Separation Logic
3/4

Arthur Charguéraud

Febuary 15th, 2016

Chapter 12

Loops in Separation Logic

Verification of a for-loop

```
let facto n =
  let r = ref 1 in
  for i = 2 to n do
    let v = !r in
    r := v * i;
  done;
  !r
```

Before the loop:

$$r \mapsto 1$$

At each iteration:

$$\text{from} \quad r \mapsto (i-1)! \quad \text{to} \quad r \mapsto i!$$

After the loop:

$$r \mapsto n!$$

Loop invariant $(I : \text{int} \to \text{Hprop})$ that applies for any $i \in [2, n+1]$:

$$I\, i \quad \equiv \quad r \mapsto (i-1)!$$

Reasoning rule for for-loops

Reasoning rule for the case $a \leqslant b$:

$$\frac{\begin{array}{c} H \rhd I\, a \\ \forall i \in [a, b].\quad \{I\, i\}\ t\ \{\lambda\_.\, I\, (i+1)\} \\ I\, (b+1) \rhd Q\, () \end{array}}{\{H\}\ (\mathsf{for}\, i = a\, \mathsf{to}\, b\, \mathsf{do}\, t)\ \{Q\}}$$

General rule, also covering the case $a > b$:

$$\frac{\begin{array}{c} H \rhd I\, a \\ \forall i \in [a, b].\quad \{I\, i\}\ t\ \{\lambda\_.\, I\, (i+1)\} \\ I\, (\mathsf{max}\, a\, (b+1)) \rhd Q\, () \end{array}}{\{H\}\ (\mathsf{for}\, i = a\, \mathsf{to}\, b\, \mathsf{do}\, t)\ \{Q\}}$$

Reasoning rule for while loops: partial correctness

The loop invariant I describes the state between every iterations.
The post-condition J describes the state after the evaluation of t_1.

$$\frac{H \rhd I \qquad \{I\}\ t_1\ \{J\} \qquad \{J\,\mathsf{true}\}\ t_2\ \{\lambda\_.\, I\} \qquad J\,\mathsf{false} \rhd Q\,()}{\{H\}\ (\mathsf{while}\, t_1\, \mathsf{do}\, t_2)\ \{Q\}}$$

where $(I : \mathsf{Hprop})$ and $(J : \mathsf{bool} \to \mathsf{Hprop})$.

For total correctness: parameterize the invariant with a measure.

Reasoning rule for while loops

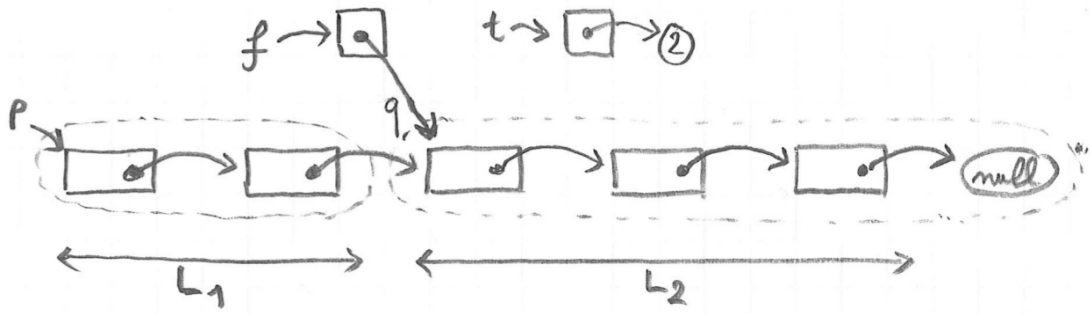
We focus on a different approach that:

- inherently supports total correctness;
- allows to apply frame during iterations.

Prove a triple $\{H\}\ (\mathsf{while}\, t_1\, \mathsf{do}\, t_2)\ \{Q\}$ by induction, using:

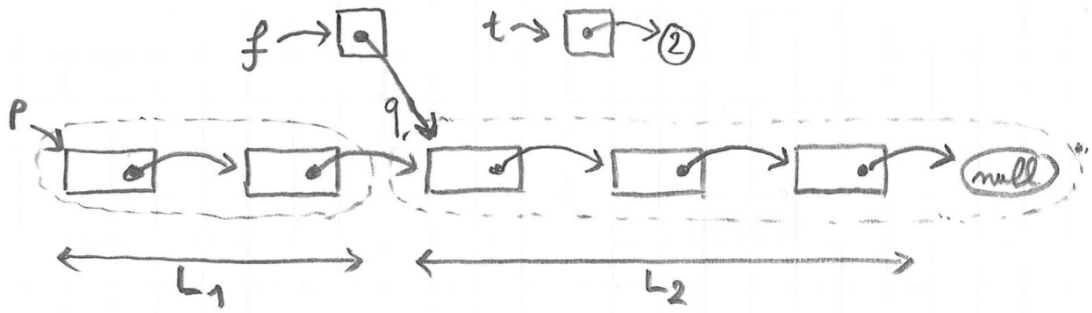
$$\frac{\{H\}\ (\mathsf{if}\, t_1\, \mathsf{then}\, (t_2\, ;\, (\mathsf{while}\, t_1\, \mathsf{do}\, t_2))\, \mathsf{else}\, ())\ \{Q\}}{\{H\}\ (\mathsf{while}\, t_1\, \mathsf{do}\, t_2)\ \{Q\}}$$

Length with a while loop




```
let rec mlength (p:'a cell) =
  let t = ref 0 in
  let f = ref p in
  while !f != null do
    incr t;
    f := (!f).tl;
  done
  !t
```

Length with a while loop: induction



We prove by induction on L_2 that for any n and q:

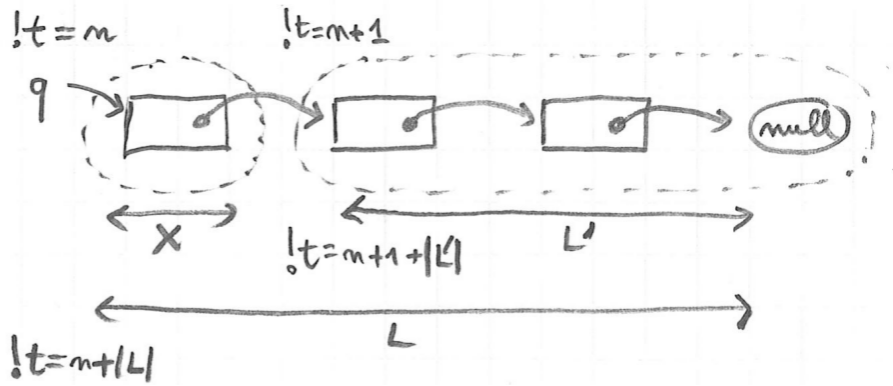
$$\{q \rightsquigarrow \mathsf{Mlist}\, L_2 \star f \mapsto q \star t \mapsto n\}$$
$$(\texttt{while !f != null do incr t; f := (!f).tl; done})$$
$$\{q \rightsquigarrow \mathsf{Mlist}\, L_2 \star f \mapsto \mathsf{null} \star t \mapsto (n + \mathsf{length}\, L_2)\}$$

The loop unfolds to:

```
if !f != null
  then (incr t; f := (!f).tl; while .. do .. done)
  else ()
```

Exercise: describe the frame process in the induction for `length`.

Length with a while loop: frame process



$q \rightsquigarrow \mathsf{Mlist}\, L_2$		$\star\ f \mapsto q$	$\star\ t \mapsto n$		begin
$q \mapsto \{\!\mid x; q' \mid\!\} \star q' \rightsquigarrow \mathsf{Mlist}\, L_2'$		$\star\ f \mapsto q$	$\star\ t \mapsto n$		unfold
$q \mapsto \{\!\mid x; q' \mid\!\} \star q' \rightsquigarrow \mathsf{Mlist}\, L_2'$		$\star\ f \mapsto q$	$\star\ t \mapsto n+1$		increment
$q \mapsto \{\!\mid x; q' \mid\!\} \star q' \rightsquigarrow \mathsf{Mlist}\, L_2'$		$\star\ f \mapsto q'$	$\star\ t \mapsto n+1$		shift head
	$\star\ q' \rightsquigarrow \mathsf{Mlist}\, L_2'$	$\star\ f \mapsto q'$	$\star\ t \mapsto n+1$		begin frame
	$\star\ q' \rightsquigarrow \mathsf{Mlist}\, L_2'$	$\star\ f \mapsto \mathsf{null}$	$\star\ t \mapsto n+1+\lvert L_2' \rvert$		induction
$q \mapsto \{\!\mid x; q' \mid\!\} \star q' \rightsquigarrow \mathsf{Mlist}\, L_2'$		$\star\ f \mapsto \mathsf{null}$	$\star\ t \mapsto n+1+\lvert L_2' \rvert$		end frame
$q \rightsquigarrow \mathsf{Mlist}\, L_2$		$\star\ f \mapsto \mathsf{null}$	$\star\ t \mapsto n+\lvert L_2 \rvert$		fold

Chapter 13

Aliasing and local state

Functions with aliasing: swap

```
let swap r s =
  let a = !r in
  let b = !s in
  r := b;
  s := a
```

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.

Functions with aliasing: 3 specifications for swap

Specification 1:

$$\forall r s n m.\ \{(r \mapsto n) \star (s \mapsto m)\}\ (\texttt{swap r s})\ \{\lambda\_.\ (r \mapsto m) \star (s \mapsto n)\}$$

Specification 2:

$$\forall r s n.\ \{[r = s] \star (r \mapsto n)\}\ (\texttt{swap r s})\ \{\lambda\_.\ r \mapsto n\}$$

or simply:

$$\forall r n.\ \{r \mapsto n\}\ (\texttt{swap r r})\ \{\lambda\_.\ r \mapsto n\}$$

Specification 3:

$$\begin{aligned}\forall r s M.\ r, s \in \mathsf{dom}\, M \ \Rightarrow\ & \{\circledast_{(p,n)\in M}\ p \mapsto n\} \\ & (\texttt{swap r s}) \\ & \{\lambda\_.\ \circledast_{(p,n)\in (M[r:=M[s]][s:=M[r]])}\ p \mapsto n\}\end{aligned}$$

Function with local state

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
```

Specification:

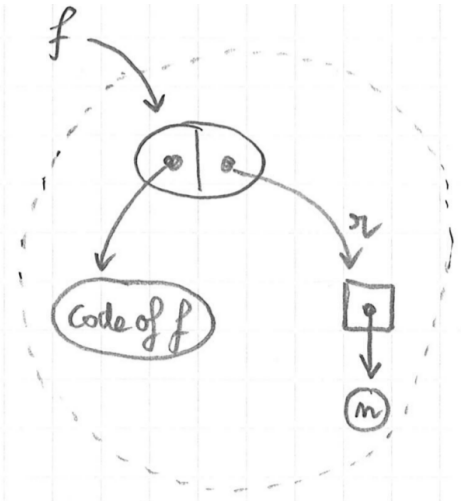
$$\forall n. \quad \{r \mapsto n\}\ (\texttt{f}\ \ ())\ \{\lambda\_.\ r \mapsto n+1\}$$

Successive states:

$$r \mapsto 3 \qquad r \mapsto 4 \qquad r \mapsto 5$$

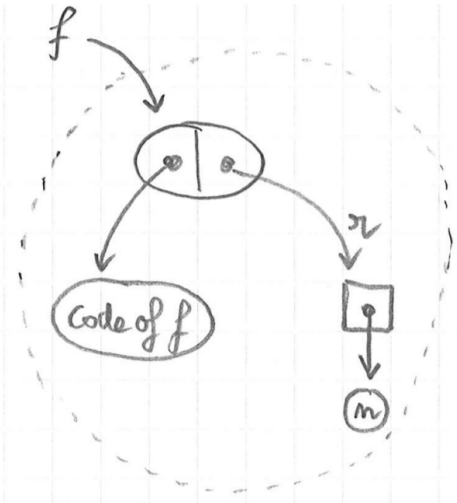
Counter function: code

```
let mkcounter () =
  let r = ref 0 in
  (fun () -> incr r; get r)
```




```
let c = mkcounter() in
let x = c() in
let y = c() in
assert (x = 1 && y = 2)
```

Counter function: specification



$$f \rightsquigarrow \mathsf{Count}\, n \;\equiv\; \exists r.\, (r \mapsto n) \\ \star\, [\forall i.\, \{r \mapsto i\}\ (f\ ())\ \{\lambda x.\, [x = i + 1] \star (r \mapsto i + 1)\}]$$

Exercise: specify a counter function, only in terms of $f \rightsquigarrow \mathsf{Count}\, n$.

$$\{[\,]\}\ (\texttt{mkcounter}\,())\ \{\lambda f.\ f \rightsquigarrow \mathsf{Count}\, 0\}$$

$$\forall f i.\quad \{f \rightsquigarrow \mathsf{Count}\, i\}\ (f\ ())\ \{\lambda x.\, [x = i + 1] \star f \rightsquigarrow \mathsf{Count}\, (i + 1)\}$$

Chapter 14

Basic higher-order functions

Apply

```
let apply f x =
  f x
```

Specification:

$$\begin{array}{lll} \forall f\, x\, H\, Q. & & \{H\}\ (f\ x)\ \{Q\} \\ & \Rightarrow & \{H\}\ (\texttt{apply}\ f\ x)\ \{Q\} \end{array}$$

This is equivalent to the form below, which involves nested triples:

$$\forall f\, x\, H\, Q. \quad \{H \star [\ \{H\}\ (f\ x)\ \{Q\}\ ]\}\ (\texttt{apply}\ f\ x)\ \{Q\}$$

Apply on a reference

```
let refapply r f =
  r := f !r
```

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'.

$$\forall r\,f\,x\,P. \quad \{[\,]\}\ (f\,x)\ \{\lambda y.\,[P\,x\,y]\}$$
$$\Rightarrow \quad \{r \mapsto x\}\ (\texttt{refapply}\,r\,f)\ \{\lambda\_.\ \exists y.\,[P\,x\,y] \star r \mapsto y\}$$

$$\forall r\,f\,x\,H\,H'\,P. \quad \{H\}\ (f\,x)\ \{\lambda y.\,[P\,x\,y] \star H'\}$$
$$\Rightarrow \quad \{(r \mapsto x) \star H\}\ (\texttt{refapply}\,r\,f)\ \{\lambda\_.\ \exists y.\,[P\,x\,y] \star (r \mapsto y) \star H'\}$$

Function twice

```
let twice f =
  f(); f()
```

Specification:

$$\begin{array}{lll} \forall f\, H'\, Q. & & \{H\}\ (f\ ())\ \{\lambda\_.\ H'\} \\ & \wedge & \{H'\}\ (f\ ())\ \{Q\} \\ & \Rightarrow & \{H\}\ (\texttt{twice}\ f)\ \{Q\} \end{array}$$

Function repeat

```
let repeat n f =
  for i = 0 to n-1 do
    f()
  done
```

Exercise: specify `repeat`, using an invariant I, of type int → Hprop.

$$\begin{array}{rl} \forall n f I. & (\forall i \in [0, n). \quad \{I\, i\}\ (f\ ())\ \{\lambda\_.\ I\,(i+1)\}) \\ \Rightarrow & \{I\, 0\}\ (\texttt{repeat}\, n\, f)\ \{\lambda\_.\ I\, n\} \end{array}$$

The premise consists of a family of hypotheses describing the behavior of applications of f to particular arguments.

Chapter 15

Higher order iteration

Iteration over a pure list

```
let rec iter f l =
  match l with
  | [] -> ()
  | x::t -> f x; iter f t
```

Exercise: specify `iter`, using an invariant I, of type list $\alpha \to$ Hprop.

$$\begin{array}{rll} \forall f l I. & & \big(\forall x k.\ \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k \& x)\}\big) \\ & \Rightarrow & \{I\,\mathsf{nil}\}\ (\texttt{iter}\,f\,l)\ \{\lambda\_.\ I\,l\} \end{array}$$

where $k \& x \equiv k +\!\!+ (x :: \mathsf{nil})$.

Length using iter

$$\begin{array}{rl} & \left(\forall x k.\ \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k \& x)\}\right) \\ \Rightarrow & \{I\,\mathsf{nil}\}\ (\mathsf{iter}\,f\,l)\ \{\lambda\_.\ I\,l\} \end{array}$$

```
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.

Invariant: $I \equiv \lambda k.\ r \mapsto |k|$.

$$\begin{array}{rl} & \left(\forall x k.\ \{r \mapsto |k|\}\ (\texttt{incr r})\ \{\lambda\_.\ r \mapsto |k| + 1\}\right) \\ \Rightarrow & \{r \mapsto 0\}\ (\mathsf{iter}\,f\,l)\ \{\lambda\_.\ r \mapsto |l|\} \end{array}$$

Sum using iter

$$\begin{array}{rl} & (\forall x k.\ \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k \& x)\}) \\ \Rightarrow & \{I\,\mathsf{nil}\}\ (\mathsf{iter}\,f\,l)\ \{\lambda\_.\ I\,l\} \end{array}$$

```
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`.

$$I \ \equiv\ \lambda k.\ r \mapsto \mathsf{Sum}\,k$$

where:

$$\mathsf{Sum}\,k \ \equiv\ \mathsf{Fold}\,(+)\,0\,k$$

Constraints over the items

$$\begin{array}{rl} & \left(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\right) \\ \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

Given a list $x_1 :: x_2 :: ... :: x_n ::$ nil, let us compute: $\sqrt{x_1} + ... + \sqrt{x_n}$.

```
iter (fun x -> r := !r +. sqrt x) [2.0; 3.0]
```

The above specification of `iter` is too weak. More general specification:

$$\begin{array}{lll} \forall f\, I\, l. & & \left(\forall x k.\ x \in l \Rightarrow \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\right) \\ & \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

Constraints over the items, in order

$$\begin{array}{lll} \forall f\, I\, l. & & \big(\forall x k.\ x \in l \Rightarrow \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\big) \\ & \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

Given a list $x_1 :: x_2 :: \ldots :: x_n ::$ nil, let us compute: $\sqrt{\sqrt{\sqrt{x_1} + x_2} + x_3}$.

```
iter (fun x -> r := sqrt (!r +. x)) [2.; -1.; 3.]
```

The above specification of `iter` is too weak. Most-general specification:

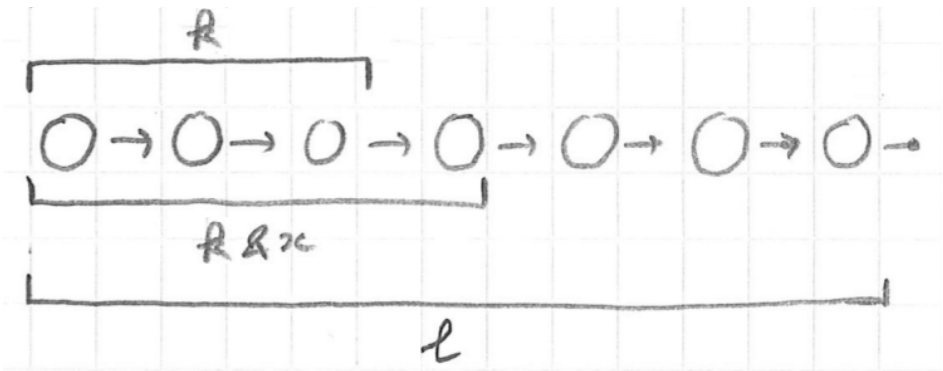
$$\begin{array}{lll} \forall f\, I\, l. & & \big(\forall x k s.\ l = k +\!\!+ x :: s \Rightarrow \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\big) \\ & \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathtt{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

Verification of iter

$$\begin{array}{rl} & \left(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\right) \\ \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

```
let rec iter f l =
  match l with
  | [] -> ()
  | x::t -> f x; iter f t
```

How to prove that the code satisfies its specification?



Verification of iter: generalized principle

Assume:

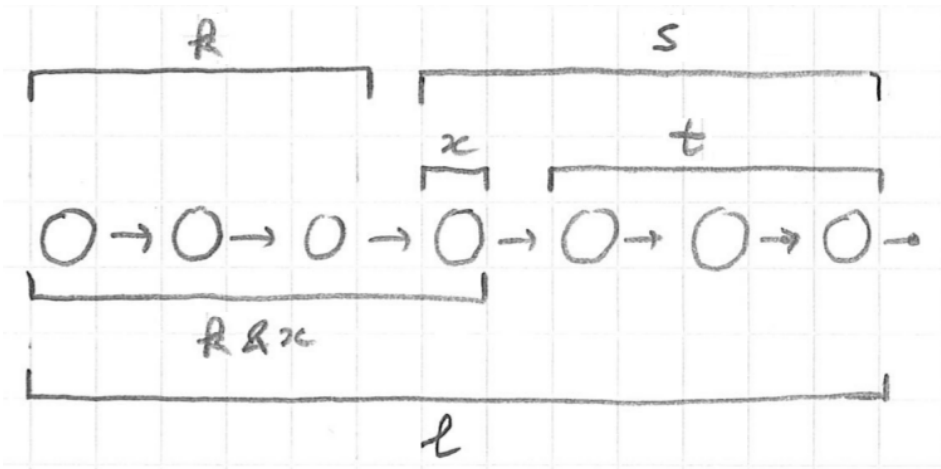
$$\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}$$

Prove:

$$\{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\}$$

Proof by induction over a generalized statement:

$$\forall k s.\quad \{I\ k\}\ (\mathsf{iter}\ f\ s)\ \{\lambda\_.\ I\ (k +\!\!+ s)\}$$



Verification of iter: induction

```
let rec iter f s =
  match s with
  | [] -> ()
  | x::t -> f x; iter f t
```

Assume: $\forall xk.\quad \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k \& x)\}$

Prove: $\forall ks.\quad \{I\,k\}\ (\mathsf{iter}\,f\,s)\ \{\lambda\_.\ I\,(k +\!\!+ s)\}$

By induction on s:

- Case $s = \mathsf{nil}$. Goal is: $\{I\,k\}\ (\mathsf{iter}\,f\,\mathsf{nil})\ \{\lambda\_.\, I\,(k +\!\!+ \mathsf{nil})\}$.
  This triple simplifies to: $\{I\,k\}\ ()\ \{\lambda\_.\, I\,k\}$, which is correct.
- Case $s = x :: t$. Goal is: $\{I\,k\}\ (\mathsf{iter}\,f\,(x :: t))\ \{\lambda\_.\, I\,(k +\!\!+ (x :: t))\}$.

$$\dfrac{\dfrac{}{\{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k \& x)\}}\ \text{HYPOTHESIS-ON-F} \qquad \dfrac{}{\{I\,(k \& x)\}\ (\mathsf{iter}\,f\,t)\ \{\lambda\_.\ I\,((k \& x) +\!\!+ t)\}}\ \text{INDUCTION-HYPOTHESIS}}{\{I\,k\}\ (f\,x;\ \mathsf{iter}\,f\,t)\ \{I\,((k \& x) +\!\!+ t)\}}\ \text{SEQ}$$

Invariant on remaining items

$$\begin{array}{rl} & \big(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\big) \\ \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

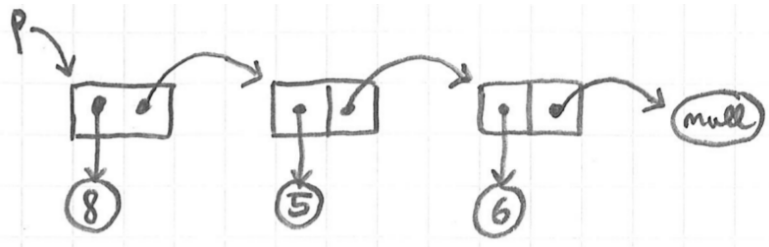
$$\begin{array}{rl} & \big(\forall \ldots.\ \{\ldots\}\ (f\ x)\ \{\lambda\_.\ \ldots\}\big) \\ \Rightarrow & \{I'\ l\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I'\ \mathsf{nil}\} \end{array}$$

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.

$$\begin{array}{rl} & \big(\forall x s.\ \{I'\ (x :: s)\}\ (f\ x)\ \{\lambda\_.\ I'\ s\}\big) \\ \Rightarrow & \{I'\ l\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I'\ \mathsf{nil}\} \end{array}$$

Derivable using: $I \equiv \lambda k.\ \exists s.\ [l = k +\!\!+ s] \star I'\ s$.

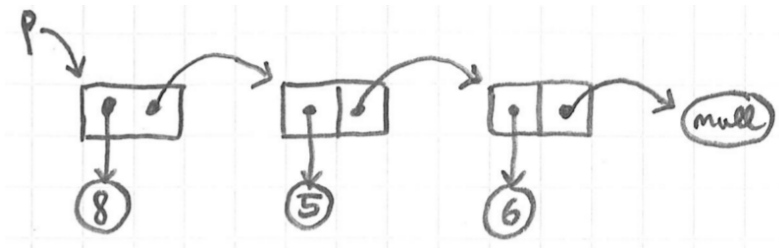
Iterating over a mutable list




```
let rec miter f p =
  if p == null
    then ()
    else (f p.hd; miter f p.tl)
```

Iterating over a mutable list

$$\forall f l I.\quad \big(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\big)$$
$$\Rightarrow\quad \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\}$$



Specification:

$$\forall f p I l.\quad \big(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\big)$$
$$\Rightarrow\quad \{p \rightsquigarrow \mathsf{Mlist}\ l\ \star\ I\ \mathsf{nil}\}\ (\mathtt{miter}\ f\ p)\ \{\lambda\_.\ p \rightsquigarrow \mathsf{Mlist}\ l\ \star\ I\ l\}$$

Remark: calls to f cannot modify the structure of the list while iterating.

Summary

Simplified:

$$\begin{array}{rl} & \left(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\right) \\ \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

Order-irrelevant:

$$\begin{array}{rl} & \left(\forall x k.\ x \in l \Rightarrow \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\right) \\ \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

Most-general:

$$\begin{array}{rl} & \left(\forall x k s.\ l = k +\!\!+ x :: s \Rightarrow \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\right) \\ \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda\_.\ I\ l\} \end{array}$$

Extension to mutable lists:

$$\begin{array}{rl} & \left(\forall x k s.\ l = k +\!\!+ x :: s \Rightarrow \{I\ k\}\ (f\ x)\ \{\lambda\_.\ I\ (k \& x)\}\right) \\ \Rightarrow & \{p \rightsquigarrow \mathsf{Mlist}\ l \star I\ \mathsf{nil}\}\ (\mathtt{miter}\ f\ p)\ \{\lambda\_.\ p \rightsquigarrow \mathsf{Mlist}\ l \star I\ l\} \end{array}$$

Chapter 16

Other classic higher-order functions

Fold-left

```
let rec fold_left f a l =
  match l with
  | [] -> a
  | x::k -> fold_left f (f a x) k
```

Example:

$$\texttt{fold\_left}\; f\; a\; [6 :: 4 :: 7] \;=\; f\,(f\,(f\,a\,6)\,4)\,7$$

Specification:

$$\begin{aligned} \forall f\, a\, l\, J. \quad & \big(\forall x\, i\, k.\ \{J\, i\, k\}\ (f\, i\, x)\ \{\lambda j.\ J\, j\, (k \& x)\}\big) \\ \Rightarrow\ & \{J\, a\, \mathsf{nil}\}\ (\texttt{fold\_left}\, f\, a\, l)\ \{\lambda b.\ J\, b\, l\} \end{aligned}$$

Application of fold-left

$$\begin{array}{rl} \forall f\, a\, l\, J. & \left(\forall x\, i\, k.\ \{J\, i\, k\}\ (f\, i\, x)\ \{\lambda j.\ J\, j\, (k \& x)\}\right) \\ \Rightarrow & \{J\, a\, \mathsf{nil}\}\ (\texttt{fold\_left}\ f\, a\, l)\ \{\lambda b.\ J\, b\, l\} \end{array}$$

```
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.

$$J\, i\, k \ \equiv\ (r \mapsto |k|) \star [i = \mathsf{Sum}\, k]$$

where $\mathsf{Sum}\, k \equiv \mathsf{Fold}\, (+)\, 0\, k$.

Fold-right

```
let rec fold_right f l a =
  match l with
  | [] -> a
  | x::k -> f x (fold_right f k a)
```

Example:

$$\mathsf{fold}\ f\ [6 :: 4 :: 7]\ a \ \equiv \ f\ 6\ (f\ 4\ (f\ 7\ a))$$

Exercise: give a specification for `fold_right`.

$$\begin{array}{lll} \forall f l a J. & & \big(\forall x i k.\ \{J\ i\ k\}\ (f\ x\ i)\ \{\lambda j.\ J\ j\ (x :: k)\}\big) \\ & \Rightarrow & \{J\ a\ \mathsf{nil}\}\ (\texttt{fold\_right}\ f\ l\ a)\ \{\lambda b.\ J\ b\ l\} \end{array}$$

Map: simple specification for pure functions

```
let rec map f l =
  match l with
  | [] -> []
  | x::k -> (f x)::(map f k)
```

Simple specification, for the case where `f` is pure:

$$\begin{array}{lll} \forall f\, l\, P. & & (\forall x.\ \{[\,]\}\ (f\ x)\ \{\lambda x'.\ [P\ x\ x']\}) \\ & \Rightarrow & \{[\,]\}\ (\texttt{map}\ f\ l)\ \{\lambda l'.\ [\mathsf{Forall2}\ P\ l\ l']\} \end{array}$$

where:

$$\frac{}{\mathsf{Forall2}\ P\ \mathsf{nil}\ \mathsf{nil}} \qquad \frac{P\ x\ x' \qquad \mathsf{Forall2}\ P\ l\ l'}{\mathsf{Forall2}\ P\ (x :: l)\ (x' :: l')}$$

Map: general specification

Specification of `map`:

$$\begin{array}{rl} \forall f\, l\, P. & (\forall x.\ \{[\,]\}\ (f\ x)\ \{\lambda x'.\ [P\ x\ x']\}) \\ \Rightarrow & \{[\,]\}\ (\mathsf{map}\ f\ l)\ \{\lambda l'.\ [\mathsf{Forall2}\ P\ l\ l']\} \end{array}$$

Specification of `iter`:

$$\begin{array}{rl} \forall f\, l\, I. & \big(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda \_.\ I\ (k \& x)\}\big) \\ \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{iter}\ f\ l)\ \{\lambda \_.\ I\ l\} \end{array}$$

Combining the two:

$$\begin{array}{rl} \forall f\, l\, P\, I. & \big(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda x'.\ [P\ x\ x'] \star I\ (k \& x)\}\big) \\ \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{map}\ f\ l)\ \{\lambda l'.\ [\mathsf{Forall2}\ P\ l\ l'] \star I\ l\} \end{array}$$

Map: general specification, alternative

$$\begin{array}{lll} \forall f\, l\, P\, I. & & \left(\forall x k.\ \{I\ k\}\ (f\ x)\ \{\lambda x'.\, [P\ x\ x'] \star J\ (k \& x)\}\right) \\ & \Rightarrow & \{I\ \mathsf{nil}\}\ (\mathsf{map}\ f\ l)\ \{\lambda l'.\ [\mathsf{Forall2}\ P\ l\ l'] \star I\ l\} \end{array}$$

Alternative specification:

$$\begin{array}{lll} \forall f\, l\, J'. & & \left(\forall x k k'.\ \{J'\ k\ k'\}\ (f\ x)\ \{\lambda x'.\, J'\ (k \& x)\ (k' \& x')\}\right) \\ & \Rightarrow & \{J'\ \mathsf{nil}\ \mathsf{nil}\}\ (\mathsf{map}\ f\ l)\ \{\lambda l'.\ J'\ l\ l'\} \end{array}$$

Above specification derivable from the previous one:

$$J'\ k\ k' \ \equiv\ [\mathsf{Forall2}\ P\ k\ k'] \star I\ k$$

Sorting with comparison function

Example:

```
List.sort (fun x y -> x - y) [2;4;5;3;2;9]
```

Specification:

$$\begin{aligned}
&\forall f\, l.\ \forall (\preceq).\\
&\qquad \mathsf{total\text{-}order}\ (\preceq)\\
&\quad \wedge\ \ (\forall x y.\ \{[\,]\}\ (f\,x\,y)\ \{\lambda n.\,[n \leqslant 0 \Leftrightarrow x \preceq y]\})\\
&\Rightarrow\ \ \{[\,]\}\ (\mathtt{sort}\ f\ l)\ \{\lambda l'.\ [\mathsf{permut}\ l\ l'\ \wedge\ \mathsf{sorted}\ (\preceq)\ l']\}
\end{aligned}$$

More general specification of comparison functions:

$$\{[\,]\}\ (f\,x\,y)\ \{\lambda n.\,[\mathsf{if}\ n = 0\ \mathsf{then}\ x \approx y\ \mathsf{else\ if}\ n < 0\ \mathsf{then}\ x \prec y\ \mathsf{else}\ x \succ y]\}$$

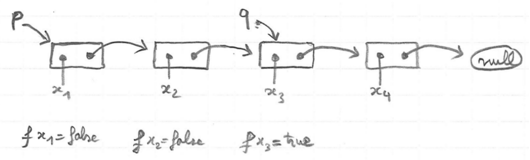
Find with a boolean predicate, on pure lists

```
let rec find f l =
  match l with
  | [] -> None
  | x::k -> if f x
            then Some x
            else find f k
```

Specification:

$$\forall f\, l\, P.\quad (\forall x.\ \{[\,]\}\ (f\ x)\ \{\lambda b.\ [b = \mathsf{true} \Leftrightarrow P\ x]\})$$
$$\Rightarrow\quad \{[\,]\}\ (\texttt{find}\ f\ l)\ \{\lambda o.\ [\ \mathsf{match}\ o\ \mathsf{with}\ ]\}$$
$$\mid \mathsf{None} \Rightarrow \mathsf{Forall}\ (\neg P)\ l$$
$$\mid \mathsf{Some}\ x \Rightarrow \exists k t.\ l = k +\!\!+ x :: t \wedge \mathsf{Forall}\ (\neg P)\ k\ \wedge\ P\ x$$

Find with a boolean predicate, on mutable lists



Specification:

$$\forall f p l P. \quad (\forall x.\ \{[\,]\}\ (f\,x)\ \{\lambda b.\,[b = \mathsf{true} \Leftrightarrow P\,x]\})$$

$$\Rightarrow \quad \{p \rightsquigarrow \mathsf{Mlist}\,l\}$$

$$(\mathtt{mfind}\ f\ p)$$

$$\{\lambda o.\,\mathsf{match}\ o\ \mathsf{with}$$

$$\mid \mathsf{None} \Rightarrow p \rightsquigarrow \mathsf{Mlist}\,l\ \star\ [\mathsf{Forall}\,(\neg P)\,l]$$

$$\mid \mathsf{Some}\,q \Rightarrow \exists kt.\ p \rightsquigarrow \mathsf{MlistSeg}\,q\,k\ \star\ q \rightsquigarrow \mathsf{Mlist}\,(x :: t)$$

$$\star\,[l = k +\!\!+ x :: t\ \wedge\ \mathsf{Forall}\,(\neg P)\,k\ \wedge\ P\,x]\ \}$$

Summary

$$\begin{array}{ll} & \left(\forall xk.\ \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k\&x)\}\right) \\ \Rightarrow & \{I\,\mathsf{nil}\}\ (\mathsf{iter}\,f\,l)\ \{\lambda\_.\ I\,l\} \end{array}$$

$$\begin{array}{ll} & \left(\forall xik.\ \{J\,i\,k\}\ (f\,i\,x)\ \{\lambda j.\ J\,j\,(k\&x)\}\right) \\ \Rightarrow & \{J\,a\,\mathsf{nil}\}\ (\mathsf{fold}\,f\,a\,l)\ \{\lambda b.\ J\,b\,l\} \end{array}$$

$$\begin{array}{ll} & \left(\forall xkk'.\ \{J\,k\,k'\}\ (f\,x)\ \{\lambda x'.\,J\,(k\&x)\,(k'\&x')\}\right) \\ \Rightarrow & \{J\,\mathsf{nil}\,\mathsf{nil}\}\ (\mathsf{map}\,f\,l)\ \{\lambda l'.\ J\,l\,l'\} \end{array}$$

- Add the hypothesis $l = k +\!\!+ x :: s$ if the position of x matters.

- Boolean predicates: $\forall x.\ \{[\,]\}\ (f\,x)\ \{\lambda b.\,[b = \mathsf{true} \Leftrightarrow P\,x]\}$.
- Order functions: $\forall xy.\ \{[\,]\}\ (f\,x\,y)\ \{\lambda n.\,[n \leqslant 0 \Leftrightarrow x \preceq y]\}$.

Chapter 17

Principle of Characteristic Formulae

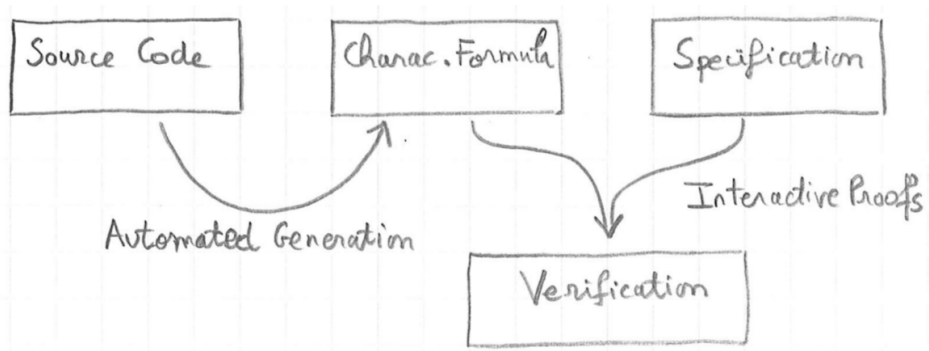
Idea of characteristic formulae

Goal: perform all the Separation Logic reasoning inside Coq.

Idea: build a logical formula $[\![t]\!]$ satisfying the equivalence below.

$$\forall H Q. \quad [\![t]\!]\, H\, Q \;\Leftrightarrow\; \{H\}\, t\, \{Q\}$$

Schema:



Properties of characteristic formulae

The characteristic formula $[\![t]\!]$ of a term t is a predicate such that:

$$\forall HQ. \quad [\![t]\!]\, H\, Q \;\Leftrightarrow\; \{H\}\, t\, \{Q\}$$

Properties:

- $[\![t]\!]$ has type $(\mathsf{Heap} \to \mathsf{Prop}) \to (\mathsf{Val} \to \mathsf{Heap} \to \mathsf{Prop}) \to \mathsf{Prop}$
- $[\![t]\!]$ characterizes the set of valid specifications for t
- $[\![t]\!]$ is a higher-order logic formula built using $\wedge$, $\vee$, $\Rightarrow$, $\exists$, ...
- $[\![t]\!]$ is built automatically and compositionally
- $[\![t]\!]$ has size linear in the size of t and is easy to read

Characteristic formula for sequence

$$\forall H Q.\quad [\![t]\!]\, H\, Q \;\Leftrightarrow\; \{H\}\, t\, \{Q\}$$

$$\frac{\{H\}\, t_1\, \{Q'\} \qquad \{Q'\,()\}\, t_2\, \{Q\}}{\{H\}\, (t_1\,;\, t_2)\, \{Q\}}\ \textsc{seq}$$

Goal:

$$\forall H Q.\quad [\![t_1\,;\, t_2]\!]\, H\, Q \;\Leftrightarrow\; \{H\}\, (t_1\,;\, t_2)\, \{Q\}$$

Exercise: define the characteristic formula for sequences.

$$[\![t_1\,;\, t_2]\!] \;\equiv\; \lambda H.\, \lambda Q.\;\; \exists Q'.\;\; [\![t_1]\!]\, H\, Q' \;\wedge\; [\![t_2]\!]\, (Q'\,())\, Q$$

