1 Circular lists

This section focuses on the formalization of circular lists and doubly linked circular lists in Separation Logic. For simplicity, we consider only lists of integers in this exercise.

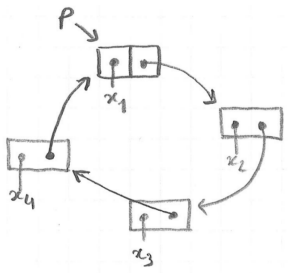
Recall the type of mutable list cells.

```
type cell = { mutable hd : int; mutable tl : cell }
```

Recall also the definition of the representation predicates MlistSeg and Mlist.

$$\begin{array}{lcl} p \rightsquigarrow \mathsf{MlistSeg}\, q\, L & \equiv & \mathsf{match}\, L\, \mathsf{with} \\ & & \mid \mathsf{nil} \Rightarrow [p = q] \\ & & \mid x :: L' \Rightarrow \exists p'.\ p \mapsto \{|\mathrm{hd}{=}x;\ \mathrm{tl}{=}p'|\} * p' \rightsquigarrow \mathsf{MlistSeg}\, q\, L' \\ p \rightsquigarrow \mathsf{Mlist}\, L & \equiv & p \rightsquigarrow \mathsf{MlistSeg}\, \mathsf{null}\, L \end{array}$$

Question 1.1. When the last cell of the list, instead of pointing to `null`, points back to the first cell of the list, we have a circular list.



Give a definition for the representation predicate $p \rightsquigarrow \mathsf{CircList}\, L$, which asserts that, when starting at the cell p and following the pointers until reaching p again, we find the items described by the list L. Your definition should build directly on top of "MlistSeg" (i.e., it should not mention any cell predicate of the form $q \mapsto \{|\mathrm{hd}{=}x;\ \mathrm{tl}{=}y|\}$), and it should use the equivalence "$p = \mathsf{null} \Leftrightarrow L = \mathsf{nil}$" to handle the particular case where the list is empty.

Answer.

$$p \rightsquigarrow \mathsf{CircList}\, L \quad \equiv \quad p \rightsquigarrow \mathsf{MlistSeg}\, p\, L * [p = \mathsf{null} \Leftrightarrow L = \mathsf{nil}]$$

Question 1.2. The function `read` takes as argument a pointer on a nonempty circular list, and returns the content of the item at this position.

```
let read p = p.hd
```

Give a specification for `read`, with a pre-condition of the form "$p \rightsquigarrow \mathsf{CircList}\, (v :: L)$".

Answer.

$$\{p \rightsquigarrow \mathsf{CircList}\, (v :: L)\}\ (\texttt{read p})\ \{\lambda x.\ [x = v] * p \rightsquigarrow \mathsf{CircList}\, (v :: L)\}$$

Question 1.3. The function `forward` takes as argument a pointer on a nonempty circular list, and returns the pointer on the next item.

```
let forward p = p.tl
```

Specify `forward`, with a post-condition of the form "$\lambda q.\ q \rightsquigarrow \mathsf{CircList}\, K$", for some list K.

Answer.

$$\{p \rightsquigarrow \mathsf{CircList}\, (v :: L)\}\ (\texttt{forward p})\ \{\lambda q.\ q \rightsquigarrow \mathsf{CircList}\, (L \& v)\}$$

Question 1.4. Prove that the code of forward satisfies the specification that you proposed in the previous question (15 lines max). Clearly indicate where focus or defocus rules are involved in the proof (no need to prove these rules).

Answer. By definition of CircList, we have:

$$p \rightsquigarrow \mathsf{CircList}\,(v :: L) \quad \rhd \quad p \rightsquigarrow \mathsf{MlistSeg}\,p\,(v :: L) * [p = \mathsf{null} \Leftrightarrow v :: L = \mathsf{nil}]$$

So, $p \neq \mathsf{null}$. Let q be the value of `p.tl`. By definition of MlistSeg,

$$p \rightsquigarrow \mathsf{MlistSeg}\,p\,(v :: L) \quad \rhd \quad p \mapsto \{|\mathrm{hd}{=}v;\ \mathrm{tl}{=}q|\} * q \rightsquigarrow \mathsf{MlistSeg}\,p\,L$$

Since $q \rightsquigarrow \mathsf{MlistSeg}\,p\,L$ is part of the state, q cannot be null. (Otherwise p would have to be null as well, according to the definition of MlistSeg.)
Furthermore, by definition of MlistSeg, we have:

$$p \mapsto \{|\mathrm{hd}{=}v;\ \mathrm{tl}{=}q|\} \quad \rhd \quad p \rightsquigarrow \mathsf{MlistSeg}\,q\,(v :: \mathsf{nil})$$

By the concatenation rule for list segments, we have:

$$q \rightsquigarrow \mathsf{MlistSeg}\,p\,L \ * \ p \rightsquigarrow \mathsf{MlistSeg}\,q\,(v :: \mathsf{nil}) \quad \rhd \quad q \rightsquigarrow \mathsf{MlistSeg}\,q\,(L +\!\!+ (v :: \mathsf{nil}))$$

Using the fact that $q \neq \mathsf{null}$, we can fold the definition of CircList and deduce that the final state satisfies the heap predicate $q \rightsquigarrow \mathsf{CircList}\,(L\&v)$.

