编译原理

北方工业大学信息学院
School of Information Science and Technology,
North China University of Technology
束劼
shujie@ncut.edu.cn
瀚学楼1122，88801615

第九章 代码优化与代码生成

第九章 代码优化与代码生成

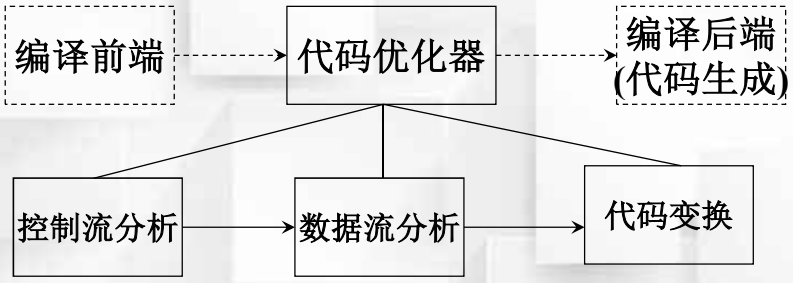
- 本章目录
- 9.1 概述
- 9.2 局部优化
- 9.3 目标代码生成

第九章 代码优化与代码生成

- 大纲要求
- 掌握：局部优化、目标代码生成。
- 理解：基本块及其DAG表示，优化遵循的原则。
- 了解：代码优化与代码生成的基本概念。

9.1 概述

9.1 概述



9.1 概述

- 代码优化的任务

  对程序进行各种**等价**变换，使得从变换后的程序出发，能生成更**有效**的目标代码。

  等价：指不改变程序的运行结果

  有效：指目标代码运行时间短，占用的存储空间小

- 代码生成的任务

  针对**目标语言**的特殊性，生成高性能的目标代码（机器相关）

9.1 概述

- 代码优化的目的

  是为了产生更高效的代码。由优化编译程序提供的对代码的各种变换必须遵循一定的原则：

  等价原则：经过优化后不应改变程序运行的结果；

  有效原则：使优化后所产生的目标代码运行时间较短，占用的存储空间较小；

  合算原则：应尽可能以较低的代价取得较好的优化效果。

9.1 概述

- 代码优化的方向
- 算法优化

  有效的数据结构和算法（领域问题）

- 编译优化

  中间代码优化：机器无关

  - 如：常数计算、公共代码段的提取

  目标代码优化：机器相关

  - 如：特殊指令、特殊结构

9.1 概述

- **中间代码**优化
- 局部优化

  基本块内部（不包括各种转移控制）

- 循环优化

  可能反复执行的代码序列，**基本块**之间

- 全局优化

  控制流分析与化简、数据流分析，多个**基本块**范围

9.1 概述

- 代码优化的种类
  - ① 删除公共子表达式(或称删除多余运算)
  - ② 代码外提
  - ③ 强度消弱
  - ④ 复写传播
  - ⑤ 删除无用代码(删除无用赋值)
  - ⑥ 删除归纳变量

9.1 概述

- 删除公共子表达式(或称删除多余运算)

  一个表达式E在前面已经计算过，并且在这之后E中变量的值没有改变，则称E为**公共子表达式**。对该表达式，可以避免重复计算，称为**删除公共子表达式**。

  例如：T4: =4*j

  ……

  T8:=4*j

  T9:= a [T8]

  替换公共子表达式

  4*j

  → T8:=T4

9.1 概述

- 代码外提

  循环中某些代码的结果在循环中是不变的，可以把它提到循环外，避免重复计算。

  例如： while(i<=limit-2)……

  如果limit-2的值始终不变，则可以变为：

  t:=limit-2;

  while(i<=t)……

9.1 概述

- 强度消弱

  利用加减法运算比乘法快的优势，把循环中计算的乘法运算，变换为在循环前进行一次乘法运算，而在循环中进行加减法运算。

  例如：某个循环中 j:=j-1; T4:=4*j; ……

  j在循环中每次减少1；

  T4在循环前没有被赋值，在循环中每次减少4；

  可以在循环开始前，给T4:=4*j

  在该循环中**T4:=T4-4**;

