Hoare Logic With State Updates - Examples (2)

Ramón Béjar Torres

March 21, 2024

Introduction

In this document we perform the verification of programs with loops. What we verify for them is its *partial correctnes*, i.e. if they finish then their result is correct. Total correctness is considered in a separate document.

Single Loops

Integer Multiplication

We are going to analyze a program for performing the multiplication of two integer numbers x_0 and y_0 with $y_0 \geq 0$. This is the program:

```
{ x = $x_0$ & y = $y_0$ & $y_0$ >= 0}
[]
m = 0
while (y > 0)
{
  m = m + x;
  y = y - 1;
}
{m = $x_0$ * $y_0$}
```

The invariant we are going to use for verifying the loop of this program is:

```
I = { m = $x_0$ * ($y_0$ - y) & x = $x_0$ & y >= 0}
```

That is, the loop achieves the desired result by obtaining in each iteration a partial result ($x_0 * (y_0 - y)$). Observe that when the loop achieves the particular case $y = 0$, then it is when it finally computes the desired multiplication.

These are the three properties we have to check to verify that the loop is correct (it finally reaches an state that satisfies the postcondition of the program):

1. Invariant initially valid. After absorbing the initial sentence (`m = 0;`), that comes before the loops enters, in an state update, what we have to prove is that:

   ```
   ( x = $x_0$ & y = $y_0$ & $y_0$ >= 0) ->
    [m := 0;]( m = $x_0$ * ($y_0$ - y) &  x = $x_0$ & y >= 0)
   ```

So, after applying the state update to the invariant the FOL formula to prove valid is:

```
( x = x0 & y = y0 & y0 >= 0) ->
 ( 0 = x0 * (y0 - y) & x = x0 & y >= 0)
```

If the premise of this formula is true, that means that $y = y_0$, so the product we have at the conclusion will be equal to zero and also if `y0 >= 0` then `y >= 0`. So, the invariant is true just before entering into the loop.

2. Invariant preserved in each iteration. Once we know that the invariant is true at the beginning, we have to prove that if it is true at the beginning of an iteration (and so the body guard condition is true), then it is also true at the end of the iteration. So, we have to prove that:

```
{ m = x0 * (y0 - y) & x = x0 & y >= 0 & y > 0}
  m = m + x;
  y = y - 1;
{ m = x0 * (y0 - y)  & x = x0 & y >= 0 }
```

If we apply the assignment rule to each of the two assignments and then the Exit rule, we get this final formula to prove:

```
( m = x0 * (y0 - y) & x = x0 & y >= 0 & y > 0) ->
  [ m := m + x; y := y - 1; ]
( m = x0 * (y0 - y)  & x = x0 & y >= 0 )
```

That after applying the final state update to the conclusion of the implication we get the FOL formula:

```
( m = x0 * (y0 - y) & x = x0 & y >= 0 & y > 0) ->
( m + x = x0 * (y0 - (y-1))  & x = x0 & y-1 >= 0 )
```

Because at the premise we have that `x = x0` and `y > 0`, observe that the formula at the conclusion of the implication can be rewritten as:

```
( m  = x0 * (y0 - y) + x0 - x0 & x = x0 & y-1 >= 0 )
```

That is therefore true, if the premise is true, because the equation $m = x_0 * (y_0 - y) + x_0 - x_0$ simplifies to $m = x_0 * (y_0 - y)$ and $y - 1 \geq 0$ if $y > 0$. So, the formula is valid. [1]

[1] As we have emphasized in the previous document, it is worth noticing that here we have simplified the formula in a few steps and concluded very quickly that it is valid, because **we quickly recognize how to simplify the formula at the conclusion to get the same formula that we have at the premise** ($m = x_0 * (y_0 - y)$) and we also reason very quickly that $y - 1 \geq 0$ if $y > 0$ (assuming we are working with integer arithmetic equations). However, again, getting a formal proof of this with KeY-Hoare needs a high number of simple inference steps that use the FOL integer arithmetic inference rules of Key-Hoare.

3. If the loops finishes, the postcondition is satisfied. That is, we have to verify the validity of the FOL formula:

```
( m = x0 * (y0 - y) &  x = x0 & y >= 0 & !(y > 0)) ->
( m = $x_0$ * $y_0$ )
```

The quick way to show that this FOL formula is valid in integer arithmetic comes from the observation that if $y \geq 0$ and $!(y > 0) = y \leq 0$ then necessarily $y = 0$. So, as in the premise we have that $m = x_0 * (y_0 - y)$ if we substitute y by 0 is when we get that the equation at the conclusion is also true. So, the FOL formula is valid, and we have proved that when the loop ends (so the loop guard condition is false) and the invariant is true, the postcondition is also true. Because at the previous step we have proved that the invariant is preserved in each loop iteration, we have finally proved that when the loop ends, it achieves the postconditon of the program.

