编译原理7-10章习题课

2020级计算机科学与技术

第 7 章 语法制导翻译

第 1 题 理解下列术语：

（1）语法制导翻译

（2）综合属性、继承属性

（3）S-属性定义（S_SDD）

（4）L-属性定义（L-SDD）

（1）什么是语法制导翻译

- 编译的阶段
  - 词法分析
  - 语法分析
  - 语义分析
  - 中间代码生成
  - 代码优化
  - 目标代码生成

语义翻译

语法制导翻译
(Syntax-Directed Translation)

语法制导翻译使用*CFG*来引导对语言的翻译，是一种面向文法的翻译技术

（1）什么是语法制导翻译

答:
所谓语法制导翻译，是指在语法规则的制导下，通过计算语义规则，完成对输入符号的翻译。

由于使用属性文法时把语法规则和语义规则分开，但在使用语法规则进行推导或归约的同时又使用这些语义规则来指导翻译与最终产生目标代码，所以称为语法制导翻译。

语法制导翻译的基本思想

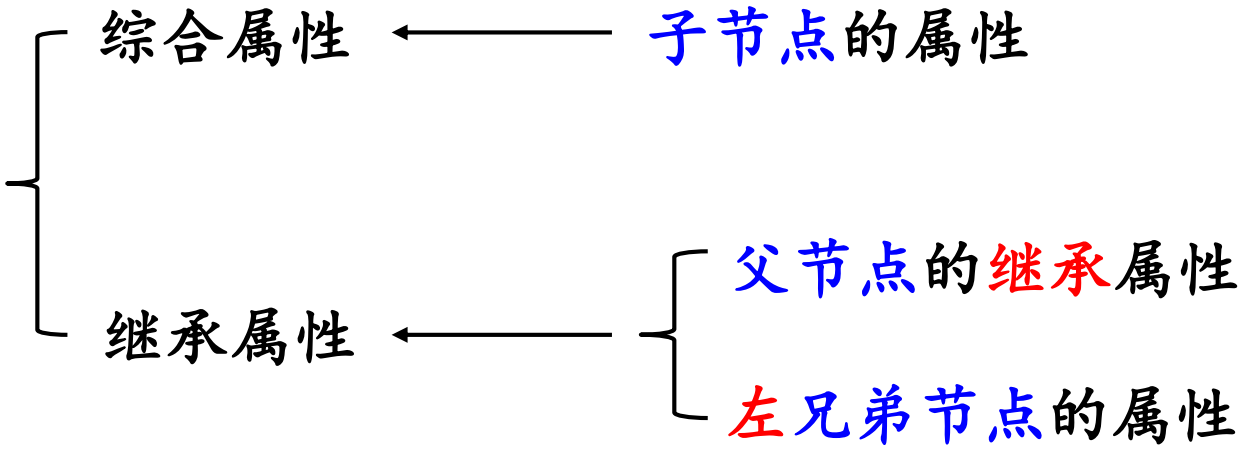
- 如何表示语义信息？

  语义属性：综合属性、继承属性

- 如何计算语义属性？
  - 文法符号的语义属性值是用与文法符号所在产生式（语法规则）相关联的语义规则来计算的
  - 对于给定的输入串x，构建x的语法分析树，并利用与产生式（语法规则）相关联的语义规则来计算分析树中各结点对应的语义属性值

S-SDD与L-SDD

- S-SDD：仅仅使用综合属性的SDD
- L-SDD：



第 7 章 语法制导翻译

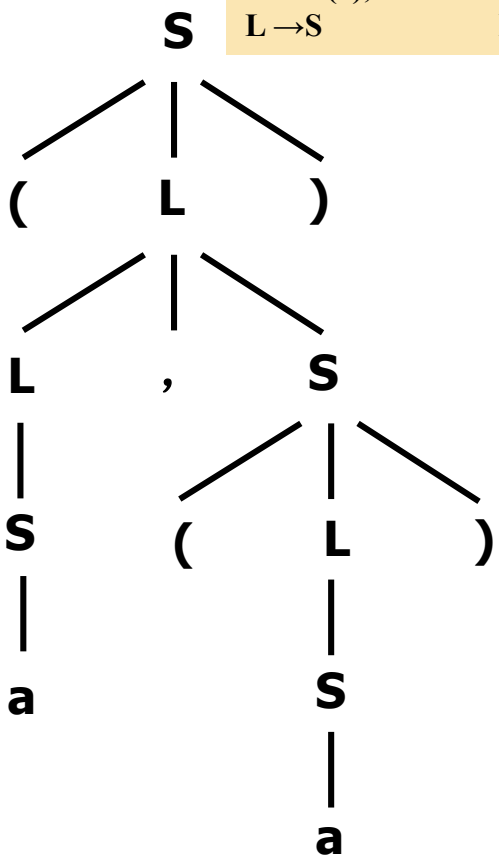
第 2 题 文法G[S]及其SDD定义如下：

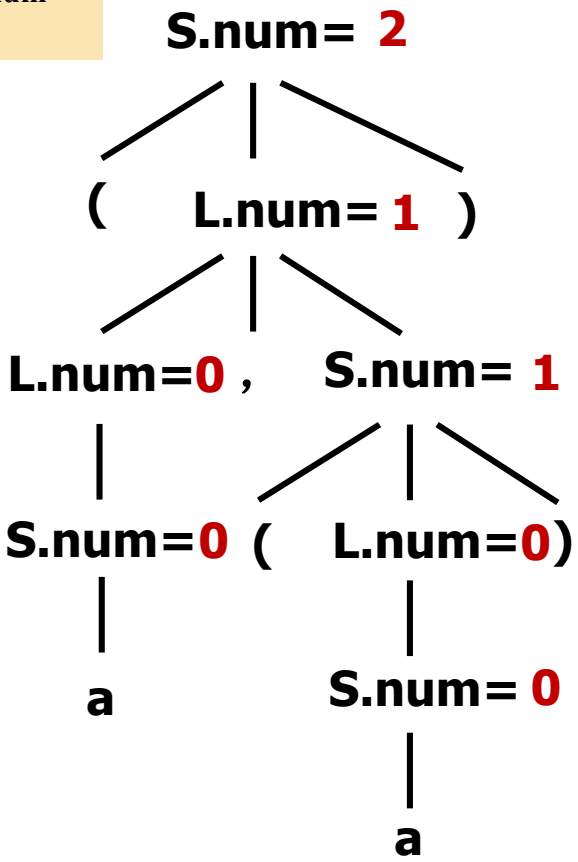
产生式	语义动作
S’ → S	print(S.num)
S → (L)	S.num = L.num +1
S → a	S.num = 0
L →L(1), S	L.num = L(1).num + S.num
L →S	L.num = S.num

若输入为(a,(a))，且采用自底向上的分析方法，则输出为(C)。

A.0　　B.1　　C.2　　D.4

S’ → S	print(S.num)
S → (L)	S.num = L.num +1
S → a	S.num = 0
L →L(1), S	L.num = L(1).num + S.num
L →S	L.num = S.num





第 7 章 语法制导翻译

第3题有一语法制导定义如下：

S→bAb　　print “1”

