第九章 中间代码优化

- 9.1 优化及其分类
- 9.2 优化技术简介
- 9.3 局部优化

9.1 优化及其分类

- 优化：编译时为改进目标程序的质量而进行的各项工作，包括提高：
  - 空间效率
  - 时间效率
- 空间效率和时间效率有时是一对矛盾，有时不能兼顾。
- 优化的基本要求：
  - 必须是等价变换
  - 为优化的努力必须是值得的。

9.1 优化及其分类

- 优化分类：
  - 机器相关优化：寄存器优化，多处理器优化，特殊指令优化，无用指令消除等。
  - 机器无关优化：中间代码优化，源程序优化
- 优化范围：
  - 局部优化：单个基本块范围内的优化：合并已知量，消除公共子表达式，削减计算强度和删除无用代码。
  - 循环优化：主要是基于循环的优化：循环不变式外提，归纳变量删除，计算强度削减。
  - 全局优化：主要是在整个程序范围内进行的优化。

9.2 优化技术简介

- 合并常量计算
- 消除公共子表达式
- 削减计算强度
- 删除无用代码
- 循环不变表达式外提
- 归纳变量删除

9.2.1 合并常量计算

- 例子：d = 2*3.14*r

  (1) (*, 2, 3.14, t1)

  (2) (*, t1 , r, t2)

  (3) (=, t2, , d)

- 2*3.1415926的值在编译时刻就可以确定。

  (1) (*, 6.28, r, t2)

  (2) (=, t2, , d)

9.2.2 消除公共子表达式

- 公共子表达式：如果某个表达式E先前已经计算，且从上次计算到现在，E中的变量的值没有改变。那么E的这次出现称为公共子表达式。
- 利用先前的计算结果，可以避免对公共子表达式的重复计算。

  例 (1)(+，b，c，a)　　(2)(-，a，d，b)

  (3)(+，b，c，c)　　(4)(-，a，d，d)

- 显然，第2和4个四元式计算的是同一个值，所以第四个四元式可以修改称为(=,b, ，d)。
- 对于第1和3个四元式，虽然都是计算b+c，但是他们的值其实是不同的，所以不能完成处理。

例子

- x+y*t-a*(x+y*t)/(y*t)

(1)(*， y， t， t1)　　(2)(+， x， t1， t2)

(3)(*， y， t， t3)　　(4)(+， x， t3， t4)

(5)(*， a，t4，t5)　　(6)(*， y， t， t6)

(7)(/， t5，t6，t7)　　(8)(-， t2， t7， t8)

- 消除公共子表达式之后：

(1)(*， y， t， t1)　　(2)(+， x， t1， t2)

(3)(*， a，t2，t5)　　(4)(/， t5， t1， t7)

(5)(-， t2，t7，t8)

9.2.3 削减计算强度

实现同样的运算可以有多种方式。用计算较快的运算代替较慢的运算。如：

- x^2 变成 x*x。
- 2*x或2.0*x 变成 x+x
- x/2 变成 x*0.5
- $a_nx^n+a_{n-1}x^{n-1}+\cdots+a_1x+a_0$变成

  $((\cdots(a_nx+a_{n-1})x+a_{n-2})\cdots)x+a_1)x+a_0$

9.2.4 删除无用代码

- 如果四元式(op，x，y，z)之后，z的值再也没有被使用到，那么这个四元式是无用的。
- 无用的四元式往往意味着程序的错误，一般不会出现在正确的程序里面。
- 多数无用四元式是由优化引起的。

9.2.5 循环不变式外提

- 有些表达式位于循环之内，但是该表达式的值不随着循环的重复执行而改变，该表达式被称为循环的不变表达式。
- 如果按照前面讲的代码生成方案，每一次循环都将计算一次。
- 如果把这个表达式提取到循环外面，该计算就只被执行一次。从而可以获得更加好的效率。

循环不变式的例子

- 计算半径为r的从10度到360度的扇形的面积：

```
for(n=1; n<36; n++)
{S:=10/360*pi*r*r*n; printf(“Area is %f”, S); }
```

- 显然，表达式10/360*pi*r*r中的各个量在循环过程中不改变。可以修改程序如下：

```
C= 10/360*pi*r*r;
for(n=1; n<36; n++)
{S:=C*n; printf(“Area is %f”, S); }
```

- 修改后的程序中，C的值只需要被计算一次，而原来的程序需要计算36次。

四元式的循环不变式

(1)= 1		n	(2)>	n	36	(21)
(3)GOF		(4)	*(4)/*	*10*	*360*	*t1*
(5) t1*	*pi*	*t2*	*(6)**	*t2*	*r*	*t3*
(7) t3*	*r*	*t4*	(8)*	t4	n	t5
(9)=t5		S	…	…	…	…
(18)+ n	1	t9	(19)=	t9		n
(20)GO	(2)		(21)			

- 其中，四元式4,5,6,7是循环不变四元式。

循环不变四元式的相对性

- 对于多重嵌套的循环，循环不变四元式是相对于某个循环而言的。可能对于更加外层的循环，它就不是循环不变式。
- 例子：

```
for(i = 1; i<10; i++)
    for(n=1; n<360/(5*i); n++)
    {S:=(5*i)/360*pi*r*r*n;...}
```

- 表达式(5*i)/360*pi*r*r对于n的循环（内层循环）是不变表达式，但是对于外层循环，它们不是循环不变表达式。

9.2.6 归纳变量的删除

- 在循环中，如果变量i的值随着循环的每次重复都固定地增加或者减少某个常量，则称i为循环的归纳变量。
- 如果在一个循环中有多个归纳变量，归纳变量的个数往往可以减少，甚至减少到1个。减少归纳变量的优化称为归纳变量的删除。

