Separation Logic 2/4

Jean-Marie Madiot

Inria Paris

February 10, 2026

part 2/4: some initial material from Arthur Charguéraud

Frame rule examples

Length of a mutable list, recursively

```
let rec mlength (p:'a cell) =
  if p == null then
    0
  else
    let n' = mlength p.tl in
    1 + n'
```

Specification:

$$\forall pL. \quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ \ulcorner n = \mathsf{length}\, L \urcorner * p \rightsquigarrow \mathsf{MList}\, L\}$$

Length of a mutable list, recursively

```
let rec mlength (p:'a cell) =
  if p == null then
    0
  else
    let n' = mlength p.tl in
    1 + n'
```

Specification:

$$\forall pL. \quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ \ulcorner n = \mathsf{length}\, L \urcorner \ast p \rightsquigarrow \mathsf{MList}\, L\}$$

We prove this specification by induction on L.

Verification of mlength: nil case

Case $L =$ **nil**. Then $p = \text{null}$. Goal is:

$$\{p \rightsquigarrow \mathsf{MList}\,\mathsf{nil}\}\ (\mathtt{0})\ \{\lambda n.\ \ulcorner n = \mathsf{length}\,\mathsf{nil}\urcorner * p \rightsquigarrow \mathsf{MList}\,\mathsf{nil}\}$$

Same as:

$$\{\ulcorner p = \mathsf{null}\urcorner\}\ (\mathtt{0})\ \{\lambda n.\ \ulcorner n = 0\urcorner * \ulcorner p = \mathsf{null}\urcorner\}$$

(true by definition of triples, because $p = \mathsf{null} \Rightarrow 0 = 0 \wedge p = \mathsf{null}$.)

Verification of mlength: using the frame rule

I.H.: $\forall L p.\quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ \ulcorner n = \mathsf{length}\, L \urcorner * p \rightsquigarrow \mathsf{MList}\, L\}$

Assume $p \neq$ null. So $L = x :: L'$ for some x, L'.

$\{p \rightsquigarrow \mathsf{MList}\, L\}$

$\{p \rightsquigarrow \mathsf{MList}\, (x :: L')\}$

$\{p \mapsto (x, p') * p' \rightsquigarrow \mathsf{MList}\, L'\}$

```
let n' = mlength p.tl in
```

// by induction hypothesis and framing $p \mapsto (x, p')$

$\{p \mapsto (x, p') * p' \rightsquigarrow \mathsf{MList}\, L' * \ulcorner n' = |L'| \urcorner\}$

```
let n = 1 + n' in
```

$\{p \mapsto (x, p') * p' \rightsquigarrow \mathsf{MList}\, L' * \ulcorner n' = |L'| \urcorner * \ulcorner n = 1 + n' \urcorner\}$

$\{p \mapsto (x, p') * p' \rightsquigarrow \mathsf{MList}\, L' * \ulcorner n = 1 + |L'| \urcorner\}$

$\{p \rightsquigarrow \mathsf{MList}\, (x :: L') * \ulcorner n = 1 + |L'| \urcorner\}$

$\{p \rightsquigarrow \mathsf{MList}\, L * \ulcorner n = |L| \urcorner\}$

Exercise!

Instantiation of the frame rule

Induction hypothesis:

$$\{p' \rightsquigarrow \mathsf{MList}\, L'\}$$
$$(\texttt{mlength p'})$$
$$\{\lambda n'.\ \ulcorner n' = \mathsf{length}\, L' \urcorner \ast p' \rightsquigarrow \mathsf{MList}\, L'\}$$

By the frame rule:

$$\{p' \rightsquigarrow \mathsf{MList}\, L' \ast p \mapsto (x, p')\}$$
$$(\texttt{mlength p'})$$
$$\{\lambda n.\ \ulcorner n = \mathsf{length}\, L' \urcorner \ast p' \rightsquigarrow \mathsf{MList}\, L' \ast p \mapsto (x, p')\}$$

Verification of mlength: rocq

Rocq: `mlength_spec` – use of frame.

Verification of in-place increment

```
let rec list_incr (p:'a cell) =
  if p != null then begin
    p.hd <- p.hd + 1;
    list_incr p.tl
  end
```

$$\forall p L. \quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{list\_incr p})\ \{\lambda\_.\ p \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L)\}$$

Verification of in-place increment

```
let rec list_incr (p:'a cell) =
  if p != null then begin
    p.hd <- p.hd + 1;
    list_incr p.tl
  end
```

$$\forall p L. \quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{list\_incr p})\ \{\lambda\_.\ p \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,L)\}$$

Exercise: proof sketch for in-place increment.

