Loop finding

1. Dominator finding

根据控制流图（CFG）利用公式求出每一个节点 b 的 dominator

Formula：out[b] = {b} ∪ (∩ { p=pred(b) } out[p]) (即 p 是 b 的 direct dominator)

2. Backedge finding

①形成深度优先搜索树

②对于每一个 retreating edge t->h，检查 h 是不是 t 的 dominator，如果是就是 backedge

（一般情况下在 reducible flow graph 中均满足 retreating edge = backedge）

3. Constructing loop

不能从 loop 外跳入 loop，即除 header（唯一 entry）有 predecessor 之后其余不可有。

对于每一个 backedge t->h： delete h，找到一些节点可以到达 t，使得加上 h 之后可以形成 t->h 的 loop。

loop invariant (INV)

一、inv 的种类

1. basic case:

（1）常量

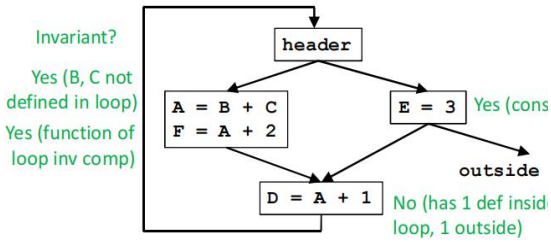
（2）变量使用：该变量的所有定义在 loop 外

2. inductive case:

（1）由不变量组成的表达式

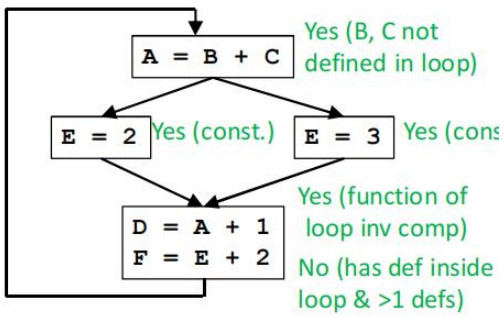
（2）变量使用：该变量在 loop 内只有一个到达定义，且该定义的右边是 loop invariant。

Example 1:



“D = A + 1”中 A 的到达定义，一个在 loop 内(经过 hearder 的左分支),另一个在 loop 外（经过 header 的右分支）。表现为此节点处变量 A 的 def-use path 有两条。

Example 2:



“F = E + 2”中 E 有两个到达定义，一个来自左分支，一个来自右分支。表现为此节点处变量 E 的 def-use path 有两条。

二、def-use(DU) path

- 变量 x 的一个 def-use path（DU path），是从 x 的一次 def 为起点，到下一次 def 之

前 x 被使用的路径。

- 算法思路：

对 GraphNode 中的每个使用变量 x，将节点 id 加入 x 的 du_path 中；

对 GraphNode 中的每个 def 变量 x，x 的上一个 du_path 完成，并新增一条以此节点 id 为起点的 x 的 du_path。

Example：

```
Example:
id 1     x = 3;
id 2     y = 4;
id 3     z = x + 1;
id 4     x = 3;
...
```

在 id 4 节点处，得到 x 的一条 du_path 为{1,3}，路径上第一个元素 1 是它的 def 位置 id 1；并新增 x 的 du_path{4,...}

三、INV 的查找

根据 INV 的类型，先找 basic INV 再找 inductive INV。

1. basic INV（常量和单个变量）

- 算法思路

遍历 loop 每个结点：

常数加入 List<> loop_constant；

对每个使用变量 x，若 x 当前位置的所有 def 都在循环外，则 x 加入 List<> basic_variable_invariant。

2. List<> inductive_invariant

每条语句(=右边),构造 BinaryTree,得到它的所有子表达式 List<> subExpressionList 和所有操作数 List<> operand.

（1） 找循环不变表达式

- 算法思路

对每个子表达式 subExpression：

若它的所有操作数都是 basic INV，那么该表达式为 inductive INV。

（2） 找在循环中仅有一次 def 的不变量 List<> def_inductive_inv

- 算法思路：使用 def-use path，dominatorTree

一条语句的操作数 X，若 X 不是 basic loop inv，则判断它是否为归纳的不变量。

X 是归纳的不变量须同时满足以下条件：

① X 在此 loop 中只有一个 def，且该 def 的右边整体表达式是不变量(DU path)

② X 的 def 所在节点支配该 loop 所有 exists 节点（domunatorTree）

③ 在 loop 中使用了 X 的每个节点 N,N 中 X 的到达定义是且仅是该 def(DU path)

