Multiply Example (Hoare Triples)

The program for multiplying is:

```
z = 0;
while( x > 0 )
  {
  z = z + y;
  x = x – 1;
  }
```

The pre-condition is: $x = x_0 \wedge x_0 \geq 0$.

The post-condition is: $z = x_0 * y$.

I first give a narrative, then I give the annotated program in “tableau form”, and finally the step-by-step derivation as a proof.

(Note: In the help session I carried $y = y_0$, but it is less cluttered without this.)

The final triple to be derived is thus:

1. $([(x = x_0) \wedge (x_0 \geq 0)])$ `z = 0; while(x > 0) {z = z + y; x = x – 1;}` $([z = x_0 * y])$

The proposed loop invariant is: $(z = (x_0 – x) * y) \wedge (x \geq 0)$.

We will need to derive these triples, which can be composed using the composition rule to get 1:

1.1 $([(x = x_0) \wedge (x_0 \geq 0)])$ `z = 0;` $([(z = (x_0 – x) * y) \wedge (x \geq 0)])$

1.2 $([(z = (x_0 – x) * y) \wedge (x \geq 0)])$ `while(x > 0) {z = z + y; x = x – 1;}` $([z = x_0 * y])$

To derive 1.1, we use the assignment rule, determining the weakest pre-condition from the invariant as post-condition:

1.1a $([(0 = (x_0 - x) * y) \square (x \geq 0)])$ `z = 0;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$

From properties of arithmetic, we have the logical implication:

$([(x = x_0) \square (x_0 \geq 0)]) \square ([(0 = (x_0 - x) * y) \square (x \geq 0)])$

To see this, assume the LHS of $\square$. Using equality elimination, the RHS becomes $([(0 = (x_0 - x_0) * y) \square (x_0 \geq 0)])$, which simplifies to $([(0 = 0) \square (x_0 \geq 0)])$, which follows from the LHS. So by the implication rule, we have derived the triple 1.1 from 1.1a, which is an instance of the assignment rule.

To derive 1.2, we are working in the domain of integers, so the *negation* of the test condition is $x \leq 0$. This conjoined with the loop invariant gives us $x = 0$, and when that is substituted for x, we get $z = (x_0 - 0) * y_0$, which simplifies to $z = x_0 * y$, the overall post-condition. So it suffices to derive 1.2a, and 1.2 follows from 1.2a by the implication rule:

1.2a $([(z = (x_0 - x) * y) \square (x \geq 0)])$ `while(x > 0){z = z + y; x = x – 1;}` $([(z = (x_0 - x) * y) \square (x = 0)])$

To derive 1.2a, we will use the while rule with our proposed loop invariant, to derive:

1.2a.1 $([(z = (x_0 - x) * y) \square (x \geq 0) \square (x > 0)])$ `z = z + y; x = x – 1;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$
Here we identify $x > 0$ with the test in the while statement, and everything else with the loop invariant. Once again, the negation of the test is equivalent to $x \leq 0$, and $x \geq 0$ together with $x \leq 0$ will give $x = 0$.

From the assignment rule, we get:

1.2a.2: $([(z = (x_0 - (x-1)) * y) \square (x - 1 \geq 0)])$ `x = x – 1;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$

From the assignment rule, we also get

1.2a.1: ([$(z + y = (x_0 – (x\text{-}1)) * y)$ □ $(x - 1 \geq 0)$]) `z = z + y;` ([$(z = (x_0 – (x\text{-}1)) * y)$ □ $(x - 1 \geq 0)$])

The pre-condition to 1.2a.1 simplifies, by arithmetic equalities and the fact that $x > 0$ is equivalent to $x \geq 1$ when dealing with integers, to ([$(z = (x_0 – x) * y)$ □ $(x > 0)$]) which is implied by (in fact, equivalent to) ([$(z = (x_0 – x) * y)$ □ $(x \geq 0)$ □ $(x > 0)$]). (The $(x \geq 0)$ part of the invariant is not used, since it is subsumed by the test condition $x > 0$.)

So we are now done.