9.1 概述

- 删除归纳变量

在循环中变量i每变化一次，变量T的赋值跟随进行线性变化，变量i称为归纳变量。当循环外有跟i相关的条件判断时，且i在其他地方不再被引用，则可以用变量T替代i用作条件判断。

例如： if i>j goto B6

i相关的线性变量为T2，j相关的线性变量为T4，

则该条件判断可以改为:

if T2>T4 goto B6

9.1 概述

- 复写传播

如果T6:=T2，且T6在下一步使用前没有改变值，则把x:=a[T6]变换为x:=a[T2]，这种变换称为复写传播。

- 删除无用代码(删除无用赋值)

由于进行了复写传播，T6将不再被程序中任何地方使用，这些变量的赋值对程序没有任何作用，可以删除这些变量赋值的代码，例如删除T6。

9.2 局部优化

第九章代码优化与代码生成 9.2 局部优化

局部优化

9.2 局部优化

- 基本名词定义
- **基本块**：指程序中一**顺序执行**语句序列，其中**只有**一个入口和一个出口。入口就是其中第一个语句，出口就是其中最后一个语句。
- **定值、引用**：如果一条三地址语句为x:=y+z，则称对x**定值**并**引用**y和z。
- **活跃**：基本块中的一个名字，所谓在程序中的某个给定点是**活跃的**，是指如果在程序中(包括在本基本块或在其它基本块中)它的值在该点以后被引用。

9.2 局部优化

- 局部优化

  局限于基本块范围内的优化称为基本块内的优化，或称局部优化。

  优化方式

  ① 删除公共子表达式
  ② 删除无用代码(删除无用赋值)
  ③ 合并已知量
  ④ 临时变量改名
  ⑤ 交换语句的位置
  ⑥ 代数变换

9.2 局部优化

- 优化方式

③ **合并已知量**：T_1:=2….T_2:=4*T_1，如果T_1赋值后没有再改变过，则可以在编译时计算T_2的值，减少程序运行时的计算量。例如，可以把T_2：=8。

④ **临时变量改名**：基本块里某个语句为：T:=b+c，其中T为临时变量。把这个语句改为:S:=b+c，其中S是新的临时变量。这样的改变不会改变基本块的值，同时可以把基本块中的所有T变为S。

9.2 局部优化

- 优化方式

⑤ **交换语句的位置**：如果基本块中有两个语句：T_1:=b+c；T_2:=x+y；如果这两个语句中的b和c、x和y分别与T_2、T_1不相关，则可以交换这两个语句的位置不影响基本块的值。

⑥ **代数变换**：如果有语句x:=x+0或x:=x*1，这样的计算没有意义，计算不会改变x的值，这样的计算可以删除。

基本块

9.2 局部优化

- 基本块的划分

对四元式按规则进行划分，规则有**三步**：

1. 求出四元式程序中各个基本块的**入口语句**，它们是：

（1）程序的**第一个**语句；

（2）能由条件转移语句或无条件转移语句**转移到的语句**；

（3）紧跟在条件转移语句**后面的语句**。

9.2 局部优化

- 基本块的划分

对四元式按规则进行划分，规则有三步：

2. 构造基本块。由该入口语句到另一入口语句，或到转移语句(包含)，或到停语句(包含)之间的语句序列组成。

3. 未被纳入基本块的语句，都是控制流程无法达到的语句，可以从程序中删除。**删除非基本块语句**。

9.2 局部优化

- 基本块的划分

例如：三地址代码程序如下

(1) read X
(2) read Y
(3) R:=X mod Y
(4) if R=0 goto (8)
(5) X:=Y
(6) Y:=R
(7) goto (3)
(8) write Y
(9) halt