Integer Division

Next, we are going to verify that the following program correctly performs the integer division (computing quotient and reminder) of a dividend `D >= 0` by a divisor `d > 0`:

```
{  D >= 0 & d > 0 }
[]
r = D;
q = 0;
while (r >= d)
{
  q = q + 1;
  r = r - d;
}
{D = (q * d) + r &  r >= 0  &  r < d}
```

But before processing the loop, we apply the assignment rule to the two first assignments so we get the next equivalent program with an initial state update:

```
{  D >= 0 & d > 0 }
[r := D; q := 0;]
while (r >= d)
{
  q = q + 1;
  r = r - d;
}
{D = (q * d) + r &  r >= 0  &  r < d}
```

The invariant we are going to use for verifying the loop of this program is:

```
I = {  D = (q*d) + r  &  r >= 0 }
```

Observe that this invariant is a *relaxed* version of the postcondition of the program. That is, the invariant satisfies the two first conditions at the postcondition. The missing one is `r < d`. This is because the program performs the division by first setting `r` to `D` and `q` to 0, and then at each iteration it decreases `r` and increases `q` until `r` is below `d`. The key point is to decrease `r` the same amount that `q*d` is increased, so that the equation `D = (q*d) + r` at the invariant is satisfied at every iteration.

These are the three properties we have to check to verify that the loop is correct:

1. Invariant initially valid. What we have to prove is that:

```
(  D >= 0 & d > 0 ) ->
 [r := D; q := 0;]( D = (q*d) + r  &  r >= 0 )
```

   So, after applying the state update to the invariant the FOL formula to prove valid is:

```
(  D >= 0 & d > 0 ) -> ( D = (0*d) + D  &  D >= 0 )
```

   This FOL formula is valid, given that `D = (0*d) + D` is equal to `D = D`.

2. Invariant preserved in each iteration. So, we have to prove that:

```
{  D = (q*d) + r  &  r >= 0 & r >= d }
  q = q + 1;
  r = r - d;
{  D = (q*d) + r  &  r >= 0  }
```

   If we apply the assignment rule to the two assignments we have and apply the Exit rule, what we have to prove is the validity of the formula:

```
(  D = (q*d) + r  &  r >= 0 & r >= d ) ->
 [ q := q + 1; r := r - d; ]
(  D = (q*d) + r  &  r >= 0  )
```

   That becomes a FOL formula after applying the final state update to the conclusion of the implication:

```
(  D = (q*d) + r  &  r >= 0 & r >= d ) ->
(  D = ((q + 1)*d) + r-d  &  r - d >= 0  )
```

   To check the validity of this FOL formula, observe that we can simplify the equation we have at the conclusion in this way:

```
 D = ((q + 1)*d) + r-d
   = ((q*d)+d) + r-d
   = (q*d) + r -d + d = (q*d) + r
```

So, we obtain the same equation that we have at the premise, and because `r >= d` is true at the premise then `r - d >= 0`, so we have that the FOL formula is valid. That is, the invariant is true at the end of an iteration if it is true at the beginning of the iteration.

3. If the loops finishes, the postcondition is satisfied. That is, we have to verify the validity of the FOL formula:

```
(  D = (q*d) + r  &  r >= 0 & !(r >= d) ) ->
(  D = (q * d) + r &  r >= 0  &  r < d  )
```

This FOL formula is valid because if the invariant is true and `(r >= d)` is false, then we have that all the conditions at the postcondition are true.

Finding the Greatest Number in an Array

Finally, we are going to verify that the following program correctly finds the greatest number on an array of integer numbers:

```
{ N >= 1 }

  max = a[1];
  i = 2;
  while (i <= N) {
     if (a[i] > max) max = a[i];
     i = i + 1;
  }

{
  \forall int j; ( (1 <= j < (N+1)) -> max >= a[j] ) &
  \exists int j; ( (1 <= j < (N+1)) &  max = a[j] )
}
```

The invariant we are going to use for verifying the loop of this program is:

```
I = {  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
       \exists int j; ( (1 <= j < i) &  max = a[j] ) & i <= (N+1) }
```

That is, the loop achieves the desired result by obtaining in each iteration a partial version of the postcondition, i.e. the postcondition is satisfied but only for a limited range of the array (up to element $i-1$). Observe that when the loop reaches the particular case $i = N+1$, then it is when it finally achieves the desired postcondition.

