编译原理

局部优化

优化的级别

- 局部优化
- 循环优化
- 全局优化

优化的级别

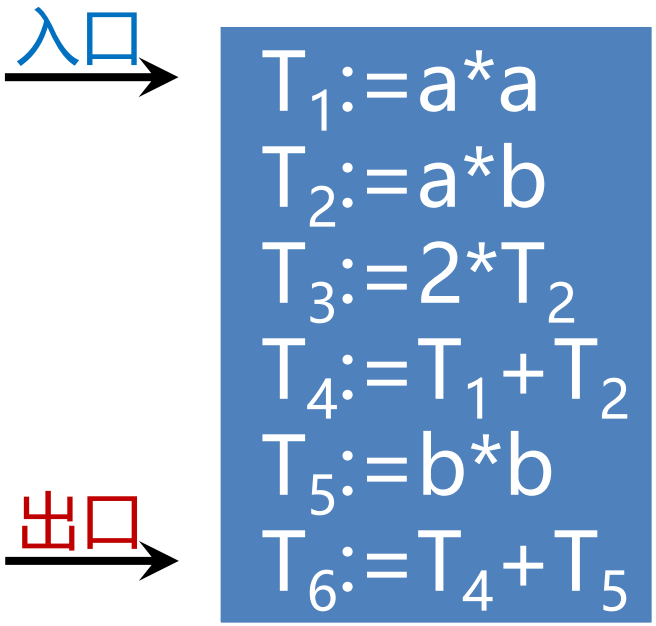
- 局部优化
  - 局限于基本块范围内的优化
- 循环优化
- 全局优化

编译原理

基本块

基本块

- 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。



- 对三地址语句为x:=y+z，称对x定值并引用y和z
- 基本块中的一个名字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

编译原理

基本块划分算法

划分基本块

- 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
- 如何将程序划分成基本块？
  - 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。

划分基本块的算法

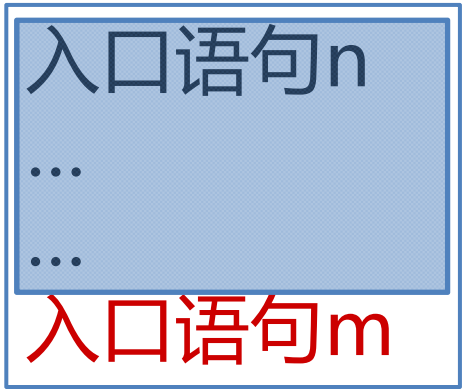
1. 找出中间语言(三地址语句)程序中各个基本块的入口语句:
   1) 程序第一个语句，或
   2) 能由条件转移语句或无条件转移语句转移到的语句，或
   3) 紧跟在条件转移语句后面的语句

