Reasoning About Programs

Week 7 Tutorial - Loop Invariants and Variants

Sophia Drossopoulou and Mark Wheelhouse

1st Question:

Consider the following Java method that multiplies its arguments through repeated addition:

```
int calcMult(int m, int n)
// PRE: m ≥ 0  ∧  n ≥ 0
// POST: r = m ∗ n
{
    int cntr = 0;
    int acc = 0;
    // INV: I
    // VAR: V
    while (cntr < m) {
        acc = acc + n;
        cntr = cntr + 1;
    }
    // MID: M
    return acc;
}
```

a) Write a midcondition M which holds after the loop and is strong enough to prove partial correctness of the `calcMult` method. (You do not need to prove anything.)

b) Write a loop invariant I which is strong enough to prove partial correctness of the `calcMult` method. (You do not need to prove anything.)

c) Write a loop variant V which is strong enough to prove total correctness of the `calcMult` method. (You do not need to prove anything.)

A possible answer:

a) $M \longleftrightarrow \texttt{acc} = \texttt{m} * \texttt{n}$

b) $I \longleftrightarrow 0 \leq \texttt{cntr} \leq \texttt{m} \ \wedge \ \texttt{acc} = \texttt{n} * \texttt{cntr}$

c) $V \ = \ \texttt{m} - \texttt{cntr}$

2nd Question:

Consider the following Java method that multiplies its arguments through repeated addition with an alternative implementation to that in Question 1. above:

```
int calcMultB(int m, int n)
// PRE: m ≥ 0  ∧  n ≥ 0
// POST: r = m ∗ n
{
    int cntr = n;
    int acc = 0;
    // INV: I
    // VAR: V
    while (cntr > 0) {
        acc = acc + m;
        cntr = cntr -1 ;
    }
    // MID: M
    return acc;
}
```

a) Write a midcondition M which holds after the loop and is strong enough to prove partial correctness of the `calcMultB` method. (You do not need to prove anything.)

b) Write a loop invariant I which is strong enough to prove partial correctness of the `calcMultB` method. (You do not need to prove anything.)

c) Write a loop variant V which is strong enough to prove total correctness of the `calcMultB` method. (You do not need to prove anything.)

A possible answer:

a) $M \;\longleftrightarrow\; \texttt{acc} = \texttt{m} * \texttt{n}$

b) $I \;\longleftrightarrow\; \land\; 0 \leq \texttt{cntr} \leq \texttt{n} \;\land\; \texttt{acc} = \texttt{m} * (\texttt{n} - \texttt{cntr})$

c) $V \;=\; \texttt{cntr}$

3rd Question:

Consider the following Java method that raises its first argument to the power of its second argument through repeated multiplication:

```java
int calcPower(int m, int n)
// PRE: n ≥ 0
// POST: r = m^n
{
    int cntr = 0;
    int acc = 1;
    // INV: I
    // VAR: V
    while (n != cntr) {
        acc = m * acc;
        cntr = cntr + 1;
    }
    // MID: M
    return acc;
}
```

a) Write a midcondition M which holds after the loop and is strong enough to prove partial correctness of the `calcPower` method. (You do not need to prove anything.)

b) Write a loop invariant I which is strong enough to prove partial correctness of the `calcPower` method. (You do not need to prove anything.)

c) Write a loop variant V which is strong enough to prove total correctness of the `calcPower` method. (You do not need to prove anything.)

A possible answer:

a) $M \longleftrightarrow \mathtt{acc} = \mathtt{m}^{\mathtt{n}}$

b) $I \longleftrightarrow 0 \leq \mathtt{cntr} \leq \mathtt{n} \ \wedge\ \mathtt{acc} = \mathtt{m}^{\mathtt{cntr}}$

c) $V = \mathtt{n} - \mathtt{cntr}$

4th Question:

Consider the following Java method that raises its first argument to the power of its second argument through repeated multiplication with an alternative implementation to that in Question 3. above:

```
int calcPowerB(int m, int n)
// PRE: n ≥ 0
// POST: r = m^n
{
    int cntr = n;
    int acc = 1;
    // INV: I
    // VAR: V
    while (cntr > 0) {
        acc = m * acc;
        cntr = cntr - 1;
    }
    // MID: M
    return acc;
}
```

a) Write a midcondition M which holds after the loop and is strong enough to prove partial correctness of the `calcPowerB` method. (You do not need to prove anything.)

b) Write a loop invariant I which is strong enough to prove partial correctness of the `calcPowerB` method. (You do not need to prove anything.)

c) Write a loop variant V which is strong enough to prove total correctness of the `calcPowerB` method. (You do not need to prove anything.)

