Loop Invariant Using Alt-Ergo

Prepared by: K.P. Jevitha

Recap: Weakest Preconditions

- The Weakest Preconditions (WP) methodology involves generating **Verification Conditions (VCs).**
- These VC are proved using a **Theorem Prover (Alt-Ergo).**
- Given program P with pre-condition I and post-condition O, the overall VC to be proved is:

  I ==> wp (P, O)

- If P consists only of assignment statements, sequencing, and if/else, only one VC needs to be proved I ==> **wp (P, O)**
  - **Assignment statement, x = expr:**

    wp (x = expr, O) = O [x <-- expr]

    i.e., replace all occurrences of x in O by expr.

  - **Statement sequence, S1; S2** :

    wp (S1; S2, O) = wp (S1, wp (S2, O))

  - **If-Else, if(B) S1 else S2 :**
    - **wp (if (B) S1 else S2, O) = [B ==> wp (S1, O) && not(B) ==> wp (S2, O)]**
    - **Wp(if(B) S1 else S2, O) = [B && wp (S1, O) || not(b) && wp (S2, O)]**

If, if(B) S1 :

- **wp (if (B) S1, O) = [B ==> wp (S1, O) && not(B) ==> O]**
- **Wp (if (B) S1, O) = [B && wp (S1, O) || not(B) && O]**

Loops, while (B) S :

Loops are verified using Loop invariant {Inv}:

Given a while statement,

while (B) S,

since S can be executed an unbounded number of times (0,1, 2,...), wp cannot be derived simply from B, S, and O.

Hence, we define **Loop Invariant {Inv} such that**

wp (while (B) S, O) = Inv

where **Inv** is a loop invariant condition to be supplied by the user and satisfying:

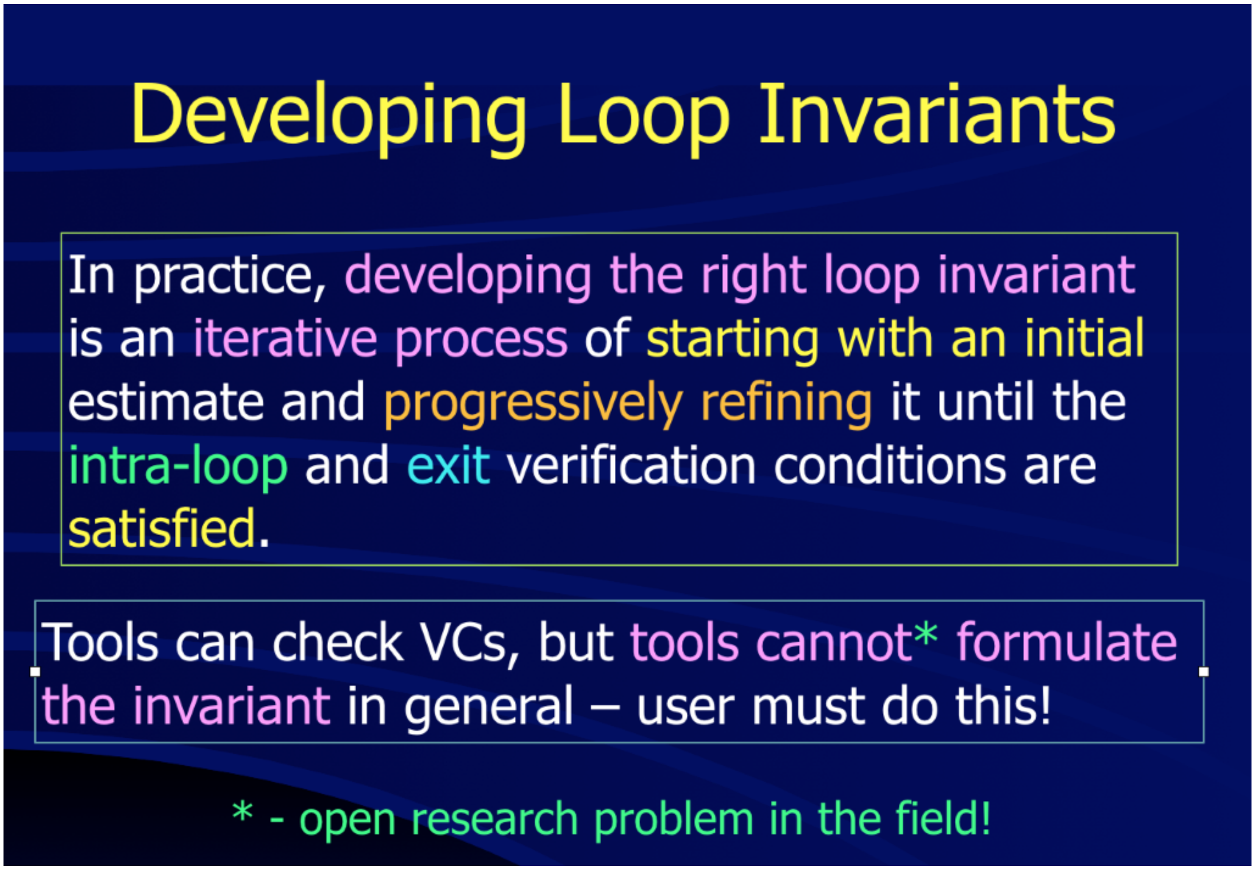
- **Start VC: P ==> wp(initstmts, Inv)**
- **An intra-loop VC: Inv && B ==> wp (LoopStmts, Inv)**
- **An exit VC: Inv && not(B) ==> Q**