划分基本块的算法

2. 对以上求出的每个入口语句，确定其所属的基本块。

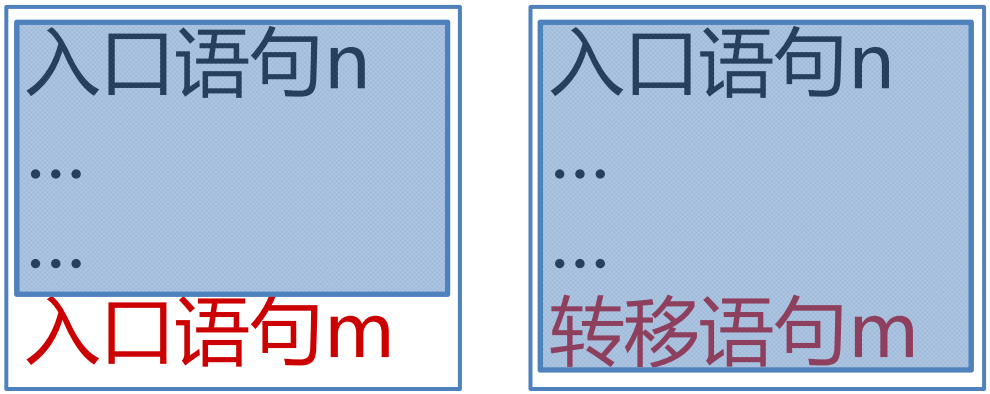
划分基本块的算法

2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句(不包括该入口语句) 之间的语句序列组成的



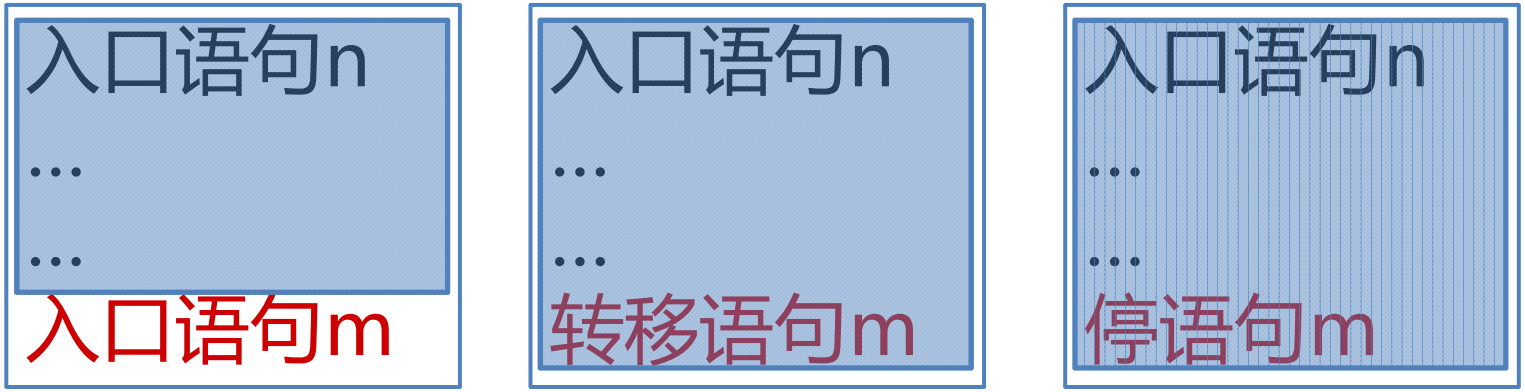
划分基本块的算法

2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句(不包括该入口语句)、或到一转移语句(包括该转移语句) 之间的语句序列组成的



划分基本块的算法

2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句(不包括该入口语句)、或到一转移语句(包括该转移语句)、或一停语句(包括该停语句)之间的语句序列组成的。



划分基本块的算法

2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句(不包括该入口语句)、或到一转移语句(包括该转移语句)、或一停语句(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中的语句，可以从程序中删除。

划分基本块示例

```
(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.求出四元式程序中各个基本块的入口语句:

1) 程序第一个语句，或

2) 能由条件转移语句或无条件转移语句转移到的语句，或

3) 紧跟在条件转移语句后面的语句

划分基本块示例

```
(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
```

2. 对以上求出的每个入口语句，确定其所属的基本块。它是由该入口语句到下一入口语句(不包括该入口语句)、或到一转移语句(包括该转移语句)、或一停语句(包括该停语句)之间的语句序列组成的。

划分基本块示例

