编译原理习题课（二）

第五章

- 属性
  - 综合属性
  - 继承属性
- 属性文法
  - S属性文法
  - L属性文法
- 带注释的语法树

第五章

- 某属性文法包含下面的定义，该文法是L-属性文法吗

  A. 是

  B. 否

产生式	语义规则
A→LM	L.i := g(A.i)
	M.i :=m(L.s)
A→QR	R.i := r(A.i)
	Q.i :=q(R.s)
	A.s :=f(Q.s)

第五章

- 某属性文法包含下面的定义，该文法是L-属性文法吗

  A. 是

  B. 否

产生式	语义规则
A→LM	L.i := g(A.i)
	M.i :=m(L.s)
A→QR	R.i := r(A.i)
	Q.i :=q(R.s)
	A.s :=f(Q.s)

答案：B. 否

原因：Q的继承属性的计算依赖右侧符号R的属性

1. 扩展右图中的SDD，使他可以像左图所示那样处理表达式。

	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

	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*

1. 扩展右图中的SDD，使他可以像左图所示那样处理表达式。

	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

	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*

观察可以发现，左图的SDD只包含综合属性，右图的SDD包含继承属性和综合属性。
因此扩展即为将左图的+和*改为继承左运算分量的形式。
继承属性的语义规则（书例5.8）：

产生式	语义规则
$T \to FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \to *FT_1'$	$T_1.inh = T'.inh * F.val$ $T.syn = T_1.syn$

第五章

1. 扩展右图中的SDD，使他可以像左图所示那样处理表达式。

	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

	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*

	产生式	语法规则
1)	$L \to En$	$L.val = E.val$
2)	$E \to TE'$	$E'.inh = T.val$ $E.val = E'.syn$
3)	$E' \to +TE_1'$	$E_1.inh = E'.inh + T.val$ $E.syn = E_1.syn$
4)	$E' \to \varepsilon$	$E'.syn = E'.inh$
5)	$T \to FT'$	$T'.inh = F.val$ $T.val = T'.syn$
6)	$T' \to *FT_1'$	$T_1.inh = T'.inh * F.val$ $T.syn = T_1.syn$
7)	$T' \to \varepsilon$	$T'.syn = T'.inh$
8)	$F \to (E)$	$F.val = E.val$
9)	$F \to digit$	$F.val = digit.lexval$

2. 对于图中的SDD，给出int x, y, z对应的注释语法分析树。

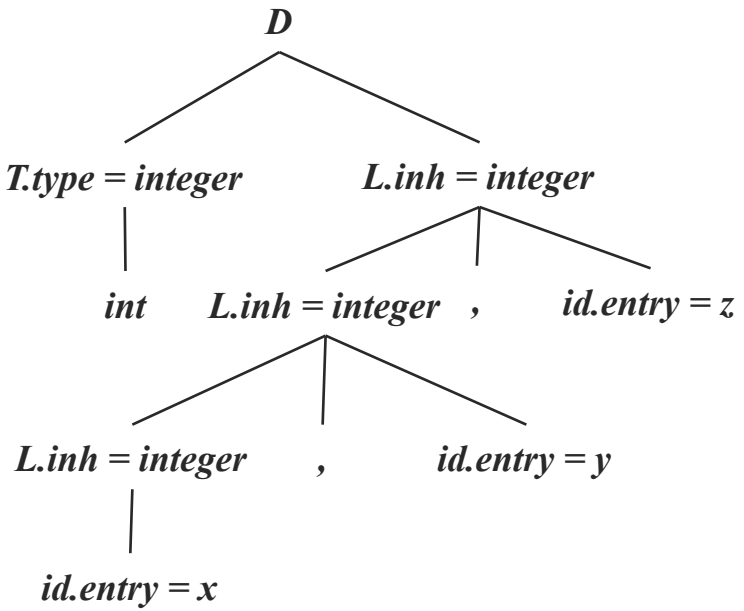
	产生式	语义规则
1)	$D \rightarrow T\ L$	$L.inh = T.type$
2)	$T \rightarrow \mathbf{int}$	$T.type = \text{integer}$
3)	$T \rightarrow \mathbf{float}$	$T.type = \text{float}$
4)	$L \rightarrow L_1\ ,\ \mathbf{id}$	$L_1.inh = L.inh$ $addType(\mathbf{id}.entry, L.inh)$
5)	$L \rightarrow \mathbf{id}$	$addType(\mathbf{id}.entry,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

2. 对于图中的SDD，给出int x, y, z对应的注释语法分析树。

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

L属性定义的SDD，依赖图的边一定是从左到右的！

（书图5-9）的简单改写：



副作用为哑属性，不出现在树中

3. 图中的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做按照正二进制数来解释。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.val的值:

$$
\begin{array}{lcl}
B & \rightarrow & B_1\ 0\ \{B.val = 2 \times B_1.val\} \\
 & | & B_1\ 1\ \{B.val = 2 \times B_1.val + 1\} \\
 & | & 1\ \{B.val = 1\}
\end{array}
$$

3. 图中的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做按照正二进制数来解释。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.val的值:

$$
\begin{array}{lll}
B & \rightarrow & B_1\ 0\ \{B.val = 2 \times B_1.val\} \\
 & | & B_1\ 1\ \{B.val = 2 \times B_1.val + 1\} \\
 & | & 1\ \{B.val = 1\}
\end{array}
$$

SDT消除左递归（书5.4.4节）：

如果不涉及属性值计算，将动作看作终结符进行处理；

如果涉及属性值计算，则通用解决方案为：

产生式	语义规则
$T \rightarrow FT'$	$T'.inh = F.val$ $T.val = T'.syn$
$T' \rightarrow *FT_1'$	$T_1.inh = T'.inh * F.val$ $T.syn = T_1.syn$

- 假设
  - $A \rightarrow A_1 Y \{ A.a = g(A_1.a, Y.y) \}$
  - $A \rightarrow X \{ A.a = f(X.x) \}$
- 那么
  - $A \rightarrow X \{ R.i = f(X.x) \}\ R \{ A.a = R.s \}$
  - $R \rightarrow Y \{ R_1.i = g(R.i, Y.y) \}\ R_1 \{ R.s = R_1.s \}$
  - $R \rightarrow \varepsilon \{ R.s = R.i \}$

3. 图中的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做按照正二进制数来解释。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.val的值:

$$
\begin{array}{lll}
B & \rightarrow & B_1\ 0\ \{B.val = 2 \times B_1.val\} \\
 & | & B_1\ 1\ \{B.val = 2 \times B_1.val + 1\} \\
 & | & 1\ \{B.val = 1\}
\end{array}
$$

基础文法为：$B \rightarrow B_1\,0 \mid B_1\,1 \mid 1$

不提取左公因子:

消除左递归：$B \rightarrow 1\ A$

$A \rightarrow 0\ A_1 \mid 1\ A_1 \mid \varepsilon$

改写后的SDT：$B \rightarrow 1\ \{A.i = 1\}\ A\ \{B.val = A.val\}$

$A \rightarrow 0\ \{A_1.i = 2 \times A.i\}\ A_1\ \{A.val = A_1.val\}$

$\mid 1\ \{A_1.i = 2 \times A.i + 1\}\ A_1\ \{A.val = A_1.val\}$

$\mid \varepsilon\ \{A.val = A.i\}$

- 假设
  - $A \rightarrow A_1 Y\ \{\ A.a = g(A_1.a,\ Y.y)\ \}$
  - $A \rightarrow X\ \{\ A.a = f(X.x)\ \}$
- 那么
  - $A \rightarrow X\ \{\ R.i = f(X.x)\ \}\ R\ \{\ A.a = R.s\ \}$
  - $R \rightarrow Y\ \{\ R_1.i = g(R.i,\ Y.y)\ \}\ R_1\ \{\ R.s = R_1.s\ \}$
  - $R \rightarrow \varepsilon\ \{\ R.s = R.i\ \}$

3. 图中的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做按照正二进制数来解释。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.val的值:

$$
\begin{array}{lll}
B & \rightarrow & B_1\ 0\ \{B.val = 2 \times B_1.val\} \\
 & | & B_1\ 1\ \{B.val = 2 \times B_1.val + 1\} \\
 & | & 1\ \{B.val = 1\}
\end{array}
$$

基础文法为：$B \rightarrow B_1\,0 \mid B_1\,1 \mid 1$

提取左公因子的SDT:

$B \rightarrow B_1$ digit {B.val = $2 \times B_1$.val + digit}
　　| 1 {B.val = 1}
digit → 0 {digit.val = 0}
　　| 1 {digit.val = 1}

消除左递归的SDT:

$B \rightarrow 1$ {A.i = 1} A {B.val = A.val}
$A \rightarrow$ digit {A_1.i = $2 \times$A.i + digit} A_1 {A.val = A_1.val} | ε {A.val = A.i}
digit → 0 {digit.val = 0} | 1 {digit.val = 1}

- 假设
  - $A \rightarrow A_1 Y\ \{ A.a = g(A_1.a, Y.y) \}$
  - $A \rightarrow X\ \{ A.a = f(X.x) \}$
- 那么
  - $A \rightarrow X\ \{ R.i = f(X.x) \}\ R\ \{ A.a = R.s \}$
  - $R \rightarrow Y\ \{ R_1.i = g(R.i, Y.y) \}\ R_1\ \{ R.s = R_1.s \}$
  - $R \rightarrow \varepsilon\ \{ R.s = R.i \}$

第六章

- 中间代码表示
  - 抽象语法树
  - 三地址代码
- 中间代码生成
  - 表达式
  - 类型检查
  - 控制流

- 设AS 为文法的综合属性集，AI 为继承属性集，则对于下面的属性文法G(P)定义中，AS和AI正确描述是

A. AS={ Q.a, Q.b } AI={ R.c, R.d, R.e, R.f }
B. AS={ Q.a, R.d, R.f } AI={ Q.b, R.c, R.e }
C. AS={ Q.b, R.c, R.f } AI={ Q.a, R.d, R.e }
D. AS={ Q.a, R.c, R.e } AI={ Q.b, R.d, R.f }

body变量位于SDD的左值一定为继承属性，
head变量位于SDD左值一定为综合属性

产生式	语义规则
P → xQR	Q.b:=R.d
	R.c:=1
	R.e:=Q.a
Q → u	Q.a:=3
R → v	R.d:=R.c
	R.f:=R.e

第六章

- 设AS 为文法的综合属性集，AI 为继承属性集，则对于下面的属性文法G(P)定义中，AS和AI正确描述是

原因：从 P → xQR 的语义规则 Q.b:=R.d, R.c:=1, R.e:=Q.a 可得Q.b, R.c和R.e 为继承属性，而R.d, Q.a的性质则需要其他产生式的语义规则一起加以分析才能确定;

从Q → u的语义规则Q.a:=3

可得 Q.a为综合属性

…

产 生 式	语 义 规 则
P → xQR	Q.b:=R.d
	R.c:=1
	R.e:=Q.a
Q → u	Q.a:=3
R → v	R.d:=R.c
	R.f:=R.e

答案：B. AS={ Q.a, R.d, R.f }

AI={ Q.b, R.c, R.e }

- (a+b)/(c-d)对应的逆波兰式(后缀式)是

- (a+b)/(c-d)对应的逆波兰式(后缀式)是

答案：ab+cd-/

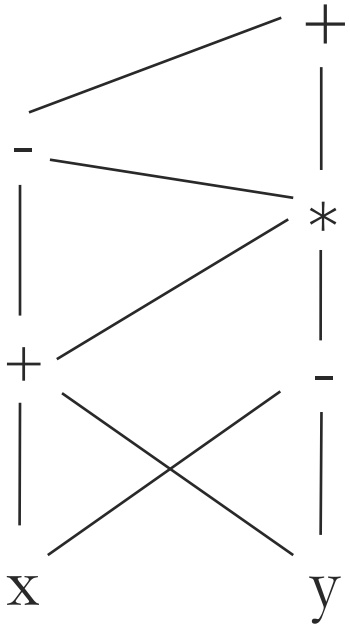
第六章

1. 为下面的表达式构造DAG:

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

1. 为下面的表达式构造DAG:

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



书6.1.1节
表达式的有向无环图

2. 将下列将下列赋值语句翻译为四元式序列，三元式序列，间接三元式序列:

（1）a = b[i] + c[j]

（2）a[i] = b * c - b * d

2. 将下列将下列赋值语句翻译为四元式序列，三元式序列，间接三元式序列:

（1）a = b[i] + c[j]

带下标的复制指令:
书222页（8）

四元式				
0)	=[]	b	i	t1
1)	=[]	c	j	t2
2)	+	t1	t2	t3
3)	=	t3		a

三元式			
0)	=[]	b	i
1)	=[]	c	j
2)	+	(0)	(1)
3)	=	a	(2)

	间接三元式（假设地址为100）			
	0)	=[]	b	i
	1)	=[]	c	j
	2)	+	(0)	(1)
	3)	=	a	(2)
100	0)			
101	1)			
102	2)			
103	3)			

2. 将下列将下列赋值语句翻译为四元式序列，三元式序列，间接三元式序列:

（2）a[i] = b * c - b * d

四元式				
0)	*	b	c	t1
1)	*	b	d	t2
2)	-	t1	t2	t3
3)	[]	a	i	t4
4)	=	t3		t4

三元式			
0)	*	b	c
1)	*	b	d
2)	-	(0)	(1)
3)	[]	a	i
4)	=	(3)	(2)

	间接三元式（假设地址为100）			
	0)	*	b	c
	1)	*	b	d
	2)	-	(0)	(1)
	3)	[]	a	i
	4)	=	(3)	(2)
100	0)			
101	1)			
102	2)			
103	3)			
104	4)			

3.使用下图所示的翻译方案来翻译赋值语句 x = a[i][j] + b[i][j]。

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

图 6-22　处理数组引用的语义动作

3.使用下图所示的翻译方案来翻译赋值语句 x = a[i][j] + b[i][j]。

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

图 6-22　处理数组引用的语义动作

目标：为带数组引用的表达式生成三地址代码

关键字解释：

E.addr：E存放值的地址

gen：构造一条新的三地址指令，添加至指令序列之后

top：当前符号表

非终结符号L的三个综合属性：

L.addr：临时变量，计算数组引用的偏移量

L.array：指向数组名字的符号表的指针（L.array.base代表基地址）

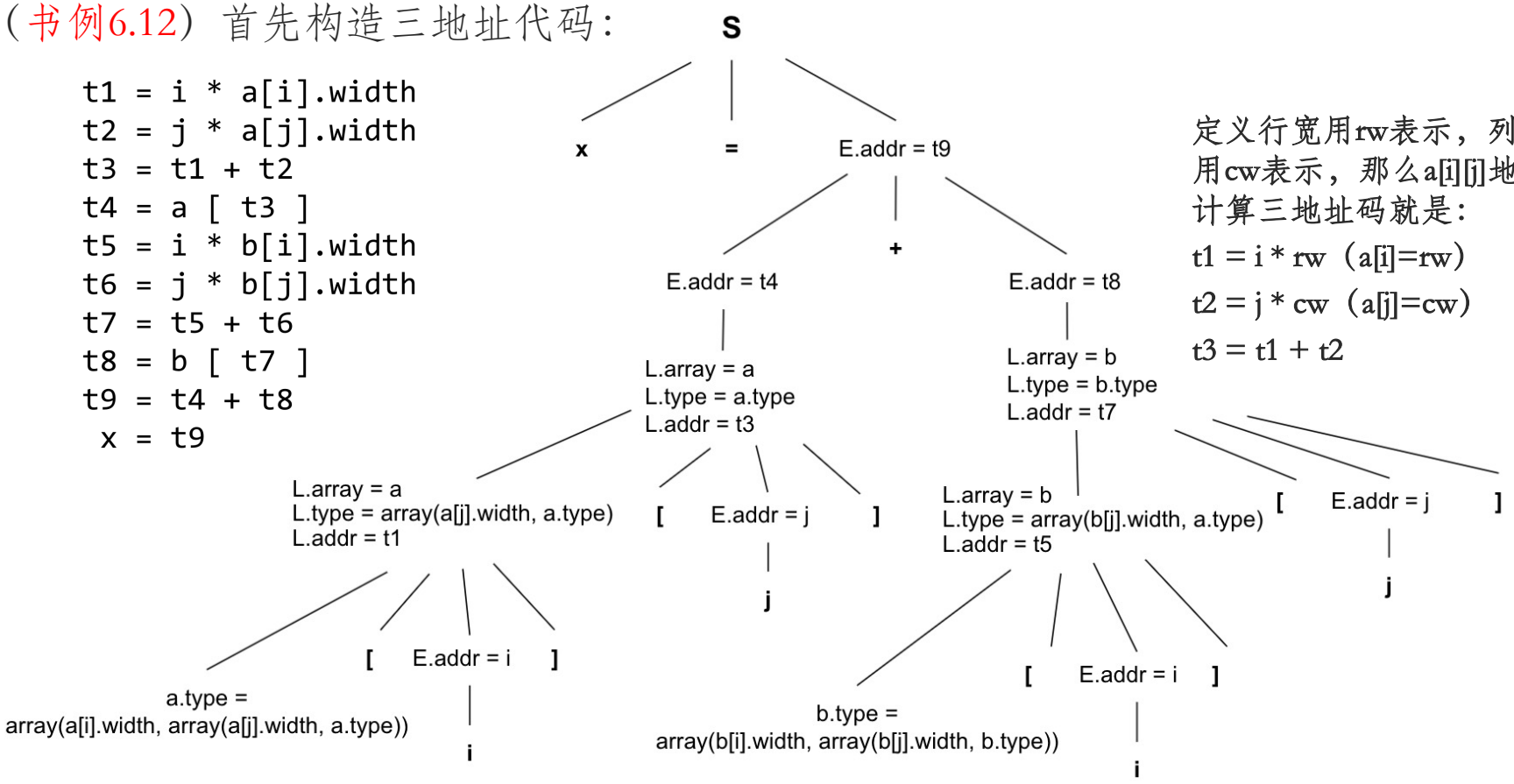
L.type：L生成子数组的类型

第六章

3.使用下图所示的翻译方案来翻译赋值语句 x = a[i][j] + b[i][j]。

（书例6.12）首先构造三地址代码：

```
t1 = i * a[i].width
t2 = j * a[j].width
t3 = t1 + t2
t4 = a [ t3 ]
t5 = i * b[i].width
t6 = j * b[j].width
t7 = t5 + t6
t8 = b [ t7 ]
t9 = t4 + t8
 x = t9
```



定义行宽用rw表示，列宽用cw表示，那么a[i][j]地址计算三地址码就是：

t1 = i * rw （a[i]=rw）

t2 = j * cw （a[j]=cw）

t3 = t1 + t2

注释语法分析树

4. 一个按行存放的实数型数组 A[i, j, k] 的下标 i 的范围为 1~4，下标 j 的范围为 0~4，且下标 k 的范围为 5~10。每个实数占 8 个字节。假设数组 A 从 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]

4. 一个按行存放的实数型数组 A[i, j, k] 的下标 i 的范围为 1~4，下标 j 的范围为 0~4，且下标 k 的范围为 5~10。每个实数占 8 个字节。假设数组 A 从 0 字节开始存放，计算下列元素的位置:

（1）A[3, 4, 5]

（书6.4.3）数组在A[i]上的宽度为5*6，在A[i, j]上的宽度为6，根据公式6.3和6.7，元素位置为： ((3 - 1)*5*6 + (4 - 0)*6 + (5 - 5)) *8 = 672

（2）A[1, 2, 7]

元素位置为： ((1 - 1)*5*6 + (2 - 0)*6 + (7 - 5)) *8 = 112

（3）A[4, 3, 9]