P – Precondition, Q – Postcondition, B – Loop condition (or Loop Guard)

Hence to verify a loop, the above mentioned 3 verification conditions need to be checked.

Three Verification Conditions state that:

1. Start VC: {I} holds immediately before the loop, i.e., immediately after the initialization steps.
2. Intra-Loop VC: {I} holds at the end of the loop body, given that {I Λ B} hold at the beginning of the loop body.
3. Exit VC: {I Λ !B}=>{Q} holds after exit of loop.



Steps to find the Loop Invariant:

STEP 1: Trace our program

STEP 2: Assume an invariant based on the relation between the values of the variables

STEP 3: Use Alt-ergo to check the 3 Verification Conditions (VC) [Intra loop VC, Exit VC, Start VC].

> **STEP 3a:** If all the 3 VCs are valid, then the INVARIANT assumed is correct. Hence, we can stop.
>
> **STEP 3b:** If any one of the VC is invalid (unknown), then it means the loop invariant we assumed is insufficient. Repeat step 3c, till all the VCs become valid.
>
> **STEP 3c:** STRENGTHEN THE INVARIANT, generate the 3 VCs and check again in Alt-ergo till all the VCs become valid.

Annotations used in program verification :

@requires - Used to state the Pre-condition or the Input condition for the program.

@ensures – Used to state the Post-condition for the program.

@program – Used to provide the program statements.

@var – used to list the variables and their datatypes used.

@invariant – used to specify the loop invariant.

While Loop General Form:

P – Precondition,

Q – Postcondition,

B – Loop condition (or Loop Guard)

S1, S2 – Loop Statements

```
{P}
[ initalization statements]
While (B) {
        S1;
        S2;
}
{Q}
```

Rewriting using verification annotations:

@requires P

@ensures Q

@program

@var comma – separated variables: datatype

[initalization statements]

@invariant: loop-invariant

```
While (B) {
        S1;
        S2;
 }
```

@end

Example 1 : Sum of n integers

```
@requires n >= 1
@ensures s = n*(n+1)/2
@program
        @var i, n, s : int
        s = 1;
        i = 1;
        @invariant :
        while (i < n) {
        i = i+1;
        s = s+i;
        }
@end
```

STEP 1: Trace our program

Precondition: n > =1. Let us assume n = 5

I	**S**
1	**1**
2	**3**
3	**6**
4	**10**
5	**15**

S = I * (I+1)/2

STEP 2: Assume an invariant based on the relation between the values of the variables:

Inv: S = I * (I+1)/2

STEP 3: Use Alt-ergo to check the 3 Verification Conditions (VC) [Intra loop VC, Exit VC, Start VC].