These are the three properties we have to check to verify that the loop is correct (it finally reaches an state that satisfies the postcondition of the program):

1. Invariant initially valid. After absorbing the two initial sentences (`max = a[1]; i = 2;`), that come before the loops enters, in an state update, what we have to prove is that:

```
( N >= 1 ) ->
 [max := a[1]; i := 2;]( \forall int j; ( (1 <= j < i) -> max >= a[j]
 ) &
  \exists int j; ( (1 <= j < i) &  max = a[j] ) & i <= (N+1))
```

   So, after applying the state update to the invariant the FOL formula to prove valid is:

```
( N >= 1 ) ->
 ( \forall int j; ( (1 <= j < 2) -> a[1] >= a[j] ) &
   \exists int j; ( (1 <= j < 2) &  a[1] = a[j] ) & 2 <= (N+1)  )
```

   Given that the range of values for `j` in the conclusion of the implication on the above formula only contains the value 1, the formula is valid, as `a[1] >= a[j]` and `a[1] = a[j]` are valid for `j=1`. Also, if `N >= 1` then it holds that `2 <= (N+1)` .

2. Invariant preserved in each iteration. Once we know that the invariant is true at the beginning, we have to prove that if it is true at the beginning of an iteration (and so the body guard condition is true), then it is also true at the end of the iteration. So, we have to prove that:

```
{  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) &
   i <= (N+1) & (i <= N) }

     if (a[i] > max) max = a[i];
     i = i + 1;

{  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) & i <= (N+1) }
```

   Because the first sentence inside the loop body is a conditional sentence, we have to prove two different cases:

   - Left branch (condition true). The equivalent program to prove for this branch is:

```
{  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) &
   i <= (N+1) & (i <= N) & [](a[i] > max) }

     max = a[i];
     i = i + 1;

{  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) & i <= (N+1) }
```

where we have extended the precondition with the expression of the conditional sentence and included as first sentence the assignment sentence associated with the condition being true. If we observe both sentences in an state update and apply then the Exit rule, we have that the formula to prove is:

```
(  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) &
   i <= (N+1) & (i <= N) & [](a[i] > max) )   ->

 [ max := a[i]; i := i + 1; ]
(  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) & i <= (N+1) )
```

and after applying the final state update to the conclusion of the implication we get the FOL formula:

```
(  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) &
   i <= (N+1) & (i <= N) & [](a[i] > max) )   ->

(  \forall int j; ( (1 <= j < i+1) -> a[i] >= a[j] ) &
   \exists int j; ( (1 <= j < i+1) &  a[i] = a[j] ) & (i+1) <= (N
+1) )
```

This FOL formula is valid because if `max` is the greatest value in the range $[1, i-1]$ and `a[i] > max`, then it follows that `a[i]` is the greatest value in the range $[1, i]$. Also, if `(i <= N)` then `(i+1) <= (N+1)`. [2]

[2] However, it is interesting to check the amount of work needed by Key-Hoare to prove the validity of this formula, where it needs to use its inference rules for handling universal and existential quantifiers.

- Right branch (condition false). The equivalent program to prove for this branch is:

```
{  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) &
   i <= (N+1) & (i <= N) & [](!(a[i] > max))}

     i = i + 1;

{  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
```

```
   \exists int j; ( (1 <= j < i) &  max = a[j] ) & i <= (N+1) }
```

where we have extended the precondition with the negation of the expression of the conditional sentence and included as unique sentence in the program the second sentence of the loop body, as in this case the conditional sentence does not execute any sentence. If we absorb the unique sentence in an state update and apply then the Exit rule, we have that the formula to prove is:

```
(  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) &
   i <= (N+1) & (i <= N) & [](!(a[i] > max)) )  ->

    [ i := i + 1; ]
(  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) & i <= (N+1) )
```

and after applying the final state update to the conclusion of the implication we get the FOL formula:

```
(  \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i) &  max = a[j] ) &
   i <= (N+1) & (i <= N) & [](!(a[i] > max)) )  ->

(  \forall int j; ( (1 <= j < i + 1) -> max >= a[j] ) &
   \exists int j; ( (1 <= j < i + 1) &  max = a[j] ) & i + 1 <= (N
+1) )
```

This FOL formula is valid because if `max` is the greatest value in the range $[1, i-1]$ and `a[i] <= max`, then it follows that `max` is also the greatest value in the range $[1, i]$. Also, if `(i <= N)` then `(i+1) <= (N+1)`.