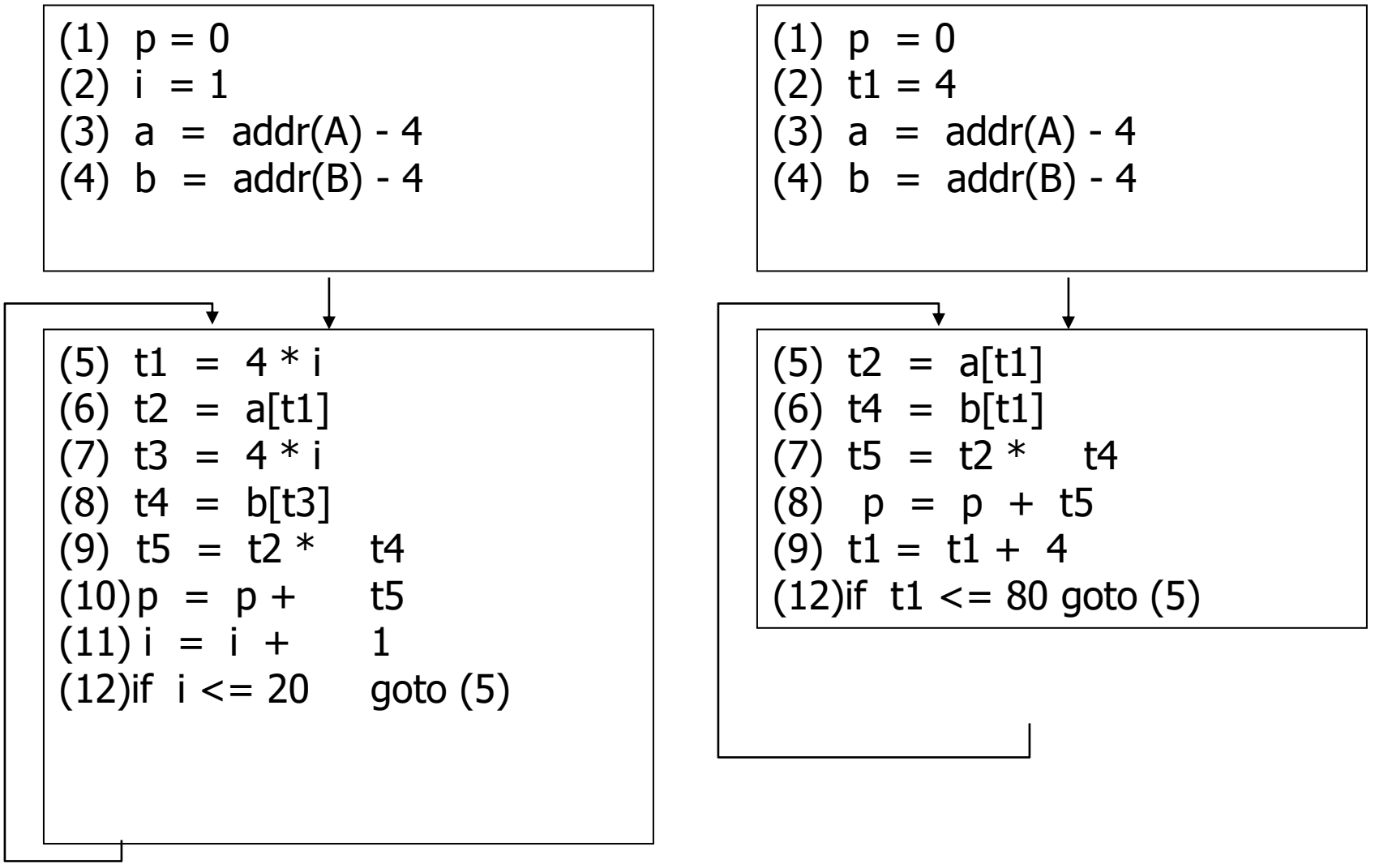
归纳变量的删除（例子）

- 例子：

```
p=0;
for(i = 1; i<= 20; i++)
    p= p+A[i]*B[i];
```

- i作为计数器。每次重复，i的值增加1，而A[i], B[i]对应的地址t1, t3增加4。
- 我们可以删除i，而使用t1或者t3进行循环结束条件的测试。

p=0; for(i = 1; i<= 20; i++) p= p+A[i]*B[i];



9.3 局部优化

- 局部优化：基本块内的优化
- 基本块：是指程序中一顺序执行的语句序列，其中只有一个入口语句和一个出口语句。
- 入口语句：
  1. 程序的第一个语句；或者，
  2. 条件转移语句或无条件转移语句的转移目标语句；或者
  3. 紧跟在条件转移语句后面的语句。

划分基本块的算法：

1. 求出四元式程序之中各个基本块的入口语句。

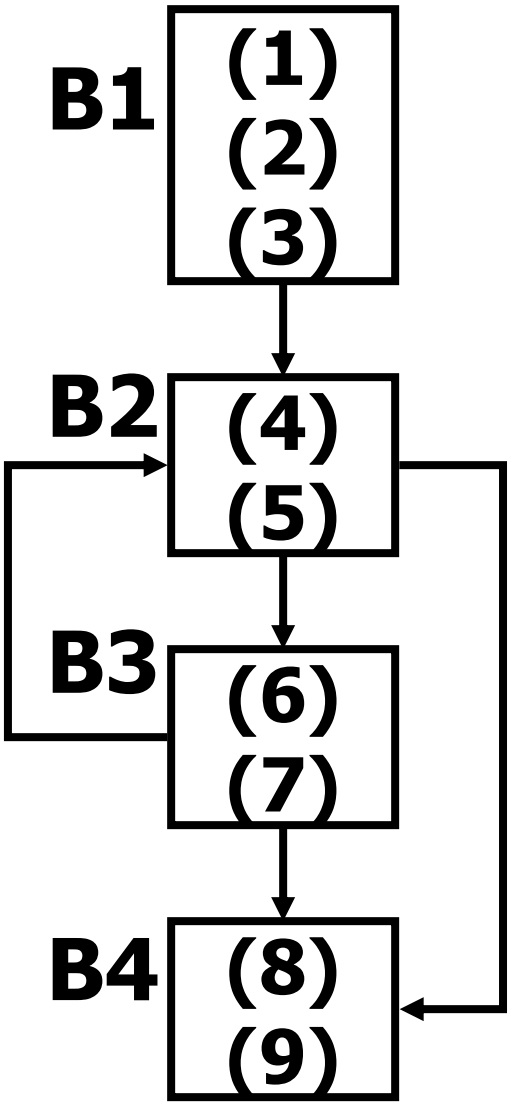
2. 对每一入口语句，构造其所属的基本块。它是由该语句到下一入口语句（不包括下一入口语句），或到一转移语句（包括该转移语句），或到一停语句（包括该停语句）之间的语句序列组成的。

3. 凡未被纳入某一基本块的语句，都是程序中控制流程无法到达的语句，因而也是不会被执行到的语句，我们可以把它们删除。

例：划分基本块

```
(1)         read (C)
(2)         A:= 0
(3)         B:= 1
(4) L1:  A:=A + B
(5)         if  B>= C  goto  L2
(6)         B:=B+1
(7)         goto  L1
(8) L2:  write (A)
(9)         halt
```

划分成四个基本块B1，B2，B3，B4



基本块内实行的优化：

合 并 已 知 量

删除多余运算

删除无用赋值

```
(1)      read (C)
(2)      A:= 0
(3)      B:= 1
(4) L1: A:=A + B
(5)      if  B>= C  goto  L2
(6)      B:=B+1
(7)      goto  L1
(8) L2:  write (A)
(9)      halt
```