元素位置为： ((4 - 1)*5*6 + (3 - 0)*6 + (9 - 5)) *8 = 896

5. 使用下图中的翻译方案翻译表达式a==b && (c==d||e==f)，并给出每个子表达式的真值列表与假值列表，你可以假设第一条被生成的指令的地址是100:

1)	$B \rightarrow B_1$ || M B_2	{ *backpatch*(B_1.*falselist*, M.*instr*); B.*truelist* = *merge*(B_1.*truelist*, B_2.*truelist*); B.*falselist* = B_2.*falselist*; }
2)	$B \rightarrow B_1$ && M B_2	{ *backpatch*(B_1.*truelist*, M.*instr*); B.*truelist* = B_2.*truelist*; B.*falselist* = *merge*(B_1.*falselist*, B_2.*falselist*); }
3)	$B \rightarrow$! B_1	{ B.*truelist* = B_1.*falselist*; B.*falselist* = B_1.*truelist*; }
4)	$B \rightarrow$ (B_1)	{ B.*truelist* = B_1.*truelist*; B.*falselist* = B_1.*falselist*; }
5)	$B \rightarrow E_1$ **rel** E_2	{ B.*truelist* = *makelist*(*nextinstr*); B.*falselist* = *makelist*(*nextinstr* + 1); *gen*('`if` E_1.*addr* **rel**.*op* E_2.*addr* '`goto` _'); *gen*('`goto` _'); }
6)	$B \rightarrow$ **true**	{ B.*truelist* = *makelist*(*nextinstr*); *gen*('`goto` _'); }
7)	$B \rightarrow$ **false**	{ B.*falselist* = *makelist*(*nextinstr*); *gen*('`goto` _'); }
8)	$M \rightarrow \epsilon$	{ M.*instr* = *nextinstr*; }

图 6-43　布尔表达式的翻译方案

5. 使用下图中的翻译方案翻译表达式a==b && (c==d||e==f)，并给出每个子表达式的真值列表与假值列表，你可以假设第一条被生成的指令的地址是100:

1)	$B \rightarrow B_1$ || M B_2	{ *backpatch*(B_1.*falselist*, M.*instr*); B.*truelist* = *merge*(B_1.*truelist*, B_2.*truelist*); B.*falselist* = B_2.*falselist*; }
2)	$B \rightarrow B_1$ && M B_2	{ *backpatch*(B_1.*truelist*, M.*instr*); B.*truelist* = B_2.*truelist*; B.*falselist* = *merge*(B_1.*falselist*, B_2.*falselist*); }
3)	$B \rightarrow$! B_1	{ B.*truelist* = B_1.*falselist*; B.*falselist* = B_1.*truelist*; }
4)	$B \rightarrow$ (B_1)	{ B.*truelist* = B_1.*truelist*; B.*falselist* = B_1.*falselist*; }
5)	$B \rightarrow E_1$ **rel** E_2	{ B.*truelist* = *makelist*(*nextinstr*); B.*falselist* = *makelist*(*nextinstr* + 1); *gen*('if' E_1.*addr* **rel**.*op* E_2.*addr* 'goto _'); *gen*('goto _'); }
6)	$B \rightarrow$ **true**	{ B.*truelist* = *makelist*(*nextinstr*); *gen*('goto _'); }
7)	$B \rightarrow$ **false**	{ B.*falselist* = *makelist*(*nextinstr*); *gen*('goto _'); }
8)	$M \rightarrow \epsilon$	{ M.*instr* = *nextinstr*; }

图 6-43　布尔表达式的翻译方案

目标：为布尔表达式生成三地址代码
方法：回填完成一趟式目标代码生成

B有两个综合属性：

两个list都是包含跳转指令的列表

B.truelist：B为真时控制流应该转向的标号

B.falselist：B为假时控制流应该转向的标号

构建这个列表需要三个函数：

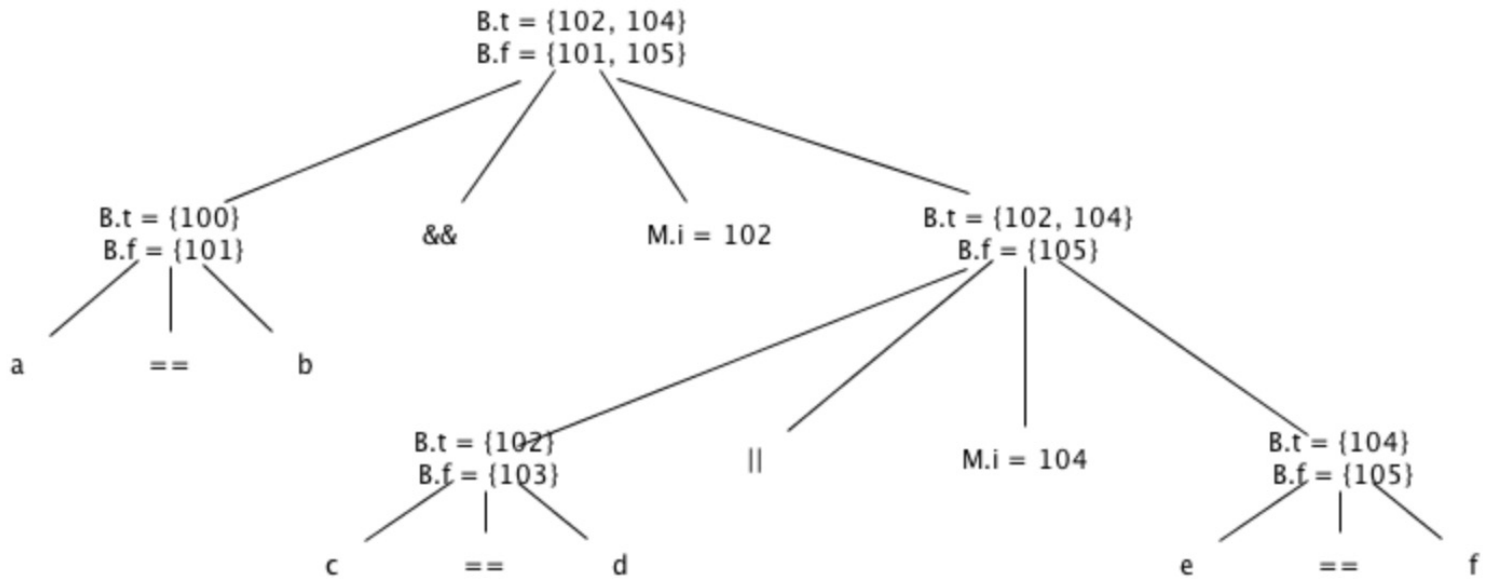
(1)makelist(i): 创建一个只包含指令数组下标i的列表，返回一个新列表的指针

