Separation Logic 3/4

Jean-Marie Madiot

Inria Paris

February 7, 2023

using some material from Arthur Charguéraud

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
```

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:

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

After the loop:

$$r \mapsto n!$$

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:

from $r \mapsto (i-1)!$ to $r \mapsto i!$

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 $a \leqslant b$:

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

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

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.

Reasoning rule for while loops

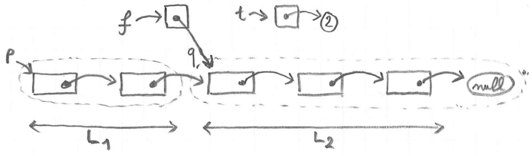
We focus on a different approach that:

- 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:

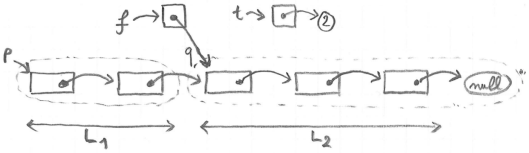
$$\frac{\{H\}\ (\mathsf{if}\, t_1\, \mathsf{then}\, (t_2\, ;\ (\mathsf{while}\, t_1\, \mathsf{do}\, t_2))\, \mathsf{else}\, ())\ \{Q\}}{\{H\}\ (\mathsf{while}\, t_1\, \mathsf{do}\, t_2)\ \{Q\}}$$

Length with a while loop



```
let 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:

$$\{q \rightsquigarrow \mathsf{MList}\, L_2 \,*\, f \mapsto q \,*\, t \mapsto n\}$$

$$\big(\texttt{while !f != null do incr t; f := (!f).tl; done}\big)$$

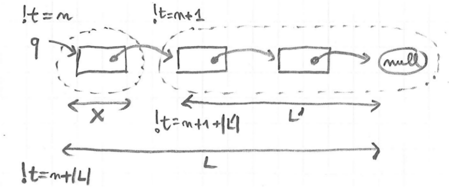
$$\{q \rightsquigarrow \mathsf{MList}\, L_2 \,*\, f \mapsto \mathsf{null} \,*\, t \mapsto (n + \mathsf{length}\, L_2)\}$$

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 `mlength`.

Length with a while loop: frame process



$q \leadsto \mathsf{MList}\, L_2$	$* \;\; f \mapsto q$	$* \;\; t \mapsto n$	begin
$q \mapsto \{\lvert x; q' \rvert\} * q' \leadsto \mathsf{MList}\, L_2'$	$* \;\; f \mapsto q$	$* \;\; t \mapsto n$	unfold
$q \mapsto \{\lvert x; q' \rvert\} * q' \leadsto \mathsf{MList}\, L_2'$	$* \;\; f \mapsto q$	$* \;\; t \mapsto n + 1$	increment
$\underline{q \mapsto \{\lvert x; q' \rvert\}} * q' \leadsto \mathsf{MList}\, L_2'$	$* \;\; f \mapsto q'$	$* \;\; t \mapsto n + 1$	shift head
$q \mapsto \{\lvert x; q' \rvert\} * q' \leadsto \mathsf{MList}\, L_2'$	$* \;\; f \mapsto \mathsf{null}$	$* \;\; t \mapsto n + 1 + \lvert L_2' \rvert$	frame+Ind.hyp.
$q \leadsto \mathsf{MList}\, L_2$	$* \;\; f \mapsto \mathsf{null}$	$* \;\; t \mapsto n + \lvert L_2 \rvert$	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
```

Exercise: 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).

Functions with aliasing: 3 specifications for swap

Specification 1:

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

Functions with aliasing: 3 specifications for swap

Specification 1:

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

Specification 2:

$$\forall r s n.\ \{\ulcorner r = s \urcorner * (r \mapsto n)\}\ (\texttt{swap r s})\ \{\lambda\_.\ r \mapsto n\}$$

or simply:

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

Functions with aliasing: 3 specifications for swap

Specification 1:

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

Specification 2:

$$\forall rsn.\ \{\ulcorner r = s \urcorner * (r \mapsto n)\}\ (\texttt{swap r s})\ \{\lambda\_.\ r \mapsto n\}$$

or simply:

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

Specification 3:

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

alternatively, $\forall rsM.\ r, s \in \mathsf{dom}\, M \ \Rightarrow$

$$\{\mathsf{Cells}'(M)\}\ (\texttt{swap r s})\ \{\lambda\_.\ \mathsf{Cells}'\ (M[r := M[s]][s := M[r]])\}$$

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
```