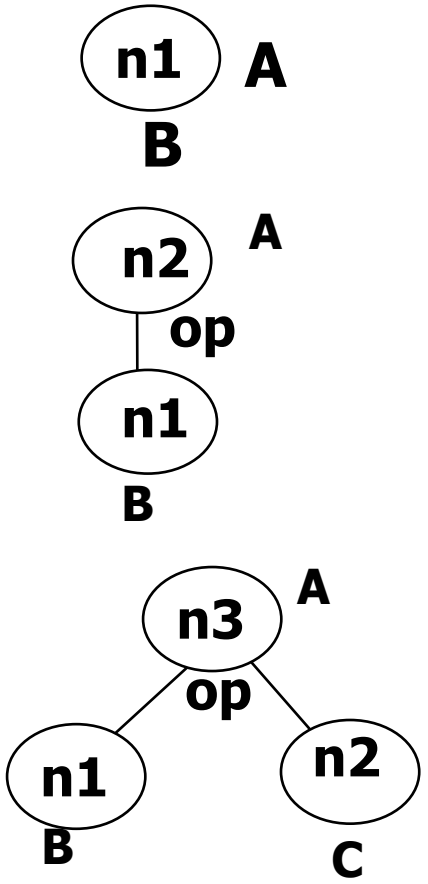
基本块优化的实现

- 基本块内部优化的实现主要工具为DAG(Directed Acyclic Graph)图。
- 用DAG图表示各个值的计算/依赖关系。
- 图中的标记或附加信息：
  - 叶子节点的标记为标识符（变量名）或常数作为唯一的标记。叶子节点是标识符时，用0表示它时初值。
  - 内部节点用运算符号作为标记，表示计算的值。每个节点的值都可以用关于变量初始值的表达式表示。
  - 各节点可能附加有一个或者多个标识符。同一个节点的标识符表示相同的值。

四元式的分类

- 0型：（:=， B，， A）
- 1型：（op，B，， A）(单目运算)
- 2型：（op，B，C，A）(双目运算)

DAG结点



基本块DAG图构造算法

- 输入：一个基本块　　输出：相应DAG图
- 算法说明：
  - 通过逐个扫描四元式来逐渐建立DAG图。
  - 函数node(A)的值或者是一个结点的编号n或者无定义。如果是前一种情况，代表存在一个结点n，A是其上的标记或附加标识符。

对基本块的每一四元式(op,B,C,A)，依次执行：

1. 如果NODE（B）无定义，则构造一标记为B的叶结点并定义NODE（B）为这个结点；
- 如果当前四元式是0型，则记NODE(B)的值为n，转4。
- 如果当前四元式是1型，则转2(1)。
- 如果当前四元式是2型，则：(I) 如果NODE(C)无定义，则构造一标记为C的叶结点并定义NODE(C) 为这个结点；(II) 转2 (2)

基本块DAG图构造算法（续）

2．（合并已知量）

（1）如果NODE(B)是标记为常数的叶结点，则转2(3)，否则转3（1）。

（2）如果NODE(B)和NODE(C)都是标记为常数的叶结点，则转2（4），否则转3（2）。

（3）执行op B（即合并已知量），令得到的新常数为P。如果NODE(B)是处理当前四元式时新构造出来的结点，则删除它。如果NODE(P)无定义，则构造一用P做标记的叶结点n。置NODE(P)=n，转4。

（4）执行B op C（即合并已知量），令得到的新常数为P。如果NODE(B)或NODE(C)是处理当前四元式时新构造出来的结点，则删除它。如果NODE(P)无定义，则构造一用P做标记的叶结点n。置NODE(P)=n，转4。

基本块DAG图构造算法（续）

3．（找公共子表达式）

（1）检查DAG中是否已有一结点，其唯一后继为NODE(B)，且标记为op（即找公共子表达式）。如果没有，则构造该结点n，否则就把已有的结点作为它的结点并设该结点为n，转4。

（2）检查中DAG中是否已有一结点，其左后继为NODE(B)，其右后继为NODE(C)，且标记为op（即找公共子表达式）。如果没有，则构造该结点n，否则就把已有的结点作为它的结点并设该结点为n，转4。

4．（删除无用赋值语句）

如果NODE(A)无定义，则把A附加在结点n上并令NODE(A)=n；否则先把A从NODE(A)结点上附加标识符集中删除（注意，如果NODE(A)是叶结点，则其标记A不删除），把A附加到新结点n上并令NODE(A)=n。转处理下一四元式。

例：构造下列四元式序列的DAG图

(1) T_0:=3.14

(2) T_1:=2*T_0

(3) T_2:=R+r

(4) A:=T_1*T_2

(5) B:=A

(6) T_3:=2*T_0

(7) T_4:=R+r

(8) T_5:=T_3*T_4

(9) T_6:=R-r

(10) B:=T_5*T_6

例：

(1) T_0:=3.14 node(T_0)=n1

(2) T_1:=2*T_0

(3) T_2:=R+r

(4) A:=T_1*T_2

(5) B:=A

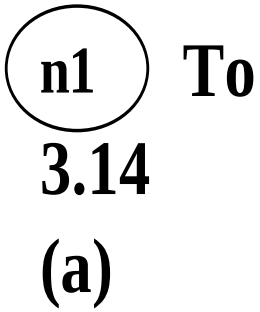
(6) T_3:=2*T_0

(7) T_4:=R+r

(8) T_5:=T_3*T_4

(9) T_6:=R-r

(10) B:=T_5*T_6



例：

(1) T_0:=3.14 node(T_0)=n1

(2) T_1:=2*T_0 node(T_1)=n2

(3) T_2:=R+r

(4) A:=T_1*T_2

(5) B:=A

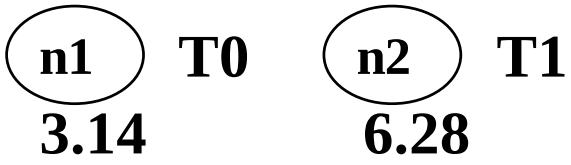
(6) T_3:=2*T_0

(7) T_4:=R+r

(8) T_5:=T_3*T_4

(9) T_6:=R-r

(10) B:=T_5*T_6



(b)

