Final exam, March 8, 2022

- Duration: 3 hours.
- Answers may be written in English or French.
- Lecture notes and personal notes are allowed. **Mobile phones must be switched off.** Electronic notes are allowed, but **all network connections must be switched off**, and the use of any electronic device must be restricted to reading notes: no typing, no using proof-related software.
- There are 4 pages and **4** exercises. Exercise 1 is about verification using weakest preconditions. Exercises 2 to 4 are about separation logic.
- Don't forget to put your name on each sheet of paper, and page numbers under the form 1/7, 2/7, etc. **Please write your answers for Exercise 1 on a separate piece of paper.**

1 Cumulative Sums of Arrays

In this exercise, we are interested in specifying and proving correct data structures that support efficient computation of *array range sums*: given an array a of integers, and given a range delimited by indices i (inclusive) and j (exclusive), we wish to compute the value $\sum_{k=i}^{j-1} a[k]$. Such a sum can be computed using the following recursive program.

```
let rec sum (a: array int) (i j:int) : int
  = if j <= i then 0 else a[i] + sum a (i+1) j
```

Question 1.1. *Which annotations (e.g. pre-conditions, loop invariants, etc.) should be given to the program above so as to be able to prove it safe and terminating? Justify informally (5 lines max) why your annotations suffice.*

The program above being pure and terminating, it provides a logic function `sum` that can be used further in logical annotations. We consider now an imperative code for computing a range sum as follows.

```
let query_sum (a:array int) (i j:int) : int
  ensures { result = sum a i j }
= let ref s = 0 in
  for k=i to j-1 do s ← s + a[k] done;
  s
```

Question 1.2. *Which annotations should be given to the program above so as to be able to prove it safe, terminating and satisfying its post-condition? Justify informally (10 lines max) why your annotations suffice. If any logical property is needed on the* `sum` *logic function, state it explicitly as a lemma, and explain how this lemma should be proved.*

The *cumulative array* data structure is an idea to compute the range sum queries in constant time, the cost being transferred to the array update. The cumulative array of an array a of length l is an array of length $l + 1$ whose value at index i is $\sum_{k=0}^{i-1} a[k]$. To express that c is the cumulative array of a, we provide the predicate

```
predicate is_cumulative_array_for (c:array int) (a:array int) =
  c.length = a.length + 1 /\
  ∀ i. 0 <= i < c.length → c[i] = sum a 0 i
```

The creation of the cumulative array of an array a is thus made as follows

```
let create (a:array int) : array int
  ensures { is_cumulative_array_for result a }
= let l = a.length in let s = Array.make (l+1) 0 in
  for i=1 to l do
      s[i] ← s[i-1] + a[i-1]
  done;
  s
```

Question 1.3. *Which annotations should be given to the program above so as to be able to prove it safe, terminating and satisfying its postcondition? Justify informally (10 lines max) why your annotations suffice.*

After creation of the cumulative array for a, it is not intended to keep a copy of a, so in the following a is kept only as a ghost parameter, for specification purposes. The constant-time function for range sum query is

```
let query (c:array int) (i j:int) (ghost a:array int): int
  requires { is_cumulative_array_for c a }
  ensures { result = sum a i j }
= c[j] - c[i]
```

Question 1.4. *Which annotations should be given to the program above so as to be able to prove it safe and satisfy its post-condition? Justify informally (10 lines max) why your annotations suffice. If any logical property is needed on the* `sum` *logic function, state it explicitly as a lemma, and explain how this lemma should be proved.*

An update of the original array a should impose an update to its cumulative array. This is done by the following program.

```
let update (c:array int) (i:int) (v:int) (ghost a:array int) : unit
    requires { is_cumulative_array_for c a }
    writes { a, c }
    ensures { is_cumulative_array_for c a }
    ensures { a[i] = v }
    ensures { ∀ k. 0 <= k < a.length /\ k <> i → a[k] = (old a)[k] }
  = let incr = v - c[i+1] + c[i] in a[i] ← v;
    for j=i+1 to c.length-1 do c[j] ← c[j] + incr done
```

Question 1.5. *Which annotations should be given to the program above so as to be able to prove it safe and satisfy its post-condition? Justify informally (20 lines max) why your annotations suffice. If any logical property is needed on the* `sum` *logic function, state it explicitly as a lemma, and explain how this lemma should be proved.*