```
(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
```

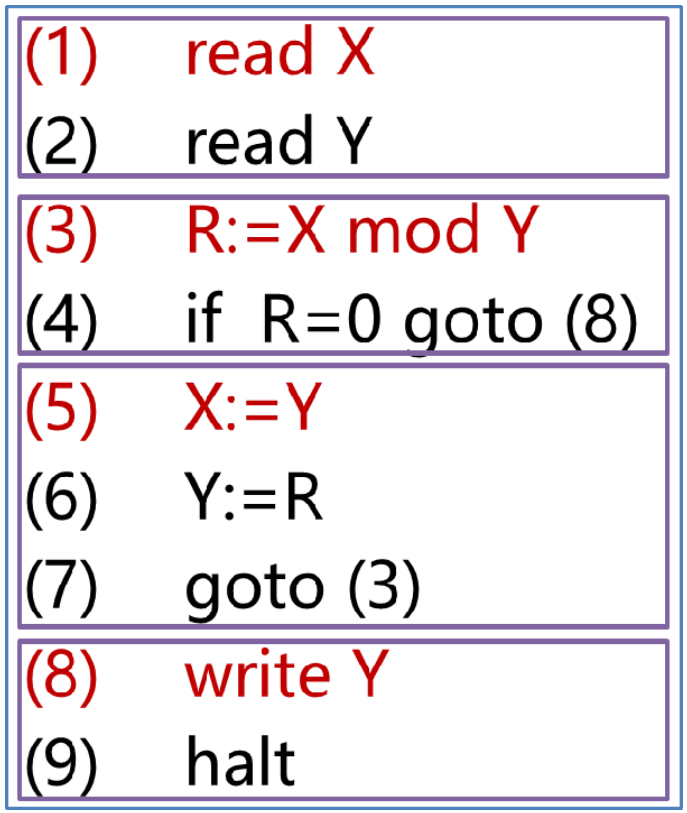
3.凡未被纳入某一基本块中的语句，可以从程序中删除。

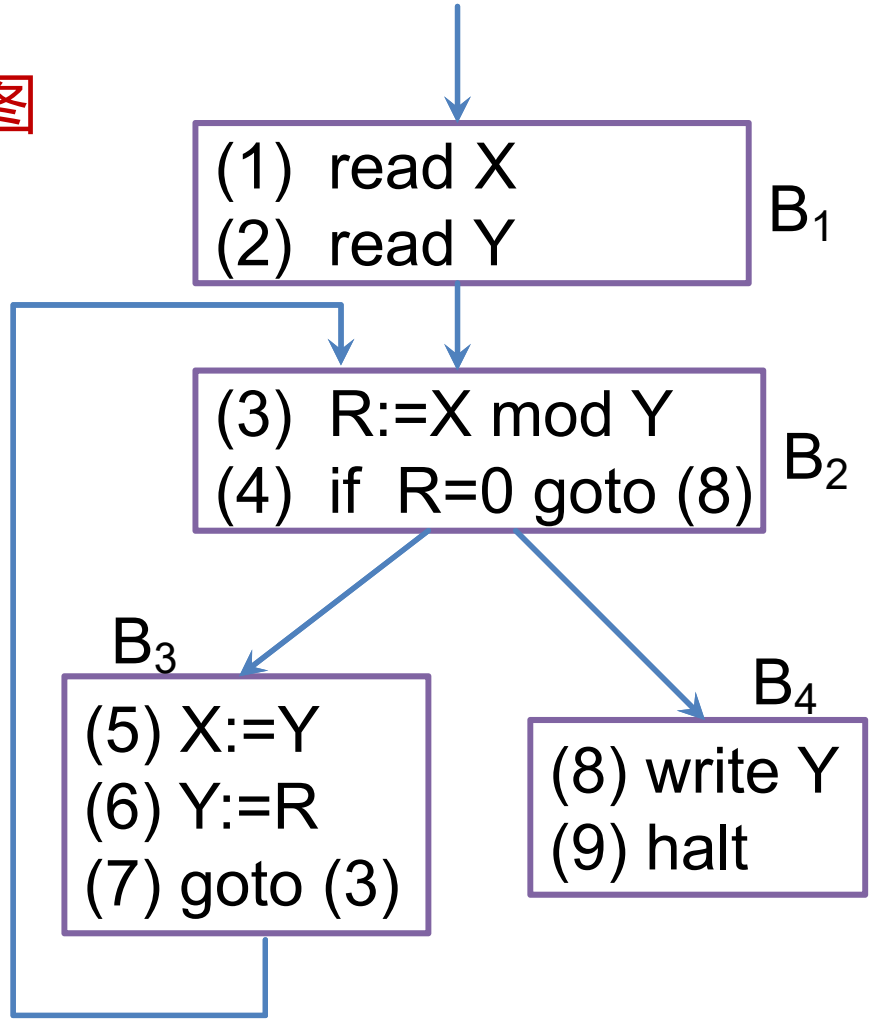
编译原理

流图

流图

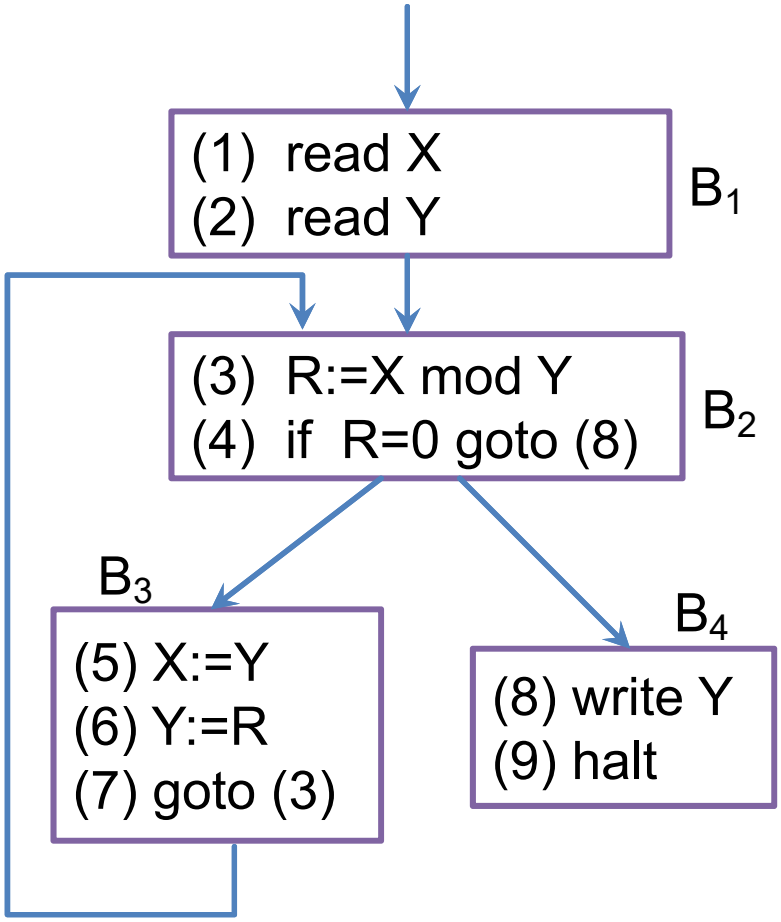
- 以基本块为结点构成流图





流图

- 以基本块为结点构成流图
- 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（B_1是B_2的前驱，B_2是B_1的后继）。
  - 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；或者
    - 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。

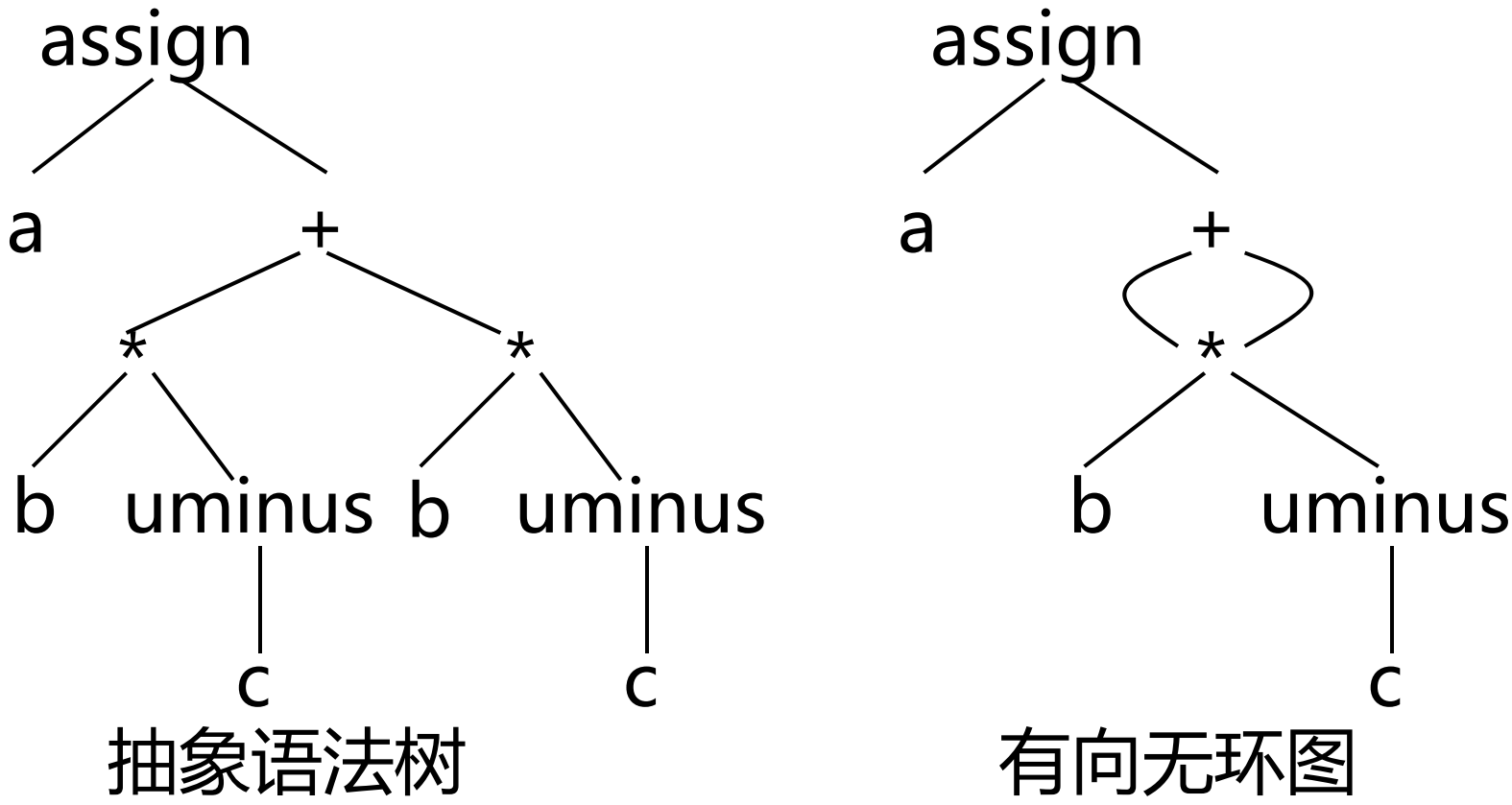


编译原理

基本块的DAG表示

抽象语法树 vs. 有向无环图

- a:=b*(-c)+b*(-c)的图表示法

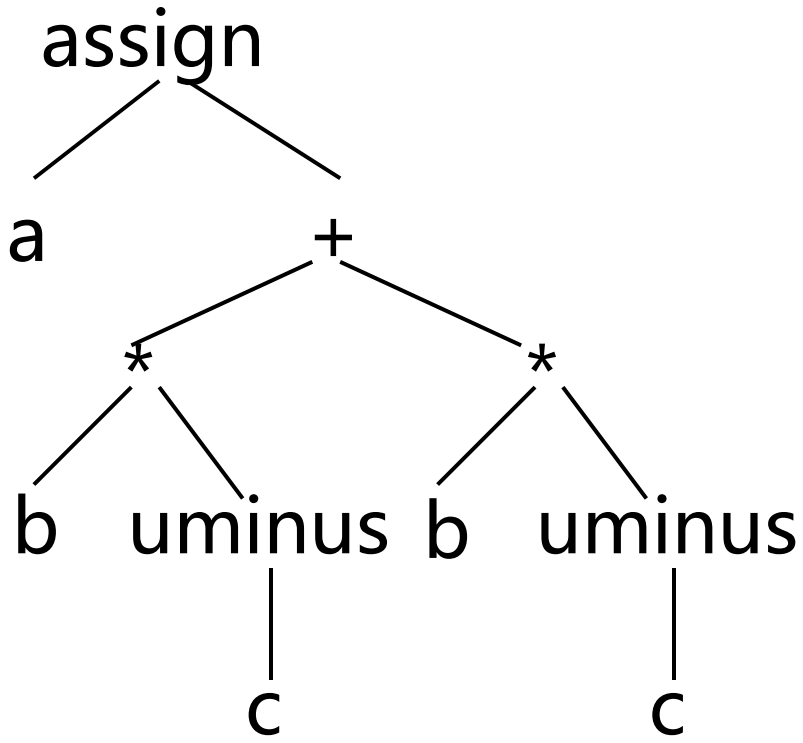


抽象语法树 vs. 三地址代码

- a:=b*(-c)+b*(-c)的图表示法

抽象语法树对应的三地址代码:

$T_1 := -c$
$T_2 := b * T_1$
$T_3 := - c$