(2)merge(p1, p2): 将p1和p2指向的列表进行合并，返回合并列表的指针

(3)backpatch(p, i): 将i作为目标标号插入p列表

5. 使用下图中的翻译方案翻译表达式a==b && (c==d||e==f)，并给出每个子表达式的真值列表与假值列表，你可以假设第一条被生成的指令的地址是100:

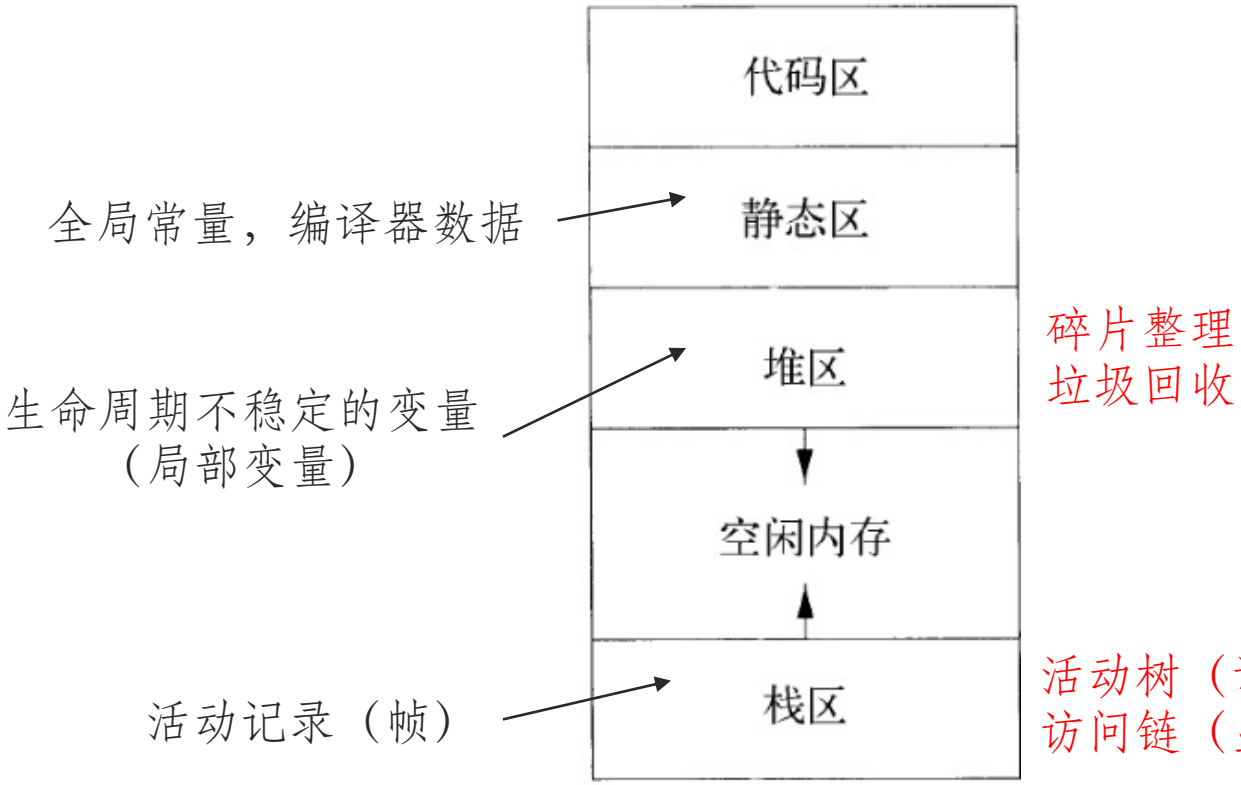
书6.7.2 节 例6.17



第七章

- 运行时刻环境
  - 为数据分配安排存储位置
  - 确定访问变量时使用的机制
  - 过程之间的连接
  - 参数传递
  - 和操作系统、输入输出设备相关的其它接口
- 主题
  - 存储管理：栈分配、堆管理、垃圾回收
  - 对变量、数据的访问

第七章


代码区
静态区
堆区
空闲内存
栈区
全局常量，编译器数据
生命周期不稳定的变量
（局部变量）
活动记录（帧）
碎片整理
垃圾回收
活动树（调用/返回序列）
访问链（显示表）
栈区在7，8章中地址是从低到高
增长的，与实际相反

1. 在一个通过引用传递参数的语言中，有一个函数f(x, y)完成下面的计算：

   x = x+1; y = y+2; return x+y;

   如果将a赋值为3，然后调用f(a, a)，那么返回值是什么？

1. 在一个通过引用传递参数的语言中，有一个函数f(x, y)完成下面的计算：

   x = x+1; y = y+2; return x+y;

   如果将a赋值为3，然后调用f(a, a)，那么返回值是什么？

引用传递，就是传递地址，调用f(a, a)会改变地址上a的值

x = x+1, 会把形参x地址上的值改为4，实参a地址上的值也改为4

y = y+2, 会把形参y地址上的值改为6，实参a地址上的值也改为6

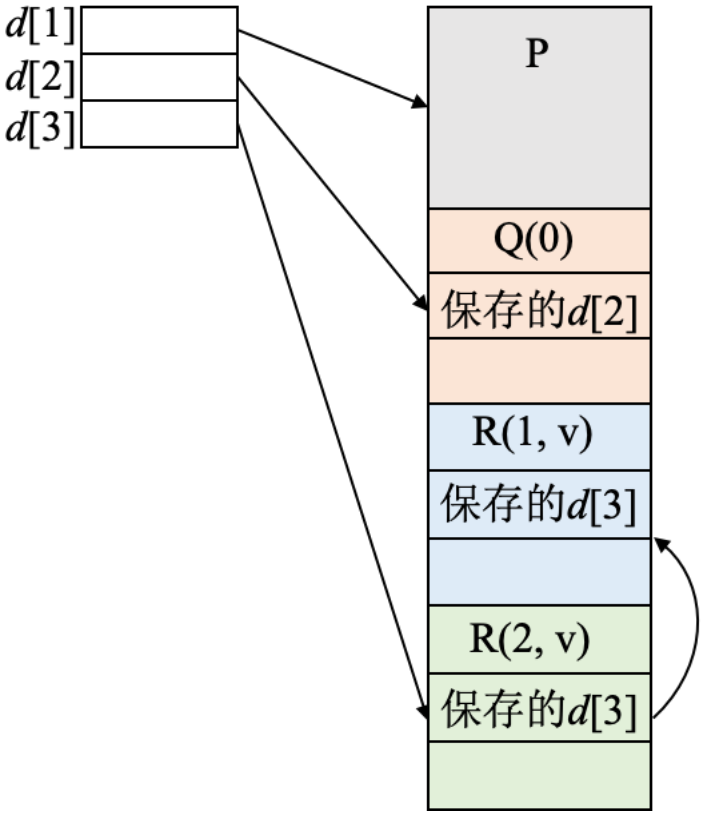
return x+y，返回值是形参x和形参y地址上的值求和，即实参a和实参a地址上的值求和，即12

2. 考虑下面的类PASCAL的嵌套过程语言程序，对于调用过程：主程序→过程Q→过程R→过程R，过程R的第2次调用的活动记录中的显示表是什么？

```
program P;
    var a, x : integer;
    procedure S;
        var c, i:integer;
    begin
        a:=1;
        ......
    end {S}
    procedure Q(b: integer);
        var i: integer;
        procedure R(u: integer; var v: integer);
          var c, d: integer;
        begin
          if u=1 then R(u+1, v)
          ......
          v:=(a+c)*(b-d);
          ......
        end {R}
    begin
        ......
        R(1, x);
        ......
    end {Q}
begin
    a:=0;
    Q(a);
    ......
end. {P}
```

2. 考虑下面的类PASCAL的嵌套过程语言程序，对于调用过程：主程序→过程Q→过程R→过程R，过程R的第2次调用的活动记录中的显示表是什么？

```
program P;
    var a, x : integer;
    procedure S;
        var c, i:integer;
    begin
        a:=1;
        ......
    end {S}
    procedure Q(b: integer);
        var i: integer;
        procedure R(u: integer; var v: integer);
          var c, d: integer;
        begin
          if u=1 then R(u+1, v)
          ......
          v:=(a+c)*(b-d);
          ......
        end {R}
    begin
        ......
        R(1, x);
        ......
    end {Q}
begin
    a:=0;
    Q(a);
    ......
end. {P}
```



3. 当下列事件发生时，图7-19中的对象的引用计数会发生哪些变化？

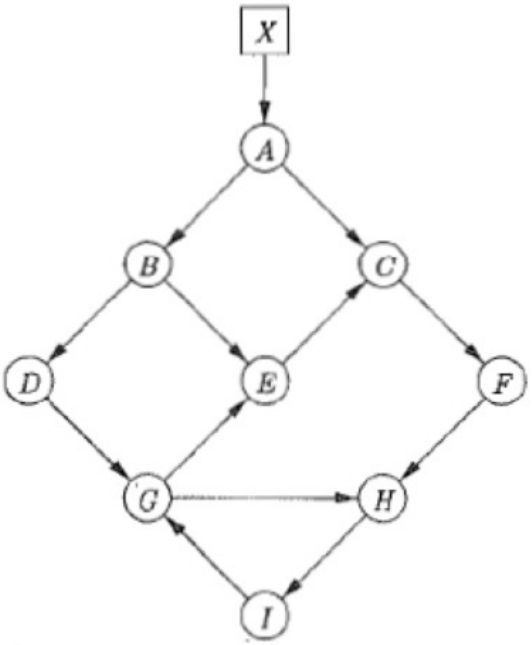


图 7-19　一个对象网络

（1）从A指向B的指针被删除。

（2）从X指向A的指针被删除。

（3）结点C被删除。

3. 当下列事件发生时，图7-19中的对象的引用计数会发生哪些变化？

（1）从A指向B的指针被删除。

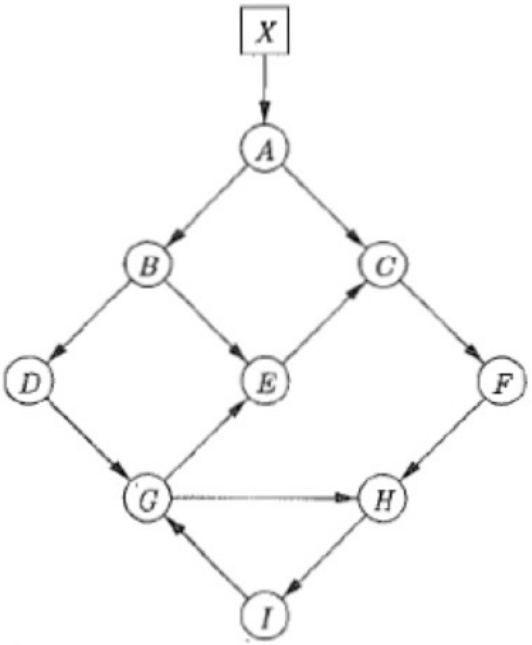
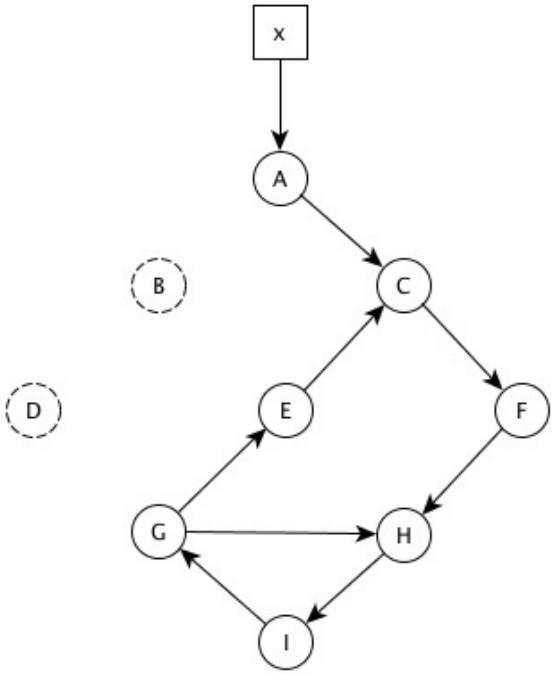