Verification of in-place increment: frame rule

I.H.: $\forall L p.\quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{list\_incr p})\ \{\lambda\_.\ p \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,L)\}$

$\{p \rightsquigarrow \mathsf{MList}\,(x :: L')\}$

$\{p \mapsto (x, p') * p' \rightsquigarrow \mathsf{MList}\, L'\}$

```
p.hd <- p.hd + 1;
```

$\{p \mapsto (x+1, p') * p' \rightsquigarrow \mathsf{MList}\, L'\}$

```
list_incr p.tl
```

// by induction hypothesis and framing $p \mapsto (x+1, p')$

$\{p \mapsto (x+1, p') * p' \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,L')\}$

$\{p \rightsquigarrow \mathsf{MList}\,(x+1 :: \mathsf{map}\,(+1)\,L')\}$

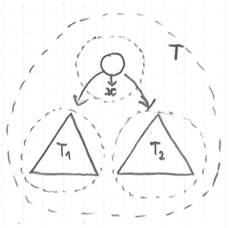
$\{p \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,(x :: L'))\}$

Verification of list_incr: rocq

Rocq: `list_incr_spec` – guess state of proof after applying IH

Specification of tree copy

```
let rec copy (p:node) : node =
  if p == null then null else
  let p1' = copy p.left in
  let p2' = copy p.right in
  { item = p.item;
    left = p1';
    right = p2' }
```

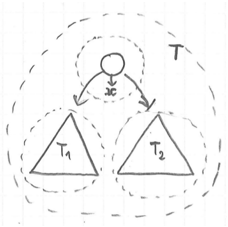


Exercise: specify the tree copy function.

Exercise: proof sketch for tree copy

Specification of tree copy

```
let rec copy (p:node) : node =
  if p == null then null else
  let p1' = copy p.left in
  let p2' = copy p.right in
  { item = p.item;
    left = p1';
    right = p2' }
```



Exercise: specify the tree copy function.

$$\forall pT. \quad \{p \rightsquigarrow \mathsf{Mtree}\, T\}\ (\texttt{copy p})\ \{\lambda p'.\ p \rightsquigarrow \mathsf{Mtree}\, T \ast p' \rightsquigarrow \mathsf{Mtree}\, T\}$$

Exercise: proof sketch for tree copy

Verification of tree copy: frame rule

$p \rightsquigarrow \mathsf{Mtree}\,T$	by pre-condition
$\underline{p \mapsto (x, p_1, p_2)} * p_1 \rightsquigarrow \mathsf{Mtree}\,T_1 \underline{* p_2 \rightsquigarrow \mathsf{Mtree}\,T_2}$	by unfolding
$p \mapsto \underline{(x, p_1, p_2)} * \underline{p_1 \rightsquigarrow \mathsf{Mtree}\,T_1} * p_2 \rightsquigarrow \mathsf{Mtree}\,T_2$ $* \underline{p'_1 \rightsquigarrow \mathsf{Mtree}\,T_1}$	 $\underline{\text{frame}}$+induction
$p \mapsto (x, p_1, p_2) * p_1 \rightsquigarrow \mathsf{Mtree}\,T_1 * p_2 \rightsquigarrow \mathsf{Mtree}\,T_2$ $* p'_1 \rightsquigarrow \mathsf{Mtree}\,T_1 * p'_2 \rightsquigarrow \mathsf{Mtree}\,T_2$	 $\underline{\text{frame}}$+induction
$p \mapsto (x, p_1, p_2) * p_1 \rightsquigarrow \mathsf{Mtree}\,T_1 * p_2 \rightsquigarrow \mathsf{Mtree}\,T_2$ $* p' \mapsto (x, p'_1, p'_2) * p'_1 \rightsquigarrow \mathsf{Mtree}\,T_1 * p'_2 \rightsquigarrow \mathsf{Mtree}\,T_2$	 by allocation
$p \rightsquigarrow \mathsf{Mtree}\,T$ $* p' \rightsquigarrow \mathsf{Mtree}\,T$	 by folding

Small footprint specifications

Small footprint access to records

$$p \mapsto (v, q) \;\equiv\; p \mapsto v * p + 1 \mapsto q$$

Specification of a write on the head field:

$$\{p \mapsto (w, q)\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \mapsto (v, q)\}$$

Small footprint access to records

$$p \mapsto (v, q) \;\equiv\; p \mapsto v \ast p+1 \mapsto q$$

Specification of a write on the head field:

$$\{p \mapsto (w, q)\}\,(p.\mathsf{hd} \texttt{ <- } v)\,\{\lambda\_.\ p \mapsto (v, q)\}$$

