Sorting Flags

Sample Answers

Sophia Drossopoulou and Mark Wheelhouse

(based on exercise sheet by Krysia Broda)

**1st Question:**

i) If we removed the Perm(a, a0) condition from the post-condition of sort method then we would be able to make modifications to the array using methods other than swap. In particular, we could satisfy the post-condition by simply replacing every stripe by a Red stripe. Thus, the sort method would not necessarily solve the 4 Stripe Flag Problem.

ii) If we changed the last line of the post-condition from

| k1 - (k3 -k2) | ≤ 1 ∧ | (k2 -k1)- (a.length - k3) | ≤ 1

to

k1 - (k3 -k2) ≤ 1 ∧ (k2 -k1)- (a.length - k3) ≤ 1

then we would not need to keep the size of the Red and White stripes within one of each

other. In particular, we could satisfy the post-condition by sorting the array into just two

stripes (i.e. we could leave the first Red and White stripes empty). Thus, the sort method

would not necessarily solve the 4 Stripe Flag Problem.

**2nd Question:**

i) There is only one possible output in this case:  
 a = [R|R|W|W|R|R|W|W]

ii) There are four possible outputs in this case:  
 a = [R|R|R|W|W|R|R|W]  
 a = [R|R|R|W|R|R|W|W]  
 a = [R|R|W|W|R|R|R|W]  
 a = [R|R|W|R|R|R|W|W]

**3rd Question:**

We follow a similar strategy as set out for the Dutch National Flag problem given in the

lecture notes.

1. ***From post-condition to invariant:***

The post-condition states:

Perm(a, a0) ∧   
 k1, k2,k3  [0..a.length]  
 a[0..k1) = Red ∧ a[k1..k2) = White ∧ a[k2..k3) = Red ∧ a[k3..a.length) = White   
 ∧ | k1 - (k3 -k2) | ≤ 1 ∧ | (k2 -k1)- (a.length - k3) | ≤ 1

The invariant is a generalization of the post-condition, it states:

(I1) Perm(a, a0) ∧   
 (I2) 0 ≤ w1 ≤ g ≤ r ≤w2 ≤ a.length ∧  
 (I3) a[0..w1) = Red ∧ a[w1..g) = White ∧ a[r..w2) = Red ∧ a[w2..a.length) = White ∧  
 (I4) | w1 - (w2 -r) | ≤ 1 ∧ | (g-w1)- (a.length -w2) | ≤ 1 ∧  
 (I5) evenR ↔ w1 = (w2 -r) ∧ evenW ↔ (g-w1)= (a.length -w2)

1. ***From invariant to loop condition:***

We need a loop condition such that INV and **¬**cond implies MID.  
Similar to the lecture notes, the only choice is **¬**cond ≡ (g = r) so we let the loop condition be (g < r).

1. ***From invariant to loop code:***

**while (**g < r) { **switch**(a[g]){

**case** Red:

**if**( evenR ){

swap(a,w1,g);

w1++;

g++;

}

**else**{

swap(a,g,r-1);

r--;

}

evenR = !evenR;

**break;**

**case** White:  
 **if**( evenW ){

g++;

}

**else**{

swap(a,g,w2-1);

**if**(w2 != r){

swap(a,g,r-1);

}

w2--;

r--;

}

evenW = !evenW;

**break;**

}

}

**iv) *From invariant to initialization:***

**int** w1=0; **int** g=0; **int** r=a.length; **int** w2=a.length; **boolean** evenR = **true**; **boolean** evenW = **true**;

Note: the ranges [0..0) and [a.length..a.length) are empty, therefore the invariant trivially holds.

**v) *Array Accesses:***

From the loop condition we have that g < r. From the invariant we also have that   
w1 ≤ g ≤ r, and therefore we can obtain that 0 ≤ g < a.length. Thus, all of the array access a[g] are valid. Any array accesses within the sort method will be handled by checking that the precondition of sort holds before it is called (we cover this in part viii).

**vi) *Termination:***

The variant (r-g) decreases after each loop iteration, aseither g increases, or r decreases. From the invariant, we can see that the variant has 0 as its lower bound.

**vii) *Putting it all together:***

(see separate answer sheet.)

**viii) *Prove that the code satisfies its specification*,** (proof to follow, but it is similar to the proof for the Dutch National Flag code in the lecture notes.)