$T_4 := b * T_3$
$T_5 := T_2 + T_4$
$a := T_5$



抽象语法树

有向无环图 vs. 三地址代码

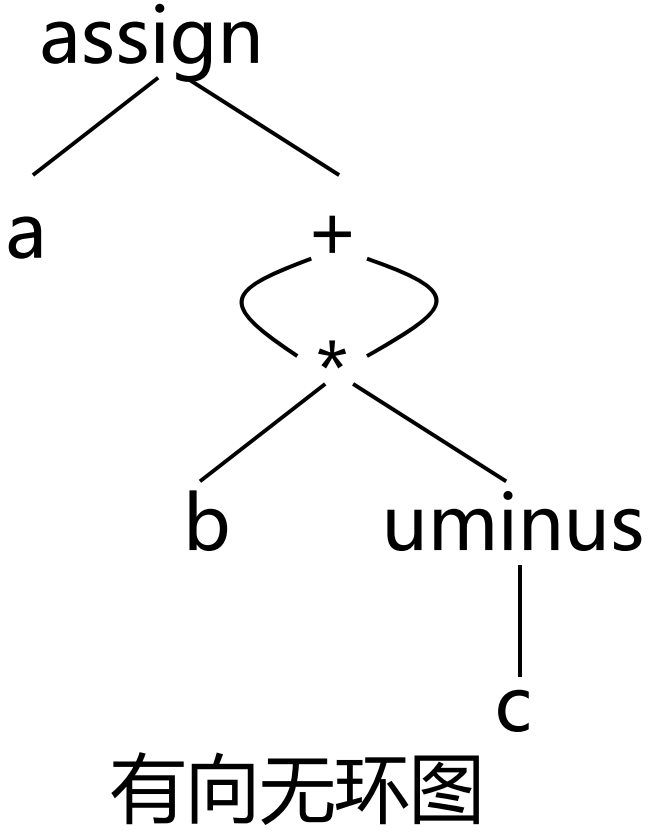
- a:=b*(-c)+b*(-c)的图表示法

抽象语法树对应的三地址代码：

$T_1 := -c$
$T_2 := b * T_1$
$T_3 := -c$
$T_4 := b * T_3$
$T_5 := T_2 + T_4$
$a := T_5$

有向无环图对应的三地址代码：

$T_1 := -c$
$T_2 := b * T_1$
$T_5 := T_2 + T_2$
$a := T_5$



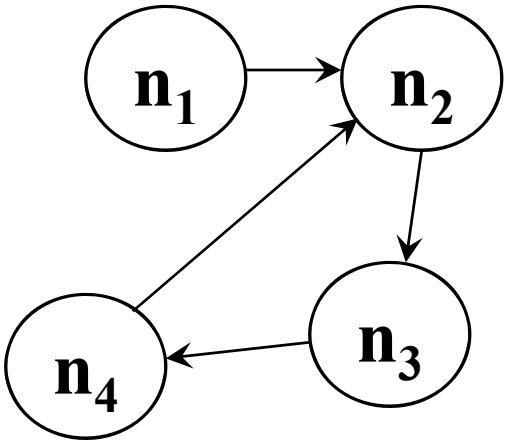
有向无环图

有向无环图(DAG)

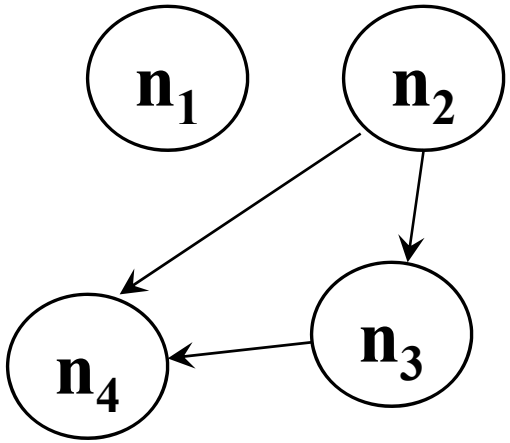
- 有向图(Directed Graph)
  - 有向边: $n_i \rightarrow n_j$
  - 前驱：n_i是n_j的前驱
  - 后继： n_j是n_i的后继
  - 通路: $n_1 \rightarrow n_2$, $n_2 \rightarrow n_3$,… , $n_{k-1} \rightarrow n_k$
  - 环路： $n_1 = n_k$
- 有向无环图(Directed Acyclic Graph，简称DAG)
  - 没有环路

测试

- 下面图中哪个是有向无环图(DAG)?



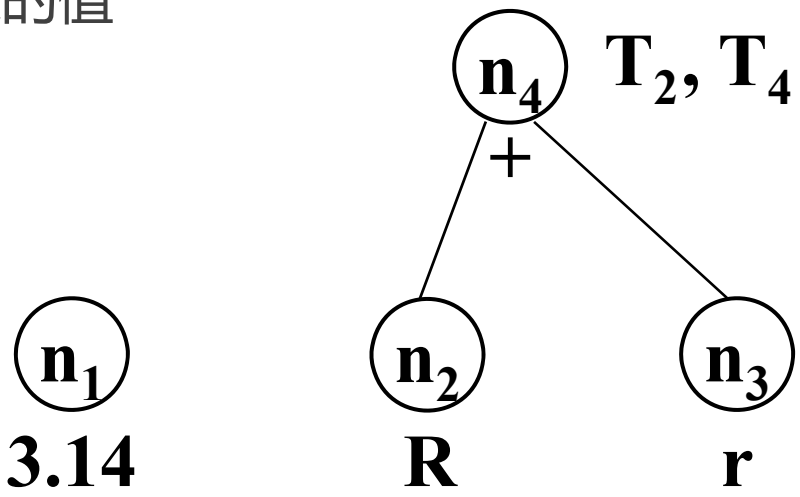