A→(B　　print “2”

A→a　　print “3”

B→aA)　　print “4”

若输入序列为b(a(a(aa)))b，且采用自底向上的分析方法，则输出序列为(B)。

A.32224441　　B.34242421

C.12424243　　D.34442212

第 7 章 语法制导翻译

第4题 使用( )可以定义一个程序的意义。
A.语义规则
B.词法规则
C.产生规则
D.词法规则

第5题 以下说法正确的是( )。
A.语义规则中的属性有两种：综合属性与继承属性
B.终结符只有继承属性，它由词法分析器提供
C.非终结符可以有综合属性，但不能有继承属性
D.属性值在分析过程中可以进行计算，但不能传递

第 7 章 语法制导翻译

第6题 对下面的文法，设计其SDD，使得只利用综合属性获得类型信息

D→L,id | L

L→T id

T→int | real

答：为D,T,L 引入综合属性type，代表类型。

SDD

产生式	语义规则
D→L,id	D.type:=L.type addtype(id.entry,L.type)
D→L	D.type:=L.type
L→T id	L.type:=T.type addtype(id.entry,T.type)
T→int	T.type:=integer
T→real	T.type:=real

第 7 章 语法制导翻译

第7题 文法G的产生式如下：

S→(L) | a

L→L,S | S

试写出一个语法制导定义(SDD)，使其输出配对括号个数。

答：为S,L 引入综合属性num，代表配对括号个数。

SDD

产生式	语义动作
S'→S	print(S.num)
S→(L)	S.num:=L.num+1
S→a	S.num:=0
L→L1,S	L.num:=L1.num+S.num
L→S	L.num:=S.num

语法制导翻译方案*SDT*

- 语法制导翻译方案(*SDT*)是在产生式右部中嵌入了程序片段(称为语义动作)的*CFG*

- 例

$D \rightarrow T \{ L.inh = T.type \} L$

$T \rightarrow \text{int} \{ T.type = int \}$

$T \rightarrow \text{real} \{ T.type = real \}$

$L \rightarrow \{ L_1.inh = L.inh \} L_1, \text{id}$

…

将S-SDD转换为SDT

- 将一个S-SDD转换为SDT的方法：将每个语义动作都放在产生式的最后
  - 例

S-SDD

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

SDT

(1) $L \rightarrow E$ n { $L.val = E.val$}
(2) $E \rightarrow E_1 + T${$E.val = E_1.val + T.val$}
(3) $E \rightarrow T$ { $E.val = T.val$}
(4) $T \rightarrow T_1 * F$ { $T.val = T_1.val \times F.val$}
(5) $T \rightarrow F$ { $T.val = F.val$}
(6) $F \rightarrow (E)$ { $F.val = E.val$}
(7) $F \rightarrow$ digit { $F.val =$ digit.$lexval$}

S-SDD的SDT

- 综合属性的计算时机
  - 所有子节点分析完
  - 语义动作置于产生式末尾

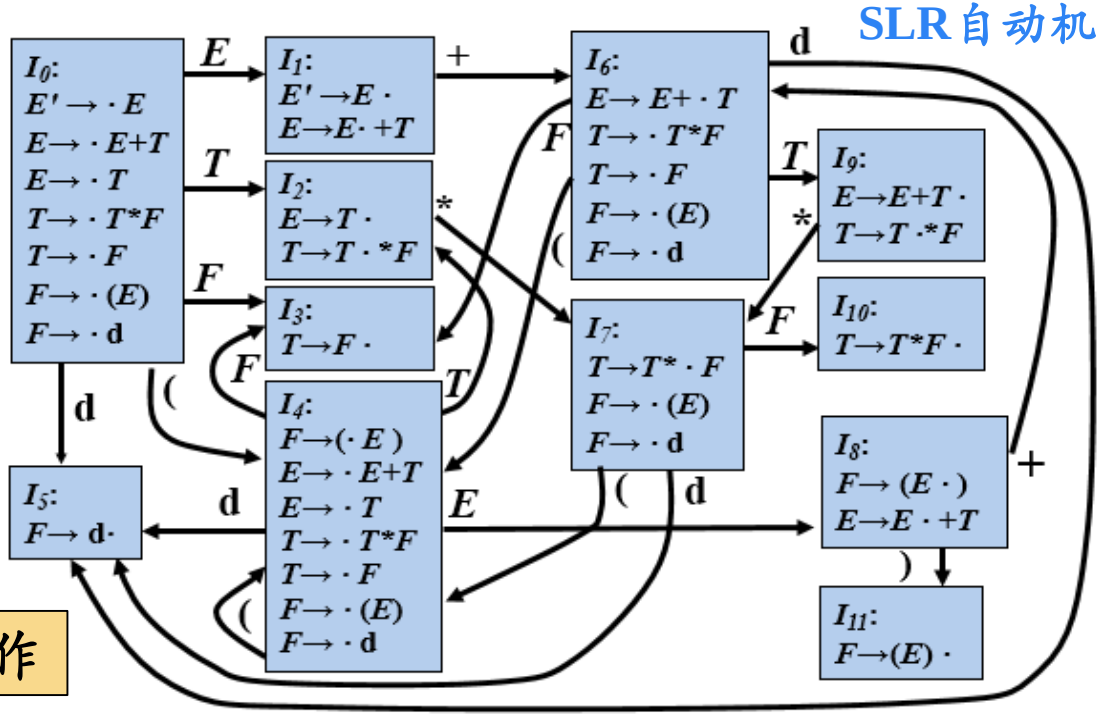
S-属性定义的SDT实现

- 如果一个S-SDD的基本文法可以使用LR分析技术，那么它的SDT可以在LR语法分析过程中实现
- 例

S-SDD

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*

当归约发生时执行相应的语义动作



将*L-SDD*转换为*SDT*

- 将*L-SDD*转换为*SDT*的规则
  - 将计算某个非终结符号*A*的继承属性的动作插入到产生式右部中紧靠在*A*的本次出现之前的位置上
  - 将计算一个产生式左部符号的综合属性的动作放置在这个产生式右部的最右端

L-SDD的SDT

- 继承属性的计算时机
  - 即将分析A之前
  - 语义动作置于紧靠在A的本次出现之前的位置上
- 综合属性的计算时机
  - 所有子节点分析完毕
  - 语义动作置于产生式末尾

例

➢ *L-SDD*

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

➢ *SDT*

1) $T \rightarrow F\ \{\ T'.inh = F.val\ \}\ T'\ \{\ T.val = T'.syn\ \}$

2) $T' \rightarrow *F\ \{\ T_1'.inh = T'.inh \times F.val\ \}\ T_1'\ \{\ T'.syn = T_1'.syn\ \}$

