_k.false。

a) S_4.next

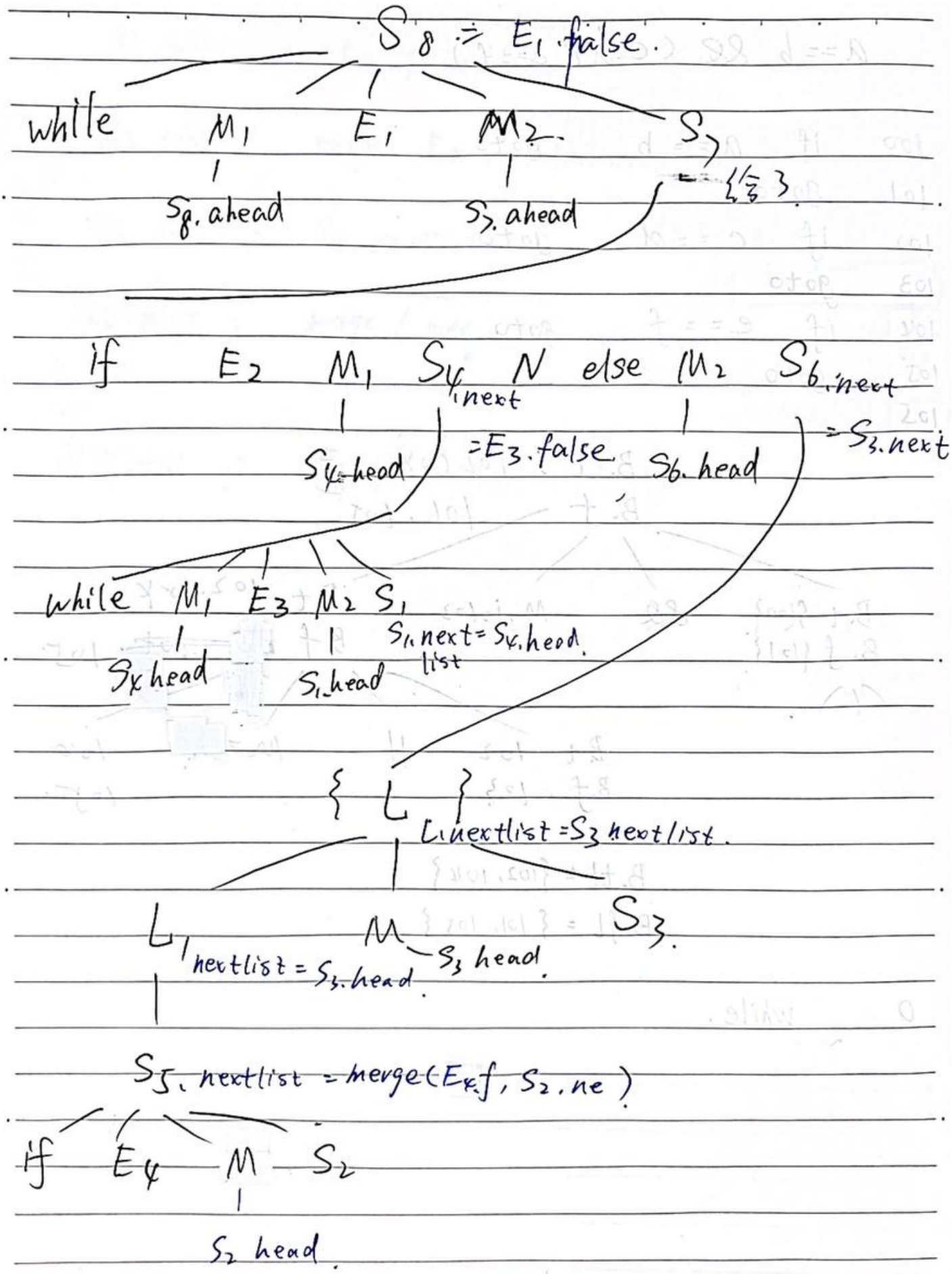
b) S_5.next

c) S_6.next

d) S_7.next

e) S_8.next

（提示：我们直接给出最麻烦的情况S_7的答案以供参考：S_7.next = merge(merge(S_4.nextlist, N.nextlist), S_6.nextlist)，因为它还依赖于一个隐含的节点N。这是个简单的习题，你的答案应该尽量与实际生成代码时的行为保持一致）



1. $S_4.\text{next} = E_3.\text{falselist}$
2. $S_5.\text{next} = \text{merge}(E_4.\text{falselist}, S_2.\text{nextlist})$

3. $S_6.\text{next} = S_3.\text{nextlist}$
4. $S_7.\text{next} = \text{merge}(\text{merge}(S_4.\text{nextlist}, N.\text{nextlist}), S_6.\text{nextlist})$
5. $S_8.\text{next} = E1.\text{falselist}$