（1）程序的第一个语句；
入口: (1)

（2）能由条件转移语句或无条件转移语句转移到的语句；
入口: (8), (3)

（3）紧跟在条件转移语句后面的语句。
入口: (5)

9.2 局部优化

- 基本块的划分

  例如：三地址代码程序如下

(1) read X
(2) read Y
(3) R:=X mod Y
(4) if R=0 goto (8)
(5) X:=Y
(6) Y:=R
(7) goto (3)
(8) write Y
(9) halt

由该入口语句到另一入口语句，或到转移语句(包含)，或到停语句(包含)之间的语句序列组成。

入口: (1), (3), (5), (8)

基本块: (1) (2), (3) (4), (5) (6) (7), (8) (9)

27

流图

9.2 局部优化

- 流图

每个流图以基本块为**结点**。

如果一个结点的基本块的入口语句是程序的第一条语句，则称此结点为**首结点**。

如果在某个执行顺序中，基本块B_2紧接在基本块B_1之后执行，则从B_1到B_2有一条**有向边**。

9.2 局部优化

- 流图

基本块之间的顺序由以下两个条件决定：

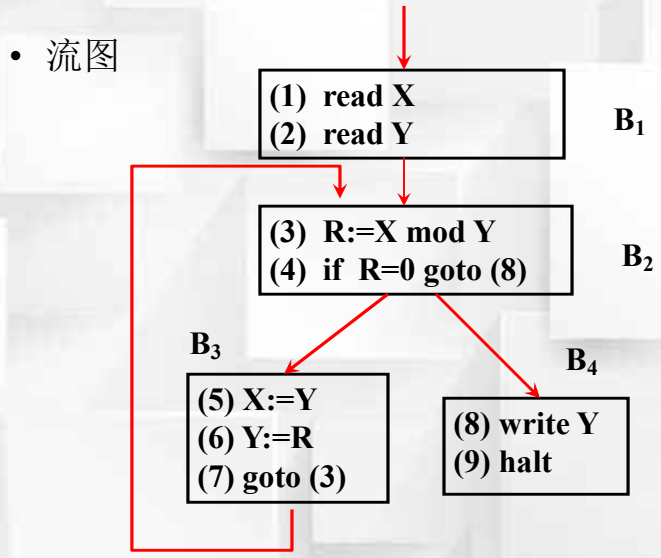
① 有一个条件或无条件转移语句从B_1的最后一条语句转移到B_2的第一条语句；

② 在程序的序列中，B_2紧接在B_1的后面，并且B_1的最后一条语句不是一个无条件转移语句。

B_1是B_2的**前驱**，B_2是B_1的**后继**。

9.2 局部优化

- 流图



31

DAG有向无环图
Directed Acyclic Graph

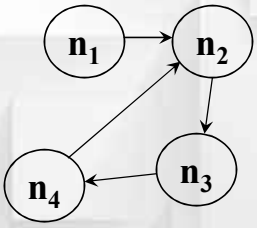
9.2 局部优化

- DAG有向无环图

  对表达式中的每个子表达式，DAG中都有一个结点；

  一个内部结点代表一个操作符，它的孩子代表操作数；

  在一个DAG中代表公共子表达式的结点具有多个父结点。

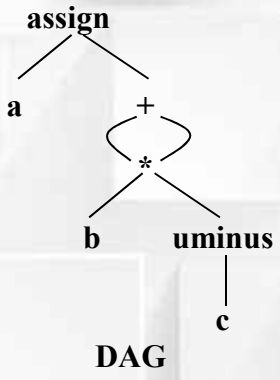


33

9.2 局部优化

- DAG有向无环图

  例如：a:=b*(-c)+b*(-c)的DAG图



DAG

DAG对应的代码：

T_1:=-c

T_2:=b*T_1

T_3:=T_2+T_2

a:=T_3

34

9.2 局部优化

- DAG有向无环图

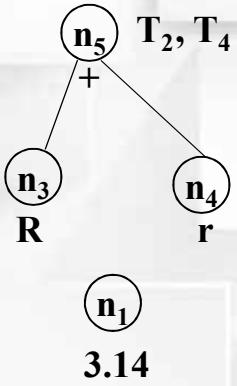
  图的**叶结点**以一标识符或常数作为标记，表示该结点代表该变量或常数的值;