Same, but with a smaller footprint:

$$\{p \mapsto w\}\,(p.\mathsf{hd} \texttt{ <- } v)\ \{\lambda\_.\ p \mapsto v\}$$

$$\text{or}\quad \{p \mapsto -\}\ (p.\mathsf{hd} \texttt{ <- } v)\ \{\lambda\_.\ p \mapsto v\}$$

Small footprint access to records

$$p \mapsto (v, q) \;\equiv\; p \mapsto v \ast p + 1 \mapsto q$$

Specification of a write on the head field:

$$\{p \mapsto (w, q)\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \mapsto (v, q)\}$$

Same, but with a smaller footprint:

$$\{p \mapsto w\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \mapsto v\}$$

or $\{p \mapsto -\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \mapsto v\}$

From small to large footprint using frame:

$$\frac{\{p \mapsto w\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \mapsto v\}}{\{p \mapsto w \ast p + 1 \mapsto q\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \mapsto v \ast p + 1 \mapsto q\}}\ \textsc{Frame}$$

Representation predicate for arrays

Representation predicate for C arrays:

$$p \rightsquigarrow \mathsf{Array}\, L \quad \equiv \quad \mathop{\Huge\ast}_{i=0}^{|L|-1} p + i \mapsto L[i]$$

Programmers know that `p[i]` is short for `*(p+i)`… and `i[p]` for `*(i+p)`, so the memory layout is transparent.

Representation predicate for arrays

Representation predicate for C arrays:

$$p \rightsquigarrow \mathsf{Array}\, L \quad \equiv \quad \mathop{\Large\ast}_{i=0}^{|L|-1} p+i \mapsto L[i]$$

Programmers know that `p[i]` is short for `*(p+i)`... and `i[p]` for `*(i+p)`, so the memory layout is transparent.

Representation predicate for ML arrays:

$$p \rightsquigarrow \mathsf{Array}\, L \quad \equiv \quad p.\mathsf{length} \mapsto |L| \ast \mathop{\Large\ast}_{i=0}^{|L|-1} p[i] \mapsto L[i]$$

where $p.\mathsf{length} \mapsto n$ and $p[i] \mapsto v$ are abstract definitions for the user. Memory safety/GC assume we respect abstraction barrier.

Small footprint specifications for C arrays

Small footprint specification for C array is the same as for any pointer, since `p[i]` is just `*(p + i)`

$$
\begin{array}{lll}
\{p \mapsto -\} & (\texttt{*p = v}) & \{\lambda\_.\ p \mapsto v\} \\
\{p \mapsto v\} & (\texttt{*p}) & \{\lambda x.\ \ulcorner x = v \urcorner * p \mapsto v\}
\end{array}
$$

large footprint specifications e.g. for array read/write are derivable

Small footprint specifications of ML array operations

Recall the large footprint specifications:

$$i \in \mathsf{dom}\, L \;\Rightarrow$$

$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{p.(i)})\ \{\lambda x.\ \ulcorner x = L[i] \urcorner * p \rightsquigarrow \mathsf{Array}\, L\}$$

$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{p.(i) <- v})\ \{\lambda \_.\ p \rightsquigarrow \mathsf{Array}\, (L[i := v])\}$$

$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{Array.length p})\ \{\lambda n.\ \ulcorner n = |L| \urcorner * p \rightsquigarrow \mathsf{Array}\, L\}$$

Small footprint specifications of ML array operations

Recall the large footprint specifications:

$$i \in \mathsf{dom}\, L \;\Rightarrow$$

$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{p.(i)})\ \{\lambda x.\ \ulcorner x = L[i] \urcorner * p \rightsquigarrow \mathsf{Array}\, L\}$$

$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{p.(i) <- v})\ \{\lambda \_.\ p \rightsquigarrow \mathsf{Array}\,(L[i := v])\}$$

$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{Array.length p})\ \{\lambda n.\ \ulcorner n = |L| \urcorner * p \rightsquigarrow \mathsf{Array}\, L\}$$

Small footprint specifications:

$$\begin{array}{lll}
\multicolumn{3}{l}{\forall p, i, v, n,} \\
\{p[i] \mapsto v\} & (\texttt{p.(i)}) & \{\lambda x.\ \ulcorner x = v \urcorner * p[i] \mapsto v\} \\
\{p[i] \mapsto -\} & (\texttt{p.(i) <- v}) & \{\lambda \_.\ p[i] \mapsto v\} \\
\{p.\mathsf{length} \mapsto n\} & (\texttt{Array.length p}) & \{\lambda x.\ \ulcorner x = n \urcorner * p.\mathsf{length} \mapsto n\}
\end{array}$$