例：

(1) T_0:=3.14 node(T_0)=n1

(2) T_1:=2*T_0 node(T_1)=n2

(3) T_2:=R+r node(R)=n3 node(r)=n4 node(T_2)=n5

(4) A:=T_1*T_2

(5) B:=A

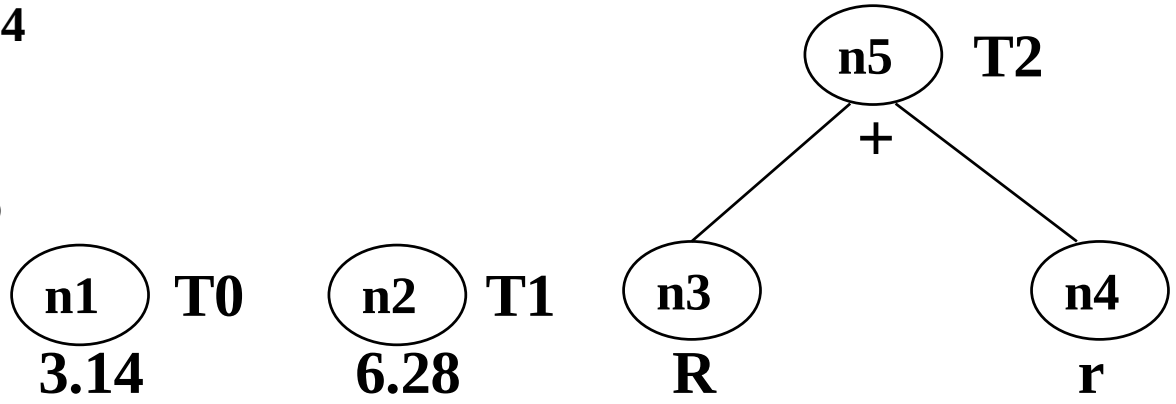
(6) T_3:=2*T_0

(7) T_4:=R+r

(8) T_5:=T_3*T_4

(9) T_6:=R-r

(10) B:=T_5*T_6



(c)

例：

(1) T_0:=3.14 node(T_0)=n1

(2) T_1:=2*T_0 node(T_1)=n2

(3) T_2:=R+r node(R)=n3 node(r)=n4 node(T_2)=n5

(4) A:=T_1*T_2 node(A)=n6

(5) B:=A

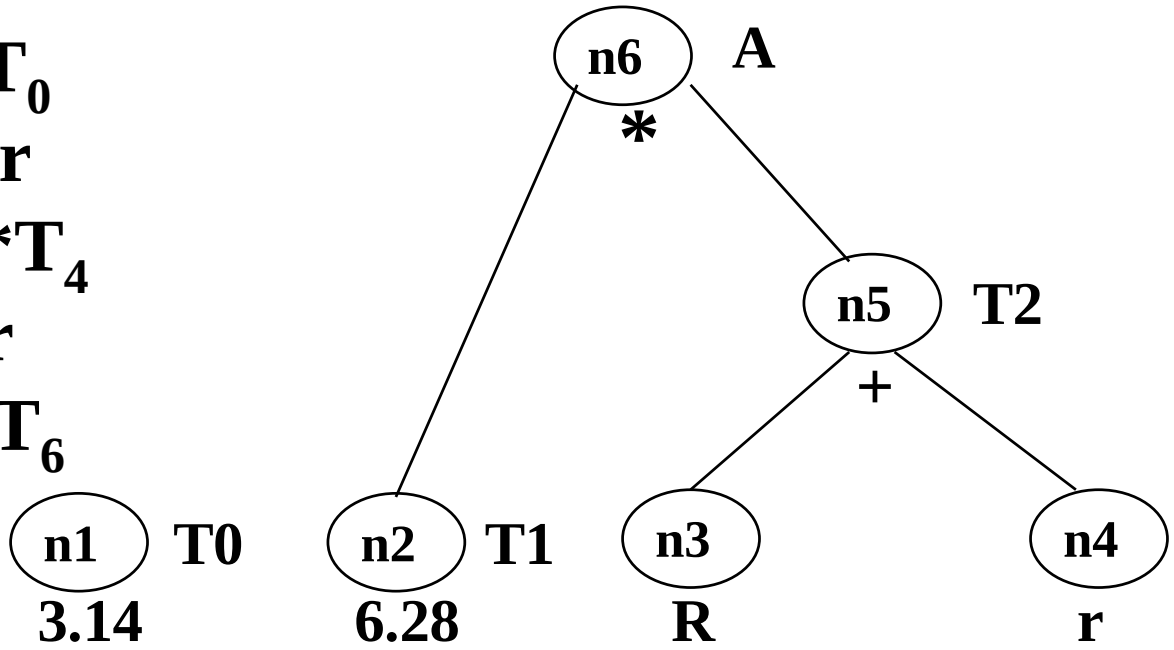
(6) T_3:=2*T_0

(7) T_4:=R+r

(8) T_5:=T_3*T_4

(9) T_6:=R-r

(10) B:=T_5*T_6



(d)

例：

(1) T_0:=3.14 node(T_0)=n1

(2) T_1:=2*T_0 node(T_1)=n2

(3) T_2:=R+r node(R)=n3 node(r)=n4 node(T_2)=n5

(4) A:=T_1*T_2 node(A)=n6

(5) B:=A node(B)=n6

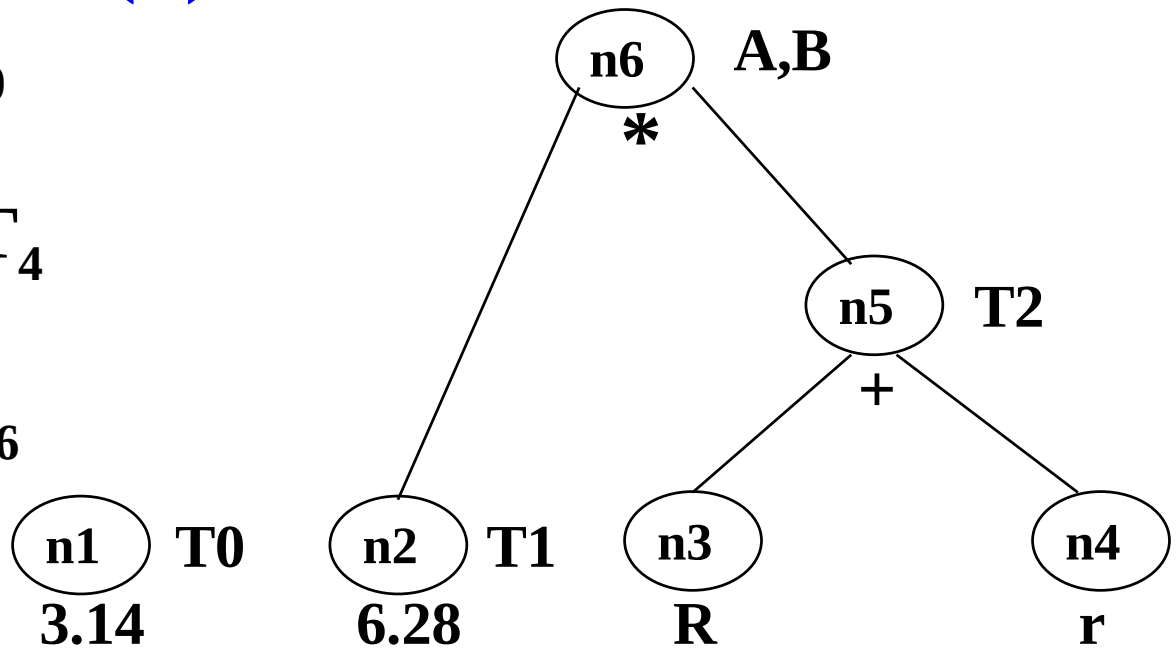
(6) T_3:=2*T_0

(7) T_4:=R+r

(8) T_5:=T_3*T_4

(9) T_6:=R-r

(10) B:=T_5*T_6



(e)