1. **Intra-loop VC: Inv && B ==> wp (LoopStmts, Inv)**

Inv: S = I * (I+1)/2

B: I < n

Wp (Loopstmts,Inv) :

Loop Stmts:

S1: i = i+1;

S2: s = s+i;

Wp(Loopstmts, Inv)

=> wp (S1; S2, Inv) = wp (S1, wp(S2, Inv))

=> wp (i = i+1; s = s+i; , s = i * (i+1)/2)

=> wp (i=i+1, **wp (s=s+i, s=i*(i+1)/2)**

[Substituting for s = s+i in s=i*(i+1)/2]

=> s+i = i * (i+1)/2

[Substituting for i=i+1]

=> s+(i+1) = (i+1) *(i+1+1)/2

=> s+(i+1) = (i+1) *(i+2)/2

=> s+(i+1) = (i^2 + 2*i+1*i+2)/2

=> s+(i+1) = (I^2 + 3*i +2)/2

=> 2*(s+(i+1)) = (i^2 + 3*i +2)

=> 2*s + 2*(i+1) = i^2 + 3*i + 2

=> 2*s = i^2 +3*i+2 - (2*(i+1))

=> 2*s= i^2 +I

=> 2*s = i*(i+1)

WP (LoopStmts, Inv) => s = i*(i+1)/2

Condition to be checked in alt-ergo :

Inv && B ==> wp (LoopStmts, Inv)

S = i * (i+1)/2 && i <n ==> s = i*(i+1)/2

2*s = i*(i+1) && i<n ==> 2*s = i*(i+1)

Alt-ergo : Intraloop Verification Condition is Valid

```
goal intraloop:

forall i,n,s: int.

2*s = i*(i+1) and i<n
     ->
     2*s = i*(i+1)

# [answer] Valid (0.1240 seconds) (3 steps)
```

2. **Exit VC: Inv && not(B) ==> Q**

   Inv: S = I * (I+1)/2

   Not(B) : not(i <n) => i >=n

   Q: s = n*(n+1)/2

   Exit VC:

   S= i*(i+1)/2 && (i>=n) ==> s=n*(n+1)/2

   2*s = i*(i+1) && (i>=n) ==> 2*s = n*(n+1)

Alt-ergo: Exit Verification Condition is unknown

Inv && not(B) ==> Q is not valid

```
goal exit:

forall i,n,s: int.

  2*s = i*(i+1) and i>=n  ->  2*s = n*(n+1)

# [answer] unknown (0.0700 seconds) (4 steps)
```

STEP 3b: If any one of the VC is invalid (unknown), then it means the loop invariant we assumed is insufficient. Repeat step 3c, till all the VCs become valid.

STEP 3c: STRENGTHEN THE INVARIANT, generate the 3 VCs and check again in Alt-ergo till all the VCs become valid.

STEP 3c: STRENGTHENING THE INVARIANT:

Question : How to **strengthen** the invariant such that **Inv && not(B) ==> Q becomes valid?**

2*s = i*(i+1) && (i>=n) ==> 2*s = n*(n+1)

Question: Why is the following code "**unknown"** as per alt-ergo?

goal exit:

forall i,n,s: int.

2*s = i*(i+1) and i>=n -> 2*s = n*(n+1)

forall i,n,s: int.

LHS : 2*s = i*(i+1) && (i>=n)

If (i=n),

LHS ==> 2*s = n*(n+1) == RHS

If(i>n), say n+1,

LHS ==> 2*s = (n+1) *(n+2)! = RHS

So the condition is valid only for i =n, for all other i>n, the condition is invalid. Hence alt-ergo says its unknown.

So, our invariant should ensure that the value of "i" should not exceed "n" in the condition

Inv && not(B).

Inv: s = i * (i+1)/2 && i<=n

We are strengthening the Invariant by adding another condition to limit the value of n.

Stronger conditions --> reduces the number of items in the matching set

Weaker conditions --> increases the number of items in the matching set

Revisiting - Exit VC: Inv && not(B) ==> Q

Inv: s = i * (i+1)/2 && i<=n

Not(B) : not(i <n) => i >=n

Q: s = n*(n+1)/2

Exit VC : s= i * (i+1)/2 && i<=n && i >=n ==> s = n*(n+1)/2

LHS ==> 2*s = i * (i+1) && i<=n && i >=n

==> 2*s = i * (i+1) && i=n

==> 2*s= n*(n+1) = RHS.