A



B

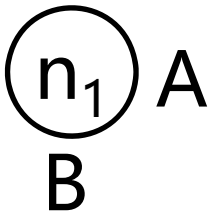
DAG的扩充

- 在DAG增加标记和附加信息
  - 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
  - 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



四元式的DAG表示

- 与各四元式相对应的DAG结点形式:

四元式	DAG 图
(0) 0型: A:=B (:=, B, -, A)	

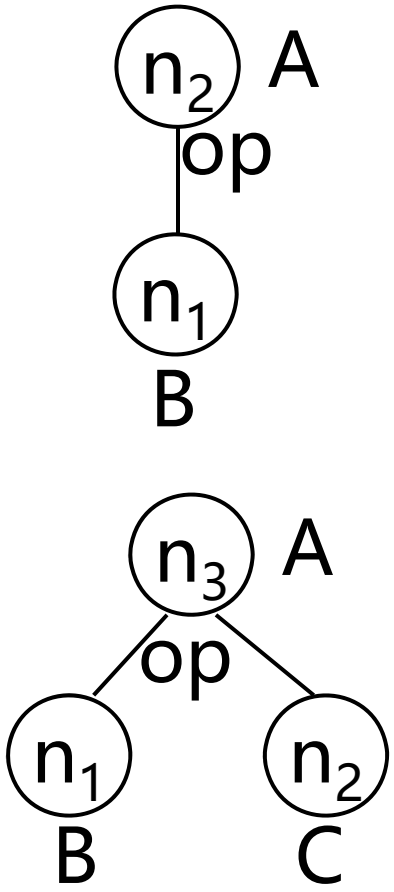
四元式的DAG表示

四元式

(1) 1型: A:=op B
(op，B，-，A)

(2) 2型: A:=B op C
(op，B，C，A)

DAG 图

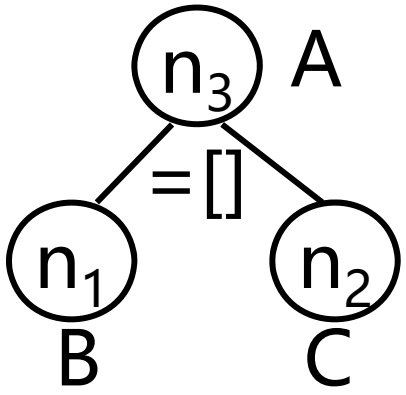


四元式的DAG表示

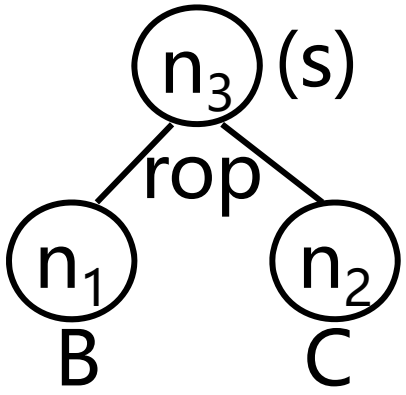
四元式

DAG 图

(3) 2型: A:=B[C]
(=[]，B，C，A)



(4) 2型: if B rop C goto (s)
(jrop，B，C，(s))

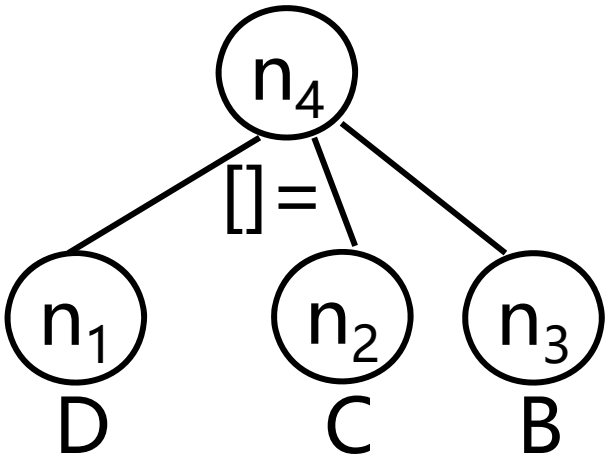


四元式的DAG表示

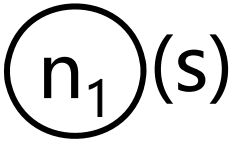
四元式

(5) 3型: D[C]:=B
([]=，B，D，C)

DAG 图



