Problem Sheet 3: Separation Logic

Chris Poskitt
ETH Zürich

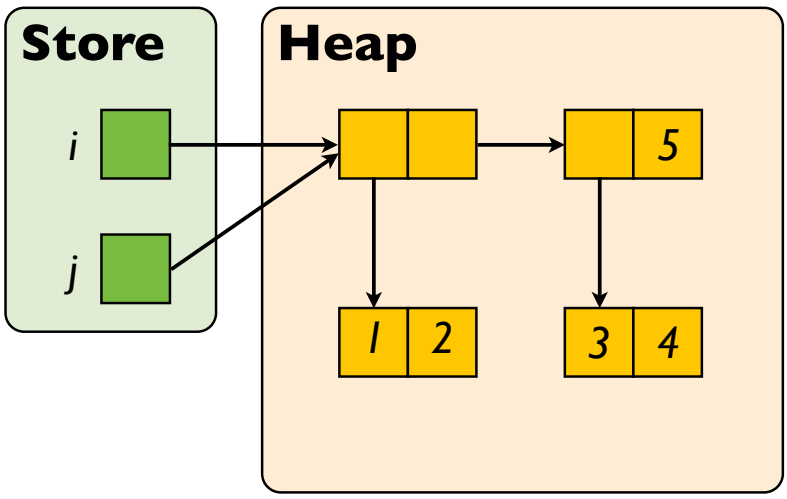
Starred exercises (∗) are more challenging than the others.

1 Separation Logic Assertions

These exercises are about assertions in separation logic, in particular, the *separating conjunction* ∗. For a reminder of the semantics of assertions (as well as numerous examples), please refer to the first set of separation logic lecture slides:

http://se.inf.ethz.ch/courses/2013b_fall/sv/slides/05-SeparationLogic-1.pdf

i. Consider the following state:



Which of the following assertions hold in this state? For the ones that do not: why not?

(a) $\exists z.\ i \mapsto z$

(b) $\exists z.\ i \mapsto z * j \mapsto z$

(c) $\exists z.\ i \mapsto z * \text{true}$

(d) $\exists z.\ (i \mapsto z * \text{true}) \wedge i = j \wedge (z \mapsto 1, 2 * \text{true})$

(e) $\exists x, x', y, z.\ i \mapsto x * j \mapsto x' * x \mapsto 1, 2 * j + 1 \mapsto y * y \mapsto z, 5 * z \mapsto 3, 4$

(f) $\text{true} * \text{emp}$

ii. Do the following implications hold for all states and predicates p, q? If not, why not?

$$p \text{ imples } p * p$$
$$p * q \text{ imples } [(p \wedge q) * \text{true}]$$

2 Separation Logic Proofs

These exercises involve *proving* specifications of heap-manipulating programs using separation logic. For some Hoare-style proof rules, please refer to the first problem sheet. For the small axioms and the frame rule of separation logic, please refer to the first set of separation logic lecture slides:

http://se.inf.ethz.ch/courses/2013b_fall/sv/slides/05-SeparationLogic-1.pdf

The second set of lecture slides demonstrates their use on two simple heap-manipulating programs:

http://se.inf.ethz.ch/courses/2013b_fall/sv/slides/05-SeparationLogic-2.pdf

i. Consider the following program:

```
l := cons(1);
r := cons(2,3);
temp1 := [r+1];
temp2 := [l];
[l] := temp1;
[r] := temp2;
```

Starting from precondition {emp}, apply the axioms and inference rules of separation logic to derive a postcondition expressing exactly the contents of the store and heap at termination. Then, depict this state using the store and heap diagrams presented in the lectures.

ii. We can assert that a heap portion contains a linked list by using the following inductively defined predicate:

$$\begin{aligned} \text{list}([], i) &\iff \text{emp} \wedge i = nil \\ \text{list}(a :: as, i) &\iff \exists j.\ i \mapsto a, j * \text{list}(as, j) \end{aligned}$$

where nil is a constant used to terminate the list.

Using the list predicate, verify the following program that deletes the first item in a non-empty linked list (assume that i points to the first node).

```
dispose(i);
k := [i+1];
dispose(i+1);
i := k;
```

iii. **(∗)** Consider the following inductively defined predicate:

$$\begin{aligned}\text{tree}(e,\tau) \iff & \texttt{if } (\text{isAtom}(e) \land e = \tau) \texttt{ then emp} \\ & \texttt{else } \exists x, y, \tau_1, \tau_2.\ \tau = \langle \tau_1, \tau_2 \rangle \\ & \land\ e \mapsto x, y * \text{tree}(x, \tau_1) * \text{tree}(y, \tau_2)\end{aligned}$$

where $\text{tree}(p, \tau)$ holds if p points to a data structure in memory representing the "mathematical" tree τ (i.e. the tree as an abstract mathematical object; not a representation in computer memory). Consider the following specification:

$$\{\text{tree}(p, \tau)\}\ \text{CopyTree}(p, q)\ \{\text{tree}(p, \tau) * \text{tree}(q, \tau)\}$$

which expresses that the procedure CopyTree stores—in a separate portion of memory pointed to by q—a mathematically equivalent tree to the one pointed to by p.

Define the procedure CopyTree and verify the specification.

Hint: the following derived axiom for heap lookups may simplify the proof:

$$\vdash \{e \mapsto e'\}\ x := [e]\ \{e \mapsto e' \land x = e'\}$$

provided that x does not appear free in e or e'.