例：

(1) T_0:=3.14 node(T_0)=n1

(2) T_1:=2*T_0 node(T_1)=n2

(3) T_2:=R+r node(R)=n3 node(r)=n4 node(T_2)=n5

(4) A:=T_1*T_2 node(A)=n6

(5) B:=A node(B)=n6

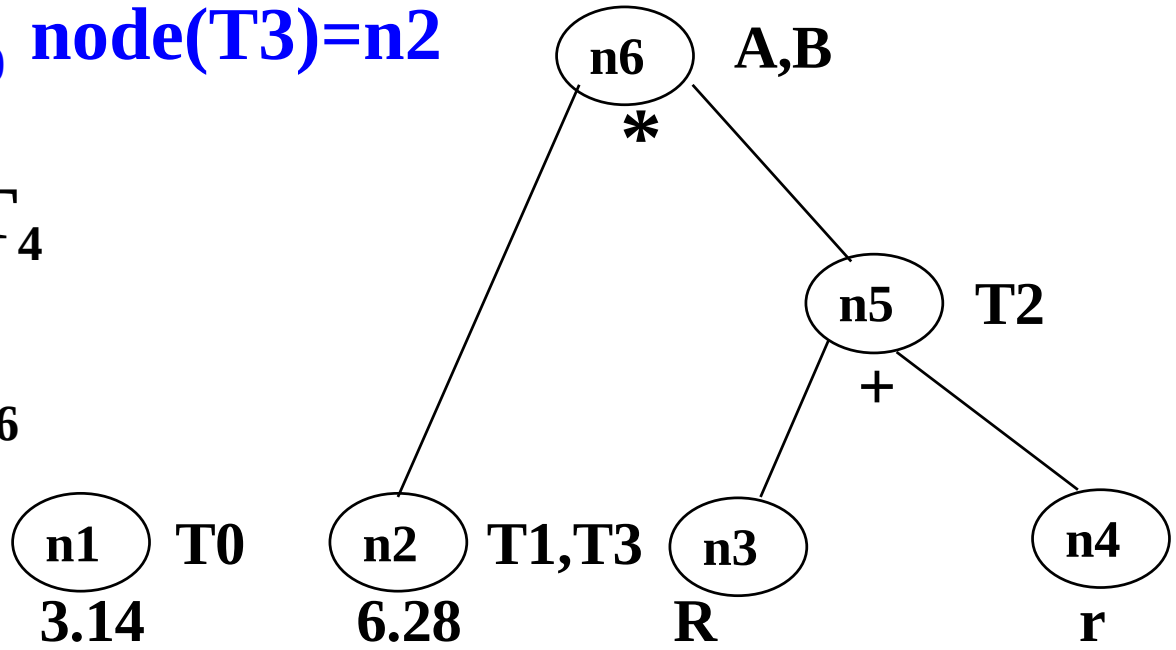
(6) T_3:=2*T_0 node(T3)=n2

(7) T_4:=R+r

(8) T_5:=T_3*T_4

(9) T_6:=R-r

(10) B:=T_5*T_6



(f)

例：

(1) T_0:=3.14 node(T_0)=n1

(2) T_1:=2*T_0 node(T_1)=n2

(3) T_2:=R+r node(R)=n3 node(r)=n4 node(T_2)=n5

(4) A:=T_1*T_2 node(A)=n6

(5) B:=A node(B)=n6

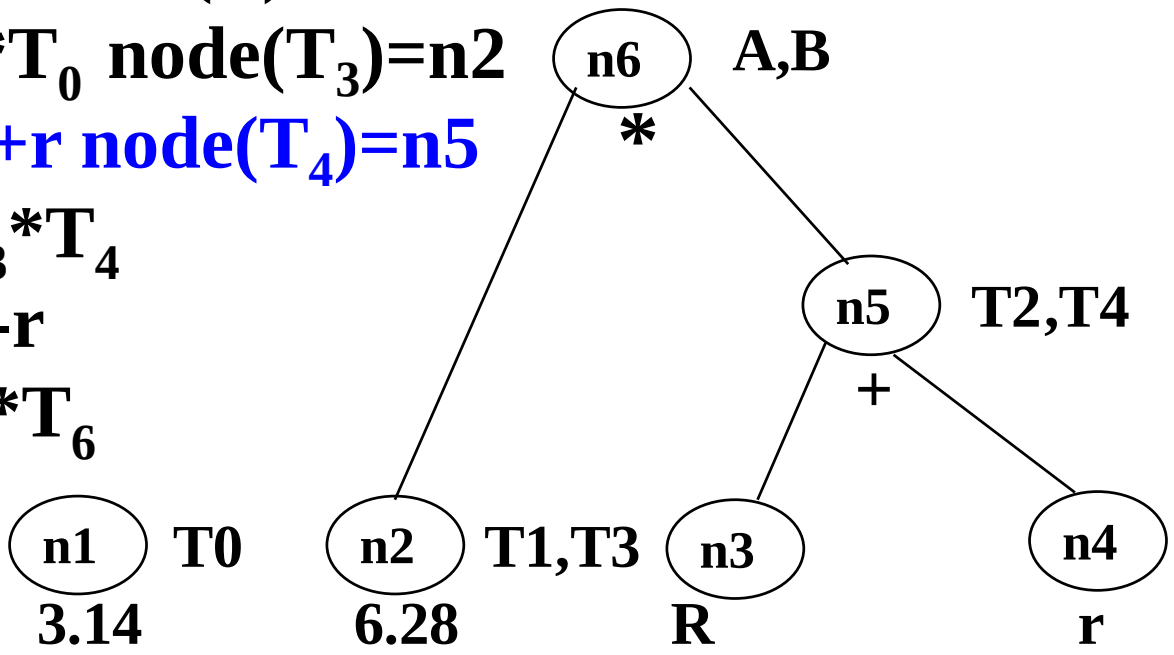
(6) T_3:=2*T_0 node(T_3)=n2

(7) T_4:=R+r node(T_4)=n5

(8) T_5:=T_3*T_4

(9) T_6:=R-r

(10) B:=T_5*T_6



(g)