  图的**内部结点**以一运算符作为标记，表示该结点代表应用该运算符对其后继结点所代表的值进行运算的结果;

  图中各个结点上可能附加**一个或多个标识符**(称附加标识符)表示这些变量具有该结点所代表的值。

9.2 局部优化

- DAG有向无环图



内部结点n_5以运算符+为标记，表示n_3+n_4的值，该值存在T_2和T_4中

叶结点n_3和n_4代表变量R, r的值

叶结点n_1代表常数3.14

9.2 局部优化

- DAG有向无环图

一个基本块，可用一个DAG来表示与各四元式相对应的DAG结点形式。

中间代码具有以下3型：

0型: A:=B
(:=，B，-，A)

2型: A:=B op C
(op，B，C，A)

1型: A:=op B
(op，B，-，A)

2型: A:=B[C]
(=[]，B[C]，-，A)

9.2 局部优化

- DAG有向无环图

假设DAG各结点信息将用某种适当的数据结构存放(如链表)。另设置一个标识符与结点的对应函数:

$$Node(A)=\begin{cases} n & \text{如果DAG存在一个结点n，A是该结点的标记，}\textbf{有定义} \\ null & \text{如果DAG不存在标记为A的结点，}\textbf{无定义} \end{cases}$$

9.2 局部优化

- 构造DAG有向无环图的算法

使用0，1，2型四元式构造DAG的算法

对基本块中每一四元式，依次执行以下步骤:

1. 准备操作数的结点
2. 合并已知量
3. 删除公共子表达式
4. 删除无用赋值

9.2 局部优化

- 构造DAG有向无环图的算法

1. 准备操作数的结点

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)。

0型: A:=B　1型: A:=op B　2型: A:=B op C　2型: A:=B[C]

9.2 局部优化

- 构造DAG有向无环图的算法

2. 合并已知量(4项操作)

(1) 如果NODE(B)是标记为**常数**的叶结点，则转2(3)；否则，转3(1)。

1型: A:=op B，转2(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)。

2型: A:=B op C　2型: A:=B[C]，转2(2)

41

9.2 局部优化

- 构造DAG有向无环图的算法

2. 合并已知量(4项操作)

(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作标记的叶结点n。置NODE(P)=n，转4。

1型: A:=op B，若B为标记好的常数，转2(3)

42

9.2 局部优化

- 构造DAG有向无环图的算法

  2. 合并已知量(4项操作)

  (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作标记的叶结点n。置NODE(P)=n，转4。

2型: A:=B op C 若B和C都为常数	2型: A:=B[C] 若B和C都为常数，转2(4)

9.2 局部优化

- 构造DAG有向无环图的算法

  3. 寻找公共子表达式(2项操作)

  (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(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。

1型: A:=op B，若B不是常数，转3(1)

9.2 局部优化

- 构造DAG有向无环图的算法

3. 寻找公共子表达式(2项操作)

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

2型: A:=B op C

2型: A:=B[C]，若B和C都不为常数，转3(2)

9.2 局部优化

- 构造DAG有向无环图的算法

4. 删除无用赋值

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)结点上的附加标识符集中删除(注意，如果NODE(A)是叶结点，则其A标记不删除)。把A附加到新结点n上并置NODE(A)=n。转处理下一四元式。

0型: A:=B

1型: A:=op B

2型: A:=B op C

2型: A:=B[C]

