Reasoning About Programs

Loops: finding the invariants, finding the substitutions, doing the proofs

Sophia Drossopoulou and Mark Wheelhouse

The function $EqLocs$ calculates the number of equal elements for arrays `cs` and `ds`, provided they have the same length:

$$EqLocs(\texttt{c}, \texttt{d}) = \begin{cases} \mid \{\ j \mid <\in [0..\texttt{cs.length}) \ \land \ \texttt{cs}[j] = \texttt{ds}[j]\ \} \mid & \text{if } \texttt{cs.length} = \texttt{ds.length} \\ \text{undefined} & \text{otherwise} \end{cases}$$

As an example, $EqLocs(\texttt{'DEFGH'}, \texttt{'DXFXH'}) = 3$. Consider the following Java method that claims to perform the same calculation:

```
int numberOverlap (int[] a, int[] b)
// PRE: a ≠ null ∧ b ≠ null ∧ a.length = b.length
// POST: r = EqLocs(a[..]_pre, b[..]_pre)
{
    // MID: M1
    int res = a.length;
    int i = a.length;
    // INV: I
    // VAR: V
    while (i > 0) {
        i--;
        if !(a[i] == b[i]) { res--; }
    }
    // MID: M2
    return res;
}
```

In the questions below, you may find it helpful to use the following lemmas. You do not need to prove them, but it is advisable to consider their meaning.

Lemma 1 $\forall m, n \in \mathbb{N}. \forall \texttt{c}, \texttt{d} \in \mathbb{N}^*.$
$$[\ n \le m \le \texttt{c.length} = \texttt{d.length} \quad \longrightarrow \quad EqLocs(\texttt{c}[m..n), \texttt{d}[m..n)) = 0\ \ ]$$

Lemma 2 $\forall m, n \in \mathbb{N}. \forall \texttt{c}, \texttt{d} \in \mathbb{N}^*.$
$$[\ k \le m \le n \quad \longrightarrow$$
$$EqLocs(\texttt{c}[k..n), \texttt{d}[k..n)) = EqLocs(\texttt{c}[k..m), \texttt{d}[k..m)) + EqLocs(\texttt{c}[m..n), \texttt{d}[m..n))\ ]$$

Lemma 3 $\forall k \in \mathbb{N}. \forall \texttt{c}, \texttt{d} \in \mathbb{N}^*.$
$$[\quad EqLocs(\texttt{c}[k..k+1), \texttt{d}[k..k+1)) \ = \ \begin{cases} 1 & \text{if } \texttt{c}[k] = \texttt{d}[k] \\ 0 & \textit{otherwise} \end{cases} \quad]$$

a) Complete the specification of the method `numberOverlap`.

   i) Write a mid-condition $M1$ which holds at the beginning of the function body.

ii) Write a loop invariant I.

iii) Write a loop variant V.

iv) Write a mid-condition $M2$ which holds after the loop.

Hint: The most challenging part will be the characterization of the value of `res` in the invariant I. To go about finding the invariant, one can use "mechanistic" or "intuitive" thinking.

In the "mechanistic" thinking we try to find some I, such that

$$I \ \wedge \ \neg(\texttt{i > 0}) \quad \rightarrow \quad M2.$$

In the particular case, we are looking for something of the form

$$\texttt{res} = \ldots\ EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..))\ldots \ \wedge \ \texttt{i} = 0 \quad \rightarrow \quad \texttt{res} = EqLocs(\texttt{a}[..), \texttt{b}[..)).$$

The reason we have $EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..))$ to the left of the arrow, is that at each step, the value of `res` depends on the value of $EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..)$.

In the "intuitive" thinking we try to understand *how* the loop achieves its goal. The following thoughts might help: the loop starts as if all entries in `a` and `b` were equal, and then decrements the value of `res` when it finds an index where they differ. At each step, the loop has exact knowledge of equalities for all entries at `i` and beyond, but assumes that all entries before `i` are equal.