例：

(1) T_0:=3.14 node(T_0)=n1

(2) T_1:=2*T_0 node(T_1)=n2

(3) T_2:=R+r node(R)=n3 node(r)=n4 node(T_2)=n5

(4) A:=T_1*T_2 node(A)=n6

(5) B:=A node(B)=n6

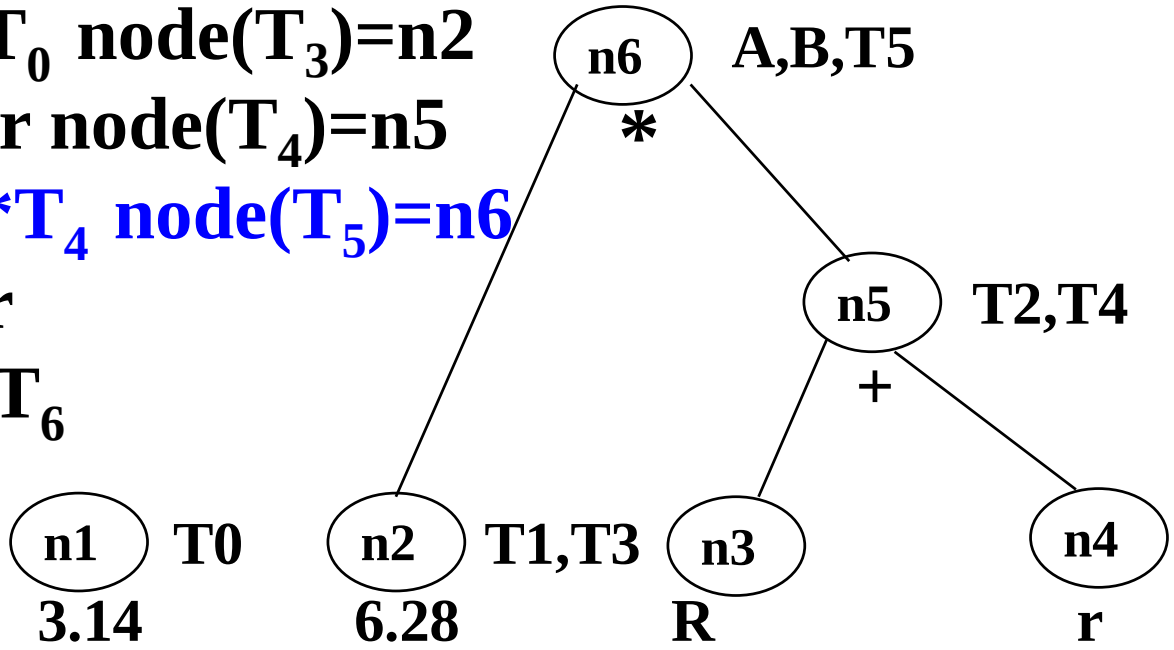
(6) T_3:=2*T_0 node(T_3)=n2

(7) T_4:=R+r node(T_4)=n5

(8) T_5:=T_3*T_4 node(T_5)=n6

(9) T_6:=R-r

(10) B:=T_5*T_6



(h)