3) $T' \rightarrow \varepsilon\ \{\ T'.syn = T'.inh\ \}$

4) $F \rightarrow$ digit $\{\ F.val =$ digit.*lexval* $\}$

L-属性定义的*SDT*实现

- 如果一个*L-SDD*的基本文法可以使用*LL*分析技术，那么它的*SDT*可以在*LL*或*LR*语法分析过程中实现
  - 例

1) $T \rightarrow F$ { $T'.inh = F.val$ } T' { $T.val = T'.syn$ }
2) $T' \rightarrow *F$ { $T_1'.inh = T'.inh \times F.val$ } T_1' { $T'.syn = T_1'.syn$ }
3) $T' \rightarrow \varepsilon$ { $T'.syn = T'.inh$ }
4) $F \rightarrow$ digit { $F.val$ = digit.*lexval* }

SELECT (1)= { digit }
SELECT (2)= { * }
SELECT (3)= { $ }
SELECT (4)= { digit }

➢ 语法制导翻译方案（SDT）

SDD的具体实施方案

SDD定义了各属性的计算方法（计算规则）怎么算？

SDT进一步明确了各属性的计算时机（计算顺序）怎么算？＋何时算？

无循环依赖SDD的SDT

SDD

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

SDT

1) $T \rightarrow F\ \{ T'.inh = F.val \}\ T'\ \{ T.val = T'.syn \}$
2) $T' \rightarrow *F\ \{ T_1'.inh = T'.inh \times F.val \}\ T_1'\ \{ T'.syn = T_1'.syn \}$
3) $T' \rightarrow \varepsilon\ \{ T'.syn = T'.inh \}$
4) $F \rightarrow$ digit $\{ F.val =$ digit.lexval $\}$

第 7 章 语法制导翻译

第8题 以下说法不正确的是(A)。

A.如果一个S-SDD的基本文法可以使用LR分析技术，那么它的SDT可以在LL语法分析过程中实现

B.如果一个S-SDD的基本文法可以使用LR分析技术，那么它的SDT可以在LR语法分析过程中实现

C.如果一个L-SDD的基本文法可以使用LL分析技术，那么它的SDT可以在LL语法分析过程中实现

D.如果一个L-SDD的基本文法可以使用LL分析技术，那么它的SDT可以在LR语法分析过程中实现

第 7 章 语法制导翻译

9有文法G及其语法制导翻译如下所示(语义规则中的*和+分别是常规意义下的算术运算符):

$$E \to E^{(1)} \wedge T\{E.val = E^{(1)}.val * T.val\}$$

$$E \to T\{E.val = T.val\}$$

$$T \to T^{(1)} \# n\{T.val = T^{(1)}.val + n.val\}$$

$$T \to n\{T.val = n.val\}$$

则分析句子2 ∧ 3 # 4其值为(C)。

A.10　　B.21　　C.14　　D.24

第 7 章 语法制导翻译

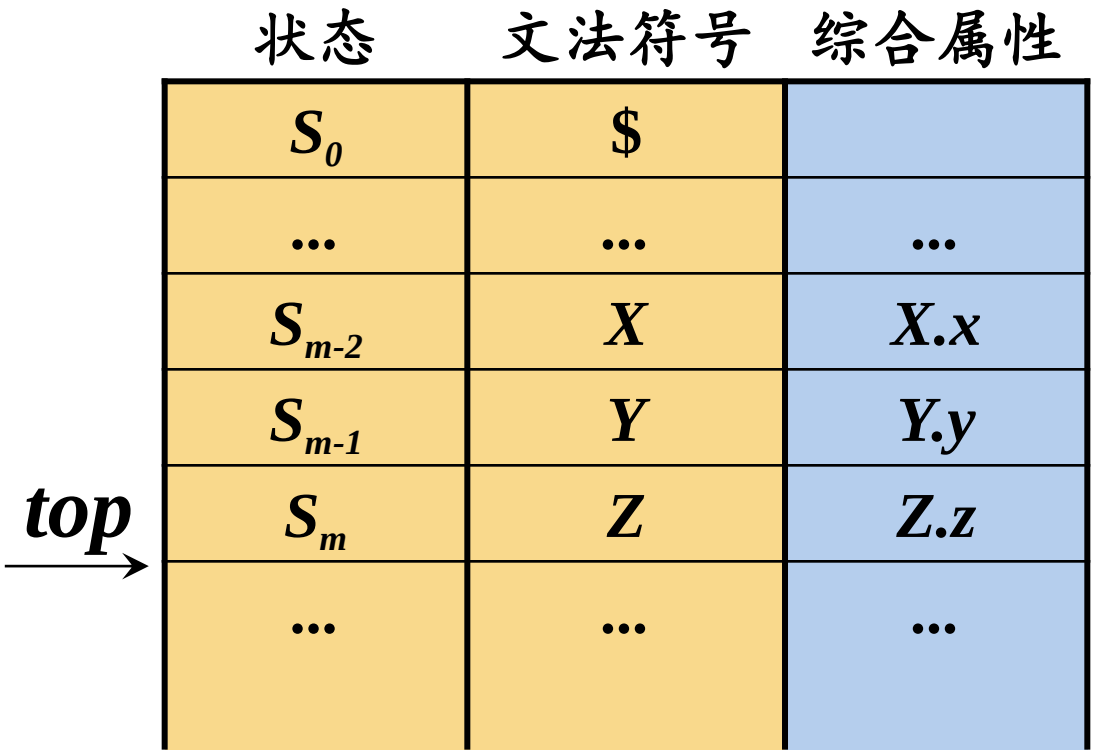
第 10 题 G[S]的属性文法如下是一个S-SDD，因此可以在自底向上的语法分析过程中通过在分析栈中增加综合属性域进行语义计算，右图是G[S]的LR分析表，给出输入串（a,（a））的语法分析和语义属性计算过程。

产生式	语义规则
(1) S’ → S	print(S.num)
(2) S → (L)	S.num = L.num +1
(3) S → a	S.num = 0
(4) L →L(1), S	L.num = L(1).num + S.num
(5) L →S	L.num = S.num

状态	ACTION					GOTO	
	a	,	(	)	#	S	L
0	s3		s2			1	
1					acc		
2	s3		s2			5	4
3		r2		r2	r2		
4		s7		s6			
5		r4		r4			
6		r1		r1	r1		
7	s3		s2			8	
8		r3		r3			

扩展的LR语法分析栈

在分析栈中使用一个附加的域来存放综合属性值



答：

产生式	语义规则
(0) S’ → S	print(S.num)
(1) S → (L)	S.num = L.num +1
(2) S → a	S.num = 0
(3) L →L(1), S	L.num = L(1).num + S.num
(4) L →S	L.num = S.num

状态	ACTION					GOTO	
	a	,	(	)	#	S	L
0	s3		s2			1	
1					acc		
2	s3		s2			5	4
3		r2		r2	r2		
4		s7		s6			
5		r4		r4			
6		r1		r1	r1		
7	s3		s2			8	
8		r3		r3			

状态栈	符号栈	剩余输入串	综合属性值	A	G
0	#	(a,(a))#	-	s2	
02	#(	a,(a))#	--	s3	
023	#(a	,(a))#	---	r2	5
025	#(S	,(a))#	--0	r4	4
024	#(L	,(a))#	--0	s7	
0247	#(L,	(a))#	--0-	s2	
02472	#(L,(	a))#	--0--	s3	
024723	#(L,(a	))#	--0---	r2	5
024725	#(L,(S	))#	--0--0	r4	4
024724	#(L,(L	))#	--0--0	s6	
0247246	#(L,(L)	)#	--0--0-	r1	8
02478	#(L,S	)#	--0-1	r3	4
024	#(L	)#	--1	s6	
0246	#(L)	#	--1-	r1	1
01	#S	#	-2		
acc					