图 7-19　一个对象网络



3. 当下列事件发生时，图7-19中的对象的引用计数会发生哪些变化？

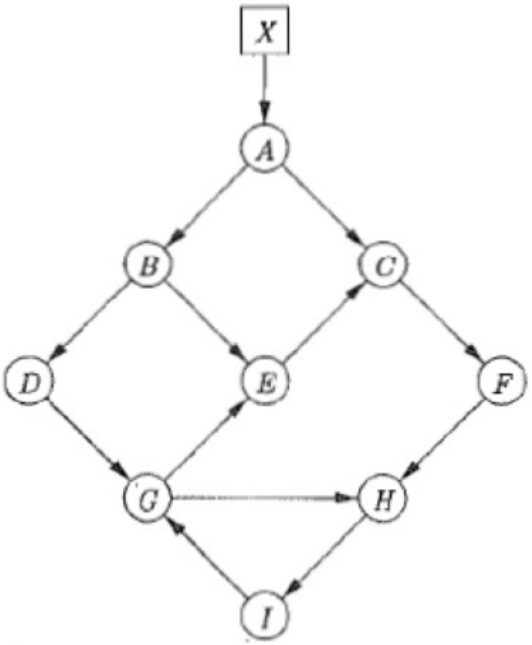
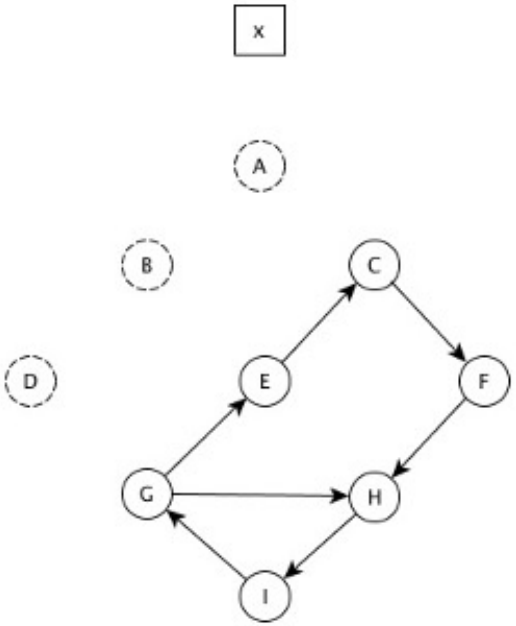


图 7-19　一个对象网络

（2）从X指向A的指针被删除。



3. 当下列事件发生时，图7-19中的对象的引用计数会发生哪些变化？

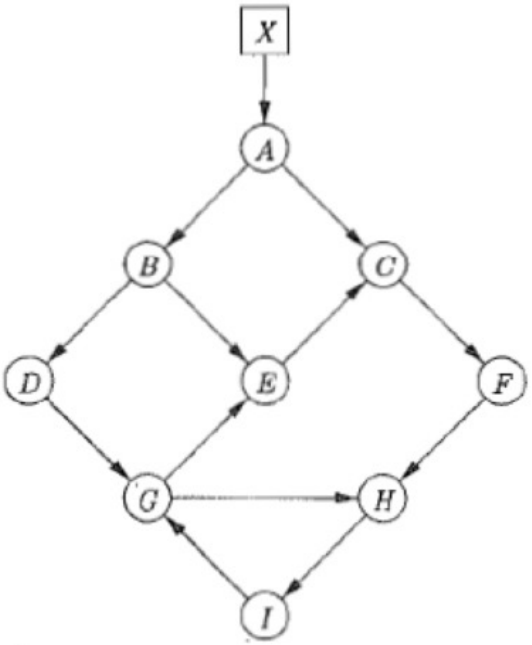
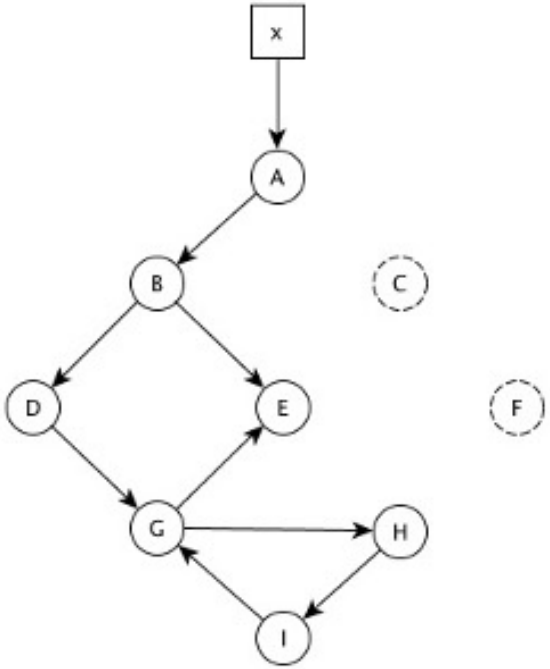


图 7-19　一个对象网络

（3）结点C被删除。



4. 对于题4中的图7-19，当下列事件发生时，给出标记-清扫式垃圾回收器的处理步骤。

（1）当指针A→B被删除。（2）当指针A→C被删除。

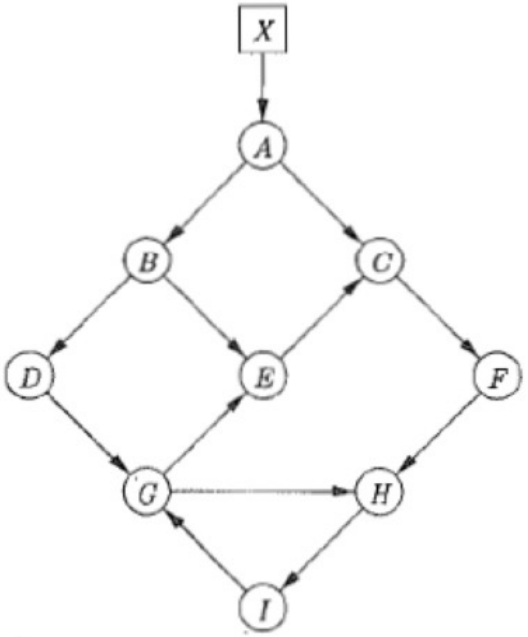


图 7-19 一个对象网络

4. 对于题4中的图7-19，当下列事件发生时，给出标记-清扫式垃圾回收器的处理步骤。

（1）当指针A→B被删除。

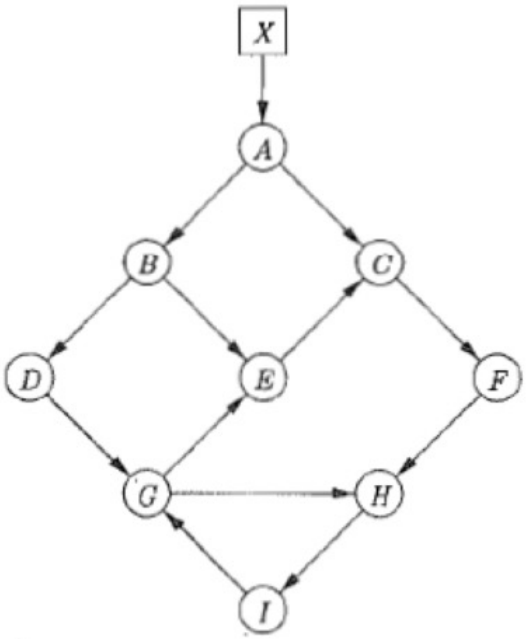


图 7-19　一个对象网络

```
    /* 标记阶段 */
1)  把被根集引用的每个对象的 reached 位设置为 1，并把它加入
      到 Unscanned 列表中；
2)  while (Unscanned ≠ ∅) {
3)        从 Unscanned 列表中删除某个对象 o；
4)        for (在 o 中引用的每个对象 o′) {
5)              if (o′ 尚未被访问到；即它的 reached 位为 0) {
6)                    将 o′ 的 reached 位设置为 1；
7)                    将 o′ 放到 Unscanned 中；
                  }
          }
    }
    /* 清扫阶段 */
8)  Free = ∅;
9)  for (堆区中的每个内存块 o) {
10)       if (o 未被访问到，即它的 reached 位为 0) 将 o 加入到 Free 中；
11)       else 将 o 的 reached 位设置为 0；
    }
```