The following “tableau” form places the assertions used in the context of the original program:

([$x = x_0$ □ $x_0 \geq 0$])
([$(0 = (x_0 – x) * y)$ □ $(x \geq 0)$])
`z = 0;`
([$(z = (x_0 – x) * y)$ □ $(x \geq 0)$])
`while( x > 0 )`
   `{`
   ([$(z = (x_0 – x) * y)$ □ $(x \geq 0)$ □ $(x > 0)$])
   ([$(z = (x_0 – (x\text{-}1)) * y)$ □ $(x - 1 \geq 0)$])
   ([$(z + y = (x_0 – (x\text{-}1)) * y)$ □ $(x - 1 \geq 0)$])
   `z = z + y;`
   ([$(z = (x_0 – (x\text{-}1)) * y)$ □ $(x - 1 \geq 0)$])
   `x = x – 1;`
   ([$(z = (x_0 – x) * y)$ □ $(x \geq 0)$])
   `}`
([$(z = (x_0 – x) * y)$ □ $(x \geq 0)$ □ □ $(x > 0)$])
([$(z = (x_0 – x) * y)$ □ $(x = 0)$])
([$z = x_0 * y$])

Here is the overall derivation, with justifications:

1.	$([(0 = (x_0 - x) * y) \square (x \geq 0)])$ `z = 0;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$	Assignment rule
2.	$([x = x_0 \square x_0 \geq 0])$ `z = 0;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$	Implication rule 1
3.	$([(z + y = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ `z = z + y;` $([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$	Assignment rule
4.	$([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ `z = z + y;` $([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$	Implication rule 3
5.	$([(z = (x_0 - x) * y) \square (x \geq 0) \square (x > 0)])$ `z = z + y;` $([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$	Implication rule 4
6.	$([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ `x = x – 1;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$	Assignment rule
7.	$([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ `z = z + y; x = x – 1;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$	Composition rule 5, 6
8.	$([(z = (x_0 - x) * y) \square (x \geq 0)])$ `while(x > 0){z = z + y; x = x – 1;}` $([(z = (x_0 - x) * y) \square (x \geq 0) \square \square (x > 0)])$	While rule 7
9.	$([(z = (x_0 - x) * y) \square (x \geq 0)])$ `while(x > 0){z = z + y; x = x – 1;}` $([(z = (x_0 - x) * y) \square (x = 0)])$	Implication rule 8
10.	$([(z = (x_0 - x) * y) \square (x \geq 0)])$ `while(x > 0){z = z + y; x = x – 1;}` $([z = x_0 * y])$	Implication rule 9
11.	$([x = x_0 \square x_0 \geq 0])$ `z = 0; while(x > 0){z = z + y; x = x – 1;}` $([z = x_0 * y])$	Composition rule 2, 10

Here is an attempt to show this as a tree.

$([(z + y = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ `z = z + y;` $([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ assignment

$([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ `z = z + y;` $([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)]$ imp

$([(z = (x_0 - x) * y) \square (x \geq 0) \square (x > 0)])$ `z = z + y;` $([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ imp

$([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ `x = x – 1;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$ assn

$([(z = (x_0 - (x\text{-}1)) * y) \square (x - 1 \geq 0)])$ `z = z + y; x = x – 1;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$ composition

$([(z = (x_0 - x) * y) \square (x \geq 0)])$ `while(x > 0){z = z + y; x = x – 1;}` $([(z = (x_0 - x) * y) \square (x \geq 0) \square \square (x > 0)])$ while

$([(0 = (x_0 - x) * y) \square (x \geq 0)])$ `z = 0;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$ assignment

$([x = x_0 \square x_0 \geq 0])$ `z = 0;` $([(z = (x_0 - x) * y) \square (x \geq 0)])$ implication

$([(z = (x_0 - x) * y) \square (x \geq 0)])$ `while(x > 0){z = z + y; x = x – 1;}` $([(z = (x_0 - x) * y) \square (x = 0)])$ implication

$([(z = (x_0 - x) * y) \square (x \geq 0)])$ `while(x > 0){z = z + y; x = x – 1;}` $([z = x_0 * y])$ implication

$([x = x_0 \square x_0 \geq 0])$ `z = 0; while(x > 0){z = z + y; x = x – 1;}` $([z = x_0 * y])$ composition