9.2 局部优化

- 例题：试构造以下基本块G的DAG

(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

0型: A:=B　1型: A:=op B

2型: A:=B op C　2型: A:=B[C]

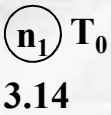
47

9.2 局部优化

- 例题：试构造以下基本块G的DAG

(1) T_0:=3.14

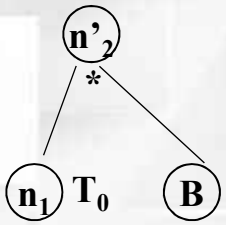
0型: A:=B，记NODE(B)的值为n，转4
4. 如果NODE(A)无定义，则把A附加在结点n上并令NODE(A)=n；T_0=n=3.14



48

9.2 局部优化

(2) $T_1:=2*T_0$



2型: A:=B op C

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;

(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；

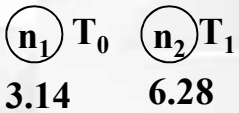
(ii)转2(2)。

2 (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)。

2 (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作标记的叶结点n。置NODE(P)=n，转4。

9.2 局部优化

(2) $T_1:=2*T_0$



2型: A:=B op C

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;

(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)。

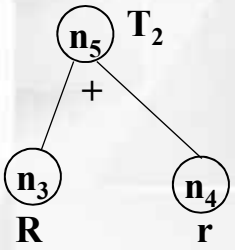
2 (2) 如果NODE(B)和NODE(C)都是标记为常数的叶结点，则转2(4)；否则，转3(2)。

2 (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作标记的叶结点n。置NODE(P)=n，转4。

9.2 局部优化

- 例题：试构造以下基本块G的DAG

(3) T_2:=R+r

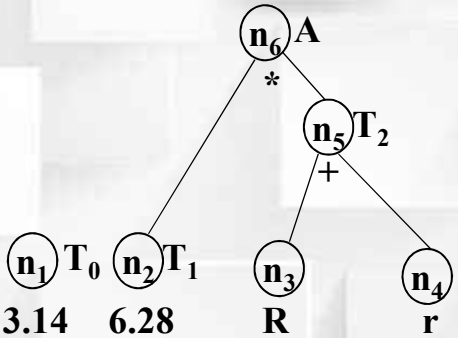


2型: A:=B op C
1(i)如果NODE(C)无定义，则构造一标记为C的叶结点并定义NODE(C)为这个结点；
(ii)转2(2)。
2 (2) 如果NODE(B)和NODE(C)都是标记为常数的叶结点，则转2(4)；否则，转3(2)。
3 (2) 检查DAG中是否已有一结点，其左后继为NODE(B)，右后继为NODE(C)，且其本身被标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。

9.2 局部优化

- 例题：试构造以下基本块G的DAG

(4) A:=T_1*T_2

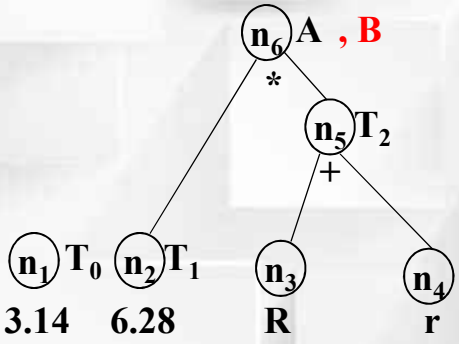


2型: A:=B op C
1(i)如果NODE(C)无定义，则构造一标记为C的叶结点并定义NODE(C)为这个结点；
(ii)转2(2)。
2 (2) 如果NODE(B)和NODE(C)都是标记为常数的叶结点，则转2(4)；否则，转3(2)。
3 (2) 检查DAG中是否已有一结点，其左后继为NODE(B)，右后继为NODE(C)，且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。