第8章 中间代码生成

➢ 本章是上一章（语法制导翻译）的延伸

- SDT
  - 实现机制（第7章）
    - 如何扩展语法分析栈
    - 如何将SDT中的抽象定义式改写成具体执行的栈操作
    - 如何扩展递归下降过程
    - 如何修改L属性定义的SDT，使之适合bottom_up翻译
  - 设计具体语法结构的SDT（第8章）
    - 声明语句的SDT
    - 赋值语句的SDT
    - 控制流语句的SDT
    - 过程调用语句的SDT

第8章 中间代码生成

1 用(C)可以把a:=b+c翻译成三地址代码序列或四元式序列。
A.语法规则
B.词法规则
C.语义规则
D.等价变换规则

2 中间代码生成时所依据的是(C)。
A.语法规则
B.词法规则
C.语义规则
D.等价变换规则

第8章 中间代码生成

3 以下说法不正确的是(B)。

A.类型自身也有结构，用类型表达式来表示这种结构

B.基本类型不是类型表达式

C.类型名也是类型表达式

D.将类型构造符作用于类型表达式可以构成新的类型表达式

4 数组元素的地址由两部分构成，一部分是基地址，另一部分是偏移量（ A ）

A.对　　B.错

5 基地址通过查符号表即可获得。（ A ）

A.对　　B.错

6 设计数组引用的SDT的关键问题是：如何将地址计算公式和数组引用的文法关联起来。（ A ）

A.对　　B.错

第8章 中间代码生成

7 在下面的语句中，(A)不需要回填技术。
A.赋值语句　　B.goto语句
C.条件语句　　D.循环语句

8 四元式（三地址代码）之间的联系是通过(B)实现的。
A.指示器　　B.临时变量
C.符号表　　D.程序变量

9 在分支和循环中会用到条件式，而用作条件式的通常是布尔表达式()
控制流语句的翻译中，布尔表达式B被翻译成由跳转指令构成的跳转代码()

10 逻辑运算符&&、|| 和！会出现在代码中()
在回填技术中，生成一个跳转指令时，暂时不指定该跳转指令的目标标号()

第8章 中间代码生成

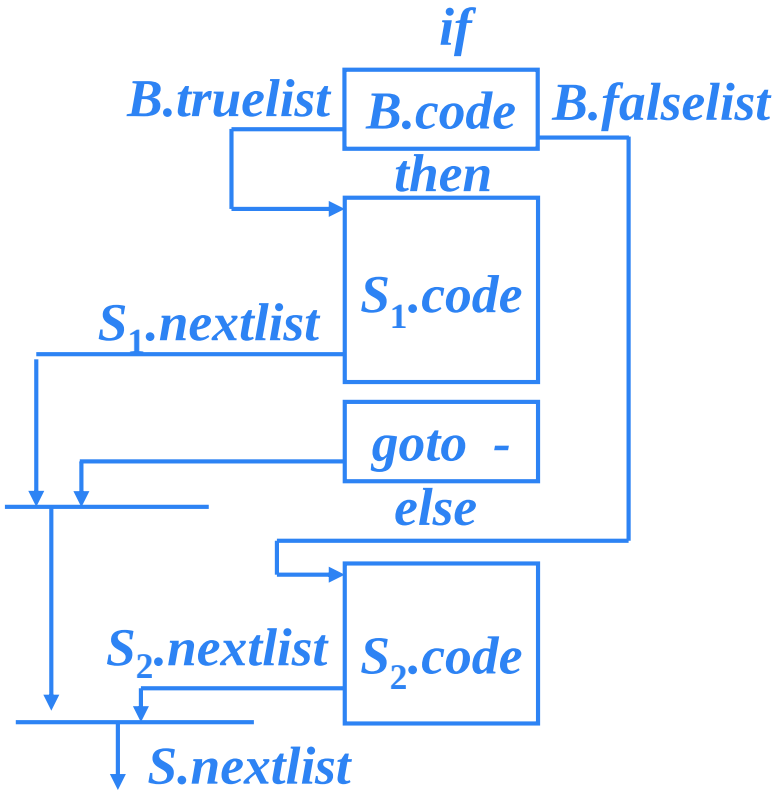
11 给出程序片段$c = a[i]$；对应的三地址代码。

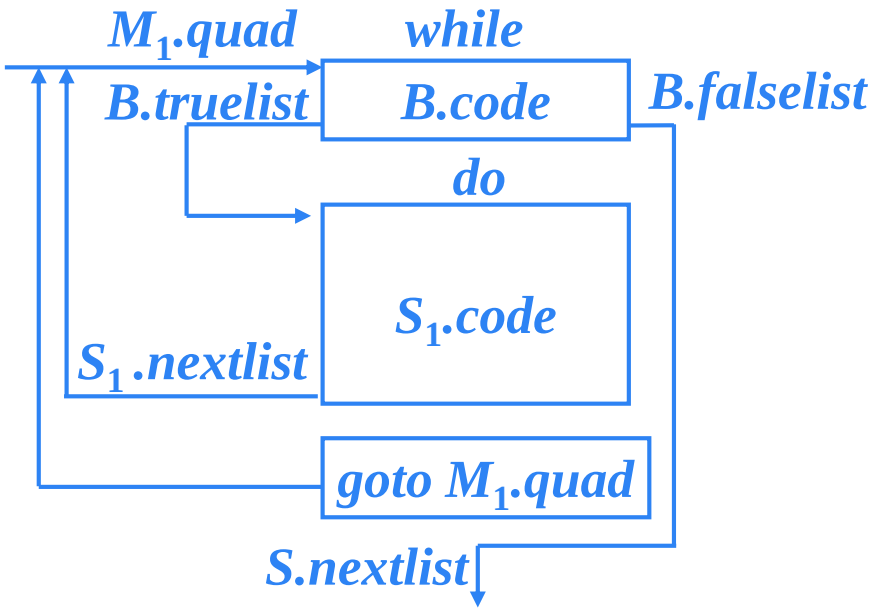
答：
```
t1  = i * 4
t2  = a [ t1 ]
c   = t2
```

12 给出下面程序片段对应的分析树并将其翻译为中间代码（三地址代码或四元式序列均可）

```
while a<b do
  if c<5 then
     while x>y do z=x+1;
  else
     x=y;
```