所有归纳不变量 X 放入 Map<variable,expression> def_inductive_inv

四、INV 外提

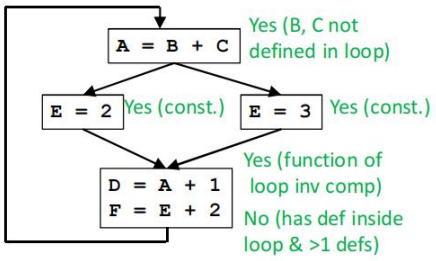
- 基本思路

将”declare temp = inv_code”作为 loop 的 preheader 移到 loop 的 header 前，遍历 loop，替换 inv_code 为 temp，更新 MAP<variable,expression>。

- 几种类型的外提：

1. 常数 c：Type temp = c
2. 子表达式 suExpression：String stemp= subExpression
3. 归纳的单个不变量 X，形如 X = Y + C，其中 Y 和 C 都是不变量。初始<X, B+C>，然后 TEMP = Y+C 外提。外提后：
   ① Ⅰ. <X,B+C>更新为<X, TEMP>
   ② 遍历 loop 节点：
   删除 A = TEMP
   若目标数 D = F(X)，=> D = F(TEMP)
   若 D 也是归纳不变量，更新<D, F(X)> => <D, F(TEMP)>

Example：



=>motion

```
TEMP = B + C;
LOOP:
    A = B + C;
    ...
    D = A + 1;
    F = E + 2;
END LOOP
```

=>transform

```
TEMP = B + C;
LOOP:
    ~~A = TEMP;~~
    //先转换成 A = TEMP，因为整个右侧整个表达式 B + C 外提（TEMP == A.def），且 A 是归纳不变量，所以消除 A = TEMP；更新<A , B+C> => <A, TEMP>
    ...
    D = TEMP + 1;
    // D = A + 1 => D = A.expression + 1，若 D 是归纳不变量，更新<D,expression>
    F = E + 2;
END LOOP
```

五、INV 的生成

针对上面三种类型的代码外提，基于已有的 basic variable inv,生成 inductive inv。

1. 前提：不变量列表 List<> basic_variable_inv 不为空

找 source code 的不变量，以及变量 List<> variable_variant。若没有 basic variable invariant，则在循环外声明一些新的变量，赋值为常数，这些变量加入 List<> basic_variable_inv。形如：

```
declare temp1 = random_number;
declare temp2 = random_number;
```

LOOP:

    ...

LOOP END

2. 生成 inductive invariant

(1) 生成表达式不变量

对 List<> basic_variable_inv 中的 X，Y，以及 List<> variable_variant 中的 P，Q 进行一些运算，在循环外定义新变量，在循环内赋值。形如：

```
declare temp;
LOOP:
    temp = X○Y○P○Q + constant;
```

(2) 生成 inductive variable invariant

对 List<>basic_variable_inv 中的 X,Y 进行一些运算，在循环外定义新变量，在循环内赋值，这个新变量就是 inductive variable invariant。形如：

```
declare temp;
LOOP:
    temp = X○Y + constant;
```

3. 生成的不变量等式放在 loop header 的直接后继位置，目的是保证这些式子支配 loop 的出口。

Induction variable（以下简称 IV）

1. IV 是指其值可以表示成以下变量的函数的任意变量：

① 循环不变量：这里指的是 constant 或者此变量所有到达 loop 的定义都在 loop 之外（loop 内 值不发生变化）

② 已执行的循环迭代次数

③ 其他 IV

2. IV 主要分为两类：

①Basic IV

满足 X = X + c 形式的变量，其中 c 是循环不变量，e.g. i

②Derived IV

在循环中只有一次定义，且值是 basic IV 的线性函数。

假设变量 B 是 basic IV， 循环中有 A = c1 * B + c2，则 A 是 derived IV。

表示成<i, c1, c2>

3. IV detection algorithm（generation algorithm?）

① 扫描 loop 中所有的 basic IV （假设为 i）i = <i, 1, 0>

② 扫描 loop 中所有满足 k = j*b 形式的 variable k，其中 j = <i, c, d> (j = c * i + d)，即 k 为 IV，表示成<i, c*b, d*b>

扫描 loop 中所有满足 k = j +/- b 形式的 variable k，其中 j = <i, c, d> (j = c * i + d)，即 k 为 IV，表示成<i, c+/-b, d+/-b>