A possible answer:

a) $M \longleftrightarrow \texttt{acc} = \texttt{m}^{\texttt{n}}$

b) $I \longleftrightarrow 0 \leq \texttt{cntr} \leq \texttt{n} \ \wedge \ \texttt{acc} = \texttt{m}^{(\texttt{n}-\texttt{cntr})}$

c) $V \ = \ \texttt{cntr}$

5th Question:

Consider the following Java method the calculate the product of the elements of an array:

```
int product (int[] a)
// PRE: a ≠ null
// POST: r = ∏ a_0[0..a_0.length)
{
    int res = 1;
    int i = 0;
    // INV: I
    // VAR: V
    while (i < a.length) {
        res = res * a[i];
        ++i;
    }
    // MID: M
    return res;
}
```

a) Write a midcondition M which holds after the loop and is strong enough to prove partial correctness of the `product` method. (You do not need to prove anything.)

b) Write a loop invariant I which is strong enough to prove partial correctness of the `product` method. (You do not need to prove anything.)

c) Write a loop variant V which is strong enough to prove total correctness of the `product` method. (You do not need to prove anything.)

A possible answer:

a) $M \longleftrightarrow \texttt{a} \approx \texttt{a}_0 \ \land\ \texttt{res} = \prod \texttt{a}[0..\texttt{a.length})$

b) $I \longleftrightarrow \texttt{a} \approx \texttt{a}_0 \ \land\ 0 \leq \texttt{i} \leq \texttt{a.length} \ \land\ \texttt{res} = \prod \texttt{a}[0..\texttt{i})$
It would not be incorrect to also add $\texttt{a} \neq \texttt{null}$ to the invariant, but notice that this can actually be derived from $\texttt{a} \approx \texttt{a}_0$ and $\texttt{a}_0 \neq \texttt{null}$ (which is given in the precondition).

c) $V \ = \ \texttt{a.length} - \texttt{i}$

6th Question:

The function *eqs* calculates the number of equal elements for arrays `a` and `b`, provided they have the same length:

$$eqs(\mathtt{a},\mathtt{b}) = \begin{cases} \mid \{\ j \mid 0 \leq j < \mathtt{a.length} \ \wedge \ \mathtt{a}[j] = \mathtt{b}[j] \ \} \mid & \text{if } \mathtt{a.length} = \mathtt{b.length} \\ \text{undefined} & \text{otherwise} \end{cases}$$

where the modulus operator applied to a set $|\{...\}|$ returns the size of that set.

As an example, $eqs(\text{`}\mathtt{DEFGH}\text{'}, \text{`}\mathtt{DXFXH}\text{'}) = 3$.

Consider the following Java method that claims to perform the same calculation for integer arrays:

```
int eqNo (int[] a, int[] b)
// PRE: a ≠ null  ∧  b ≠ null  ∧  a.length = b.length
// POST: r = eqs(a₀, b₀)
{
    int res = 0;
    int i = a.length;
    // INV: I
    // VAR: V
    while (i > 0) {
        i--;
        if (a[i] == b[i]) { res++; }
    }
    // MID: M
    return res;
}
```

a) Write a midcondition M which holds after the loop and is strong enough to prove partial correctness of the `eqNo` method. (You do not need to prove anything.)

b) Write a loop invariant I which is strong enough to prove partial correctness of the `eqNo` method. (You do not need to prove anything.)

c) Write a loop variant V which is strong enough to prove total correctness of the `eqNo` method. (You do not need to prove anything.)

Hint: You may find it helpful to use the function *eqsAux* which calculates the numbers of equal elements between index k and the end of the array for arrays `a` and `b`:

$$eqsAux(\mathtt{a},\mathtt{b},k) = \begin{cases} \mid \{j \mid k \leq j < \mathtt{a.length} \wedge \mathtt{a}[j] = \mathtt{b}[j]\} \mid & \text{if } \mathtt{a.length} = \mathtt{b.length} \text{ and } k \geq 0 \\ \text{undefined} & \text{otherwise} \end{cases}$$

For example, $eqsAux(\text{`}\mathtt{DEFGH}\text{'}, \text{`}\mathtt{DXFXH}\text{'}, 4) = 1$, and also $eqsAux(\text{`}\mathtt{DEFGH}\text{'}, \text{`}\mathtt{DXFXH}\text{'}, 2) = 2$, and finally, $eqsAux(\text{`}\mathtt{DEFGH}\text{'}, \text{`}\mathtt{DXFXH}\text{'}, 0) = 3$.