3. If the loops finishes, the postcondition is satisfied. That is, we have to verify the validity of the FOL formula:

```
( \forall int j; ( (1 <= j < i) -> max >= a[j] ) &
  \exists int j; ( (1 <= j < i) &  max = a[j] ) &
   i <= (N+1) & !(i <= N) ) ->
( \forall int j; ( (1 <= j < (N+1)) -> max >= a[j] ) &
  \exists int j; ( (1 <= j < (N+1)) &  max = a[j] ) )
```

This FOL formula is valid because if `i <= (N+1)` and `!(i <= N)` then `i == N+1`. So, replacing `i` by `N+1` in the premise of the implication we actually get the formula we have at the conclusion.

Nested Loops

Probably, the most inefficient algorithm for multiplication

We are going to verify the following program that performs the multiplication of two integer numbers x_0 and y_0 with $y_0 \geq 0$. This is the program:

```
{ x = $x_0$ & y = $y_0$ & $y_0$ >= 0}
[m := 0]
while (y > 0)
{
  t = 0;
  if (x >= 0) {
    while (t < x) { t = t+1; }
  }
   else {
      while (t > x) { t = t-1; }
   }
  m = m + t;
  y = y - 1;
}
{m = $x_0$ * $y_0$}
```

Observe that it is simply a variant of the first program we have verified in this document, but now we have replaced the single instruction we had in the original program for multiplication:

```
  m = m + x;
```

by a more complex piece of code (that contains a loop):

```
  t = 0;
  if (x >= 0) {
    while (t < x) { t = t+1; }
  }
   else {
      while (t > x) { t = t-1; }
   }
  m = m + t;
```

Obviously, they are acheiving the same effect (accumulate over `m` the value of x), but our new version acheives this with far more work, so the verification will need also more work.

We have two loops, but regarding the outer loop things are the same. That is, the invariant we are going to use for verifying the outer loop of this program is the same as before:

```
$I_0$ = { m = $x_0$ * ($y_0$ - y) & x = $x_0$ & y >= 0}
```

we will discuss about the needed invariant for the inner loop when we reach that loop in the verification process.

Because the symbolic execution firsts reaches the outer loop, the top level task to perform is as before, that is, to verify that the outer loop is correct. Actually, the only verification step that will need a different work will be the second one (verifying that I_0 is preserved in each iterarion of the outer loop), because when the outer loop iterates is when the symbolic execution reaches the inner loop.

1. Invariant initially valid. We have to prove that:

```
( x = x₀ & y = y₀ & y₀ >= 0) ->
 [m := 0;]( m = x₀ * (y₀ - y) &  x = x₀ & y >= 0)
```

   So, after applying the state update to the invariant the FOL formula to prove valid is:

```
( x = x₀ & y = y₀ & y₀ >= 0) ->
 ( 0 = x₀ * (y₀ - y) & x = x₀ & y >= 0)
```

   If the premise of this formula is true, that means that $y = y_0$, so the product we have at the conclusion will be equal to cero and also if y_0 `>= 0` then `y >= 0`. So, the invariant is true just before entering into the loop.

2. Invariant preserved in each iteration. Once we know that the invariant is true at the beginning, we have to prove that if it is true at the beginning of an iteration (and so the body guard condition is true), then it is also true at the end of the iteration. So, we have to prove that (absorving the first assignment sentence into an initial state update):

```
{ m = x₀ * (y₀ - y) & x = x₀ & y >= 0 & y > 0}
  [t := 0]
  if (x >= 0) {
    while (t < x) { t = t+1; }
  }
   else {
      while (t > x) { t = t-1; }
   }
  m = m + t;
  y = y - 1;
{ m = x₀ * (y₀ - y)  & x = x₀ & y >= 0 }
```

   Because we have a conditional sentence at the beginning of this Hoare triple, we apply the conditional rule so we have to prove two subbranches:

(a) Left subranch: $(x >= 0)$ true. The Hoare triple to prove for this case is:

```
{ m = $x_0$ * ($y_0$ - y) & x = $x_0$ & y >= 0 & y > 0 & [t := 0](x >= 0)}
  [t := 0]
  while (t < x) { t = t+1; }
  m = m + t;
  y = y - 1;
{ m = $x_0$ * ($y_0$ - y)  & x = $x_0$ & y >= 0 }
```

Because in this Hoare triple we reach an inner loop, we apply now the loops rule to prove this triple. But we need first an invariant for the inner loop. We propose the next one:

$$I_1 = I_0 \;\; \& \;\; t <= x \;\; \& \;\; x >= 0 \;\; \& \;\; y > 0$$

Then we apply the loop rule on the current branch so it is spplited into three subbranches:

i. Invariant of inner loop initially valid (before starting the execution). We have to prove that:

```
{ m = $x_0$ * ($y_0$ - y) & x = $x_0$ & y >= 0 & y > 0 & (x >= 0)} ->
 [t := 0;]( $I_0$ & $t <= x$ & $x >= 0$ & $y > 0$ )
```

So, after applying the state update to the invariant the FOL formula to prove valid is:

```
{ m = $x_0$ * ($y_0$ - y) & x = $x_0$ & y >= 0 & y > 0 & (x >= 0)} ->
 ( $I_0$ & $0 <= x$ & $x >= 0$ & $y > 0$ )
```

We do not expand the expression for I_0 because the state update does not change any of its variables and so it appears in the same way in the precondition and conclusion of this formula. Observe that this formula is valid given that x satisfies `x >= 0` in the precondition.

ii. Invariant preserved in each iteration. We have to prove that:

```
{ $I_0$ & $t <= x$ & $x >= 0$ & $y > 0$ & $t < x$ }
  t = t+1;
{ $I_0$ & $t <= x$ & $x >= 0$ & $y > 0$ }
```

That is:

$$(I_0 \;\; \& \;\; t <= x \;\; \& \;\; x >= 0 \;\; \& \;\; y > 0 \;\; \& \;\; t < x) - >$$
$$(I_0 \;\; \& \;\; t + 1 <= x \;\; \& \;\; x >= 0 \;\; \& \;\; y > 0)$$

That is a valid FOL formula.

iii. If the loops finishes, the postcondition is satisfied. We have to prove that:

```
{  $I_0$ & $t <= x$ & $x >= 0$ & $y > 0$ & $t >= x$ }
  m = m + t;
  y = y - 1;
{ m = x0 * (y0 - y)  & x = x0 & y >= 0 }
```

is valid. After executing the two final assignments and applying the final state update ($[m := m + t | y := y - 1]$) we get the formula to prove valid:

$$(\ I_0 \ \& \ t <= x \ \& \ x >= 0 \ \& \ y > 0 \ \& \ t >= x \ ) \text{ -> } (\ m + t = x_0 * (y_0 - (y-1)) \ \& \ x = x_0 \ \& \ y - 1 >= 0 \ )$$

That is valid, because if $m = x_0 * (y_0 - y)$ is true in I_0, then $m + t = x_0 * (y_0 - (y-1))$ is also true given that $t = x$ in the precondition and if $y > 0$ then $y - 1 >= 0$.

(b) Right subranch: $(x >= 0)$ false. The Hoare triple to prove for this case is:

```
{ m = x0 * (y0 - y) & x = x0 & y >= 0 & y > 0 & [t := 0](x < 0)}
  [t := 0]
  while (t > x) { t = t-1; }
  m = m + t;
  y = y - 1;
{ m = x0 * (y0 - y)  & x = x0 & y >= 0 }
```

Because in this Hoare triple we reach an inner loop, we apply now the loops rule to prove this triple. But we need first an invariant for the inner loop. We propose the next one:

$$I_2 = I_0 \ \& \ t >= x \ \& \ x < 0 \ \& \ y > 0$$

Then we apply the loop rule on the current branch so it is spplited into three subbranches:

i. Invariant of inner loop initially valid (before starting the execution). We have to prove that:

```
{ m = x0 * (y0 - y) & x = x0 & y >= 0 & y > 0 & (x < 0) } ->
 [t := 0;]( I0 & t >= x & x < 0 & y > 0 )
```

So, after applying the state update to the invariant the FOL formula to prove valid is:

```
{ m = x0 * (y0 - y) & x = x0 & y >= 0 & y > 0 & (x < 0)} ->
 ( I0 & 0 >= x & x < 0 & y > 0 )
```