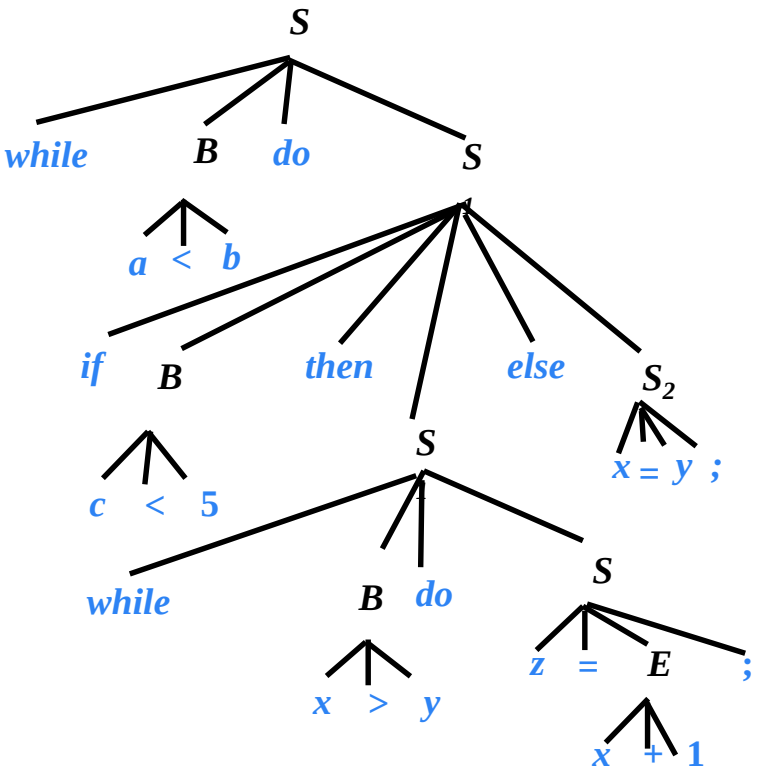
第8章 中间代码生成


if
B.truelist
B.code
B.falselist
then
S1.code
S1.nextlist
goto -
else
S2.nextlist
S2.code
S.nextlist


M1.quad
while
B.truelist
B.code
B.falselist
do
S1.code
S1 .nextlist
goto M1.quad
S.nextlist

语句“*while a<b do if c<d then x=y+z else x=y-z*”的中间代码

1: *if a < b goto* 3
2: *goto* 11
3: *if c < d goto* 5
4: *goto* 8
5: $t_1 = y + z$
6: $x = t_1$
7: *goto* 1
8: $t_2 = y - z$
9: $x = t_2$
10: *goto* 1
11:



1: (*j*<, *a*, *b* , 3)
2: (*j* , - , - , 11)
3: (*j*<, *c*, *d* , 5)
4: (*j* , - , - , 8)
5: (+ , *y* , *z* , t_1)
6: (= , t_1, - , *x*)
7: (*j* , - , - , 1)
8: (- , *y*, *z* , t_2)
9: (= , t_2, - , *x*)
10: (*j* , - , - , 1)
11:

第9章 运行时存储组织

1 有运行阶段的存储组织与管理的目的是(B)。
A.提高编译程序的运行速度
B.为运行阶段的存储分配做准备及提高目标程序的运行速度
C.优化运行空间的管理
D.节省内存空间

2 以下说法正确的是(A)。
A.编译程序除解决源程序中用户定义的量在运行时刻的存储组织与分配问题之外，还应完成为临时变量和参与运算的寄存器组织好存储空间的任务
B.由于C语言的函数允许递归调用，因此对C语言中的所有变量的单元分配一律采用动态分配方式
C.动态数组的存储空间在编译时即可完全确定
D.“运算符与运算对象类型不符”属于语法错误

第9章 运行时存储组织

3 编译方法中，动态存储分配的含义是(C)。
A.在编译阶段为源程序中的量进行分配
B.在编译阶段为源程序中的量进行分配，运行时可动态调整
C.在运行阶段为源程序中的量进行分配
D.都不正确

4 在目标代码生成阶段，符号表用于(D)。
A.目标代码生成
B.语义检查
C.语法检查
D.地址分配

第9章 运行时存储组织

每个活跃的活动都有一个位于控制栈中的活动记录

```
int  a[11];
void  readArray( ) /*将9个整数读入到a[1],…,a[9]中 */
{       int  i;
        …
}
int  partition(int  m,  int  n)
{       /*选择一个分割值v，划分a[m…n]，使得a[m…p-1]小于v，a[p]= v，
          a[p+1…n]大于等于v。返回 p */
        int  i , j ;
        …
}
void  quicksort(int  m,  int  n)
{       int  i;
        if (n＞m) {
            i=partition (m,  n);
            quicksort (m,  i-1);
            quicksort (i+1,  n);
        }
}
main( )
{       readArray ( );
        a[0] = -9999;
        a[10] = 9999;
        quicksort (1, 9);
}
```

5 代码为一个快速排序程序的概要，在程序的某次执行过程中，得到的分割值分别是4, 1, 6, 2, 8。当前正在执行的过程为*quicksort*(2,3)，要求：

（1）图示出当前栈中的全部活动记录的序列，每个活动记录只包含过程名和实参，如：*quicksort*(2,3)或简写为q(2,3)和局部变量，如：int i。

（2）为此程序画出当前对应的活动树。

第9章 运行时存储组织

答:

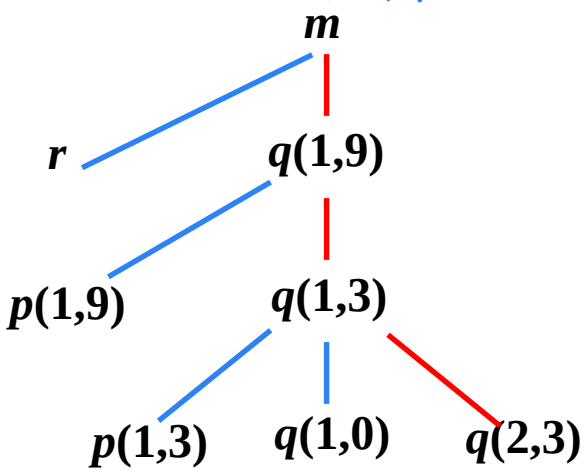
(1):

控制栈

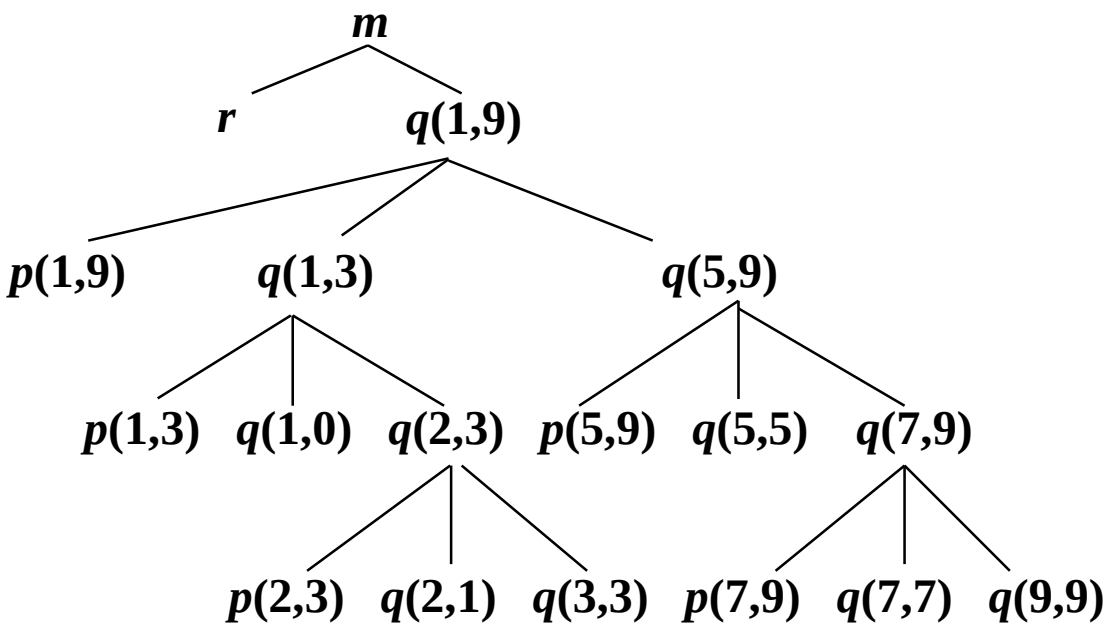
main
int *a*[11]
q(1,9)
int *i*
q(1,3)
int *i*
q(2,3)
int *i*

