Problem Sheet 5: Program Proofs
Sample Solutions

Chris Poskitt
ETH Zürich

Starred exercises (∗) are more challenging than the others.

1 Axiomatic Semantics Recap

i. I propose the axiom:

$$\vdash \{p\}\ \texttt{havoc}(\texttt{x}_0, \ldots, \texttt{x}_n)\ \{\exists x_0^{\text{old}}, \ldots, x_n^{\text{old}}.\ p[x_0^{\text{old}}/x_0, \ldots, x_n^{\text{old}}/x_n]\}$$

Essentially it is the same as the forward assignment axiom (see Problem Sheet 1), but without conjuncts about the new values of each x_i, since we do not know what they will be after the execution of `havoc`.

ii. Below is a possible program and proof outline:

$\{x \geq 0\}$
$\{x! * 1 = x! \wedge x \geq 0\}$
  $y := 1;$
$\{x! * y = x! \wedge x \geq 0\}$
  $z := x;$
$\{z! * y = x! \wedge z \geq 0\}$
  while $z > 0$ **do**
  $\{z > 0 \wedge z! * y = x! \wedge z \geq 0\}$
  $\{(z-1)! * (y * z) = x! \wedge (z-1) \geq 0\}$
    $y := y * z;$
  $\{(z-1)! * y = x! \wedge (z-1) \geq 0\}$
    $z := z - 1;$
  $\{z! * y = x! \wedge z \geq 0\}$
  end
$\{\neg(z > 0) \wedge z! * y = x! \wedge z \geq 0\}$
$\{y = x!\}$

Observe that the loop invariant $z! * y = x! \wedge z \geq 0$ is key to completing the proof.

iii. A possible inference rule would be:

$$[\text{from-until}]\ \frac{\vdash \{p\}\ A\ \{inv\} \qquad \vdash \{inv \wedge \neg b\}\ C\ \{inv\}}{\vdash \{p\}\ \texttt{from}\ A\ \texttt{until}\ b\ \texttt{loop}\ C\ \texttt{end}\ \{inv \wedge b\}}$$

iv. A possible proof outline is the following:

```
{ n >= 0 }
  from
     k := n
     found := False
  { 0 <= k <= n ∧ (found ⟹ 1 <= k <= n ∧ A[k] = v) }
  until found or k < 1 loop
    { 1 <= k <= n ∧ ¬ found ∧ (found ⟹ 1 <= k <= n ∧ A[k] = v) }
    if A[k] = v then
      { A[k] = v ∧ 1 <= k <= n ∧ ¬ found }
      { 0 <= k <= n ∧ 1 <= k <= n ∧ A[k] = v }
      found := True
      { 0 <= k <= n ∧ (found ⟹ 1 <= k <= n ∧ A[k] = v) }
    else
      { A[k] /= v ∧ 1 <= k <= n ∧ ¬ found }
      { 1 <= k <= n + 1 ∧ (found ⟹ 2 <= k <= n + 1 ∧ A[k − 1] = v) }
      k := k − 1
      { 0 <= k <= n ∧ (found ⟹ 1 <= k <= n ∧ A[k] = v) }
    end
    { 0 <= k <= n ∧ (found ⟹ 1 <= k <= n ∧ A[k] = v) }
  end
  { (found ∧ 1 <= k <= n ∧ A[k] = v) ∨ (¬found ∧ k = 0) }
{(found ⟹ 1 <= k <= n ∧ A[k] = v) ∧ ( ¬found ⟹ k < 1 )}
```

Again, note the importance of determining a strong enough loop invariant, i.e.

$$0 \leq k \leq n \wedge (found \implies 1 \leq k \leq n \wedge A[k] = v)$$

for the proof to be able to go through. Note also that we can apply backwards reasoning, as usual, when the assignment involves a Boolean value (in this case, $found[True/found] \equiv True$).

v. Assume that $\vdash$ {WP[$P, post$]} P {$post$} and $\models$ {p} P {q}. From the definition of $\models$, executing P on a state satisfying p results in a state satisfying q. By definition, WP[$P, post$] expresses the weakest requirements on the state for P to establish q; hence p is either equivalent to or stronger than WP[$P, post$], and $p \Rightarrow$ WP[$P, post$] is valid. Clearly, $q \Rightarrow q$ is also valid, so we can apply the rule of consequence [cons] and derive the result that $\vdash \{p\}\ P\ \{q\}$.