(6) 0型: goto (s)
(j，-，-，(s))



编译原理

基本块的优化算法

基本块的优化算法

- 一个基本块，可用一个DAG来表示
- 对基本块中每一条四元式代码，依次构造对应的DAG图，最后基本块中所有四元式构造出来DAG连成整个基本块的DAG。

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

构造基本块的DAG示例

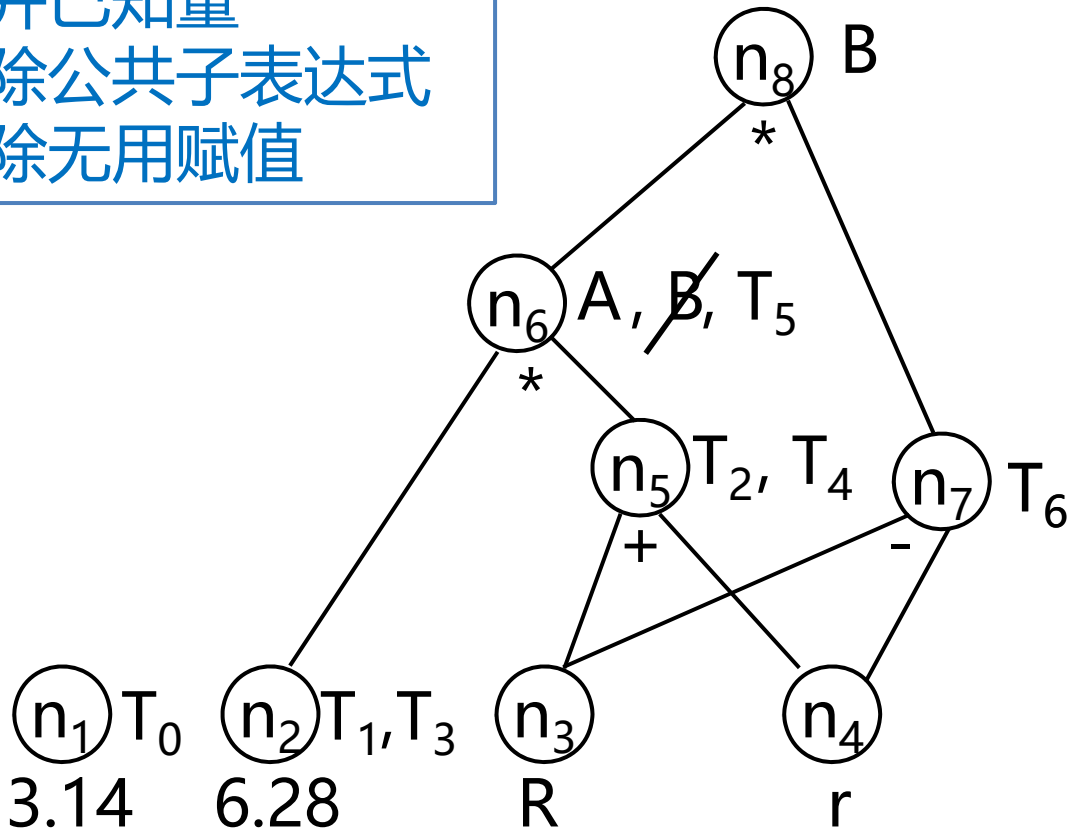
(1) $T_0:=3.14$
(2) $T_1:=2*T_0$
(3) $T_2:=R+r$
(4) $A:=T_1*T_2$
(5) $B:=A$
(6) $T_3:=2*T_0$
(7) $T_4:=R+r$
(8) $T_5:=T_3*T_4$
(9) $T_6:=R-r$
(10) $B:=T_5*T_6$

1. 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

构造基本块的DAG示例

(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. 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

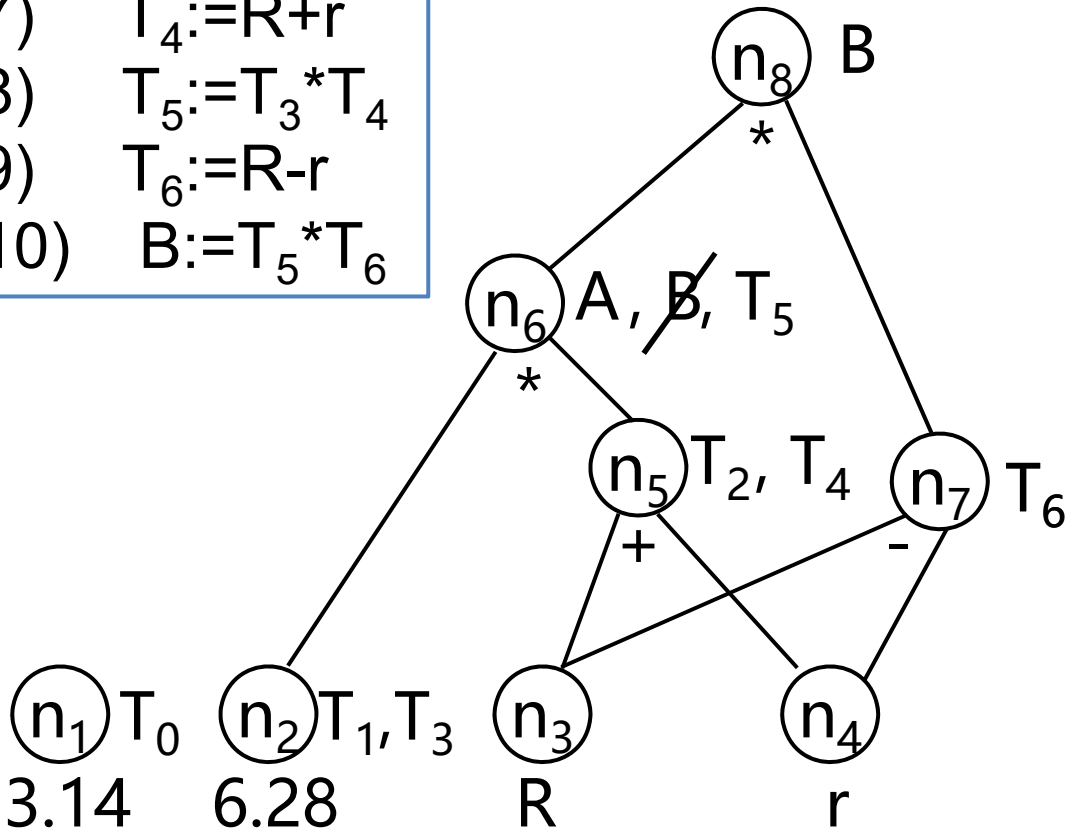


构造基本块的

- 优化后的四元式

(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