(2):

活动树



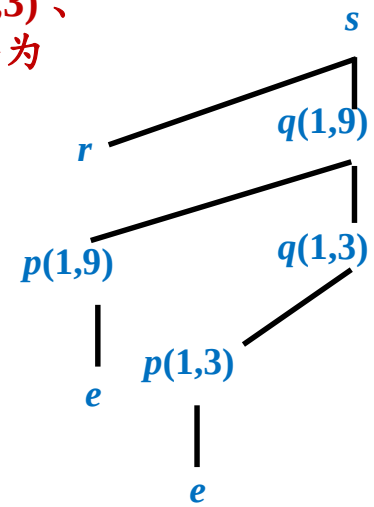
第9章 运行时存储组织


m
r
q(1,9)
p(1,9)
q(1,3)
q(5,9)
p(1,3)
q(1,0)
q(2,3)
p(5,9)
q(5,5)
q(7,9)
p(2,3)
q(2,1)
q(3,3)
p(7,9)
q(7,7)
q(9,9)

第9章 运行时存储组织

6 代码为一个快速排序程序的概要，下图是可能在程序的某次执行过程中得到的当前活动树。要求给出：
（1）每个过程的嵌套深度
（2）图示出当前栈中的活动记录，每个活动记录只包含过程名和实参如：*quicksort*(2,3)或简写为*q*(2,3)、访问链和局部变量，如：*i*。并为各个活动建立访问链。

```
program sort ( input, output );
    var a: array[0..10] of integer;
        x: integer;
    procedure readarray;
            var i: integer;
            begin … a … end {readarray} ;
    procedure exchange(i, j:integer);
            begin x=a[i];a[i]=a[j];a[j]=x; end {exchange} ;
    procedure quicksort(m, n:integer);
            var k, v : integer;
            function partition(y, z:integer):integer;
                    var i, j : integer;
                    begin … a … v … exchange(i, j) … end {partition};
            begin … a … v … partition … quicksort … end {quicksort} ;
    begin … a … readarray … quicksort … end {sort};
```



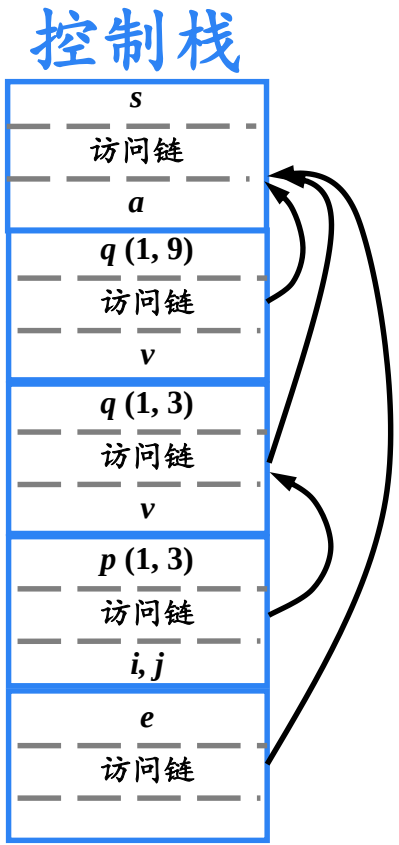
第9章 运行时存储组织

答:

(1) 每个过程的嵌套深度为:

过程	嵌套深度
sort	1
readarray	2
exchange	2
quicksort	2
partition	3

(2)活动记录和访问链为:



第10章 代码优化和目标代码生成

1 优化可生成（D）的目标代码。

A.运行时间较短　　B.占用存储空间较小

C.运行时间短但占用内存空间大

D.运行时间短且占用存储空间小

2 对一个基本块来说，(A)是正确的。

A.只有一个入口语句和一个出口语句

B.有一个入口语句和多个出口语句

C.有多个入口语句和一个出口语句

D.有多个入口语句和多个出口语句

3 数据流分析的主要应用不包括(D)。

A.到达-定值分析　　B.活跃变量分析

C.可用表达式分析　　D.自然循环分析

第10章 代码优化和目标代码生成

4 经编译得到的目标程序是(A)。
A.机器语言程序或汇编语言程序
B.四元式序列
C.三元式序列
D.二元式序列

5 (A)不可能是目标代码。
A.中间代码
B.汇编代码
C.绝对指令代码
D.可重定位指令代码

第10章 代码优化和目标代码生成

6 中间代码的优化依赖于具体的计算机。（错）

7 代码优化应以等价变换为基础，既不改变程序的运行结果，又能使生成的目标代码更有效。（对）

8 一个程序可用一个流图来表示。（对）

9 优化工作只能在中间代码这一层次上进行。（错）

10 所有编译程序都有目标代码生成阶段。（对）

11 代码生成器的设计要着重考虑目标代码的质量问题。（对）

12 目标代码生成时，无需考虑目标计算机的系统结构。（错）

第10章 代码优化和目标代码生成

13

（1）为该三地址代码序列划分为基本块并做出其流图。

（2）常用的优化技术之一是删除公共子表达式，找出每个基本块的局部公共子表达式。

（3）常用的优化技术之一是循环不变计算移出循环外，找出流图中的循环 。

B_1
(1) $i = m - 1$
(2) $j = n$
(3) $t_1 = 4 * n$
(4) $v = a[t_1]$

B_2
(5) $i = i + 1$
(6) $t_2 = 4 * i$
(7) $t_3 = a[t_2]$
(8) $if\ t_3 > v\ goto(5)$

B_3
(9) $j = j - 1$
(10) $t_4 = 4 * j$
(11) $t_5 = a[t_4]$
(12) $if\ t_5 > v\ goto(9)$