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
```

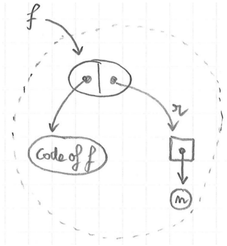
Specification:

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

Successive states: $r \mapsto 3 \qquad r \mapsto 4 \qquad r \mapsto 5$

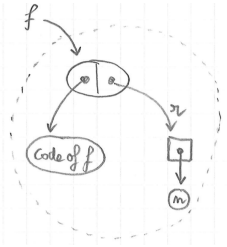
Counter function: code

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



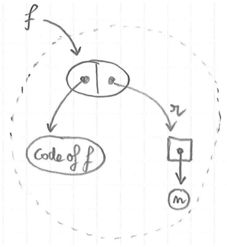
```
let c = mkcounter() in
let x = c() in
let y = c() in
assert (x = 1 && y = 2)
```

Counter function: specification



$$f \rightsquigarrow \mathsf{Count}\, n \;\equiv\; \exists r.\, (r \mapsto n) \\ \quad * \ulcorner \forall i.\ \{r \mapsto i\}\ (f\ ())\ \{\lambda x.\, \ulcorner x = i+1 \urcorner * (r \mapsto i+1)\} \urcorner$$

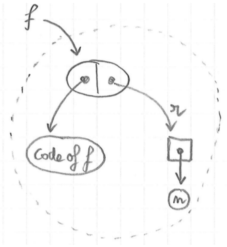
Counter function: specification



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

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

Counter function: specification



$$f \rightsquigarrow \mathsf{Count}\, n \;\equiv\; \exists r.\, (r \mapsto n) \\ \qquad * \ulcorner \forall i.\, \{r \mapsto i\}\, (f\ ())\, \{\lambda x.\, \ulcorner x = i + 1 \urcorner * (r \mapsto i + 1)\} \urcorner$$

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

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

$$\forall f i. \quad \{f \rightsquigarrow \mathsf{Count}\, i\}\, (f\ ())\, \{\lambda x.\, \ulcorner x = i + 1 \urcorner * f \rightsquigarrow \mathsf{Count}\, (i + 1)\}$$

Chapter 14

Basic higher-order functions

Apply

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

Specification:

$$\begin{aligned} \forall f x H Q. \quad & \{H\}\ (f\ x)\ \{Q\} \\ \Rightarrow \quad & \{H\}\ (\texttt{apply}\ f\ x)\ \{Q\} \end{aligned}$$

Apply

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

Specification:

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

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

$$\forall f x H Q. \quad \{H \ast \ulcorner \{H\}\ (f\,x)\ \{Q\} \urcorner\}\ (\texttt{apply}\,f\,x)\ \{Q\}$$

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'.

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'.

$$\begin{array}{rl} \forall r f x P. & \{\ulcorner\urcorner\}\ (f\ x)\ \{\lambda y.\ulcorner P\,x\,y\urcorner\} \\ \Rightarrow & \{r \mapsto x\}\ (\texttt{refapply}\ r\ f)\ \{\lambda\_.\ \exists y.\ulcorner P\,x\,y\urcorner\ *\ r \mapsto y\} \end{array}$$

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'.

$$\begin{array}{lll} \forall r f x P. & & \{\ulcorner\urcorner\}\,(f\,x)\,\{\lambda y.\,\ulcorner P\,x\,y\urcorner\} \\ & \Rightarrow & \{r \mapsto x\}\,(\texttt{refapply}\,r\,f)\,\{\lambda\_.\ \exists y.\,\ulcorner P\,x\,y\urcorner \ \ast\ r \mapsto y\} \end{array}$$

$$\begin{array}{lll} \forall r f x H H' P. & & \{H\}\,(f\,x)\,\{\lambda y.\,\ulcorner P\,x\,y\urcorner \ \ast\ H'\} \\ & \Rightarrow & \{(r \mapsto x)\ \ast\ H\} \\ & & (\texttt{refapply}\,r\,f) \\ & & \{\lambda\_.\ \exists y.\,\ulcorner P\,x\,y\urcorner \ \ast\ (r \mapsto y)\ \ast\ H'\} \end{array}$$

Function twice

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

Specification:

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

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 $\rightarrow$ Hprop.

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

⚠ For pedagogical purposes, “pure lists” are values that live outside the heap and need no representation predicate. (In practice, lists do need to be allocated.) ⚠

```
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.

Iteration over a pure list

⚠ For pedagogical purposes, "pure lists" are values that live outside the heap and need no representation predicate. (In practice, lists do need to be allocated.) ⚠

```
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.

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

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

Length using iter

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

```
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.

Length using iter

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

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

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

Sum using iter

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

```
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`.

Sum using iter

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

```
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

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

Given a list $x_1 :: x_2 :: ... :: x_n ::$ nil, let us compute:

$$\frac{10}{x_1} + \frac{10}{x_2} + ... + \frac{10}{x_n}$$

```
iter (fun x -> r := !r + 10 / x) [2; -3; 4]
```

Constraints over the items

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

Given a list $x_1 :: x_2 :: ... :: x_n ::$ nil, let us compute:

$$\frac{10}{x_1} + \frac{10}{x_2} + ... + \frac{10}{x_n}$$

```
iter (fun x -> r := !r + 10 / x) [2; -3; 4]
```

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

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

Constraints over the items, in order

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

Given a list $x_1 :: x_2 :: \ldots :: x_n :: \mathsf{nil}$, let us compute:

$$\cfrac{10}{\cfrac{10}{\cfrac{10}{0+x_1}+x_2}\ \iddots\ +x_n}$$

```
iter (fun x -> r := 10 / (!r + x)) [2; -3; 4]
```

Constraints over the items, in order

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

Given a list $x_1 :: x_2 :: \ldots :: x_n :: \mathsf{nil}$, let us compute:

$$\cfrac{10}{\cfrac{10}{\frac{10}{0+x_1}+x_2}\ \cdots +x_n}$$

```
iter (fun x -> r := 10 / (!r + x)) [2; -3; 4]
```

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

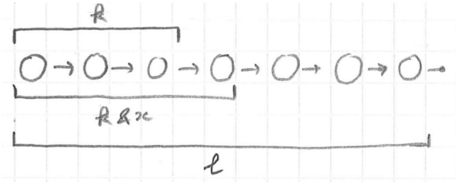
$$\forall f I l. \quad \big(\forall x k s.\ l = k \mathbin{++} x :: s \Rightarrow \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k \& x)\}\big) \\ \Rightarrow \{I\,\mathsf{nil}\}\ (\mathtt{iter}\,f\,l)\ \{\lambda\_.\ I\,l\}$$

Verification of iter

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

```
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:

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

Prove:

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

Verification of iter: generalized principle

Assume:

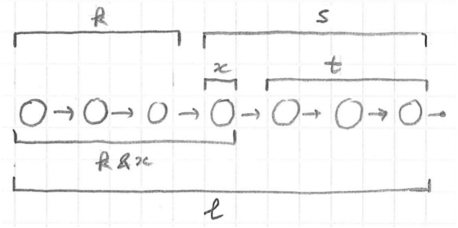
$$\forall x k.\ \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k\&x)\}$$

Prove:

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

Proof by induction over a generalized statement:

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



Verification of iter: induction

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

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

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

By induction on s:

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

Verification of iter: induction

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

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

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

By induction on s:

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

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

Invariant on remaining items

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

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

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.
- prove the old specification derivable from the new one,
- prove the new specification derivable from the old (most general) one.

Invariant on remaining items

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

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

Exercise:

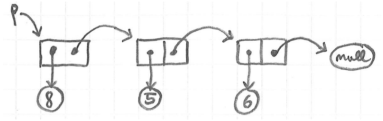
- specify `iter` using an invariant that depends on the list of items remaining to process, instead of on the list of items already processed.
- prove the old specification derivable from the new one,
- prove the new specification derivable from the old (most general) one.

$$\big(\forall xs.\ \{I'\,(x :: s)\}\ (f\,x)\ \{\lambda\_.\ I'\,s\}\big) \quad\Rightarrow\quad \{I'\,l\}\ (\mathsf{iter}\,f\,l)\ \{\lambda\_.\ I'\,\mathsf{nil}\}$$

$$I\,k \equiv \exists s.\ \ulcorner l = k \mathbin{+\!\!+} s \urcorner * I'\,s$$

$$I'\,s \equiv \exists k.\ \ulcorner l = k \mathbin{+\!\!+} s \urcorner * I\,k$$

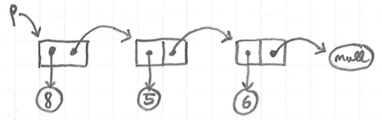
Iterating over a mutable list



```
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:

$$\forall f p I l. \quad \big(\forall x k.\ \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k\&x)\}\big) \\ \Rightarrow \quad \{p \leadsto \mathsf{MList}\,l\ *\ I\,\mathsf{nil}\}\ (\mathtt{miter}\,f\,p)\ \{\lambda\_.\ p \leadsto \mathsf{MList}\,l\ *\ I\,l\}$$

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

Summary

Simplified:

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

Order-irrelevant:

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

Most-general:

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

Using remaining items (already most general):

$$\big(\forall x s.\ \{I'\,(x :: s)\}\ (f\ x)\ \{\lambda\_.\ I'\,s\}\big)\quad\Rightarrow\quad \{I'\,l\}\ (\mathsf{iter}\,f\,l)\ \{\lambda\_.\ I'\,\mathsf{nil}\}$$

Extension to mutable lists:

$$\big(\forall x k s.\ l = k +\!\!+ x :: s\ \Rightarrow \{I\,k\}\ (f\ x)\ \{\lambda\_.\ I\,(k\&x)\}\big)$$
$$\Rightarrow\quad \{p \rightsquigarrow \mathsf{MList}\,l\ *\ I\,\mathsf{nil}\}\ (\mathtt{miter}\,f\,p)\ \{\lambda\_.\ p \rightsquigarrow \mathsf{MList}\,l\ *\ I\,l\}$$

Break?

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:

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

Fold-left

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

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

Specification:

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

Application of fold-left

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

```
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.

Application of fold-left

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

```
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.

$$J\,i\,k \;\equiv\; (r \mapsto |k|) \,*\, \ulcorner i = \mathsf{Sum}\,k \urcorner$$

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:

$$\text{fold\_right}\, f\, [6 :: 4 :: 7]\, a \;\equiv\; f\, 6\, (f\, 4\, (f\, 7\, a))$$

Exercise: give a specification for `fold_right`.

Fold-right

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

Example:

$$\text{fold\_right}\, f\, [6 :: 4 :: 7]\, a \;\equiv\; f\, 6\, (f\, 4\, (f\, 7\, a))$$

Exercise: give a specification for `fold_right`.

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

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:

$$\begin{aligned} \forall f l P. \quad & (\forall x.\ \{\ulcorner\urcorner\}\ (f\,x)\ \{\lambda x'.\ \ulcorner P\,x\,x'\urcorner\}) \\ \Rightarrow \quad & \{\ulcorner\urcorner\}\ (\texttt{map}\,f\,l)\ \{\lambda l'.\ \ulcorner \mathsf{Forall2}\,P\,l\,l'\urcorner\} \end{aligned}$$

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`:

$$\begin{aligned} \forall f l P. \quad & (\forall x.\ \{\ulcorner\urcorner\}\ (f\,x)\ \{\lambda x'.\ulcorner P\,x\,x'\urcorner\}) \\ \Rightarrow \quad & \{\ulcorner\urcorner\}\ (\mathsf{map}\,f\,l)\ \{\lambda l'.\ \ulcorner \mathsf{Forall2}\,P\,l\,l'\urcorner\} \end{aligned}$$

Specification of `iter`:

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

Map: general specification

Specification of `map`:

$$\begin{aligned}\forall f l P. \quad & (\forall x.\ \{\ulcorner\urcorner\}\ (f\,x)\ \{\lambda x'.\ulcorner P\,x\,x'\urcorner\}) \\ \Rightarrow \quad & \{\ulcorner\urcorner\}\ (\mathsf{map}\,f\,l)\ \{\lambda l'.\ \ulcorner \mathsf{Forall2}\,P\,l\,l'\urcorner\}\end{aligned}$$

Specification of `iter`:

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

Combining the two:

$$\begin{aligned}\forall f l P I. \quad & \big(\forall x k.\ \{I\,k\}\ (f\,x)\ \{\lambda x'.\ulcorner P\,x\,x'\urcorner * I\,(k\&x)\}\big) \\ \Rightarrow \quad & \{I\,\mathsf{nil}\}\ (\mathsf{map}\,f\,l)\ \{\lambda l'.\ \ulcorner \mathsf{Forall2}\,P\,l\,l'\urcorner * I\,l\}\end{aligned}$$

Map: general specification, alternative

$$\begin{array}{ll} \forall f l P I. & \big(\forall x k.\ \{I\, k\}\ (f\, x)\ \{\lambda x'.\, \ulcorner P\, x\, x' \urcorner * J\, (k \& x)\}\big) \\ & \Rightarrow\ \ \{I\ \mathsf{nil}\}\ (\mathsf{map}\, f\, l)\ \{\lambda l'.\ \ulcorner \mathsf{Forall2}\, P\, l\, l' \urcorner * I\, l\} \end{array}$$

Alternative specification:

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

Previous specification derivable from the above one:

$$J'\, k\, k' \ \equiv\ \ulcorner \mathsf{Forall2}\, P\, k\, k' \urcorner * I\, k$$

Sorting with comparison function

Example:

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

Specification:

$$\begin{array}{ll} \multicolumn{2}{l}{\forall f l.\ \forall(\preceq).} \\ & \text{total-order}\,(\preceq) \\ \wedge & (\forall x y.\ \{\ulcorner\urcorner\}\ (f\,x\,y)\ \{\lambda n.\ \ulcorner n \leqslant 0 \Leftrightarrow x \preceq y \urcorner\}) \\ \Rightarrow & \{\ulcorner\urcorner\}\ (\texttt{sort}\,f\,l)\ \{\lambda l'.\ \ulcorner \text{permut}\,l\,l' \ \wedge\ \text{sorted}\,(\preceq)\,l' \urcorner\} \end{array}$$

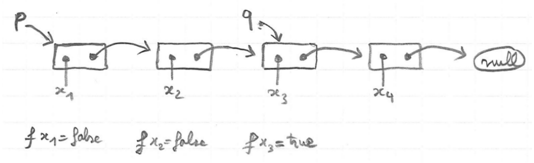
Find with a boolean predicate, on pure lists

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

Specification:

$$\begin{array}{rl} \forall f l P. & (\forall x.\ \{\ulcorner\urcorner\}\ (f\,x)\ \{\lambda b.\ \ulcorner b = \mathsf{true} \Leftrightarrow P\,x \urcorner\}) \\ \Rightarrow & \{\ulcorner\urcorner\}\ (\mathtt{find}\,f\,l)\ \{\lambda o.\ \ulcorner\ \begin{array}{l} \mathsf{match}\,o\,\mathsf{with} \\ \mid \mathsf{None} \Rightarrow \mathsf{Forall}\,(\neg P)\,l \\ \mid \mathsf{Some}\,x \Rightarrow \exists k t.\ l = k +\!\!+ x :: t \\ \qquad\qquad\quad \wedge\ \mathsf{Forall}\,(\neg P)\,k\ \wedge\ P\,x \end{array} \urcorner\} \end{array}$$

Find with a boolean predicate, on mutable lists

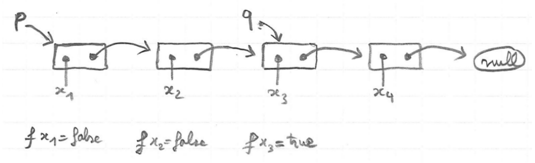


Specification:

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

$$\Rightarrow \quad \{p \rightsquigarrow \mathsf{MList}\, l\}$$
$$(\texttt{mfind}\ f\ p)$$
$$\{\lambda o.$$

Find with a boolean predicate, on mutable lists

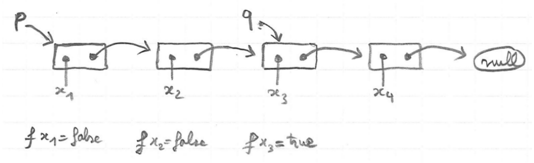


Specification:

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

$$\Rightarrow\quad \{p \rightsquigarrow \mathsf{MList}\, l\}$$
$$(\texttt{mfind}\ f\ p)$$
$$\{\lambda o.\, \mathsf{match}\ o\ \mathsf{with}$$
$$\mid \mathsf{None} \Rightarrow p \rightsquigarrow \mathsf{MList}\, l\ *\ \ulcorner \mathsf{Forall}\,(\neg P)\, l\urcorner$$

Find with a boolean predicate, on mutable lists



Specification:

$$
\begin{array}{ll}
\forall f p l P. & (\forall x.\ \{\ulcorner\urcorner\}\ (f\ x)\ \{\lambda b.\ \ulcorner b = \mathsf{true} \Leftrightarrow P\,x\urcorner\}) \\
\Rightarrow & \{p \rightsquigarrow \mathsf{MList}\, l\} \\
& (\texttt{mfind}\ f\ p) \\
& \{\lambda o.\ \mathsf{match}\ o\ \mathsf{with} \\
& \quad \mid \mathsf{None} \Rightarrow p \rightsquigarrow \mathsf{MList}\, l \ * \ \ulcorner \mathsf{Forall}\,(\neg P)\, l\urcorner \\
& \quad \mid \mathsf{Some}\, q \Rightarrow \exists k t.\ p \rightsquigarrow \mathsf{MlistSeg}\, q\, k \ * \ q \rightsquigarrow \mathsf{MList}\,(x :: t) \\
& \qquad\qquad * \ \ulcorner l = k +\!\!+ x :: t \ \wedge\ \mathsf{Forall}\,(\neg P)\, k \ \wedge\ P\, x\urcorner \quad \}
\end{array}
$$

Summary

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

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

$$\begin{aligned} & \big(\forall x k k'.\ \{J\,k\,k'\}\ (f\,x)\ \{\lambda x'.\,J\,(k\&x)\,(k'\&x')\}\big) \\ \Rightarrow\quad & \{J\ \mathsf{nil}\ \mathsf{nil}\}\ (\mathsf{map}\ f\,l)\ \{\lambda l'.\ J\,l\,l'\} \end{aligned}$$

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

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

Chapter 17

Higher-order representation predicates

Overview

1. Higher-order predicate:

$$p \rightsquigarrow \mathsf{MList}\, L \qquad \text{is generalized into} \qquad p \rightsquigarrow \mathsf{Mlistof}\, R\, L$$

2. Identity representation predicate:

$$p \rightsquigarrow \mathsf{Mlistof}\, \mathsf{Id}\, L \qquad \text{is the same as} \qquad p \rightsquigarrow \mathsf{MList}\, L$$

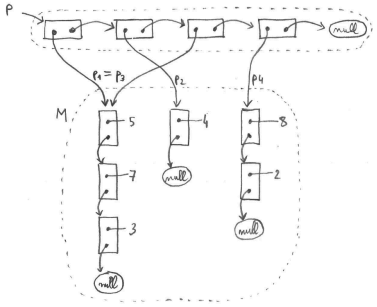
3. Control accesses:

$$\{p \rightsquigarrow \mathsf{MCellof}\, \mathsf{Id}\, v_1\, R_2\, V_2\}\ (p.\mathsf{hd})\ \{\lambda x.\, \ulcorner x = v_1 \urcorner\ *\ \ldots\}$$

4. Compose recursively:

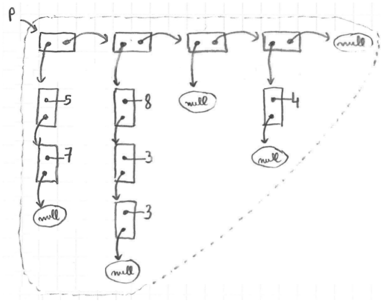
$$p \rightsquigarrow \mathsf{Nodeof}\, R\, X\, (\mathsf{Mlistof}\, (\mathsf{Narytreeof}\, R))\, L$$

Mutable list of possibly-aliased lists



$$p \rightsquigarrow \mathsf{MList}\, K \;\; * \;\; \left(\mathop{\circledast}_{(p_i, L_i) \in M} p_i \rightsquigarrow \mathsf{MList}\, L_i \right) \;\; * \;\; \ulcorner \forall p_i \in K.\;\; p_i \in \mathsf{dom}\, M \urcorner$$

Mutable list of disjoint mutable lists

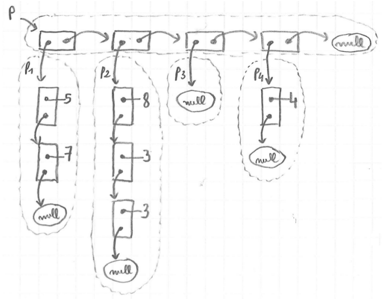


$$L \;=\; (5 :: 7 :: \mathsf{nil}) :: (8 :: 3 :: 3 :: \mathsf{nil}) \\ :: (\mathsf{nil}) :: (4 :: \mathsf{nil}) \;::\; \mathsf{nil}$$

$$p \leadsto \mathsf{MlistofMlist}\, L$$

(to be later generalized into: $p \leadsto \mathsf{Mlistof}\, R\, L$)

Representation using iterated star

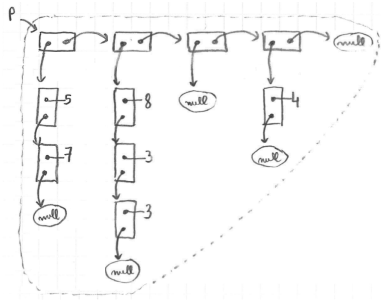


$$L = (5 :: 7 :: \mathsf{nil}) :: (8 :: 3 :: 3 :: \mathsf{nil}) :: (\mathsf{nil}) :: (4 :: \mathsf{nil}) :: \mathsf{nil}$$

$$K = p_1 :: p_2 :: p_3 :: p_4 :: \mathsf{nil}$$

$$p \rightsquigarrow \mathsf{MlistofMlist}\, L \quad \equiv \quad \exists K. \quad p \rightsquigarrow \mathsf{MList}\, K \;\; * \;\; \circledast_{i \in [0, |L|)} (K[i]) \rightsquigarrow \mathsf{MList}\,(L[i]) \;\; * \;\; \ulcorner |K| = |L| \urcorner$$

Representation using a recursive predicate



$$L = (5 :: 7 :: \mathsf{nil}) :: (8 :: 3 :: 3 :: \mathsf{nil}) :: (\mathsf{nil}) :: (4 :: \mathsf{nil}) :: \mathsf{nil}$$

$$\begin{aligned} p \rightsquigarrow \mathsf{MlistofMlist}\, L \ \equiv\ & \mathsf{match}\ L\ \mathsf{with} \\ & \mid \mathsf{nil} \Rightarrow \ulcorner p = \mathsf{null} \urcorner \\ & \mid X :: L' \Rightarrow \exists x p'.\ \ p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \\ & \qquad\qquad\qquad\quad * \ p' \rightsquigarrow \mathsf{MlistofMlist}\, L' \\ & \qquad\qquad\qquad\quad * \ x \rightsquigarrow \mathsf{MList}\, X \end{aligned}$$

Generalization to a higher-order predicate

$$
\begin{aligned}
p \leadsto \mathsf{MlistofMlist}\, L \;\equiv\; & \mathsf{match}\, L\, \mathsf{with} \\
& \mid \mathsf{nil} \;\Rightarrow\; \ulcorner p = \mathsf{null} \urcorner \\
& \mid X :: L' \;\Rightarrow\; \exists x p'.\;\; p \leadsto \{\!|\mathsf{hd}{=}x;\, \mathsf{tl}{=}p'|\!\} \\
& \qquad\qquad\qquad\quad * \; p' \leadsto \mathsf{MlistofMlist}\, L' \\
& \qquad\qquad\qquad\quad * \; x \leadsto \mathsf{MList}\, X
\end{aligned}
$$

Generalization:

$$
\begin{aligned}
p \leadsto \mathsf{Mlistof}\, R\, L \;\equiv\; & \mathsf{match}\, L\, \mathsf{with} \\
& \mid \mathsf{nil} \;\Rightarrow\; \ulcorner p = \mathsf{null} \urcorner \\
& \mid X :: L' \;\Rightarrow\; \exists x p'.\;\; p \leadsto \{\!|\mathsf{hd}{=}x;\, \mathsf{tl}{=}p'|\!\} \\
& \qquad\qquad\qquad\quad * \; p' \leadsto \mathsf{Mlistof}\, R\, L' \\
& \qquad\qquad\qquad\quad * \; x \leadsto R\, X
\end{aligned}
$$

In particular:

$$p \leadsto \mathsf{MlistofMlist}\, L \;=\; p \leadsto \mathsf{Mlistof}\, \mathsf{MList}\, L$$

Type-checking

$p \leadsto \mathsf{Mlistof}\, R\, L$ *is a notation for* $\mathsf{Mlistof}\, R\, L\, p$ (of type Hprop)
$x \leadsto R\, X$ *is a notation for* $R\, X\, x$ (of type Hprop)

$$\begin{array}{lcl} p \leadsto \mathsf{Mlistof}\, R\, L & \equiv & \mathsf{match}\ L\ \mathsf{with} \\ & & \mid \mathsf{nil}\ \Rightarrow\ \ulcorner p = \mathsf{null} \urcorner \\ & & \mid X :: L'\ \Rightarrow\ \exists x p'.\quad p \leadsto \{\!| \mathsf{hd}{=}x;\ \mathsf{tl}{=}p' |\!\} \\ & & \qquad\qquad\qquad\quad * \ \ p' \leadsto \mathsf{Mlistof}\, R\, L' \\ & & \qquad\qquad\qquad\quad * \ \ x \leadsto R\, X \end{array}$$

Exercise: since $(p : \mathsf{loc})$ and $(x : \mathsf{Val})$ and $(X : A)$ for some A, what is the type of R? What is the type of Mlistof?

Type-checking

$p \leadsto \mathsf{Mlistof}\, R\, L$	*is a notation for*	$\mathsf{Mlistof}\, R\, L\, p$	(of type Hprop)
$x \leadsto R\, X$	*is a notation for*	$R\, X\, x$	(of type Hprop)

$$\begin{array}{lcl} p \leadsto \mathsf{Mlistof}\, R\, L & \equiv & \mathsf{match}\ L\ \mathsf{with} \\ & & \mid \mathsf{nil} \ \Rightarrow\ \ulcorner p = \mathsf{null} \urcorner \\ & & \mid X :: L' \ \Rightarrow\ \exists x p'.\quad \begin{array}{ll} & p \leadsto \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \\ * & p' \leadsto \mathsf{Mlistof}\, R\, L' \\ * & x \leadsto R\, X \end{array} \end{array}$$

Exercise: since $(p : \mathsf{loc})$ and $(x : \mathsf{Val})$ and $(X : A)$ for some A, what is the type of R? What is the type of Mlistof?

- $R : A \to \mathsf{Val} \to \mathsf{Hprop}$
- $\mathsf{Mlistof} : \forall A.\ (A \to \mathsf{Val} \to \mathsf{Hprop}) \to \mathsf{list} A \to \mathsf{loc} \to \mathsf{Hprop}$

The identity representation predicate

$$\begin{array}{rcl} p \rightsquigarrow \mathsf{Mlistof}\, R\, L & \equiv & \mathsf{match}\ L\ \mathsf{with} \\ & & \mid \mathsf{nil} \Rightarrow \ulcorner p = \mathsf{null} \urcorner \\ & & \mid X :: L' \Rightarrow \exists x p'.\quad p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \\ & & \qquad\qquad\qquad\qquad * \ p' \rightsquigarrow \mathsf{Mlistof}\, R\, L' \\ & & \qquad\qquad\qquad\qquad * \ x \rightsquigarrow R\, X \end{array}$$

$$\begin{array}{rcl} p \rightsquigarrow \mathsf{MList}\, L & \equiv & \mathsf{match}\ L\ \mathsf{with} \\ & & \mid \mathsf{nil} \Rightarrow \ulcorner p = \mathsf{null} \urcorner \\ & & \mid x :: L' \Rightarrow \exists p'.\quad p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \\ & & \qquad\qquad\qquad * \ p' \rightsquigarrow \mathsf{MList}\, L' \end{array}$$

Exercise: define the identity representation predicate Id such that

$$p \rightsquigarrow \mathsf{Mlistof}\, \mathsf{Id}\, L \;=\; p \rightsquigarrow \mathsf{MList}\, L$$