(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



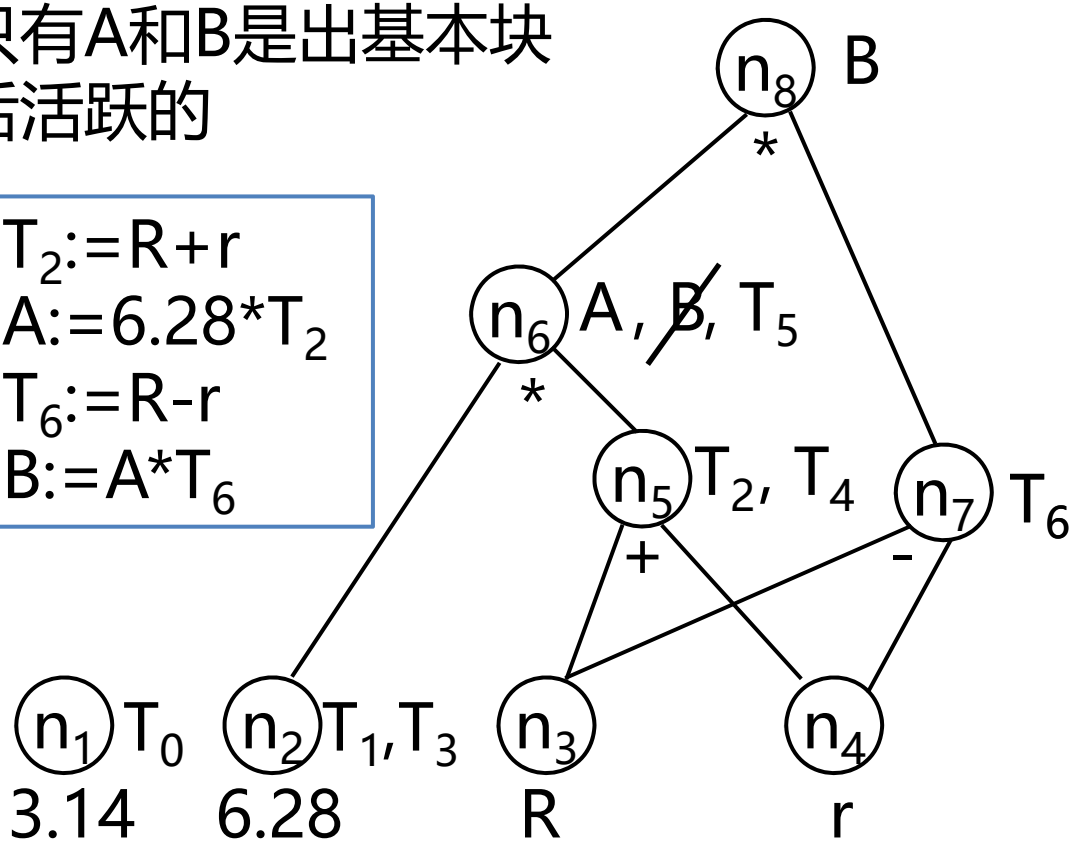
构造基本块的DAG示例

- 优化后的四元式

(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

若只有A和B是出基本块之后活跃的

(1) T_2:=R+r
(2) A:=6.28*T_2
(3) T_6:=R-r
(4) B:=A*T_6



构造基本块的DAG的算法

- 引入了一个函数Node，保存和计算DAG中标识符与结点的对应关系

$$Node(A) = \begin{cases} n, & \text{如果存在一个结点}n\text{，} \\ & A\text{是其上的标记或者附加标识符} \\ null, & \text{否则} \end{cases}$$

0,1,2型四元式的基本块的DAG构造算法

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

0,1,2型四元式的基本块的DAG构造算法

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)。

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

0,1,2型四元式的基本块的DAG构造算法

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作标记的叶结点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作标记的叶结点n。置NODE(P)=n，转4

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

0,1,2型四元式的基本块的DAG构造算法

3. 删除公共子表达式

(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) 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。

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

0,1,2型四元式的基本块的DAG构造算法

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。

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

构造基本块的DAG示例

(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. 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

从DAG中得到的优化信息

- 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各结点上的那些标记或者附加标识符

小结

- 局部优化和基本块
- 基本块的划分
- 基本块的DAG表示
- 基本块的DAG构造算法