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:

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

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:

$\{x \geq 0\}$

$\{x! * 1 = x! \land x \geq 0\}$

  $y := 1;$

$\{x! * y = x! \land x \geq 0\}$

  $z := x;$

$\{z! * y = x! \land z \geq 0\}$

  $\texttt{while}\ z > 0\ \texttt{do}$

  $\{z > 0 \land z! * y = x! \land z \geq 0\}$

  $\{(z-1)! * (y * z) = x! \land (z-1) \geq 0\}$

    $y := y * z;$

  $\{(z-1)! * y = x! \land (z-1) \geq 0\}$

    $z := z - 1;$

  $\{z! * y = x! \land z \geq 0\}$

  $\texttt{end}$

$\{\neg(z > 0) \land z! * y = x! \land z \geq 0\}$

$\{y = x!\}$

Observe that the loop invariant $z! * y = x! \land z \geq 0$ is key to completing the proof. The three implications arising from applications of [cons] can be shown to be valid through elementary mathematics and the definition of factorials.

iii. 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,

$$(x \mapsto 5), (5 \mapsto 5) \models x \mapsto x * \neg x \mapsto x$$

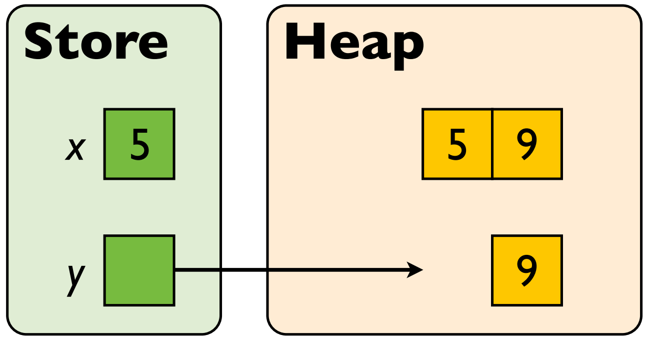
However, $x = y * \neg(x = y)$ is not satisfiable since x, y denote values in the store, which is heap-independent.

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:



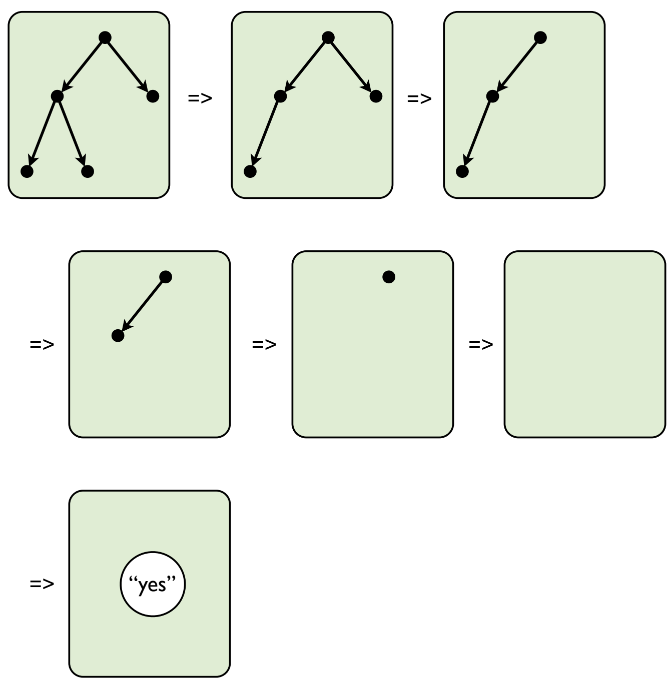
3 Graph-Based Reasoning and Verification

i. If P is executed on a graph satisfying c, then *any* graph that results will satisfy d.