9.2 局部优化

- 例题：试构造以下基本块G的DAG

(5) B:=A

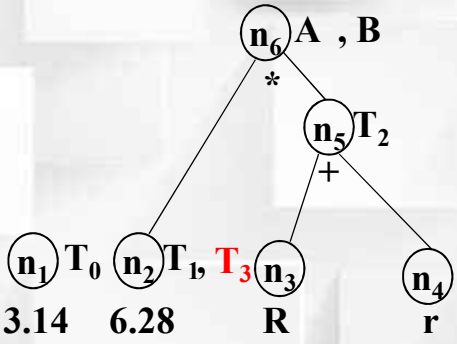


0型：A:=B，记NODE(B)的值为n，转4

4. 如果NODE(A)无定义，则把A附加在结点n上并令NODE(A)=n；B=A

9.2 局部优化

(6) T_3:=2*T_0



2型: A:=B op C

1(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；

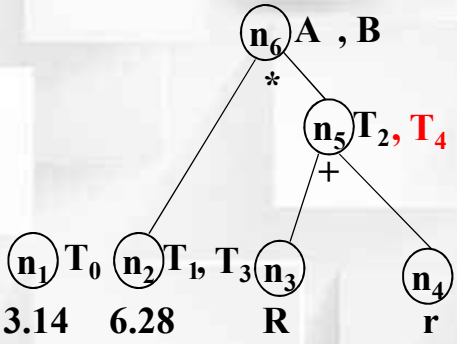
(ii)转2(2)。

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

9.2 局部优化

(7) T_4:=R+r

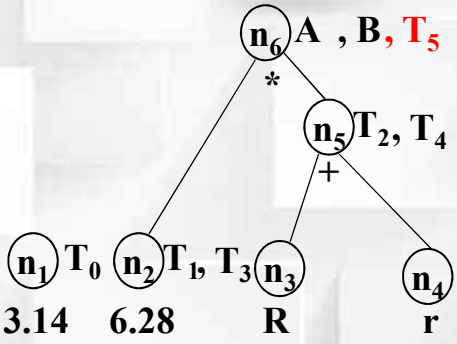


2型: A:=B op C
1(i)如果NODE(C)无定义，则构造一标记为C的叶结点并定义NODE(C)为这个结点；
(ii)转2(2)。
2 (2) 如果NODE(B)和NODE(C)都是标记为常数的叶结点，则转2(4)；否则，转3(2)。
3 (2) 检查DAG中是否已有一结点，其左后继为NODE(B)，右后继为NODE(C)，且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。

55

9.2 局部优化

(8) T_5:=T_3*T_4

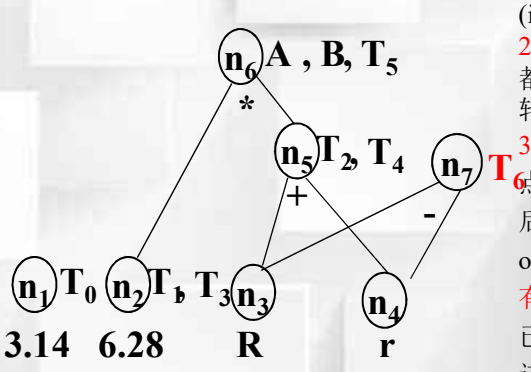


2型: A:=B op C
1(i)如果NODE(C)无定义，则构造一标记为C的叶结点并定义NODE(C)为这个结点；
(ii)转2(2)。
2 (2) 如果NODE(B)和NODE(C)都是标记为常数的叶结点，则转2(4)；否则，转3(2)。
3 (2) 检查DAG中是否已有一结点，其左后继为NODE(B)，右后继为NODE(C)，且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。