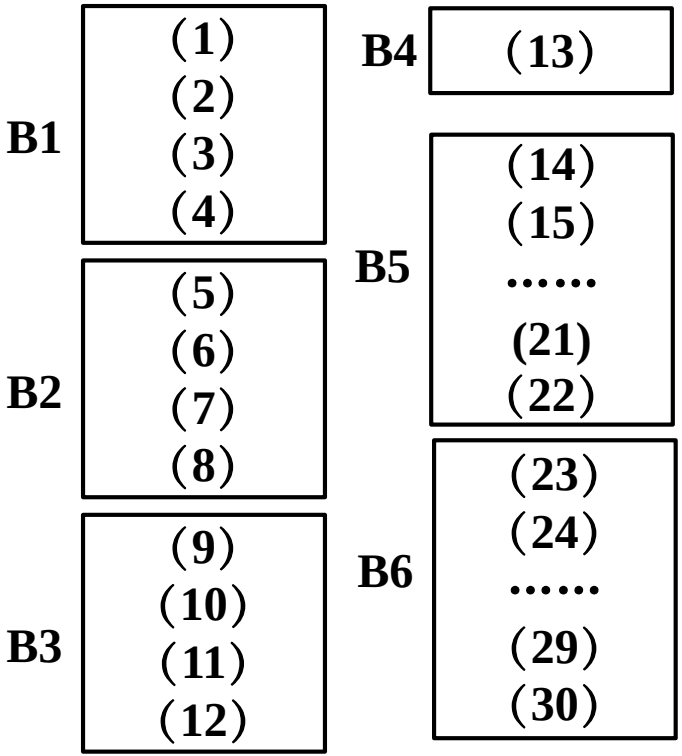
B_4
(13) $if\ i >= j\ goto(23)$
(14) $t_6 = 4 * i$
(15) $x = a[t_6]$

B_5
(16) $t_7 = 4 * i$
(17) $t_8 = 4 * j$
(18) $t_9 = a[t_8]$
(19) $a[t_7] = t_9$
(20) $t_{10} = 4 * j$
(21) $a[t_{10}] = x$
(22) $goto\ (5)$

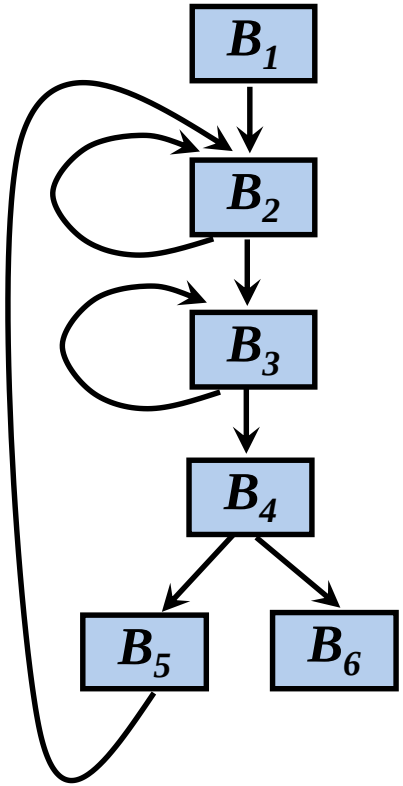
B_6
(23) $t_{11} = 4 * i$
(24) $x = a[t_{11}]$
(25) $t_{12} = 4 * i$
(26) $t_{13} = 4 * n$
(27) $t_{14} = a[t_{13}]$
(28) $a[t_{12}] = t_{14}$
(29) $t_{15} = 4 * n$
(30) $a[t_{15}] = x$

第10章 代码优化和目标代码生成

答: (1) 基本块



流图



第10章 代码优化和目标代码生成

答: (2) 各基本块的局部公共子表达式为

B5 中（14）和（16）是公共子表达式、（17）和（20）是公共子表达式
B6 中（23）和（25）是公共子表达式、（26）和（29）是公共子表达式

(3) 流图中的循环为

① {B2}
② {B3}
③ {B2，B3，B4，B5}

第10章 代码优化和目标代码生成

14 对基本块应用DAG进行优化，假设只有L在基本块后面还要被引用。

① $B = 3$

② $D = A + C$

③ $E = A * C$

④ $F = E + D$

⑤ $G = B * F$

⑥ $H = A + C$

⑦ $I = A * C$

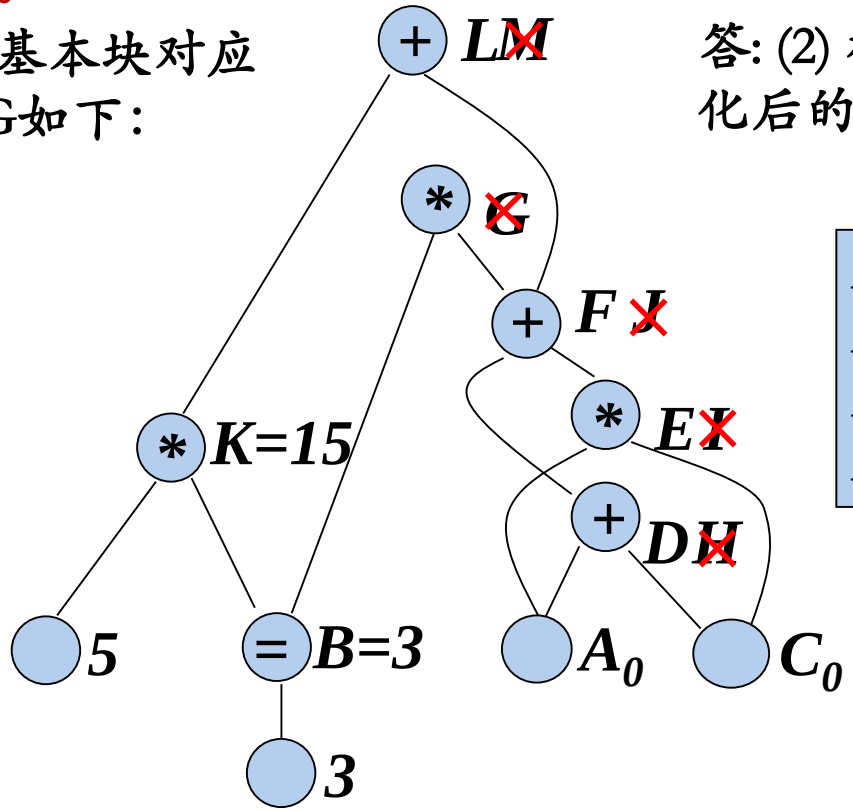
⑧ $J = H + I$

⑨ $K = B * 5$

⑩ $L = K + J$

⑪ $M = L$

答: (1) 基本块对应的DAG如下:



答: (2) 根据DAG图，优化后的语句序列为:

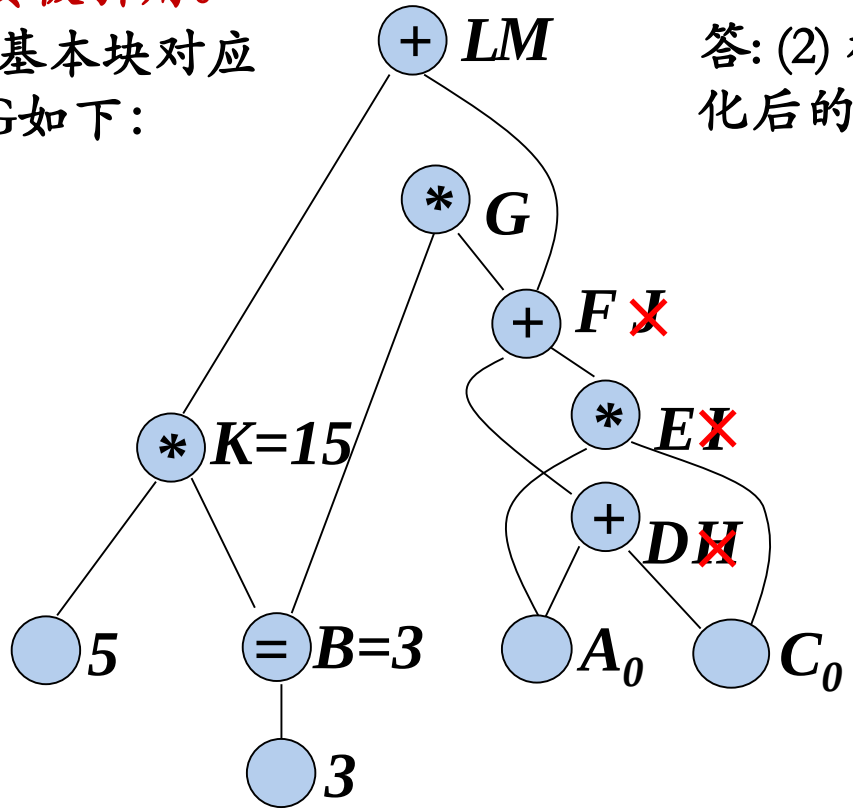
$D = A + C$
$E = A * C$
$F = E + D$
$L = 15 + F$

第10章 代码优化和目标代码生成

15 对基本块应用DAG进行优化，假设只有G、L、M在基本块后面还要被引用。

① $B = 3$
② $D = A + C$
③ $E = A * C$
④ $F = E + D$
⑤ $G = B * F$
⑥ $H = A + C$
⑦ $I = A * C$
⑧ $J = H + I$
⑨ $K = B * 5$
⑩ $L = K + J$
⑪ $M = L$

答: (1) 基本块对应的DAG如下:



答: (2) 根据DAG图，优化后的语句序列为:

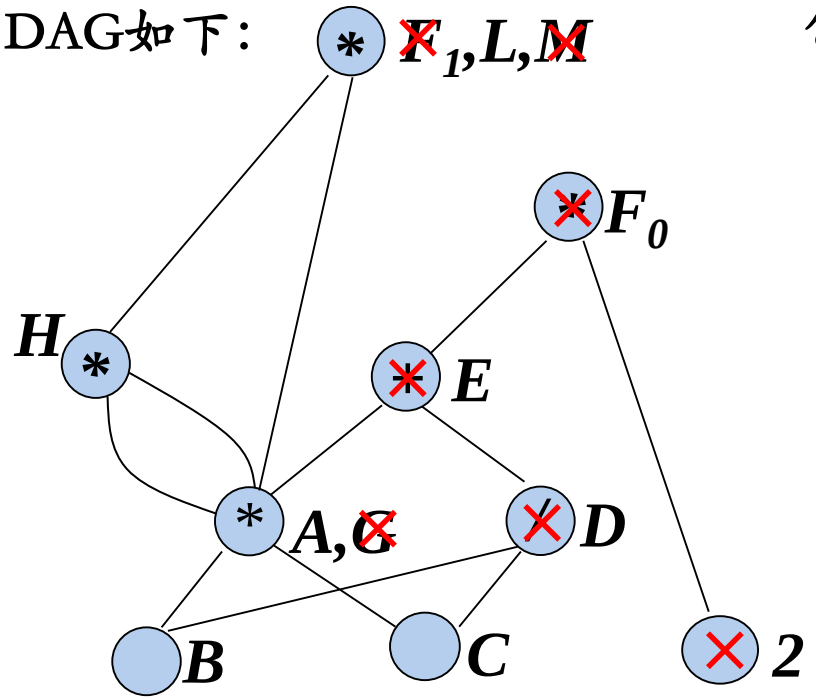
$D = A + C$
$E = A * C$
$F = E + D$
$G = 3 * F$
$L = 15 + F$
$M = L$

第10章 代码优化和目标代码生成

16 对基本块应用DAG进行优化，假设只有L在基本块后面还要被引用。

① $A = B*C$
② $D = B/C$
③ $E = A + D$
④ $F = 2 * E$
⑤ $G = B * C$
⑥ $H = G * G$
⑦ $F = H * G$
⑧ $L = F$
⑨ $M = L$

答: (1) 基本块对应的DAG如下：



答: (2) 根据DAG图，优化后的语句序列为：

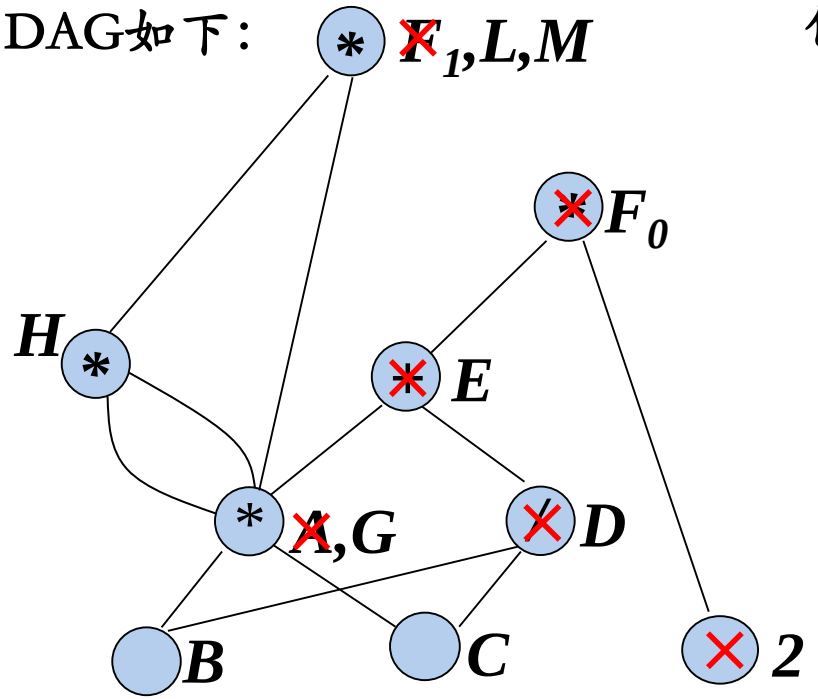
$A = B * C$
$H = A * A$
$L = H * A$

第10章 代码优化和目标代码生成

17 对基本块应用DAG进行优化，假设只有G、L、M在基本块后面还要被引用。

① $A = B*C$
② $D = B/C$
③ $E = A + D$
④ $F = 2 * E$
⑤ $G = B * C$
⑥ $H = G * G$
⑦ $F = H * G$
⑧ $L = F$
⑨ $M = L$

答: (1) 基本块对应的DAG如下：



答: (2) 根据DAG图，优化后的语句序列为：

$G = B * C$
$H = G * G$
$L = H * G$
$M = L$