2 Separation Logic: Heap Predicates

We recall that *heaps* are finite maps, or finite partial functions, from locations $l \in L$ to values $v \in V$, *i.e.* finite subsets m of $L \times V$ such that for all l, v_1, v_2: $(l, v_1) \in m \wedge (l, v_2) \in m \Rightarrow v_1 = v_2$. We write $|m|$ for the number of elements in m. We also recall the definition of two simple forms of heap predicates: the singleton heap predicate $\mapsto$, and the empty heap predicate with pure fact $[P]$:

$$\begin{array}{rcl} l \mapsto v & \equiv & \lambda m.\ m = \{(l, v)\} \quad \wedge \quad l \neq \mathsf{null} \\ [P] & \equiv & \lambda m.\ m = \varnothing \quad \wedge \quad P \end{array}$$

We also recall the definition of separating conjunction, $\star$ (note that $m_1 \perp m_2$ is short for $\mathsf{dom}(m_1) \cap \mathsf{dom}(m_2) = \varnothing$, where $\mathsf{dom}(m) \equiv \{l \mid (l, v) \in m\}$)

$$H_1 \star H_2 \quad \equiv \quad \lambda m.\ \exists m_1 m_2. \begin{cases} m_1 \perp m_2 \\ m = m_1 \uplus m_2 \\ H_1\, m_1 \\ H_2\, m_2 \end{cases}$$

Question 2.1. *Give a detailed proof that* $((a \mapsto b) \star (b \mapsto a))(m)$ *implies* $a \neq b$ *(3 lines should be enough).*

We recall the definition of heap entailment $\rhd$ and of existential quantification $\exists$:

$$\begin{array}{rcl} H_1 \rhd H_2 & \equiv & \forall m. H_1\, m \Rightarrow H_2\, m \\ \exists x. H & \equiv & \lambda m. \exists x. H m \end{array}$$

Question 2.2. *Let* GC *be the predicate* $\exists H. H$. *Give a detailed proof that* $(a \mapsto b) \star (b \mapsto a) \rhd [a \neq b] \star \mathsf{GC}$ *(6 lines should be enough).*

We also recall the definitions of (non-separating) conjunction ($\curlywedge$) and disjunction ($\curlyvee$):

$$\begin{aligned} H_1 \curlywedge H_2 &\equiv \lambda m.\ H_1\, m \wedge H_2\, m \\ H_1 \curlyvee H_2 &\equiv \lambda m.\ H_1\, m \vee H_2\, m \end{aligned}$$

Question 2.3. *Prove that* $(a \mapsto b) \curlywedge (b \mapsto a) \rhd [a = b] \star \mathsf{GC}$.

Question 2.4. *For each of the following equations of the form* $A \overset{?}{=} B$*, say whether* $A \rhd B$ *holds or not, and whether* $B \rhd A$ *holds or not. When the entailment holds, shortly explain why, and when it does not, give a counterexample.*

$$H_1 \curlywedge (H_2 \curlyvee H_3) \overset{?}{=} (H_1 \curlywedge H_2) \curlyvee (H_1 \curlywedge H_3) \tag{1}$$

$$H_1 \curlyvee (H_2 \curlywedge H_3) \overset{?}{=} (H_1 \curlyvee H_2) \curlywedge (H_1 \curlyvee H_3) \tag{2}$$

$$H_1 \star (H_2 \curlyvee H_3) \overset{?}{=} (H_1 \star H_2) \curlyvee (H_1 \star H_3) \tag{3}$$

$$H_1 \curlyvee (H_2 \star H_3) \overset{?}{=} (H_1 \curlyvee H_2) \star (H_1 \curlyvee H_3) \tag{4}$$

Let $\mathrel{-\!\!\star}$ be the "magic wand" or "separating implication" operator on heap predicates defined by:

$$H_1 \mathrel{-\!\!\star} H_2 \quad \equiv \quad \lambda m.\ \forall m_1\ (m_1 \perp m \wedge H_1\, m_1) \Rightarrow H_2(m_1 \uplus m)$$

Question 2.5. *Show that the following rules hold:*

$$\frac{H_1 \star H_2 \rhd H_3}{H_1 \rhd (H_2 \mathrel{-\!\!\star} H_3)} \qquad\qquad \frac{H_1 \rhd (H_2 \mathrel{-\!\!\star} H_3) \qquad H_4 \rhd H_2}{H_1 \star H_4 \rhd H_3}$$