Finally, we could also draw inspiration by looking at the following two variations of the code above:

```
       ...
       int res = a.length;
       int i = 0;
       // INV: I
       // VAR: V
       while (i < a.length) {
           if !(a[i] == b[i])
               { res--; }
           i++
       }
      ...
```

```
       ...
       int res = 0;
       int i = a.length;
       // INV: I
       // VAR: V
       while (i > 0) {
           i--;
           if (a[i] == b[i])
               { res++; }
       }
       ...
```

b) Prove that the loop invariant I is established before entering the loop.

c) Prove that the loop re-establishes the loop invariant I.

d) Prove that the mid-condition $M2$ holds immediately after the termination of the loop.

e) Prove that `numberOverlap` is partially correct.

f) Prove that `numberOverlap` terminates.

g) Where did you use the condition that `a.length` $=$ `b.length`.

To clarify what needs to be proven, and what substitutions are applied, the sample answer has explanations given in the following style

Comment: Here is some further discussion. □

Such explanations are not expected in exams.

A possible answer:

a) We define all the assertions in the code

a) $M1 \triangleq$ $\mathtt{a} \neq \mathtt{null} \neq \mathtt{b} \ \wedge\ \mathtt{a.length} = \mathtt{b.length}\ \wedge$
$\mathtt{a}[..] \approx \mathtt{a}[..]_{pre} \ \wedge\ \mathtt{b}[..] \approx \mathtt{b}[..]_{pre}$

b) $I \triangleq$ $\mathtt{a} \neq \mathtt{null} \neq \mathtt{b} \ \wedge\ \mathtt{a.length} = \mathtt{b.length}\ \wedge$
$\mathtt{a}[..] \approx \mathtt{a}[..]_{pre} \ \wedge\ \mathtt{b}[..] \approx \mathtt{b}[..]_{pre}\ \wedge$
$\mathtt{i} \in [0..\mathtt{a.length}] \ \wedge\ \mathtt{res} = \mathtt{i} + \mathit{EqLocs}(\mathtt{a}[\mathtt{i}..), \mathtt{b}[\mathtt{i}..))$

Comment: $\mathtt{res} = \mathtt{i} + \mathit{EqLocs}(\mathtt{a}[\mathtt{i}..), \mathtt{b}[\mathtt{i}..))$ says that at each step the loop assumes that all entries preceding `i` are equal, while for those at `i` and beyond it had exact knowledge. Note that $\mathtt{i} \in [0..\mathtt{a.length}]$, and *not* $\mathtt{i} \in [0..\mathtt{a.length})$, nor $\mathtt{i} \in (0..\mathtt{a.length})$ □

c) $M2 \triangleq$ $\mathtt{res} = \mathit{EqLocs}(\mathtt{a}[..)_{pre}, \mathtt{b}.[..)_{pre})$

d) $V = \mathtt{i}$

b) Proving that loop invariant holds before entering the loop. *Comment*: For this we prove:
$M1 \ \wedge\ \mathtt{res} = ... \ \wedge\ \mathtt{i} = ... \ \longrightarrow\ I.$
Notice that we do not apply any substitutions in $M1$ because, it $M1$ does not contain any assertions about `res` or `i`. □

Given:

(1)	$\mathtt{a} \neq \mathtt{null} \neq \mathtt{b} \ \wedge\ \mathtt{a.length} = \mathtt{b.length}$	from ($M1$)
(2)	$\mathtt{a}[..) \approx \mathtt{a}[..)_{pre} \ \wedge\ \mathtt{b}[..) \approx \mathtt{b}[..)_{pre}$	from ($M1$)
(3)	$\mathtt{res} = \mathtt{a.length}$	from code, line 6
(4)	$\mathtt{i} = \mathtt{a.length}$	from code, line 7

To show:

(α)	$\mathtt{a} \neq \mathtt{null} \neq \mathtt{b} \ \wedge\ \mathtt{a.length} = \mathtt{b.length}$	INV
(β)	$\mathtt{a}[..) \approx \mathtt{a}[..)_{pre} \ \wedge\ \mathtt{b}[..) \approx \mathtt{b}[..)_{pre}$	INV
(γ)	$\mathtt{i} \in [0..\mathtt{a.length}]$	INV
(δ)	$\mathtt{res} = \mathtt{i} + \mathit{EqLocs}(\mathtt{a}[\mathtt{i}..), \mathtt{b}[\mathtt{i}..))$	INV

Proof:

(α) follows directly from (1)
(β) follows directly from (2)
(γ) follows directly from (4)
(5)$\mathit{EqLocs}(\mathtt{a}[\mathtt{a.length}..), \mathtt{b}[\mathtt{a.length}..)) = 0$
by Lemma 1, and because `a.length=b.length`
(δ) follows from (5), (3) and (4)

c) Proving that the loop re-establishes the loop invariant. *Comment*: For this we prove:
$I[\mathtt{i} \mapsto \mathtt{i}_{old}, \mathtt{res} \mapsto \mathtt{res}_{old}] \ \wedge\ (\mathtt{i} > 0)[\mathtt{i} \mapsto \mathtt{i}_{old}] \ \wedge\ \mathtt{res} = ... \ \wedge\ \mathtt{i} = ... \ \longrightarrow\ I.$
We applied the substitution $[\mathtt{i} \mapsto \mathtt{i}_{old}, \mathtt{res} \mapsto \mathtt{res}_{old}]$ to the invariant describing the state before execution, because the variables `i` and `res` are updated by the code. Also, we substituted `i` in the condition, because the condition uses the old version of `i`. □

Given:

(1)	$\texttt{a} \neq \texttt{null} \neq \texttt{b} \quad \wedge \quad \texttt{a.length} = \texttt{b.length}$	INV
(2)	$\texttt{a}[..] \approx \texttt{a}[..]_{pre} \quad \wedge \quad \texttt{b}[..] \approx \texttt{b}[..]_{pre}$	INV
(3)	$\texttt{i}_{old} \in [0..\texttt{a.length}]$	INV
(4)	$\texttt{res}_{old} = \texttt{i}_{old} + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$	INV
(5)	$\texttt{i}_{old} > 0$	loop condition
(6)	$\texttt{i} = \texttt{i}_{old} - 1$	from code line 10
(7)	$\texttt{res}...$	according to line 12 in code

To show:

(α)	$\texttt{a} \neq \texttt{null} \neq \texttt{b} \quad \wedge \quad \texttt{a.length} = \texttt{b.length}$	INV
(β)	$\texttt{a}[..] \approx \texttt{a}[..]_{pre} \quad \wedge \quad \texttt{b}[..] \approx \texttt{b}[..]_{pre}$	INV
(γ)	$\texttt{i} \in [0..\texttt{a.length}]$	INV
(δ)	$\texttt{res} = \texttt{i} + EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..))$	INV

Proof:

(α) follows directly from (1)
(β) follows directly from (2)
(γ) follows from (3), (5) and (6).

We will now prove (δ). We first deconstruct $EqLocs(...)$ on the RHS of δ:
(8) $EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..)) = EqLocs(\texttt{a}[\texttt{i}..\texttt{i}_{old}), \texttt{b}[\texttt{i}..\texttt{i}_{old})) + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$
by Lemma 2 and (6)

We proceed by case analysis.