因为语言是强类型的，所以垃圾回收机制可以知道每个数据对象的类型，以及这个对象有哪些字段是指针

4. 对于题4中的图7-19，当下列事件发生时，给出标记-清扫式垃圾回收器的处理步骤。

（1）当指针A→B被删除。

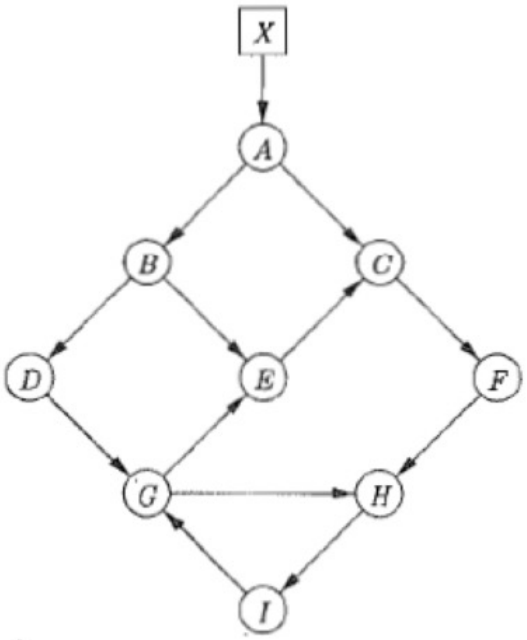


图 7-19　一个对象网络

开始时：根集是{A}，A.reached = 1

Unscanned	对象	可到达对象	动作
A	A	C	C.reached=1，C加入Unscanned
C	C	F	F.reached=1，F加入Unscanned
F	F	H	H.reached=1，H加入Unscanned
H	I	I	I.reached=1，I加入Unscanned
I	G	G	G.reached=1，G加入Unscanned
G	E	E	E.reached=1，E加入Unscanned
E	E	C	C被扫描过了，无操作
Free = {B, D}，其他对象reached被标为0，满足下次回收前置			

4. 对于题4中的图7-19，当下列事件发生时，给出标记-清扫式垃圾回收器的处理步骤。

（2）当指针A→C被删除。

开始时：根集是{A}，A.reached = 1

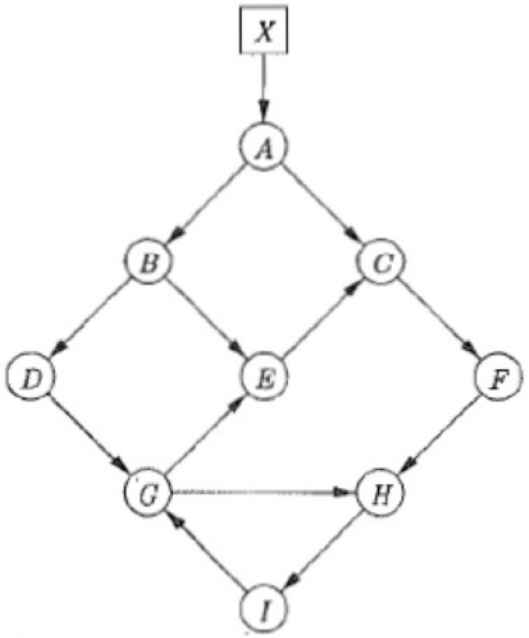


图 7-19　一个对象网络

Unscanned	对象	可到达对象	动作
A	A	B	B.reached=1，B加入Unscanned
B	B	D，E	D.reached=1，D加入Unscanned E.reached=1，E加入Unscanned
D，E	D	G	G.reached=1，G加入Unscanned
E，G	E	C	C.reached=1，C加入Unscanned
C，G	C	F	F.reached=1，F加入Unscanned
F，G	F	H	H.reached=1，H加入Unscanned
H，G	H	I	I.reached=1，I加入Unscanned
I，G	I	G	G在Unscanned中，无操作
G	G	H	H已被扫描，无操作
Free = {∅}，其他对象reached被标为0，满足下次回收前置			

运行时刻（Python）

Python 3.6
(known limitations)

```
4
5  class Staff601:
6      course = '6.01'
7      building = 34
8      room = 501
9
10     def giveRaise(self, percentage):
11         self.salary = self.salary + self.salary * per
12
13 class Prof601(Staff601):
14     salary = 100000
15
16     def __init__(self, name, age):
17         self.name = name
18         self.giveRaise((age - 18) * 0.03)
19
20     def salutation(self):
21         return self.role + ' ' + self.name
22
23 pat = Prof601('Pat', 60)
```

Frames Objects

Global frame
- Staff601
- Prof601
- pat

Staff601 class

building	34
course	"6.01"
giveRaise	function giveRaise(self, percentage)
room	501

Prof601 class [extends Staff601]

__init__	function __init__(self, name, age)
salary	100000
salutation	function salutation(self)

Prof601 instance

name	"Pat"
salary	226000.0

演示网址：https://pythontutor.com

随堂测试

- 表达式(a+b)/c-(a+b)*d对应的间接三元式表示如下，其中三元式表中第(3)号三元式应为_____

间接码表	三元式表			
		OP	ARG1	ARG2
(1)	(1)	+	a	b
(2)	(2)	/	(1)	c
(1)	(3)			
(3)				
(4)	(4)	-	(2)	(3)

随堂测试

- 考虑下面的属性文法G(S)，对于输入字符串abc进行自下而上的语法分析和属性计算，设S.u的初始值为5，属性计算完成后，S.v的值为_____

产生式	语义规则
S→ABC	B.u := S.u
	A.u := B.v + C.v
	S.v := A.v
A→a	A.v := 3*A.u
B→b	B.v := B.u
C→c	C.v := 1

随堂测试

- 如下属性文法是否为L-属性文法

G(S):

产生式	语义规则
S→XYZ	Z.h := S.a
	X.c := Z.g
	S.b := X.d -2
	Y.e := S.b
X→x	X.d :=2*X.c
Y→y	Y.f := Y.e*3
Z→z	Z.g :=Z.h+1