Problem Sheet 9: Program Proofs
Sample Solutions

Chris Poskitt
ETH Zürich

1 Axiomatic Semantics

i. I propose the axiom:

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

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 \text{havoc}.

ii. Below is a possible program and proof outline:

\[
\begin{align*}
\{ x \geq 0 \} \\
\{ x! \cdot 1 = x! \land x \geq 0 \} \\
y := 1; \\
\{ x! \cdot y = x! \land x \geq 0 \} \\
z := x; \\
\{ z! \cdot y = x! \land z \geq 0 \} \\
\textbf{while} z > 0 \textbf{ do} \\
\{ z > 0 \land z! \cdot y = x! \land z \geq 0 \} \\
\{ (z - 1)! \cdot (y \cdot z) = x! \land (z - 1) \geq 0 \} \\
y := y \cdot z; \\
\{ (z - 1)! \cdot y = x! \land (z - 1) \geq 0 \} \\
z := z - 1; \\
\{ z! \cdot y = x! \land z \geq 0 \} \\
\textbf{end} \\
\{ \neg (z > 0) \land z! \cdot y = x! \land z \geq 0 \} \\
\{ y = x! \} \\
\end{align*}
\]

Observe that the loop invariant \( z! \cdot y = x! \land z \geq 0 \) is key to completing the proof.
iii. A possible inference rule would be:

\[
\begin{align*}
  &\vdash \{ p \} \text{ inv} & \vdash \{ inv \land \neg b \} \ C \ \{ inv \} \\
  &\vdash \{ p \} \text{ from } A \text{ until } b \text{ loop } C \text{ end } \{ inv \land b \}
\end{align*}
\]

iv. A possible proof outline is the following:

\[
\begin{align*}
\{ n \geq 0 \} \\
\text{from} \\
\quad k := n \\
\quad \text{found} := \text{False} \\
\quad \{ 0 \leq k \leq n \land (\text{found} \implies 1 \leq k \leq n \land A[k] = v) \} \\
\text{until found or } k < 1 \text{ loop} \\
\quad \{ 1 \leq k \leq n \land \neg \text{found} \land (\text{found} \implies 1 \leq k \leq n \land A[k] = v) \} \\
\quad \text{if } A[k] = v \text{ then} \\
\quad \quad \{ A[k] = v \land 1 \leq k \leq n \land \neg \text{found} \} \\
\quad \quad \{ 0 \leq k \leq n \land 1 \leq k \leq n \land A[k] = v \} \\
\quad \quad \text{found} := \text{True} \\
\quad \quad \{ 0 \leq k \leq n \land (\text{found} \implies 1 \leq k \leq n \land A[k] = v) \} \\
\quad \text{else} \\
\quad \quad \{ A[k] \neq v \land 1 \leq k \leq n \land \neg \text{found} \} \\
\quad \quad \{ 1 \leq k \leq n + 1 \land (\text{found} \implies 2 \leq k \leq n + 1 \land A[k-1] = v) \} \\
\quad \quad k := k - 1 \\
\quad \quad \{ 0 \leq k \leq n \land (\text{found} \implies 1 \leq k \leq n \land A[k] = v) \} \\
\quad \text{end} \\
\quad \{ 0 \leq k \leq n \land (\text{found} \implies 1 \leq k \leq n \land A[k] = v) \} \\
\text{end} \\
\quad \{ (\text{found} \land 1 \leq k \leq n \land A[k] = v) \lor (\neg \text{found} \land k = 0) \} \\
\quad \{ (\text{found} \implies 1 \leq k \leq n \land A[k] = v) \land (\neg \text{found} \implies k < 1) \}
\end{align*}
\]

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

\[0 \leq k \leq n \land (\text{found} \implies 1 \leq k \leq n \land A[k] = v)\]

for the proof to be able to go through. Note that we can still apply backwards reasoning when the assignment involves a Boolean value (in this case, \(\text{found[True/found]} = \text{True}\)).

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

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

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

$$s, h \models x \mapsto x \neq x \mapsto x$$

if $s(x) = 5$, $h(5) = 5$, and $h$ is defined for no other values. However, $x = y \neq (x = y)$ 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:

\[
\{\text{emp}\} \\
\hspace{1cm} x := \text{cons}(5, 9); \\
\hspace{1cm} \{x \mapsto 5, 9\} \\
\hspace{1cm} y := \text{cons}(6, 7); \\
\hspace{1cm} \{x \mapsto 5, 9 + y \mapsto 6, 7\} \\
\hspace{1cm} \exists x^{\text{old}}. x \mapsto 5, 9 + y \mapsto 6, 7 \land x^{\text{old}} = x \\
\hspace{1cm} x := [x]; \\
\hspace{1cm} \exists x^{\text{old}}. x^{\text{old}} \mapsto 5, 9 + y \mapsto 6, 7 \land x = 5 \\
\hspace{1cm} [y + 1] := 9; \\
\hspace{1cm} \exists x^{\text{old}}. x^{\text{old}} \mapsto 5, 9 + y \mapsto 6, 9 \land x = 5 \\
\hspace{1cm} \text{dispose}(y); \\
\hspace{1cm} \exists x^{\text{old}}. x^{\text{old}} \mapsto 5, 9 + y + 1 \mapsto 9 \land x = 5
\]

and a depiction of the final state:

![Diagram of store and heap states](image-url)
iv. A proof outline is given below:

\[
\{ \text{tree}((1, t), i) \} \\
\{ \exists l, r. \ i \mapsto l, r \ast \text{tree}(1, l) \ast \text{tree}(t, r) \} \\
\quad x := [i]; \\
\{ \exists r. \ i \mapsto x, r \ast \text{tree}(1, x) \ast \text{tree}(t, r) \} \\
\quad [i] := 2; \\
\{ \exists r. \ i \mapsto 2, r \ast \text{tree}(1, x) \ast \text{tree}(t, r) \} \\
\quad y := [i + 1]; \\
\{ i \mapsto 2, y \ast \text{tree}(1, x) \ast \text{tree}(t, y) \} \\
\quad \text{dispose } i; \\
\{ (i + 1) \mapsto y \ast \text{tree}(1, x) \ast \text{tree}(t, y) \} \\
\{ (i + 1) \mapsto y \ast x \mapsto 1 \ast \text{tree}(t, y) \} \\
\quad \text{dispose } x; \\
\{ (i + 1) \mapsto y \ast \text{tree}(t, y) \} \\
\quad \text{dispose } (i + 1); \\
\{ \text{tree}(t, y) \}
\]