Case 1: $\texttt{a[i]} \neq \texttt{b[i]}$

(9)	$\texttt{res} = \texttt{res}_{old} - 1$	from code, line 12 and case
(10)	$EqLocs(\texttt{a}[\texttt{i}..\texttt{i}_{old}), \texttt{b}[\texttt{i}..\texttt{i}_{old})) = 0$	from (6), case, and lemma 3
(11)	$EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..)) = EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$	from (8) and (10)
(12)	$\texttt{res} = \texttt{i}_{old} + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..)) - 1$	by (9) and (4)
	$= \texttt{i}_{old} - 1 + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$	by arithmetic
	$= \texttt{i} + EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..))$	by (6) and (11)

Fte **Case 2:** $\texttt{a[i]} = \texttt{b[i]}$

(9)	$\texttt{res} = \texttt{res}_{old}$	from code, line 12 and case
(10)	$EqLocs(\texttt{a}[\texttt{i}..\texttt{i}_{old}), \texttt{b}[\texttt{i}..\texttt{i}_{old})) = 1$	from (6), case, and lemma 3
(11)	$EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..)) = 1 + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$	from (8) and (10)
(12)	$\texttt{res} = \texttt{i}_{old} + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$	by (9), and (4)
	$= \texttt{i}_{old} - 1 + 1 + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$	by arithmetic
	$= \texttt{i} + EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..))$	by (6) and (11)

(δ) follows in both cases.

d) Showing that $M2$ holds after the loop. *Comment*: Here we are proving:
$$I \;\wedge\; \neg(\texttt{i} > 0) \quad \longrightarrow \quad M2.$$
No substitution needed here, as no code is involved. $\square$

Given:

(1)	$\texttt{a} \neq \texttt{null} \neq \texttt{b} \quad \wedge \quad \texttt{a.length} = \texttt{b.length}$	INV
(2)	$\texttt{a}[..) \approx \texttt{a}[..)_{pre} \quad \wedge \quad \texttt{b}[..) \approx \texttt{b}[..)_{pre}$	INV
(3)	$\texttt{i} \in [0..\texttt{a.length}]$	INV
(4)	$\texttt{res} = \texttt{i} + EqLocs(\texttt{a}[\texttt{i}..), \texttt{b}[\texttt{i}..))$	INV
(5)	$\texttt{i} \leq 0$	negated loop condition

To show:

(α)	$\texttt{res} = EqLocs(\texttt{a}[..)_{pre}, \texttt{b}[..)_{pre})$	$(M2)$

Proof:

(6)	$\texttt{i} = 0$	from (3) and (5)
(7)	$\texttt{res} = 0 + EqLocs(\texttt{a}[0..), \texttt{b}[0..))$	from (6) and (4)
(8)	$\texttt{res} = EqLocs(\texttt{a}[..), \texttt{b}[..))$	from (7), arithmetic and that $\texttt{xs}[..) = \texttt{xs}[\texttt{o}..)$

(α) follows from (8) and (1)

e) Showing that $M2$ and return implies $POST$.
Comment: Here we are proving:

$M2 \ \wedge \ \ \texttt{r} = \texttt{res} \quad \longrightarrow \quad POST.$

No substitution needed here. $\square$

Given:

(1)	$\texttt{res} = EqLocs(\texttt{a}[..)_{pre}, \texttt{b}[..)_{pre})$	from M2
(4)	$\texttt{r} = \texttt{res}$	from code line 15

To show:

(α)	$\texttt{r} = EqLocs(\texttt{a}[..)_{pre}, \texttt{b}[..)_{pre})$	POST

Proof:

(α) follows from (1) and (2)

f) We first show that the loop terminates. *Comment*: Here we are proving:

$I[\texttt{i} \mapsto \texttt{i}_{old}, \texttt{res} \mapsto \texttt{res}_{old}] \ \wedge \ (\texttt{i} > 0)[\texttt{i} \mapsto \texttt{i}_{old}] \ \wedge \ \texttt{res} = ... \ \wedge \ \texttt{i} = ...$
$\longrightarrow$
$\texttt{i} \geq 0 \ \wedge \ \texttt{i} < \texttt{i}_{old}.$ $\square$

Given:

(1)	$\texttt{a} \neq \texttt{null} \neq \texttt{b} \quad \wedge \quad \texttt{a.length} = \texttt{b.length}$	INV
(2)	$\texttt{a}[..) \approx \texttt{a}[..)_{pre} \quad \wedge \quad \texttt{b}[..) \approx \texttt{b}[..)_{pre}$	INV
(3)	$\texttt{i}_{old} \in [0..\texttt{a.length}]$	INV
(4)	$\texttt{res} = \texttt{i}_{old} + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$	INV
(5)	$\texttt{i}_{old} > 0$	loop condition
(6)	$\texttt{i} = \texttt{i}_{old} - 1$	from code line 10

To show:

(α)	$\texttt{i} \geq 0$	variant is bounded
(β)	$\texttt{i} < \texttt{i}_{old}$	variant decreases with every iteration

Proof:

(α) follows from (5) and (6)
(β) follows from (6)

We next show that all array accesses are valid. These are on lines 6, 7 and 12. We can give a shorter or a longer answer. In exams we would give same number of points to either.

Short Answer Here is a short answer (full marks in exams):

The term `a.length` on lines 6 and 7 does not throw a null-pointer exception, because $(M1)$ guarantees that $\texttt{a} \neq \texttt{null}$.

On line 12, we have that $\texttt{i}_{old} > 0$ by condition, that $\texttt{i} = \texttt{i}_{old} - 1$ by code line 11, that $\texttt{a} \neq \texttt{null}$ and that $\texttt{i}_{old} \in [0..\texttt{a.length}]$ by INV. These together give that $\texttt{i} \in [0..\texttt{a.length})$, and therefore, `a[i]` is a safe array access.

Moreover, because $\texttt{b} \neq \texttt{null}$ and $\texttt{a.length} = \texttt{b.length}$ by INV, and because $\texttt{i} \in [0..\texttt{a.length})$ from above, we have that $\texttt{i} \in [0..\texttt{b.length}]$, and therefore, `b[i]` is also a safe array access.

Long Answer First we show that the accesses on line 6 and 7 of the code are valid.

Given:

(1)	$\texttt{a} \neq \texttt{null} \neq \texttt{b} \quad \wedge \quad \texttt{a.length} = \texttt{b.length}$	M1
(2)	$\texttt{a}[..] \approx \texttt{a}[..]_{pre} \quad \wedge \quad \texttt{b}[..] \approx \texttt{b}[..]_{pre}$	M1

To show:

(α)	$\texttt{a} \neq \texttt{null}$	lines 6, 7 do not raise exceptions

Proof:

(α) follows from (1)

Next we show that the access on line 7 of the code is valid.

Given:

(1)	$\texttt{a} \neq \texttt{null} \neq \texttt{b} \quad \wedge \quad \texttt{a.length} = \texttt{b.length}$	INV
(2)	$\texttt{a}[..] \approx \texttt{a}[..]_{pre} \quad \wedge \quad \texttt{b}[..] \approx \texttt{b}[..]_{pre}$	INV
(3)	$\texttt{i}_{old} \in [0..\texttt{a.length}]$	INV
(4)	$\texttt{res} = \texttt{i}_{old} + EqLocs(\texttt{a}[\texttt{i}_{old}..), \texttt{b}[\texttt{i}_{old}..))$	INV
(5)	$\texttt{i}_{old} > 0$	loop condition
(6)	$\texttt{i} = \texttt{i}_{old} - 1$	from code line 11

To show:

(α)	$\texttt{a} \neq \texttt{null} \neq \texttt{b} \quad \wedge \quad \texttt{i} \in [0..\texttt{a.length})$	`a[i]` line 12, valid
(β)	$\texttt{a} \neq \texttt{null} \neq \texttt{b} \quad \wedge \quad \texttt{i} \in [0..\texttt{b.length})$	`b[i]` line 12, valid

Proof:

(α) follows from (1), (3), (5) and (6)
(β) follows from (1), (α) and (1)

g) We used the fact that `a.length=b.length` in the application of Lemma 1, when proving that the code in lines 6-7 establishes the invariant. We also used it in the application of Lemma 2, when proving that the loop body re-establishes the invariant. Finally, we used it in the proof that all array accesses are valid.