We do not expand the expression for I_0 because the state update does not change any of its variables and so it appears in the same way in the precondition and conclusion of this formula. Observe that this formula is valid given that x satisfies `x < 0` in the precondition.

ii. Invariant preserved in each iteration. We have to prove that:

```
{ I0 & t >= x & x < 0 & y > 0 & t > x }
```

```
  t = t-1;
{ $I_0$ & $t >= x$ & $x < 0$ & $y > 0$ }
```

That is:

```
 ($I_0$ & $t >= x$ & $x < 0$ & $y > 0$ & $t > x$)$- >$
($I_0$ & $t - 1 >= x$ & $x < 0$ & $y > 0$)
```

That is a valid FOL formula.

iii. If the loops finishes, the postcondition is satisfied. We have to prove that:

```
{  $I_0$ & $t >= x$ & $x < 0$ & $y > 0$ & $t <= x$ }
  m = m + t;
  y = y - 1;
{ m = $x_0$ * ($y_0$ - y)  & x = $x_0$ & y >= 0 }
```

is valid. After executing the two final assignments and applying the final state update ($[m := m + t | y := y - 1]$) we get the formula to prove valid:

```
(  $I_0$ & $t >= x$ & $x < 0$ & $y > 0$ & $t <= x$ ) ->
   ( $m + t = x_0 * (y_0 - (y - 1))$ & $x = x_0$ & $y - 1 >= 0$  )
```

That is valid, because if $m = x_0 * (y_0 - y)$ is true in I_0, then $m + t = x_0 * (y_0 - (y - 1))$ is also true given that $t = x$ in the precondition and if $y > 0$ then $y - 1 >= 0$.

3. If the loops finishes, the postcondition is satisfied. That is, we have to verify the validity of the FOL formula:

```
( m = $x_0$ * ($y_0$ - y) &  x = $x_0$ & y >= 0 & !(y > 0)) ->
( m = $x_0$ * $y_0$ )
```

The quick way to show that this FOL formula is valid in integer arithmetic comes from the observation that if $y \geq 0$ and $!(y > 0) = y \leq 0$ then necessarily $y = 0$. So, as in the premise we have that $m = x_0 * (y_0 - y)$ if we substitute y by 0 is when we get that the equation at the conclusion is also true. So, the FOL formula is valid, and we have proved that when the loop ends (so the loop guard condition is false) and the invariant is true, the postcondition is also true. Because at the previous step we have proved that the invariant is preserved in each loop iteration, we have finally proved that when the loop ends, it achieves the postconditon of the program.

Array Autocorrelation

Finally, consider the following algorithm to compute the function:

$$\sum_{i=1}^{N} \sum_{j=1}^{N} a[i] * a[j]$$

```
{ }
[ i1 := 1 || res := 0]
while (i1 <= N)
{
  j1 = 1;
  ri = 0;
  while (j1 <= N)
  {
     ri = ri + (a[i1]*a[j1]);
     j1 = j1 + 1;
  }
  res = res + ri;
  i1 = i1 + 1;
}
{ res = $\sum_{i=1}^{N} \sum_{j=1}^{N}$ a[i]*a[j] }
```

Each iteration of the outer loop satisfies a version of the postcondition but with a smaller range of values for the index i. In particular, for the range $[1, i1-1]$. That is, a valid invariant for the outer loop is:

$$I_1 \equiv \{res = \sum_{i=1}^{i1-1} \sum_{j=1}^{N} a[i] * a[j] \;\; \& \;\; i1 \leq N+1\}$$

Such that **after** the last iteration (when $i1 = N+1$) the invariant tells us that $res = \sum_{i=1}^{N} \sum_{j=1}^{N} a[i] * a[j]$.

The inner loop computes the value $ri = \sum_{j=1}^{N} a[i1] * a[j]$, for the current value of $i1$. Each iteration of the inner loop satisfies that ri is equal to that value but for a smaller range of values for j. In particular, for the range $[1, j1-1]$. That is, a good invariant for the inner loop is:

$$I_2 \equiv \{ri = \sum_{j=1}^{j1-1} a[i1] * a[j] \;\; \& \;\; j1 \leq N+1\}$$

Observe that in particular, **after** the last iterarion when $j1 = N+1$, the invariant tell us that $ri = \sum_{j=1}^{N} a[i1] * a[j]$.

Exercise: Verify the program using the invariants proposed. You cannot do it using `KeyHoare`, as you cannot use the sum symbol in the specifications of `KeyHoare`.