Hence Valid.

Alt-Ergo Verification for Exit VC, Intra Loop Vc and Start VC:

1. **Exit Verification Condition:**

   Inv && not(B) ==> Q is valid

   goal new_exitvc:

   forall i,n,s: int.

   2*s = i*(i+1) and i<=n and i>=n -> 2*s = n*(n+1)

```
goal new_exitvc:

forall i,n,s: int.

  2*s = i*(i+1) and i<=n and i>=n  ->  2*s = n*(n+1)

# [answer] Valid (0.0680 seconds) (4 steps)
```

2. Intra-loop VC: Inv && B ==> wp (LoopStmts, Inv)

Inv: s = i * (i+1)/2 && i<=n

B: i < n

Wp (Loopstmts,Inv) :

Loop Stmts:

S1: i = i+1;

S2: s = s+i;

Wp(Loopstmts, Inv)

=> wp (S1; S2, Inv) = wp (S1, wp(S2, Inv))

=> wp (i = i+1; s = s+i; , s = i * (i+1)/2 && i<=n)

=> wp (i=i+1, **wp (s=s+i, s=i*(i+1)/2 && i<=n)**

wp (s=s+i, s=i*(i+1)/2&&i<=n)

[Substituting for s = s+i in s=i*(i+1)/2]

=> s+i = i * (i+1)/2 && i<=n

[Substituting for i = i+1]

$\Rightarrow s+(i+1) = (i+1) * (i+1+1)/2 \;\&\&\; i+1 <= n$

$\Rightarrow s+(i+1) = (i+1) * (i+2)/2 \;\&\&\; i <= n-1$

$\Rightarrow s+(i+1) = (i^2 + 2*i+1*i+2)/2 \;\&\&\; i <= n-1$

$\Rightarrow s+(i+1) = (i^2 + 3*i +2)/2 \;\&\&\; i <= n-1$

$\Rightarrow 2*(s+(i+1)) = (i^2 + 3*i +2) \;\&\&\; i <= n-1$

$\Rightarrow 2*s + 2*(i+1) = i^2 + 3*i + 2 \;\&\&\; i <= n-1$

$\Rightarrow 2*s = i^2 +3*i+2 - (2*(i+1)) \;\&\&\; i <= n-1$

$\Rightarrow 2*s= i^2 + i\&\& \; i <= n -1$

$\Rightarrow 2*s = i*(i+1) \;\&\&\; i <= n-1$

$\Rightarrow s = i*(i+1)/2 \;\&\&\; i <= n-1$

Wp(Loopstmts, Inv) = s = i*(i+1)/2 && i<=n-1

Condition to be checked in alt-ergo :

Inv && B ==> wp (LoopStmts, Inv)

s = i * (i+1)/2 && i<=n && i < n ==> s = i*(i+1)/2 && i<=n-1

2*s = i*(i+1) && i<=n && i < n ==> 2*s = i*(i+1) && i<=n-1

Alt-ergo: Intraloop Verification Condition is Valid

```
goal new_intraloopvc:

forall i,n,s: int.

2*s = i*(i+1) and i<=n and i < n
     ->
     2*s = i*(i+1) and i<=n-1
```

```
# [answer] Valid (0.0790 seconds) (4 steps)
```

3. **Start VC: P ==> wp(initstmts, Inv)**

   Inv: s = i * (i+1)/2 && i<=n

   P: n >= 1

   wp(initstmts, I) = wp (s=1; i=1, s=(i*(i+1)/2 && i<=n)

   - S1: S=1
   - S2: i=1

Inv: S=i*(i+1)/2 && i<=n

Wp (S2, Inv) ==> 2*s=1*(1+1) && 1 <= n [substitute for i=1]

Wp (S1, Wp (S2, Inv)) ==> 2*1 = 1*(1+1) && 1<=n [Substitute for s=1]

P ==> wp(initstmts, Inv)

Substituting for P, wp(initstmts,P) in the verification Condition (VC) for start :

P--> Wp(initstmts, Inv) =

Start VC : (n>=1) -> 2*1 = 1*(1+1) && 1<=n

```
goal new_startvc:
forall i,n,s: int.
(n>=1) -> (2*1) = (1*(1+1)) and 1<=n

# [answer] Valid (0.0800 seconds) (2 steps)
```