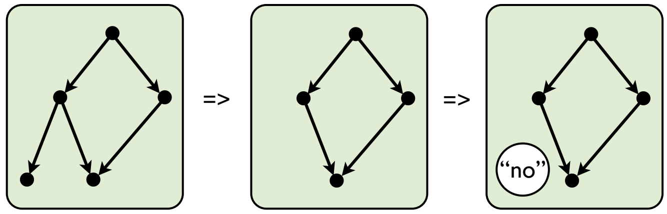
   This definition handles nondeterminism by requiring that all of the possible (proper) post-states satisfy the the postcondition. The definition does not guarantee the absence of program failures.

ii. The program (destructively) tests whether or not the input graph was a tree. It iteratively attempts to delete all the leaves by exploiting the *dangling condition* (nodes can only be deleted if all the edges they are incident to are *also* deleted by the rule), until finally only the root of the tree is left, and then deleted by `finalChop`. If at this stage the graph is empty, then the original graph was a tree; otherwise it was not.

   A possible yes-run:



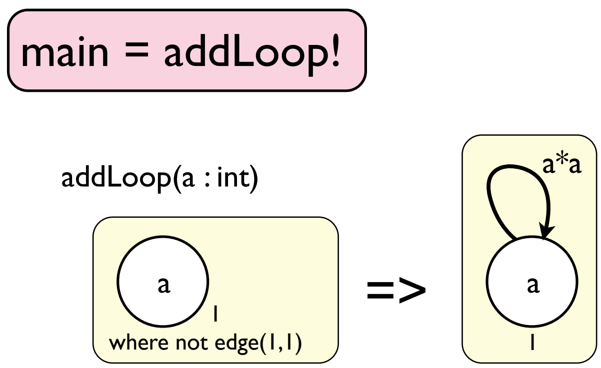
   . . . and a possible no-run:



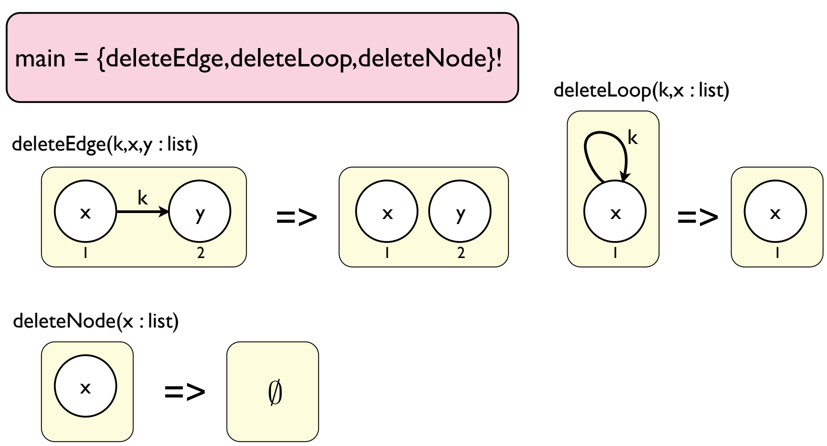
Graph reduction can be used to specify a wide range of pointer structures, see e.g.

http://www.cs.york.ac.uk/plasma/publications/pdf/BakewellPlumpRunciman.04b.pdf

iii. The following program should respect the given specification:



iv. The following program deletes the entire graph yet respects the given specification:



An obvious frame axiom would be: “the nodes, edges, and labels of the input graph are all preserved in the output graph”.

v. A possible proof rule might be:

$$[\text{or}]\ \frac{\vdash \{c\}\ P\ \{d\} \quad \vdash \{c\}\ Q\ \{d\}}{\vdash \{c\}\ P\ \texttt{or}\ Q\ \{d\}}$$

vi. A possible proof rule might be:

$$[\text{if}_2]\ \frac{\vdash \{c \wedge \text{App}(\mathcal{R})\}\ P\ \{d\} \quad c \wedge \neg\text{App}(\mathcal{R}) \Rightarrow d}{\vdash \{c\}\ \texttt{if}\ \mathcal{R}\ \texttt{then}\ P\ \{d\}}$$

vii. This expresses that there exists a node incident to a loop, and moreover, there is not another node distinct from it that also is incident to a loop: