Exercise: describe the frame process in the induction for `mlength`. Find

three useful specifications for `swap`:

1. a specification for non-aliased (distinct) arguments,
2. a specification for aliased (equal) arguments,
3. a most-general specification, stated using iterated conjunction (or another construct from Course 2).

Exercise: what is the specification of `f` in the following program?

```
let r = ref 3
let f () =
  incr r
```

Then, show that the code below returns 5.

```
f();
```

```
f();
!r
```

Exercise: specify a counter function, only in terms of $f \rightsquigarrow \mathsf{Count}\,n$.
Exercise: give two specifications for the function `refapply`. In the first,

assume `f` to be pure, and introduce a predicate $P\,x\,y$. In the second, assume that `f` also modifies the state from H to H'.

$$\begin{array}{rl} & \{\qquad\qquad\qquad\}\,(f\,x)\,\{\qquad\qquad\qquad\qquad\} \\ \Rightarrow & \{\qquad\qquad\qquad\qquad\}\,(\texttt{refapply}\,r\,f)\,\{\lambda\_.\qquad\qquad\qquad\qquad\qquad\qquad\} \end{array}$$

Exercise: specify `repeat`, using an invariant I, of type $\mathsf{int} \to \mathsf{Hprop}$.
Exercise: specify `iter`, using an invariant I, of type $\mathsf{list}\,\alpha \to \mathsf{Hprop}$.

```
let length l =
  let r = ref 0 in
  iter (fun x -> incr r) l;
  !r
```

Exercise: give the instantiatiation of the invariant I for `iter`; then, write the specialization of the specification of `iter` to I and to `(fun x -> incr r)`; finally, check that the premise is provable.

```
let sum l =
  let r = ref 0 in
  iter (fun x -> r := !r + x) l;
  !r
```

Exercise: give the invariant I involved in the above call to `iter`. **Exercise:**

specify `iter` using an invariant that depends on the list of items remaining to process, instead of on the list of items already processed. Then, prove the new specification derivable from the old one.

$$\begin{array}{ll} & (\qquad\qquad\{\qquad\qquad\qquad\}\ (f\ x)\ \{\lambda\_.\qquad\qquad\}) \\ \Rightarrow & \{I'\ l\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I'\ \mathsf{nil}\} \end{array}$$

```
let r = ref 0
let count_and_sum l =
  fold_left (fun a x -> incr r; a+x) 0 l
```

Exercise: give the instantiation of the invariant J in the code above. **Exercise:**

give a specification for `fold_right`.

$$\begin{array}{lll} \forall f l a J. & & (\qquad\qquad\qquad\qquad\qquad\qquad) \\ & \Rightarrow & \{J\ a\ \mathsf{nil}\}\ (\texttt{fold\_right}\ f\ l\ a)\ \{\lambda b.\ J\ b\ l\} \end{array}$$

Exercise: define the characteristic formula for sequences.

$$[\![t_1\,;\,t_2]\!] \;\equiv\; \lambda H.\,\lambda Q.$$

Exercise: define the characteristic formula for conditionals.

$$[\![\mathsf{if}\,b\,\mathsf{then}\,t_1\,\mathsf{else}\,t_2]\!] \;\equiv$$