Question 1.5. We next consider a function to insert an item at the head of a circular list.

```
1    let insert v p =
2      if p == null then begin
3        let q = { hd = v; tl = null } in
4        q.tl <- q;
5        q
6      end else begin
7        let q = { hd = p.hd; tl = p.tl } in
8        p.hd <- v;
9        p.tl <- q;
10       p
11     end
```

Specify the above function, then prove its correctness (15 lines max).

Answer. Specification:

$$\{p \rightsquigarrow \mathsf{CircList}\,L\}\ (\texttt{insert v p})\ \{\lambda q.\ q \rightsquigarrow \mathsf{CircList}\,(v :: L)\}$$

Case $p = \mathsf{null}$. By definition, $L = \mathsf{nil}$. At line 4, the state is $q \mapsto \{|\mathrm{hd}{=}v;\ \mathrm{tl}{=}q|\}$, which is equivalent to $q \rightsquigarrow \mathsf{MlistSeg}\,q\,(v :: \mathsf{nil})$ and thus to $q \rightsquigarrow \mathsf{CircList}\,(v :: \mathsf{nil})$.

Case $p \neq \mathsf{null}$. By definition of CircList and MlistSeg, there exists p' and x and L' such that $L = x :: L'$ and the state is described by:

$$p \mapsto \{|\mathrm{hd}{=}x;\ \mathrm{tl}{=}p'|\} * p' \rightsquigarrow \mathsf{MlistSeg}\,p\,L'$$

At line 10, the state is:

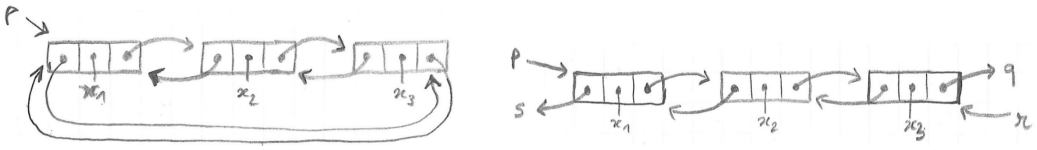
$$p \mapsto \{|\mathrm{hd}{=}v;\ \mathrm{tl}{=}q|\} * q \mapsto \{|\mathrm{hd}{=}x;\ \mathrm{tl}{=}p'|\} * p' \rightsquigarrow \mathsf{MlistSeg}\,p\,L'$$

By folding twice the definition of MlistSeg, we conclude that the final state is $p \rightsquigarrow \mathsf{MlistSeg}\,p\,(v :: x' :: L')$, and therefore $p \rightsquigarrow \mathsf{CircList}\,(v :: L)$.

Remark: in the specification of `insert`, the post-condition may be strengthen to assert that if L is nonempty, then q is equal to p.

Question 1.6. In order to be able to efficiently navigate the circular lists backwards, we extend it with reverse pointers. A cell is now represented as follows.

```
type cell = {
  mutable prev : cell;
  mutable item : int;
  mutable next : cell; }
```



The goal is to define a predicate "$p \rightsquigarrow \mathsf{DbList}\, L$" to characterize such a doubly linked list. The elements are described by the list L, when starting at the cell at address p and following the next pointers until reaching p again. Note that the predicate should use the equivalence "$p = \mathsf{null} \Leftrightarrow L = \mathsf{nil}$" to handle the particular case where the list is empty.

Hint: Start by defining a predicate of the form "$p \rightsquigarrow \mathsf{DbListSeg}\, q\, r\, s\, L$" that describes a segment of a doubly linked list storing the elements L. As shown in the rightmost picture above, the first cell, at address p, stores the address s in its prev field, and the last cell, at address r, stores the address q in its next field. When L is empty, enforce $p = q$ and $r = s$.