Characteristic formula for let bindings

$$\frac{\{H\}\ t_1\ \{Q'\} \qquad \forall x.\ \{Q'\,x\}\ t_2\ \{Q\}}{\{H\}\ (\mathsf{let}\, x = t_1\, \mathsf{in}\, t_2)\ \{Q\}}$$

Definition:

$$[\![\mathsf{let}\, x = t_1\, \mathsf{in}\, t_2]\!] \ \equiv\ \lambda H Q.\ \exists Q'.\ [\![t_1]\!]\, H\, Q' \ \wedge\ \forall x.\ [\![t_2]\!]\, (Q'\, x)\, Q$$

Technically, x has type var and:

$$[\![\mathsf{let}\, x = t_1\, \mathsf{in}\, t_2]\!] \ \equiv\ \lambda H Q.\ \exists Q'.\quad [\![t_1]\!]\, H\, Q' \\ \wedge\ \forall (X : \mathsf{Val}).\ [\![([x \to X]\, t_2)]\!]\, (Q'\, X)\, Q$$

Characteristic formula for values

$$\frac{H \rhd Q\, v}{\{H\}\ v\ \{Q\}}\ \textsc{val-frame}$$

Definition:

$$[\![v]\!] \ \equiv \ \lambda H Q.\ H \rhd Q\, v$$

Characteristic formula for conditionals

$$\frac{(b = \mathsf{true} \Rightarrow \{H\}\ t_1\ \{Q\}) \qquad (b = \mathsf{false} \Rightarrow \{H\}\ t_2\ \{Q\})}{\{H\}\ (\mathsf{if}\, b\, \mathsf{then}\, t_1\, \mathsf{else}\, t_2)\ \{Q\}}\ \text{IF}$$

Exercise: define the characteristic formula for conditionals.

$$\begin{aligned} [\![\mathsf{if}\, b\, \mathsf{then}\, t_1\, \mathsf{else}\, t_2]\!] \ \equiv\ \ \lambda H Q.\ \ & (b = \mathsf{true} \Rightarrow [\![t_1]\!]\, H\, Q) \\ \wedge\ \ & (b = \mathsf{false} \Rightarrow [\![t_2]\!]\, H\, Q) \end{aligned}$$

The App predicate

The goal of characteristic formulae is do proofs without involving triples.

Let "App" be an abstract predicate with the following interpretation:

$$\mathsf{App}\, f\, v\, H\, Q \quad \Leftrightarrow \quad \{H\}\ (f\, v)\ \{Q\}$$

Remark:

$$\mathsf{App} \ : \ \mathsf{Val} \to \mathsf{Val} \to \mathsf{Hprop} \to (\mathsf{Val} \to \mathsf{Hprop}) \to \mathsf{Prop}$$

Reasoning about function calls

Interpretation of App:	$\mathsf{App}\, f\, v\, H\, Q$	$\Leftrightarrow$	$\{H\}\ (f\ v)\ \{Q\}$
Interpretation of formulae:	$[\![f\ v]\!]\, H\, Q$	$\Leftrightarrow$	$\{H\}\ (f\ v)\ \{Q\}$

Definition:

$$[\![f\ v]\!] \ \equiv\ \lambda H Q.\ \mathsf{App}\, f\, v\, H\, Q$$

Instances of App are used on calls and introduced on function definitions.

Reasoning about function definitions

$$\frac{\begin{array}{c}\forall f.\ \ P\,f \ \Rightarrow\ \{H\}\ t_2\ \{Q\} \\ P\,f \ =\ \left(\forall x H' Q'.\ \{H'\}\ t_1\ \{Q'\} \ \Rightarrow\ \{H'\}\ (f\ x)\ \{Q'\}\right)\end{array}}{\{H\}\ (\mathsf{let\ rec}\ f\ x = t_1\ \mathsf{in}\ t_2)\ \{Q\}}\ \textsc{fix}$$

Definition:

$$[\![\mathsf{let\ rec}\ f = \lambda x.\, t_1\ \mathsf{in}\ t_2]\!] \ \equiv\ \lambda H Q.\ \ \forall f.\ \ P\,f \ \Rightarrow\ [\![t_2]\!]\ H\ Q$$

$$\text{where}\ \ P\,f \ \equiv\ (\forall x H' Q'.\ [\![t_1]\!]\ H'\ Q' \ \Rightarrow\ \mathsf{App}\ f\ x\ H'\ Q')$$

Complete definition

$$
\begin{array}{lcl}
[\![v]\!] & \equiv & \lambda H Q.\ H \rhd Q\, v \\
[\![\mathsf{let}\, x = t_1 \,\mathsf{in}\, t_2]\!] & \equiv & \lambda H Q.\ \exists Q'.\ [\![t_1]\!]\, H\, Q' \ \wedge\ \forall x.\ [\![t_2]\!]\, (Q'\, x)\, Q \\
[\![\mathsf{if}\, b \,\mathsf{then}\, t_1 \,\mathsf{else}\, t_2]\!] & \equiv & \lambda H Q.\ \ (b = \mathsf{true} \Rightarrow [\![t_1]\!]\, H\, Q) \\
 & & \qquad \wedge\ \ (b = \mathsf{false} \Rightarrow [\![t_2]\!]\, H\, Q) \\
[\![v_1\, v_2]\!] & \equiv & \lambda H Q.\ \mathsf{App}\, v_1\, v_2\, H\, Q \\
[\![\mathsf{let\,rec}\, f = \lambda x.\, t_1 \,\mathsf{in}\, t_2]\!] & \equiv & \lambda H Q.\ \forall f.\ P f \ \Rightarrow\ [\![t_2]\!]\, H\, Q
\end{array}
$$

$$
\text{where } P f \ \equiv\ (\forall x H' Q'.\ [\![t_1]\!]\, H'\, Q' \ \Rightarrow\ \mathsf{App}\, f\, x\, H'\, Q')
$$

The end!