Hence, We have got our required Loop invariant: **s = i * (i+1)/2 && i<=n**

Exercise Problems

Find the loop invariant and prove the verification conditions using Alt-Ergo :

1. Finding Sum of squares of n integers.
2. Find the post-condition and loop invariant. Assume appropriate pre-condition and prove the VCs.

```
x=c;
y=0;
While (x > 0) {
        x = x -1 ;
        y = y+1;
 }
```

3. Find the post-condition and loop invariant, for the pre-condition x=10.

```
i =0
j =0
k =0
while (j < x)
{
        j = j + k + 3 * i +1
        k = k + 6 * i + 3
        i = i + 1
}
```

****

Solutions

Find the loop invariant and prove the verification conditions using Alt-Ergo :

1. Finding Sum of squares of n integers.

```
@requires n >= 1
@ensures s = n*(n+1)*(2n+1)/6
@program
        @var i, n, s : int
        s = 1;
        i = 1;
        @invariant :
        while (i < n) {
        i = i+1;
        s = s+i*i;
        }
@end
--
```

STEP 1: Trace our program

Precondition: n > =1. Let us assume n = 5

I	S
1	1
2	5
3	14
4	30
5	55

S = i*(i+1)(2*i+1)/6

STEP 2: Assume an invariant based on the relation between the values of the variables:

Inv: S = i*(i+1) *(2*i+1)/6 && i<=n

STEP 3: Use Alt-ergo to check the 3 Verification Conditions (VC) [Intra loop VC, Exit VC, Start VC].

1. **Intra-loop VC: Inv && B ==> wp (LoopStmts, Inv)**

   Inv: S = i*(i+1) *(2*i+1)/6 && i<=n

   B: i < n

   Wp (Loopstmts,Inv) :

   Loop Stmts:

   S1: i = i+1;

   S2: s = s+i*i;

   Wp(Loopstmts, Inv)

   => wp (S1; S2, Inv) = wp (S1, wp(S2, Inv))

   => wp (i = i+1; s = s+i*i; , s = i*(i+1)* (2*i+1)/6 && i<=n)

   => wp (i=i+1, **wp (s=s+i*i, s = i*(i+1) *(2*i+1)/6 && i<=n)**

   [Substituting for s = s+i*i in s = i*(i+1) * (2*i+1)/6 && i<=n]

   => s+i*i = i*(i+1) *(2*i+1)/6 && i<=n

   [Substituting for i=i+1]

   => s+(i+1) * (i+1) = (i+1) *(i+1+1) *(2*(i+1) +1) /6

Intraloop VC: Inv && B ==> wp (LoopStmts, Inv)

(S = i*(i+1) *(2*i+1)/6 && i<=n) &&(i < n) -->

s+(i+1) * (i+1) = (i+1) *(i+1+1) *(2*(i+1)+1) /6

```
goal intraloopvc:
forall i,s,n:int.

(s = i*(i+1) *(2*i+1)/6 and i<=n) and ( i < n) ->

(s+(i+1) * (i+1) = (i+1) *(i+1+1) *(2*(i+1)+1) /6 )

# [answer] Valid (1.2620 seconds) (5 steps)
```

2. **Exit Verification Condition:**

   Inv && not(B) ==> Q

Inv: S = i*(i+1) *(2*i+1)/6 && i<=n

B: i < n ==> not(B) = not(i<n)

Q : s = n*(n+1)*(2n+1)/6

Exit VC :

S = i*(i+1) *(2*i+1)/6 && i<=n && not(i<n) --> s = n*(n+1)*(2n+1)/6

Without strengthening condition ---> invalid

```
goal exitvc:
forall i,s,n:int.

s = i*(i+1)*(2*i+1)/6 and not(i<n) ->
          s = n*(n+1)*(2*n+1)/6

# [answer] unknown (0.6120 seconds) (4 steps)
```

With Strengthening --> Valid

```
goal exitvc:
forall i,s,n:int.

s = i*(i+1)*(2*i+1)/6 and (i<=n) and not(i<n) ->
          s = n*(n+1)*(2*n+1)/6
```

```
# [answer] Valid (0.2890 seconds) (4 steps)
```