Question 2.6. *For each of the following equation* $A \overset{?}{=} B$*, prove or disprove* $A \rhd B$ *and* $B \rhd A$*.*

$$([\,] \mathrel{-\!\!\star} H) \overset{?}{=} H \tag{5}$$

$$([\mathsf{False}] \mathrel{-\!\!\star} H) \overset{?}{=} [\,] \tag{6}$$

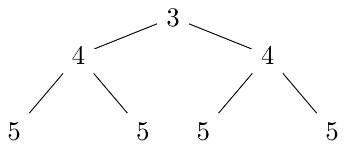
$$(H \mathrel{-\!\!\star} [\,]) \overset{?}{=} [H \rhd [\,]] \tag{7}$$

$$(l \mapsto v \mathrel{-\!\!\star} [\mathsf{False}]) \overset{?}{=} (l \mapsto \_ \star \mathsf{GC}) \tag{8}$$

(assume that $l \neq \mathsf{null}$*)*

3 Separation Logic: Mutable Trees from Mutable Lists

Hugo works in a startup that specializes in decision trees. He needs to turn lists into binary trees in such a way that the list `[3; 4; 5]` would be turned into the following tree:



He writes the following function on mutable lists:

```
let rec mtree_of_mlist (p : 'a cell) : 'a node =
  if p == null then null else
    let t = mtree_of_mlist p.tl in
    { item = p.hd; left = t; right = t }
```

He also defines the pure function TreeOfList of type $\forall A.\ \mathsf{list}\, A \to \mathsf{tree}\, A$ as:

$$\begin{aligned} \mathsf{TreeOfList}\, \mathsf{nil}) &= \mathsf{Leaf} \\ \mathsf{TreeOfList}(x :: L) &= \mathsf{Node}\, x\, (\mathsf{TreeOfList}\, L)\, (\mathsf{TreeOfList}\, L) \end{aligned}$$

Hugo now wants to relate the two using the following specification:

$$\forall p\, L,\ \{p \rightsquigarrow \mathsf{Mlist}\, L\}(\texttt{mtree\_of\_mlist p})\{\lambda r.\, r \rightsquigarrow \mathsf{Mtree}(\mathsf{TreeOfList}\, L)\} \tag{9}$$

Question 3.1. *Prove that specification* (9) *does not hold. Give another of its disadvantages.*

It turns out that Hugo's functions must be used with no modification because he is the CEO.

Question 3.2. *Find a new specification for* `mtree_of_mlist` *and a new specification for a known function* `f` *such that the composition of* `f` *and* `mtree_of_mlist` *can be shown to satisfy specification* (9)*.*

Question 3.3. *Prove that* `mtree_of_mlist` *satisfies its new specification. Be sure to state your induction hypothesis with the right quantifiers.*

Question 3.4. *Give a proof sketch that* `f` *satisfies its new specification.*

Question 3.5. *Adapt the new specification for* `f` *so that it works for Question 3.2 and so that* `f`*'s usual specification can be derived from it.*

4 Separation Logic: List Partitions

We consider the following OCaml function on pure lists:

```
let rec partition_tf (f : 'a -> bool) : 'a list -> 'a list * 'a list =
  function
  | [] -> ([], [])
  | x :: l -> let y = f x in
              let (a, b) = partition_tf f l in
              if y then (x :: a, b) else (a, x :: b)
```

Question 4.1. *Give a specification for* `partition_tf f l` *in the case when* `f` *is a pure deterministic function that can be represented by a logical function $F : A \to$* `bool` *(and* `l` *is a pure list). Your specification should show explicitly how* `f` *and F are related.*

Suppose now that `f` cannot be represented a pure deterministic function F, but is still pure in the sense that it can be represented by a binary predicate P of type $A \to$ `bool` $\to \mathsf{Prop}$, so that a precondition for the specification of `partition_tf f` is $\forall x\ \{[\,]\}(\texttt{f x})\{\lambda b.[P\,x\,b]\}$.

Question 4.2. *Give some examples where* `f` *is pure in the above sense but such that the previous specification is not sufficient to establish a satisfactory Hoare triple for* `partition_tf f`*.*

Question 4.3. *Give a more general specification for* `partition_tf f` *in the case when* `f` *is a pure function that can be represented by a binary predicate P.*

Question 4.4. *Give a specification for* `partition_tf f l` *in the more general case where* `f` *is not necessarily deterministic or pure. You may use an auxiliary predicate of type $\mathsf{list}\,A \to \mathsf{list}\,A \to \mathsf{list}\,A \to \mathsf{Hpred}$.*

Question 4.5. *Give an implementation and a specification for the variant* `mutable_partition_tf f p` *that takes a mutable list as argument and returns a mutable record of two mutable lists (no proof required).*