例：

(1) $T_0:=3.14$

(2) $T_1:=2*T_0$

(3) $T_2:=R+r$

(4) $A:=T_1*T_2$

(5) $B:=A$

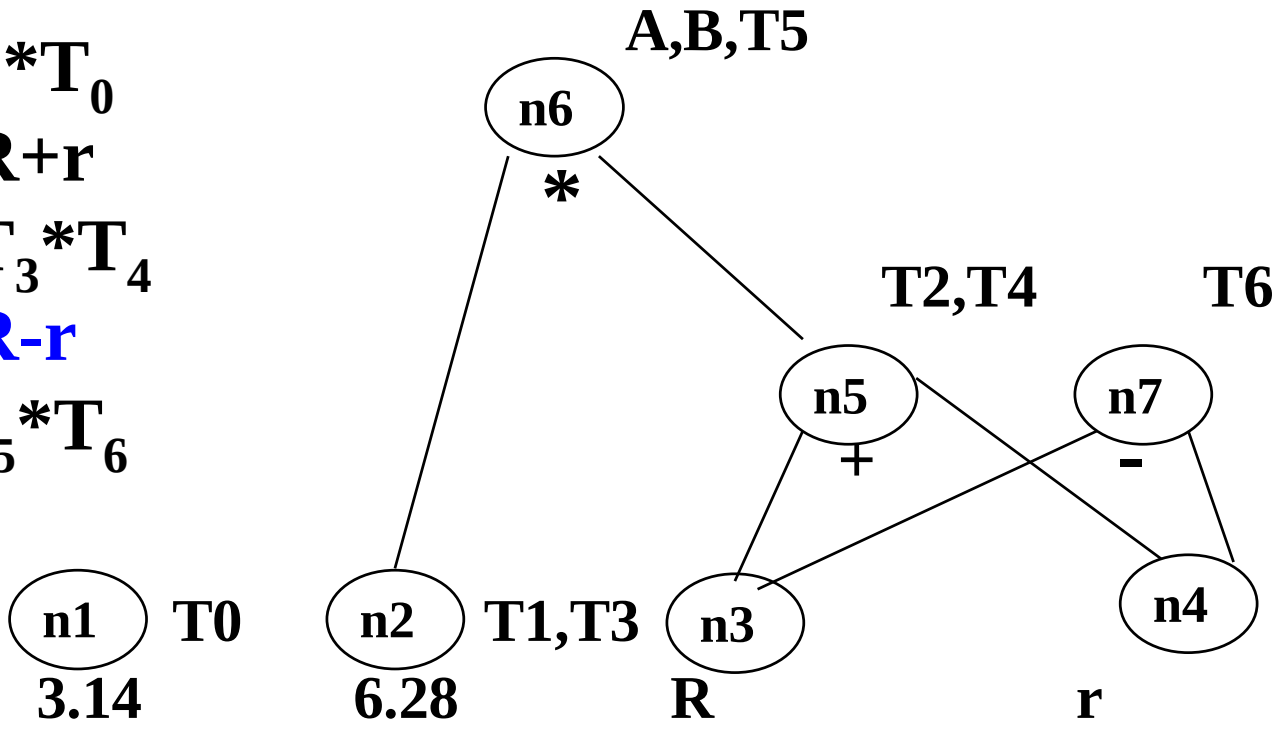
(6) $T_3:=2*T_0$

(7) $T_4:=R+r$

(8) $T_5:=T_3*T_4$

(9) $T_6:=R-r$

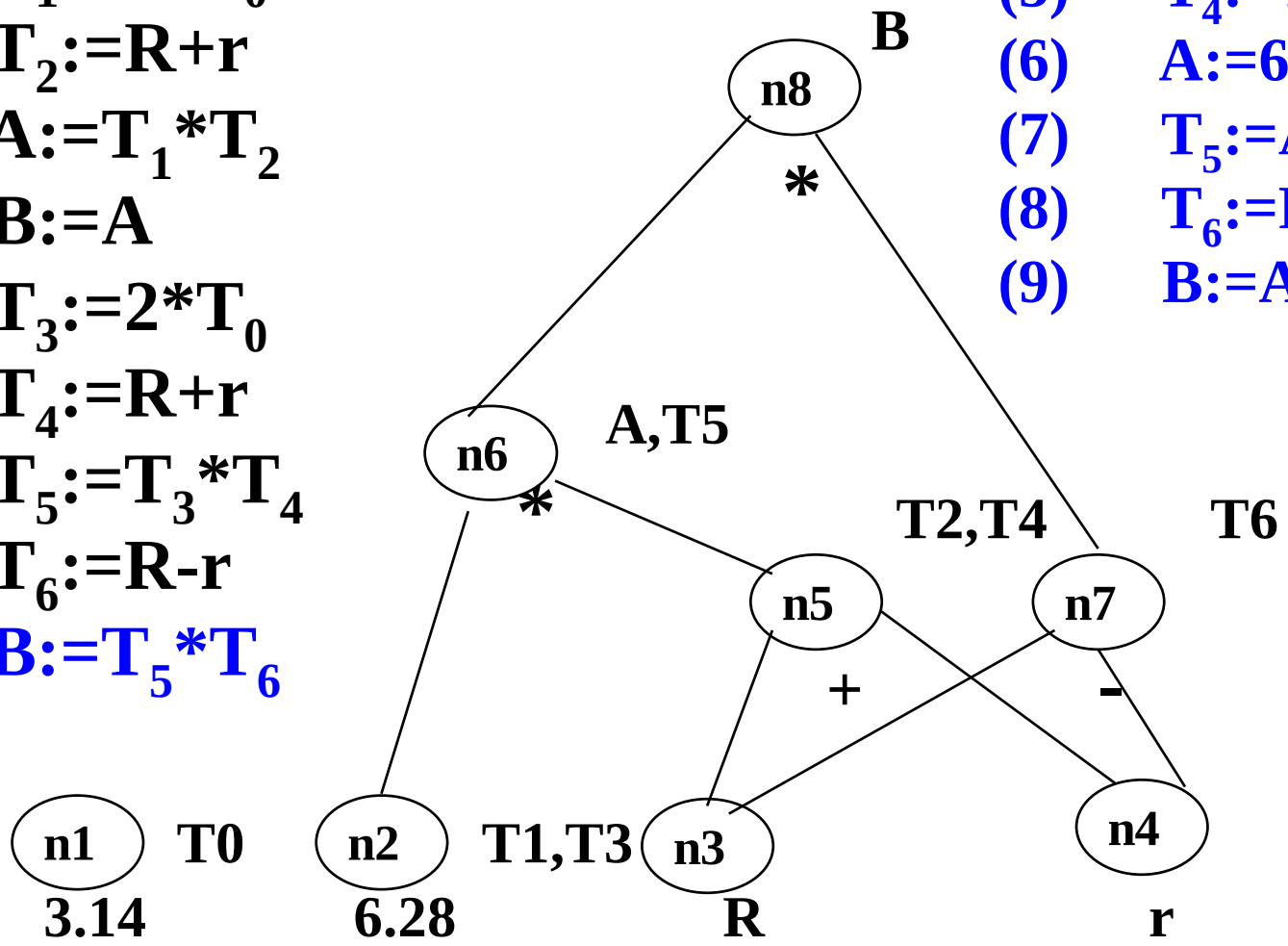
(10) $B:=T_5*T_6$



例：

(1) $T_0:=3.14$
(2) $T_1:=2*T_0$
(3) $T_2:=R+r$
(4) $A:=T_1*T_2$
(5) $B:=A$
(6) $T_3:=2*T_0$
(7) $T_4:=R+r$
(8) $T_5:=T_3*T_4$
(9) $T_6:=R-r$
(10) $B:=T_5*T_6$

(1) $T_0:=3.14$
(2) $T_1:=6.28$
(3) $T_3:=6.28$
(4) $T_2:=R+r$
(5) $T_4:=T_2$
(6) $A:=6.28*T_2$
(7) $T_5:=A$
(8) $T_6:=R-r$
(9) $B:=A*T_6$