3. **Start VC :** **P ==> wp(initstmts, Inv)**

Inv: S = i*(i+1) *(2*i+1)/6 && i<=n

P: n>=1

wp(initstmts, I) = wp (s=1; i=1, S = i*(i+1) *(2*i+1)/6 && i<=n)

1=1*(1+1)*(2*1+1)/6 && 1<=n

Start VC:

n>=1 **->** 1=1*(1+1)*(2*1+1)/6 && 1<=n

```
goal start:
forall i,s,n:int.

n>=1 -> 1=1*(1+1)*(2*1+1)/6 and 1<=n
```

```
# [answer] Valid (0.0410 seconds) (2 steps)
```

Required Invariant: S = i*(i+1) *(2*i+1)/6 && i<=n

2. Find the post-condition and loop invariant. Assume appropriate pre-condition and prove the VCs.

```
int computesum(int c){
        x=c;
        y=0;
        While (x > 0) {
                x = x -1 ;
                y = y+1;
         }
  return y;
}
```

Postcondition: Q ==> y = c ;

Some students assumed Q ⇒ x+y = c; which can also be considered.

Sol 1 :

Precondition: P ==> c >=0

Trace: Let c = 5

X	Y
5	0
4	1
3	2
2	3
1	4
0	5

Assumed Invariant: X+y = c && x>=0

1. Intraloop VC: Inv && B ==> wp(Loopstmts, Inv)

Inv : x+y =c && x>=0

B : x > 0

Wp(stmts, INV):

x = x -1 ;
y = y+1;

X+y = c && x>=0 [subs y=y+1]
X+y+1=c && x>=0 [x=x-1]
X-1+y+1=c && x-1 >=0
X+y=c && x-1>=0

Intraloop VC: x+y =c && x>=0 && (x>0) -> x+y=c && x-1>=0 – valid

2. Exit VC: Inv && not(B) ==> Q

   Inv: x+y=c && x>=0

   Not(B): not(x>0)

   Q: Y=c

   Exit VC: x+y =c && x>=0 && not(x>0) -> y=c - valid

3. Start VC: **P -> wp(initstmt, inv)**

   P : c>=0

   Initialization statements :

   x=c;

   y=0;

   Wp(initstmts, x+y =c && x>=0)

   Wp(x=c;y=0 , x+y =c && x>=0)

   => x+0 =c && x>=0 [subs y=0]

Wp => c+0=c && c>=0 [subs x=c]

Startvc: c>=0 -> c+0=c && c>=0 - valid

Required Invariant : **x+y=c && x>=0**

```
goal startvc:
forall c:int.
c>=0 -> c+0=c and c>=0


goal exitvc:

forall x,y,c: int.

x+y=c and (x>=0) and not(x>0)
    -> y=c


goal intravc:
forall x,y,c:int.

x+y =c and x>=0 and (x>0)
  ->
    x+y=c and x-1>=0
```

```
# [answer] Valid (0.0900 seconds) (2 steps) # [answer] Valid (0.1530
seconds) (4 steps) # [answer] Valid (0.1840 seconds) (4 steps)
```

Sol 2 :

Inv: c= x+y

1. Intra-loop VC: Inv && B ==> wp (LoopStmts, Inv)

Inv: c= x+y

B: x > 0

Wp (Loopstmts,Inv) :

Loop Stmts: S1: x = x -1 ; S2: y = y+1;

Wp(Loopstmts, Inv) :

=>Wp (S1, Wp (S2, Inv)) :

=>Wp (x=x-1, Wp (y=y+1, c=x+y))

=>Wp(x=x-1 , c=x+y+1)

=>c=x+y

WP (LoopStmts, Inv) => c=x+y

Intra-loop VC:

c = x + y && x>0 -> c = x + y

Alt-ergo: Exit Verification Condition is valid
2. Exit VC: Inv && not(B) ==> Q
Inv: c= x+y
Not(B): Not(x > 0) => x<=0
Q: c=x+y
Exit VC:
c = x + y and x<=0 -> c = x + y