The identity representation predicate

$$p \rightsquigarrow \mathsf{Mlistof}\, R\, L \;\equiv\; \begin{array}{l}\mathsf{match}\, L\, \mathsf{with} \\ \mid \mathsf{nil} \;\Rightarrow\; \ulcorner p = \mathsf{null} \urcorner \\ \mid X :: L' \;\Rightarrow\; \exists x p'.\quad \begin{array}{l} p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \\ *\ p' \rightsquigarrow \mathsf{Mlistof}\, R\, L' \\ *\ x \rightsquigarrow R\, X \end{array}\end{array}$$

$$p \rightsquigarrow \mathsf{MList}\, L \;\equiv\; \begin{array}{l}\mathsf{match}\, L\, \mathsf{with} \\ \mid \mathsf{nil} \;\Rightarrow\; \ulcorner p = \mathsf{null} \urcorner \\ \mid x :: L' \;\Rightarrow\; \exists p'.\quad \begin{array}{l} p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \\ *\ p' \rightsquigarrow \mathsf{MList}\, L' \end{array}\end{array}$$

Exercise: define the identity representation predicate Id such that

$$p \rightsquigarrow \mathsf{Mlistof}\, \mathsf{Id}\, L \;=\; p \rightsquigarrow \mathsf{MList}\, L$$

Definition:

$$x \rightsquigarrow \mathsf{Id}\, X \;\equiv\; \ulcorner x = X \urcorner$$

Summary

1. Higher-order predicate:

   $p \rightsquigarrow \mathsf{MList}\, L$ is generalized into $p \rightsquigarrow \mathsf{Mlistof}\, R\, L$

2. Identity representation predicate:

   $p \rightsquigarrow \mathsf{Mlistof}\, \mathsf{Id}\, L$ is the same as $p \rightsquigarrow \mathsf{MList}\, L$

Chapter 18

Separating implication

Separating implication or "magic wand"

Recalling separating conjunction:

$$(P_1 * P_2)h \quad \equiv \quad \exists h_1, h_2.\ h = h_1 \uplus h_2 \wedge P_1\,h_1 \wedge P_2\,h_2$$

Introducing *separating implication*:

$$(P \mathrel{-\!\!*} Q)h \quad \equiv \quad \forall h_1.\ h \perp h_1 \wedge P\,h_1 \Rightarrow Q(h \uplus h_1)$$

Intuition:

$$(P \mathrel{-\!\!*} Q) * P \quad \rhd \quad Q$$

Rules:

$$\frac{R * P \vdash Q}{R \vdash (P \mathrel{-\!\!*} Q)} \qquad\qquad \frac{R_1 \vdash (P \mathrel{-\!\!*} Q) \qquad R_2 \vdash P}{R_1 * R_2 \vdash Q}$$

Separating implication examples

Exercise: Give heaps satisfying the following predicates:

1. $\ulcorner\,\urcorner \mathrel{-\!\!*} (1 \mapsto 2)$
2. $\ulcorner \text{False} \urcorner \mathrel{-\!\!*} (1 \mapsto 2)$
3. $\ulcorner x \geqslant 1 \urcorner \mathrel{-\!\!*} \ulcorner x \geqslant 0 \urcorner$
4. $(1 \mapsto 4) \mathrel{-\!\!*} (1 \mapsto 4) * (2 \mapsto 3)$
5. $(1 \mapsto 2) \mathrel{-\!\!*} (1 \mapsto 2)$
6. $(1 \mapsto 2) \mathrel{-\!\!*} \ulcorner \text{False} \urcorner$
7. $(1 \mapsto 2) \mathrel{-\!\!*} \ulcorner\,\urcorner$
8. $(1 \mapsto 2) \mathrel{-\!\!*} (1 \mapsto 3)$

Separating implication examples

Exercise: Among the following heap entailments, which hold?

1. $P \rhd (Q \mathrel{-\!\!*} P * Q)$
2. $(Q \mathrel{-\!\!*} P * Q) \rhd P$
3. $(1 \mapsto 2) \mathrel{-\!\!*} (1 \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$
4. $(1 \mapsto 2) \mathrel{-\!\!*} (1 \mapsto 2 * 2 \mapsto 8) \rhd 2 \mapsto 8$
5. $\ulcorner\urcorner \mathrel{-\!\!*} P \rhd P$
6. $P \rhd \ulcorner\urcorner \mathrel{-\!\!*} P$
7. $\ulcorner\urcorner \rhd (P \mathrel{-\!\!*} Q \mathrel{-\!\!*} P * Q)$
8. $\ulcorner P \rhd Q \urcorner \rhd (P \mathrel{-\!\!*} Q)$
9. $(P \mathrel{-\!\!*} Q) \rhd \ulcorner P \rhd Q \urcorner$