56

9.2 局部优化

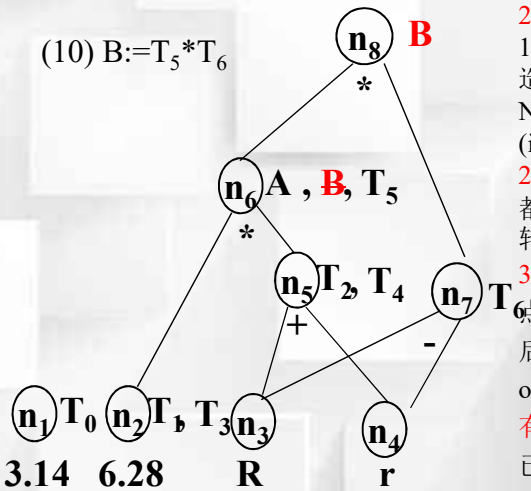
(9) T_6:=R-r



2型: A:=B op C
1(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)。
2 (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 (2) 检查DAG中是否已有一结点，其左后继为NODE(B)，右后继为NODE(C)，且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。

9.2 局部优化

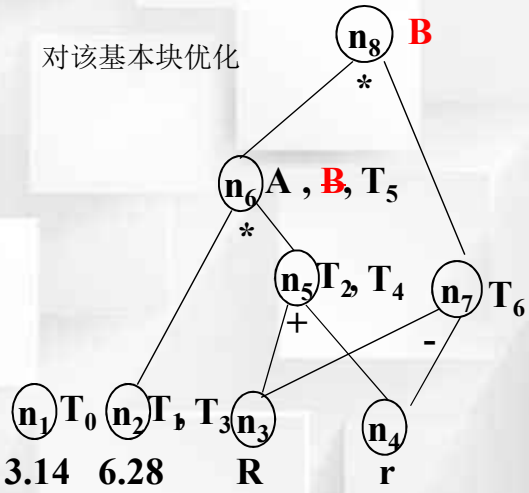
(10) B:=T_5*T_6



2型: A:=B op C
1(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)。
2 (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 (2) 检查DAG中是否已有一结点，其左后继为NODE(B)，右后继为NODE(C)，且标记为op(即公共子表达式)。如果没有，则构造该结点n，否则，把已有的结点作为它的结点并设该结点为n。转4。

9.2 局部优化

对该基本块优化



优化后：

(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

9.2 局部优化

基本块中的一个名字，所谓在程序中的某个给定点是**活跃的**，是指如果在程序中(包括在本基本块或在其它基本块中)它的值在该点以后被引用。

这里只看T_0, T_1, T_2, T_3, T_4, T_5, T_6, 这里假设它们只在当前基本块定义和使用，而A和B在后续基本块可能有使用。

优化后活跃的标识符：

(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

9.2 局部优化

- 活跃的标识符
- 假如T_0, T_1, T_2, T_3, T_4, T_5, T_6在基本块以后都不会被引用，可以删除其中重复的赋值，得到新的代码为：

优化前活跃的标识符：

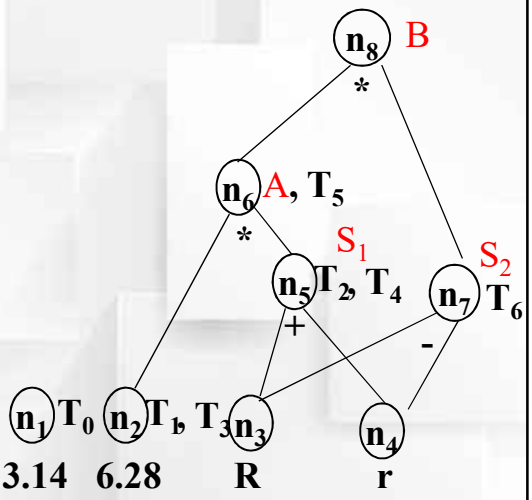
(1) T_0:=3.14
(2) T_1:=6.28
(3) T_3:=6.28
(4) T_2:=R+r
(5) T_4:=T_2
(6) A:=6.28*T_2
(7) T_5:=A
(8) T_6:=R-r
(9) B:=A*T_6