直到没有 IV 发现

4. Optimization

最对于不同种类的 IV 有不同的优化策略

① targets derived IV ： Strength Reduction (用加法替换乘法)
算法：对于所有的 derived IV 假设 j=<i, c, d>
a: 创建心变量 s
b: 将 j = c * i + d 替换成 j = s
c: 在每一个 i = i + e 紧随之后加上 s = s + c * e
d: 将 s 放在 i 的 family 中<i, c, d> （family 是关于一个 basic IV 的所有 derived IV 的集合）
e: 添加 s = c * i + d 放在 preheader，使得循环开始时 s = j

② targets basic IV (elimination) 删除仅用于测试的归纳变量
假设消除的 basic IV 是 i，需满足 loop 中有 A = c * i + d
将 loop condition: i < x 替换成 A < c * x + d

```
Example ① & ②:
//before:
for(i=0; i < 100; i++){
    t1 = 4 * i;
    t2 = 4 * i + &A
}

//After
t1’ = 0;
t2’ = &A;
for(; t2’ < 4 * 100 + &A; ){
    t1 = t1’;
    t2= t2’;
    t1’ = t1’ + 4;
    t2’ = t2’ + 4;
}
```

③ dead code elimination
假设消除的 IV 是 i，loop 中没有 A = c * i + d 形式

```
Example:
//Before
int max = 10;
int result = 0;
for(int i=0; i<max; i++){
    result += 2;
}
return result;

//After : eliminate i
int max = 10;
int result = 0;
for(; result < max * 2; result +=2);
return result;
```

思路：消除循环索引变量 i
a：首先检查循环体内是否使用到 i；
b：遍历循环体中所有进行自增或自减的变量(e.g. v_i += m_i)，记录其 name, op_number, operator; 记录 i 的自增速度 sp
c: 删除 i，将循环条件 i < x 替换为 $v_1 < x * m_1$ && $v_2 < x * m_2$ && $v_i < x * m_i$ ，将 i += sp 替换成 v_i = vi +/- sp * op_number

④ other IV elimination(standard optimization) 减小寄存器压力 减少变量数量

loop inversion

将 while 语句替换成 if + do-while 语句
while 的 loop condition 即为 if 的 condition 和 do-while 的 loop condition。

loop fusion

1. loop A, B have the same header and unequal trip counts

AB 中引用同一数据，cache miss 减少一半，若 A 中变量只是为 B 所使用，则 A 中变量可以直接带入 B 中。

```
//Unfused loop
  A(I) = B(I) + C1
ENDDO
DO I = 1, N
  D(I) = A(I) + C2
ENDDO

//Fused loop-1
DO I = 1, N

DO I = 1, N
  A(I) = B(I) + C1
  D(I) = A(I) + C2
ENDDO

//Fused loop-2
DO I = 1, N
  D(I) = B(I) + C1 + C2
ENDDO
```

2. fusing loops over a concatenated loop-index range(allow have unequal trip counts)

```
//Unfused loop
for( i=0; i<m; i++){
    loop body A;
}
for( j=0; j<n; j++){
    loop body B;
}
```

```
//Fused loop
for( ij=0; ij<(m+n); ij++){
    if(ij<m){
       int i = ij;
       loop body A;
    }
   else{
        int j = ij-m;
        loop body B;
     }
}
```

3. fusing loop with unequal trip counts

```
//Unfused loop
for( i=0; i<m; i++){
   loop body A;
}
for( j=0; j<n; j++){
   loop body B;
}

//Fused loop
for( f=0; f<max(m,n); f++){
   if(f<m){
       loop body A;
   }
   if(f<n){
       loop body B;
   }
}
```

loop unrolling: 增加每次迭代计算的次数

1. 找到满足此基本类型的 loop：form: for(i = 0; i<n; i+=k)
2. n 是一个合数，得出其所有的因数 $x_1,x_2,...,x_m$
3. 对于每一个因数 x，将 i += k 替换成 i += x * k

将循环体内部进行复制 x-1 次，共计算 x 次。

第几次迭代	每次迭代计算时 i 相应的值
inner part0	i
inner part1	i+k
inner part...	i+(...)*k
inner part(x-1)	i+(x-1)*k

loop unswitching

循环体内 if(w) w 的创建

w：boolean，定义在循环体外

1. 整型数据类型：！、|| 、&& 、> 、<、variable random composition
2. 字符串型数据类型：equals. contains...