Observe that $k \leq k'$ implies that $eqsAux(\mathtt{a},\mathtt{b},k') \leq eqsAux(\mathtt{a},\mathtt{b},k)$ for any `a` and `b`.
Also, $eqsAux(\mathtt{a},\mathtt{b},0) = eqs(\mathtt{a},\mathtt{b})$ for any `a` and `b`.

A possible answer:

a) $M \;\longleftrightarrow\; \mathtt{a} \approx \mathtt{a}_0 \;\wedge\; \mathtt{b} \approx \mathtt{b}_0 \;\wedge\; \mathtt{res} = eqsAux(\mathtt{a}, \mathtt{b}, 0)$

b) $I \;\longleftrightarrow\; \mathtt{a} \approx \mathtt{a}_0 \;\wedge\; \mathtt{b} \approx \mathtt{b}_0 \;\wedge\; 0 \leq \mathtt{i} \leq \mathtt{a.length} \;\wedge\; \mathtt{res} = eqsAux(\mathtt{a}, \mathtt{b}, \mathtt{i})$

c) $V \;=\; \mathtt{i}$

7th Question:

Remember the "mystery" tail recursive function $\mathsf{G} : \mathbb{N} \to \mathbb{N}$, discussed in week_5:

$$\begin{array}{lll} \textbf{M1} & \mathsf{G}(m) = \mathsf{GP}(m, 0, 1) & \\ \textbf{M2} & m = cnt \;\longrightarrow\; \mathsf{GP}(m, cnt, acc) = acc & \\ \textbf{M3} & m \neq cnt \;\longrightarrow\; \mathsf{GP}(m, cnt, acc) = \mathsf{GP}(m, cnt + 1, 2 * acc) & \end{array}$$

We will now consider its counterpart though a while-loop in Java:

```
int mathPower(int m)
// PRE: m ≥ 0
// POST: r = 2^m
{
    int cnt = 0;
    int acc = 1;
    // INV: I
    // VAR: V
    while ( !(m == cnt) ) {
        cnt = cnt + 1;
        acc = 2 * acc;
    }
    // MID: M
    return acc;
}
```

a) Write a midcondition M which holds after the loop and is strong enough to prove partial correctness of the `mathPower` method. (You do not need to prove anything.)

b) Write a loop invariant I which is strong enough to prove partial correctness of the `mathPower` method. (You do not need to prove anything.)

c) Write a loop variant V which is strong enough to prove total correctness of the `mathPower` method. (You do not need to prove anything.)

Hint: In order to find appropriate M and I, observe that after the loop we have that $\mathtt{m} = \mathtt{cnt}$. Therefore, M can have the shape:

$$M \;\longleftrightarrow\; \mathtt{m} = \mathtt{cnt} \wedge ???$$

Moreover, we want to have that

$$M \;\longrightarrow\; POST[\mathbf{r} \mapsto \mathtt{acc}]$$

All the above gives us that:

$$\mathtt{m} = \mathtt{cnt} \wedge ??? \quad\longrightarrow\quad \mathtt{acc} = 2^{\mathtt{m}}$$

It remains to find appropriate assertion for ???. This assertion should say something about the value of `acc` (since the right hand side of the implication is about `acc`).

Of course, you may chose a different avenue to find M and I. In fact, there exist several different possibilities for M and I.

A possible answer:

a) $M \quad \longleftrightarrow \quad \texttt{m} = \texttt{cnt} \ \wedge \ \texttt{acc} = 2^{\texttt{cnt}}$.
It can be easily shown that $M \ \rightarrow \ \texttt{acc} = 2^{\texttt{m}}$.

Note that M' defined below is another possible answer:

$$M' \quad \longleftrightarrow \quad \texttt{acc} = 2^{\texttt{m}}.$$

b) $I \quad \longleftrightarrow \quad 0 \leq \texttt{cnt} \leq \texttt{m} \ \wedge \ \texttt{acc} = 2^{\texttt{cnt}}$.

Note that $0 \leq \texttt{cnt}$ is required to ensure that $2^{\texttt{cnt}}$ has an integer value (and so can be legally stored in `acc`).

c) $V \quad = \quad \texttt{m} - \texttt{cnt}$.