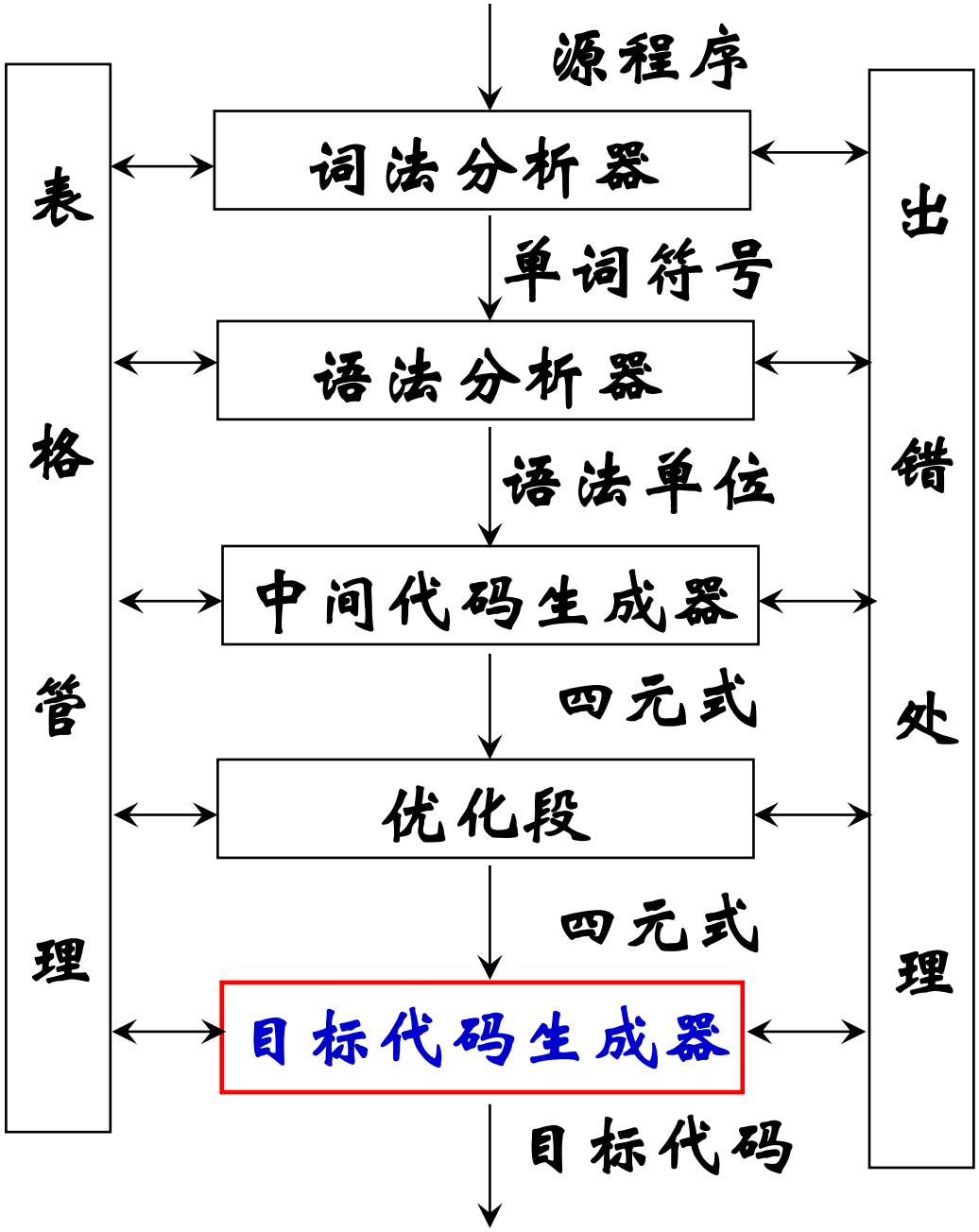
(j)

第十章 目标代码生成

- 10.1 概念
- 10.2 目标机器模型
- 10.3 一个简单的代码生成器

10.1 概念

- 代码生成是把语法分析后或优化后的中间代码变换成目标代码。
- 目标代码一般有以下三种形式：
  - 能够立即执行的机器语言代码，所有地址已经定位；
  - 待装配的机器语言模块。执行时，由连接装配程序把它们和某些运行程序连接起来，转换成能执行的机器语言代码；
  - 汇编语言代码。尚须经过汇编程序汇编，转换成可执行的机器语言代码。


源程序
词法分析器
单词符号
语法分析器
语法单位
中间代码生成器
四元式
优化段
四元式
目标代码生成器
目标代码
表格管理
出错处理

10.1 概念

- 代码生成着重考虑的问题：
  - 如何使生成的目标代码较短；
  - 如何充分利用计算机的寄存器，减少目标代码中访问存贮单元的次数。
  - 如何充分利用计算机的指令系统的特点。

10.1 概念

- 设计代码生成器时要考虑的一般问题：
  - 代码生成器的输入
    - 代码生成器的输入包括源程序的中间表示，以及符号表中的信息
    - 类型检查
  - 目标程序
    - 绝对机器代码、可再定位机器语言、汇编语言
  - 指令选择
  - 寄存器分配
  - 计算顺序选择

10.2 目标机器模型

- 考虑一个抽象的计算机模型
  - 具有多个通用寄存器，他们既可以作为累加器，也可以作为变址器。
  - 运算必须在某个寄存器中进行。
  - 含有四种类型的指令形式

10.2 目标机器模型

类　型	指令形式	意义(设 op 是二目运算符)
直接地址型	op R_i, M	(R_i) op (M) ⇒ R_i
寄存器型	op R_i, R_j	(R_i) op (R_j) ⇒ R_i
变址型	op R_i, c(R_j)	(R_i) op ((R_i)+c) ⇒ R_i
间接型	op R_i, *M op R_i, *R_j op R_i, *c(R_j)	(R_i) op ((M)) ⇒ R_i (R_i) op ((R_j)) ⇒ R_i (R_i) op (((R_j)+c)) ⇒ R_i

- op包括常见的一些运算符，如ADD(加)、SUB(减)、MUL(乘)、DIV(除)
- 如果op是一目运行符，则“op Ri, M”的意义为：op（M）⇒ Ri，其余类型可类推。

指　令	意　　义
LD R_i, B	把 B 单元的内容取到寄存器 R_i，即（B）⇒ R_i。
ST R_i, B	把寄存器 R_i 的内容存到 B 单元，即（Ri）⇒ B。
J　　X	无条件转向 X 单元。
CMP A, B	把 A 单元和 B 单元的值进行比较，根据比较情况把机器内部特征寄存器 CT 置成相应状态。CT 占两个二进位。根据 A<B 分别置 CT 为 0 或 1 或 2。
J＜X	如 CT＝0　　转 X 单元
J≤X	如 CT＝0 或 CT＝1　转 X 单元
J＝X	如 CT＝1　　转 X 单元
J≠X	如 CT≠1　　转 X 单元
J＞X	如 CT＝2　　转 X 单元
J≥X	如 CT＝2 或 CT＝1　转 X 单元

10.3 一个简单的代码生成器

- 不考虑代码的执行效率，目标代码生成是不难的，例如：
  - A:=(B+C)*D+E
- 翻译为四元式：
  - T_1:=B+C
  - T_2:=T_1*D
  - T_3:=T_2+E
  - A:=T_3

假设只有一个寄存器可供使用

- 四元式
- 目标代码：

四元式	目标代码
$T_1:=B+C$	LD R_0, B ADD R_0, C
	ST R_0, T_1
$T_2:=T_1*D$	LD R_0, T_1 MUL R_0, D
	ST R_0, T_2
$T_3:=T_2+E$	LD R_0, T_2 ADD R_0, E
	ST R_0, T_3
$A:=T_3$	LD R_0, T_3 ST R_0, A

假设T_1，T_2，T_3在基本块之后不再引用：

```
LD    R0, B
ADD   R0, C
MUL   R0, D
ADD   R0, E
ST    R0, A
```

10.3 一个简单的代码生成器

- 一般做法：

  依次把每条中间代码变换成目标代码，并在一个基本块的范围内考虑如何充分利用寄存器：

  - 在生成计算某变量值的目标代码时，尽可能让该变量保留在寄存器中。
  - 后续的目标代码尽可能引用变量在寄存器中的值，而不访问内存。
  - 在离开基本块时，把存在寄存器中的现行的值放到主存中。