Answer.

$$\begin{array}{lcl} p \rightsquigarrow \mathsf{DbListSeg}\, q\, r\, s\, L & \equiv & \mathsf{match}\, L\, \mathsf{with} \\ & & \mid \mathsf{nil} \Rightarrow [p = q \wedge r = s] \\ & & \mid x :: L' \Rightarrow \exists p'.\ \{\!|\mathrm{item}{=}x;\ \mathrm{prev}{=}s;\ \mathrm{next}{=}p'|\!\} \\ & & \qquad * p' \rightsquigarrow \mathsf{DbListSeg}\, q\, r\, p\, L' \end{array}$$

Then:

$$p \rightsquigarrow \mathsf{DbList}\, L \ \equiv\ \exists q.\ p \rightsquigarrow \mathsf{DbListSeg}\, p\, q\, q\, L * [p = \mathsf{null} \Leftrightarrow L = \mathsf{nil}]$$

Remark: if $L = \mathsf{nil}$, we have $p = q = \mathsf{null}$; and if $L = v :: \mathsf{nil}$, we have $p = q \neq \mathsf{null}$.

2 Filtering

In this exercise, we are interested in specifying and verifying, in Separation Logic, functions that filters particular elements from a list. We assume the existence of a predicate called Even, which characterizes even integers. We also assume the existence of a logical function Filter, which applies to a predicate of type "$A \to \mathsf{Prop}$" and to a list of type "$\mathsf{list}\, A$". For example, "$\mathsf{Filter}\, \mathsf{Even}\, L$" filters the even elements from a list L, in the logic.

Question 2.1. We first consider a purely-functional implementation of filter on a pure list of integers, specialized to filtering even numbers.

```
let rec list_filter_even l =
  match l with
  | [] -> []
  | x::t -> let q = list_filter_even t in
            if x mod 2 = 0 then x::q else q
```

Give a specification for this function, using a Separation Logic triple.

Answer.

$$\{[\,]\}\ (\texttt{list\_filter\_even}\ l)\ \{\lambda l'.\ [l' = \mathsf{Filter}\ \mathsf{Even}\ l]\}$$

Question 2.2. We now adapt the above function to mutable lists as follows:

```
let rec mlist_filter_even p =
  if p == null then null else begin
    let q = mlist_filter_even p.tl in
    if p.hd mod 2 = 0
      then (p.tl <- q; p)
      else q
```

Give a Separation Logic specification for this function, making use of the predicate Mlist.

Answer.

$$\{p \rightsquigarrow \mathsf{Mlist}\ L\}\ (\texttt{mlist\_filter\_even}\ p)\ \{\lambda q.\ q \rightsquigarrow \mathsf{Mlist}\ (\mathsf{Filter}\ \mathsf{Even}\ L)\}$$

Question 2.3. Prove that `mlist_filter_even` satisfies the specification that you claimed in the previous question (in 20 lines max).

Answer. Proof by induction on the length of L.

- Case $p = \mathsf{null}$. From the pre-condition $p \rightsquigarrow \mathsf{Mlist}\ L$, we get $[L = \mathsf{nil}]$. So, "$\mathsf{Filter}\ \mathsf{Even}\ L = \mathsf{nil}$". This validates the post-condition for $q = \mathsf{null}$, since we have $[\,] \rhd (\mathsf{null} \rightsquigarrow \mathsf{Mlist}\ \mathsf{nil})$.
- Case $p \neq \mathsf{null}$. From the pre-condition, by the focus rule, there exists x and L' and t such that: $L = x :: L'$ and the state is $p \mapsto \{\mathrm{hd}{=}x;\ \mathrm{tl}{=}t\} * t \rightsquigarrow \mathsf{Mlist}\ L'$. To reason about the recursive call, we apply the frame rule to keep only $t \rightsquigarrow \mathsf{Mlist}\ L'$, and invoke the induction hypothesis. We learn that q is such that "$q \rightsquigarrow \mathsf{Mlist}\ (\mathsf{Filter}\ \mathsf{Even}\ L')$".
  - Case $\mathsf{Even}\ x$. We have $\mathsf{Filter}\ \mathsf{Even}\ L = x :: \mathsf{Filter}\ \mathsf{Even}\ L'$. The state is modified to: $p \mapsto \{\mathrm{hd}{=}x;\ \mathrm{tl}{=}q\} * q \rightsquigarrow \mathsf{Mlist}\ (\mathsf{Filter}\ \mathsf{Even}\ L')$. Applying the defocus rule gives $p \rightsquigarrow \mathsf{Mlist}\ (\mathsf{Filter}\ \mathsf{Even}\ L)$, which validates the post-condition since p is returned.
  - Case $\neg\ \mathsf{Even}\ x$. We have $\mathsf{Filter}\ \mathsf{Even}\ L = \mathsf{Filter}\ \mathsf{Even}\ L'$. Applying the garbage collection rule to $p \mapsto \{\mathrm{hd}{=}x;\ \mathrm{tl}{=}t\}$, we are left with "$q \rightsquigarrow \mathsf{Mlist}\ (\mathsf{Filter}\ \mathsf{Even}\ L')$", which validates the post-condition since q is returned.

Question 2.4. We generalize the function so that it may take as argument a filter function `f`, of type `int -> bool`.

```
let rec mlist_filter f p =
  if p == null then null else begin
    let q = mlist_filter p.tl in
    if f p.hd
      then (p.tl <- q; p)
      else q
```

Give the specification of this function, assuming that `f` does not perform any visible side effect. For this question, quantify explicitly all the variables involved.

Answer.

$$\begin{aligned}\forall f p L.\quad & (\forall x.\ \{[\,]\}\ (f\ x)\ \{\lambda b.\ [b = \mathsf{isTrue}\ (P\ x)]\}) \Rightarrow \\ & \{p \rightsquigarrow \mathsf{Mlist}\ L\}\ (\texttt{mlist\_filter}\ f\ p)\ \{\lambda q.\ q \rightsquigarrow \mathsf{Mlist}\ (\mathsf{Filter}\ P\ L)\}\end{aligned}$$

Question 2.5. We are now interested in the generalization to mutable lists containing items of arbitrary types. For this purpose, we need in particular to quantify over the representation predicate R associated with the items stored in the list, and use Mlistof instead of Mlist.

Moreover, we assume that the function `f` may read (but not write) values in the heap; in particular, it may not alter any of the heap invariants. So, the function `f`, in addition to being able to access the representation of the items that it receives as argument, should also be able to read in other parts of the heap. These pieces of heap should be described by an invariant, call it "I", that should appear in both the pre- and the post-condition of `mlist_filter`.

Give the corresponding specification of `mlist_filter`.

Answer.

$$(\forall x X.\ \{x \rightsquigarrow R\,X \ast I\}\ (f\,x)\ \{\lambda b.\ [b = \mathsf{isTrue}\,(P\,X)] \ast x \rightsquigarrow R\,X \ast I\}) \Rightarrow$$
$$\{p \rightsquigarrow \mathsf{Mlistof}\,R\,L \ast I\}\ (\texttt{mlist\_filter}\ f\,p)\ \{\lambda q.\ q \rightsquigarrow \mathsf{Mlistof}\,R\,(\mathsf{Filter}\,P\,L) \ast I\}$$

Question 2.6. Consider the application of `mlist_filter` to a mutable list "`p`" made of possibly-aliased reference cells, each of these reference storing an integer. The goal is to filter the reference cells that store an even number. Using the "Group" representation predicate to describe the possibly-aliased reference cells, state a triple that specifies the behavior of the following expression, which you may refer to as "*expr*".

```
mlist_filter (fun r -> !r mod 2 = 0) p
```

Answer.

$$\{p \rightsquigarrow \mathsf{Mlist}\,L \ast \mathsf{GroupOf\,Ref}\,G \ast [\forall x \in L.\ x \in \mathrm{dom}(G)]\}$$
$$expr$$
$$\{\lambda q.\ q \rightsquigarrow \mathsf{Mlist}\,(\mathsf{Filter}\,(\lambda x.\,\mathsf{Even}(G(x)))\,L) \ast \mathsf{GroupOf\,Ref}\,G\}$$

where G has type "fmap loc int".

Question 2.7. Prove that the function "`fun r -> !r mod 2 = 0`" satisfies the hypothesis made on `f` in the specification of `mlist_filter` in question 2.5. Clearly state the focus rule that you exploit (max 12 lines for the proof).

Answer. Instantiate the specification of `mlist_filter` with $R = \mathsf{Id}$ and $I = \mathsf{GroupOf\,Ref}\,G$ and $P = \lambda x.\,\mathsf{Even}(G(x))$. Since $x \rightsquigarrow R\,X$ is equivalent to $[x = X]$, the hypothesis on f simplifies to:

$$\forall r.\ \{\mathsf{GroupOf\,Ref}\,G\}\ (f\,r)\ \{\lambda b.\ [b = \mathsf{isTrue}\,(P\,r)] \ast \mathsf{GroupOf\,Ref}\,G\}$$

We wish to relate the above specification to:

$$\forall r n.\ \{r \rightsquigarrow \mathsf{Ref}\,n\}\ (\texttt{!r mod 2 = 0})\ \{\lambda b.\ [b = \mathsf{isTrue}\,(\mathsf{Even}\,n)] \ast r \rightsquigarrow \mathsf{Ref}\,n\}$$

To that end, it suffices to exploit the following focus rule in both directions:

$$\mathsf{GroupOf\,Ref}\,(G' \uplus (x \mapsto X)) \quad = \quad (x \rightsquigarrow \mathsf{Ref}\,X) \ast (\mathsf{GroupOf\,Ref}\,G')$$

and frame out $(\mathsf{GroupOf\,Ref}\,G')$ during the evaluation of "`!r mod 2 = 0`".