优化后活跃的标识符：

- (1) S_1:=R+r
  (2) A:=6.28*S_1
  (3) S_2:=R-r
  (4) B:=A*S_2

9.2 局部优化

- 活跃的标识符
- 在基本块外被定值并在基本块内被引用的所有标识符，就是作为叶子结点上标记的那些标识符。
- 在基本块内被定值并且该值在基本块后面可以被引用的所有标识符，就是DAG各结点上的那些附加标识符。



9.3 目标代码生成

9.3 目标代码生成

- 输入：中间代码序列

  三地址代码、语法结构树、后缀式

- 输出：目标代码

  绝对机器代码、可重定位机器指令代码、汇编指令代码

- 目标机

  多通用寄存器、控制栈、堆

9.3 目标代码生成

- 质量要求
  目标代码效率高
  目标程序短
- 实现方法
  充分利用寄存器
  参考目标机的结构与指令形式

9.3 目标代码生成

- 代码生成
  计算各变量的的引用次数，据此分配寄存器
  对于每条三地址代码，参照目标机允许的指令形式，进行翻译
  例

x := y op z => mov R_i, y
op R_i, z
mov x, R_i

9.3 目标代码生成

- 指令种类（此处版本与书上不一致，请大家以书上的指令为准）

  赋值 MOV

  比较 CMP

  条件转移 JLE(Jump if Less or equal than)汇编语言中的条件转移指令。小于或等于，或者不大于则转移

  转移 JMP

  累加 ADD

  JG(Jump if Greater than) 若大于则转移
  JGE (Jump if Greater or equal than)若大于等于则转移

9.3 目标代码生成

- 三地址代码生成目标代码（此处版本与书上不一致，请大家以书上的指令为准）

(1) t_1 := z * 6
(2) x := -4
(3) if x >= 76 goto (7)
(4) x := x + 4
(5) y := t_1 + x
(6) goto (3)

(1) MOV R_0, z
(2) MUL R_0, 6
(3) MOV R_1, R_0
(4) MOV R_2, -4
(5) CMP R_2, 76
(6) JGE (12)
(7) ADD R_2, 4
(8) MOV R_0, R_1
(9) ADD R_0, R_2
(10) MOV Y, R_0
(11) JMP (5)

一般寄存器R_0
专用寄存器t_1=R_1
专用寄存器x=R_2

第九章 小结

第九章 小结

- 9.1 概述
- 9.2 局部优化
- 9.3 目标代码生成

Coursework

- 9.1试把以下程序划分为基本块并作出其程序流图。

```
read A，B
F：= 1
C：= A * A
D：= B * B
if C < D goto L1
E：= A * A
F：= F + 1
E：= E + F
write E
halt
L1：E := B * B
    F：= F + 2
    E：= E + F
    write E
    if E > 100 goto L2
    halt
L2：F：= F - 1
    goto L1
```


Coursework

- 9.2 试对以下基本块B_1和B_2分别应用DAC对它们进行优化，并就以下两种情况分别写出优化后的四元式序列：
  (1) 假设只有G，L，M在基本块后面还要被引用；
  (2) 假设只有L在基本块后还要被引用。

```
B1：A：= B * C
    D: =B / C
    E：= A + D
    F：= 2 * E
    G：= B * C
    H：= G * G
    F：= H * G
    L：= F
    M：= L
B2：B：=3
    D：= A + C
    E：= A * C
    F：= D+E
    G：= B * F
    H：= A + C
    I：= A * C
    J：= H + I
    K：=B * 5
    L：= K + J
    M：= L
```

Coursework

- 9.3 设有算术表达式 a+b*c-(c*b+a-e)，要求：

  （1）请写出该表达式的四元式中间代码；

  （2）将上述四元式中间代码理解成一个基本块，构造该基本块所对应的DAG图；

  （3）由DAG图重新产生该表达式优化后的四元式中间代码。