Alt-ergo: Exit Verification Condition is valid
3. Start VC: P ==> wp(initstmts, Inv)
Inv: c= x+y
P: x>0
wp(initstmts, I) = wp (x=c,y=0,c=x+y)
S1: x=c
S2: y=0
Wp (S1, Wp (S2, Inv)) :
=>Wp (x=c, Wp (y=0, c=x+y))
=>wp(x=c , c=x)
=>c=c
Start VC : x>0 -> c=c

Required Loop invariant: c=x+y

```
goal a:
forall x,y,c: int.
c = x + y and x>0 -> c = x + y


goal b:
forall x,y,c: int.
c = x + y and not(x>0) -> c = x + y

goal c:
forall x,y,c:int.
 x>0 -> c=c
```

```
# [answer] Valid (0.0640 seconds) (3 steps) # [answer] Valid (0.0980
seconds) (3 steps) # [answer] Valid (0.1140 seconds) (2 steps)
```

3. Find the post-condition and loop invariant, for the pre-condition x=10.

```
i =0
j =0
k =0
while (j < x)
{
        j = j + k + 3 * i +1
        k = k + 6 * i + 3
        i = i + 1
}
```

Precondition : P : x=10

Postcondition : Q: j = i*i*i and k = 3*i*i

1. **Tracing:**

I j k

0 0 0

1 1 3

2 8 12

3 27 27

Invariant: j = i*i*i and k = 3*i*i

1. Intraloop VC: Inv && B ==> wp(loopstmts, inv)

Inv: j = I*I*I and k = 3*I*I

B : j < x

Wp(loopstmts, inv)

j = j + k + 3 * i +1

k = k + 6 * i + 3

i = i + 1

Wp ==> j = i*i*i and k = 3*i*i

[subs i=i+1]

==> j = (I+1)*(I+1)*(I+1) and k = 3*(I+1)*(I+1)

[subs k = k + 6 * i + 3]

==> j = (I+1)*(I+1)*(I+1) and (k + 6 * i + 3) = 3*(I+1)*(I+1)

[subs j = j + k + 3 * i +1]

==> j + k + 3 * i +1 = (I+1)*(I+1)*(I+1) and (k + 6 * i + 3) = 3*(I+1)*(I+1)

Intraloop VC: (j = I*I*I and k = 3*I*I) && (j < x) -->

j + k + 3 * i +1 = (I+1)*(I+1)*(I+1) and (k + 6 * i + 3) = 3*(I+1)*(I+1)

Alt-Ergo:

goal intravc:

forall i,j,k,x:int.

(j = i*i*i and k = 3*i*i) and (j < x) ->

(j + k + 3 * i +1 = (i+1)*(i+1)*(i+1) and (k + 6 * i + 3) = 3*(i+1)*(i+1))

2. Exit VC: Inv && not (B) ==> Q

Inv: j = I*I*I and k = 3*I*I

B : j < x ==> not (b) = not(j<x)

Q: j = i*i*I and k = 3*i*i

ExitVc: j = I*I*I and k = 3*I*I && not(j<x) => j = i*i*I and k = 3*i*i

j = i*i*i and k = 3*i*i and not(j<x) ->

j = i*i*I and k = 3*i*i

Start VC: P -> wp (initstmts, inv)

P: **x=10**

wp (initstmts, inv) :

i =0

j =0

k =0

Inv: j = I*I*I and k = 3*I*I

Wp (stmts, inv) => 0 = 0*0*0 and 0 = 3*0*0

Start VC: x=10 -> 0 = 0*0*0 and 0 = 3*0*0

```
goal intravc:

forall i,j,k,x:int.

 (j = i*i*i and k = 3*i*i) and (j < x)  ->
    (j + k + 3 * i +1 = (i+1)*(i+1)*(i+1) and (k + 6 * i + 3) =
3*(i+1)*(i+1) )


goal exitvc:
forall i,j,k,x:int.

j = i*i*i and k = 3*i*i and not(j<x) ->
 j = i*i*i and k = 3*i*i

goal startvc:

forall x:int.

x=10 -> 0 = 0*0*0 and 0 = 3*0*0
```

```
# [answer] Valid (0.1850 seconds) (4 steps) # [answer] Valid (0.2700
seconds) (4 steps) # [answer] Valid (0.2840 seconds) (2 steps)
```

Inv: **j = I*I*I and k = 3*I*I**