Note: this property is called *relative completeness*, i.e. all valid triples can be proven in the Hoare logic, relative to the existence of an oracle for deciding the validity of implications (such as those in [cons]).

2 Separation Logic Recap

i. There are instances of s, h and p such that the state satisfies the first assertion. For example,

$$s, h \models x \mapsto x * \neg x \mapsto x$$

if $s(x) = 5$, $h(5) = 5$, and h is defined for no other values. However, $x = y * \neg(x = y)$ is not satisfiable since x, y denote values in the store, which is heap-independent.

ii. (a) Satisfies.
  (b) Does not satisfy (the heap only contains two locations).
  (c) Does not satisfy (the heap contains more than one location).
  (d) Satisfies. The variables x and y are indeed evaluated to the same location by the store. The second conjunct expresses that there is a location in the heap determined by evaluating y (clearly true).
  (e) Satisfies.

iii. A proof outline is given below:

$\{\text{emp}\}$

  $x := \text{cons}(5, 9);$

$\{x \mapsto 5, 9\}$

  $y := \text{cons}(6, 7);$

$\{x \mapsto 5, 9 * y \mapsto 6, 7\}$

$\{\exists x^{\text{old}}.\ x \mapsto 5, 9 * y \mapsto 6, 7 \wedge x^{\text{old}} = x\}$

  $x := [x];$

$\{\exists x^{\text{old}}.\ x^{\text{old}} \mapsto 5, 9 * y \mapsto 6, 7 \wedge x = 5\}$

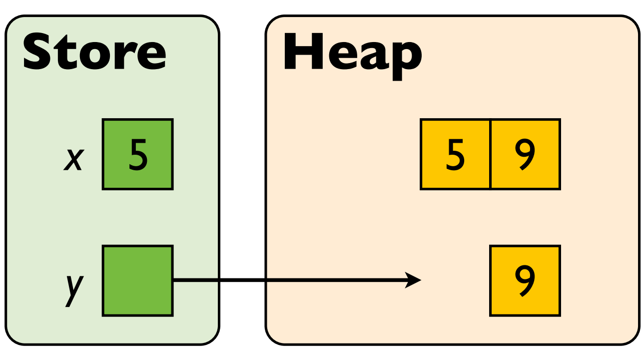
  $[y + 1] := 9;$

$\{\exists x^{\text{old}}.\ x^{\text{old}} \mapsto 5, 9 * y \mapsto 6, 9 \wedge x = 5\}$

  $\text{dispose}(y);$

$\{\exists x^{\text{old}}.\ x^{\text{old}} \mapsto 5, 9 * y + 1 \mapsto 9 \wedge x = 5\}$

and a depiction of the final state:



iv. A proof outline is given below:

$\{tree\ (1, t)\ i\}$
$\{\exists l, r \cdot i \mapsto l, r * tree\ 1\ l * tree\ t\ r\}$

$x := [i];$

$\{\exists r \cdot i \mapsto x, r * tree\ 1\ x * tree\ t\ r\}$

$[i] := 2;$

$\{\exists r \cdot i \mapsto 2, r * tree\ 1\ x * tree\ t\ r\}$

$y := [i + 1];$

$\{i \mapsto 2, y * tree\ 1\ x * tree\ t\ y\}$

dispose i;

$\{(i + 1) \mapsto y * tree\ 1\ x * tree\ t\ y\}$

$\{(i + 1) \mapsto y * x \mapsto 1 * tree\ t\ y\}$

dispose x;

$\{(i + 1) \mapsto y * tree\ t\ y\}$

dispose $(i + 1)$;

$\{tree\ t\ y\}$