Small footprint specifications of ML array operations

Recall the large footprint specifications:

$$
\begin{array}{l}
i \in \mathsf{dom}\, L \;\Rightarrow \\
\quad \{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{p.(i)})\ \{\lambda x.\ \ulcorner x = L[i] \urcorner \ast p \rightsquigarrow \mathsf{Array}\, L\} \\
\quad \{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{p.(i) <- v})\ \{\lambda \_.\ p \rightsquigarrow \mathsf{Array}\, (L[i := v])\} \\
\quad \{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{Array.length p})\ \{\lambda n.\ \ulcorner n = |L| \urcorner \ast p \rightsquigarrow \mathsf{Array}\, L\}
\end{array}
$$

Small footprint specifications:

$$
\begin{array}{lll}
\forall p, i, v, n, & & \\
\{p[i] \mapsto v\} & (\texttt{p.(i)}) & \{\lambda x.\ \ulcorner x = v \urcorner \ast p[i] \mapsto v\} \\
\{p[i] \mapsto -\} & (\texttt{p.(i) <- v}) & \{\lambda \_.\ p[i] \mapsto v\} \\
\{p.\mathsf{length} \mapsto n\} & (\texttt{Array.length p}) & \{\lambda x.\ \ulcorner x = n \urcorner \ast p.\mathsf{length} \mapsto n\}
\end{array}
$$

Derive large from small using frame and:

$$
\begin{array}{rcl}
p \rightsquigarrow \mathsf{Array}\, L & = & p[i] \mapsto L[i] \ast\ p.\mathsf{length} \mapsto |L| \\
& & \ast\ \mathop{\Large\ast}_{j=0, j \neq i}^{|L|-1} p[j] \mapsto L[j]
\end{array}
$$

Dynamic access of ML arrays

Dynamic checks in ocaml:

```
# let v = Array.make 5 0 in v.(7);;
```

Dynamic access of ML arrays

Dynamic checks in ocaml:

```
# let v = Array.make 5 0 in v.(7);;

Exception: Invalid_argument "index out of bounds".
```

This means preconditions must retain some header information.

Dynamic access of ML arrays

Dynamic checks in ocaml:

```
    # let v = Array.make 5 0 in v.(7);;

    Exception: Invalid_argument "index out of bounds".
```

This means preconditions must retain some header information.

When running programs proved in separation logic, one can disable those dynamic checks `ocamlopt -unsafe` and get faster code.

(Same story with `null`.)

Heap entailment

Heap entailment

Definition:

$$H_1 \rhd H_2 \quad \equiv \quad \forall m.\, H_1\, m \;\Rightarrow\; H_2\, m$$

For example:

$$(r \mapsto 6) \;\rhd\; \exists n.\, (r \mapsto n) * \ulcorner \text{even}\, n \urcorner$$

Thanks to $(\rhd)$, we never need to manipulate heaps explicitly.

Some rules:

$$\frac{}{H \rhd H}\ \rhd\text{-REFL} \qquad\qquad \frac{H_1 \rhd H_2 \qquad H_2 \rhd H_3}{H_1 \rhd H_3}\ \rhd\text{-TRANS}$$

Frame property for heap entailment

$$\frac{H_1 \ \rhd H_1'}{H_1 * H_2 \rhd H_1' * H_2}\ \text{ENTAIL-FRAME}$$

For example, to prove:

$$(r \mapsto 2) * (s \mapsto 3) \ \rhd \ (r \mapsto 2) * (t \mapsto n)$$

it suffices to prove:

$$(s \mapsto 3) \ \rhd \ (t \mapsto n).$$

Heap implications: true or false?

1. $(r \mapsto 3) * (s \mapsto 4) \ \rhd \ (s \mapsto 4) * (r \mapsto 3)$
2. $(r \mapsto 3) \ \rhd \ (s \mapsto 4) * (r \mapsto 3)$
3. $(s \mapsto 4) * (r \mapsto 3) \ \rhd \ (r \mapsto 4)$
4. $(s \mapsto 4) * (r \mapsto 3) \ \rhd \ (r \mapsto 3)$
5. $\ulcorner \mathsf{False} \urcorner * (r \mapsto 3) \ \rhd \ (s \mapsto 4) * (r \mapsto 4)$
6. $(r \mapsto 4) * (s \mapsto 3) \ \rhd \ \ulcorner \mathsf{False} \urcorner$
7. $(r \mapsto 4) * (r \mapsto 3) \ \rhd \ \ulcorner \mathsf{False} \urcorner$
8. $(r \mapsto 3) * (r \mapsto 3) \ \rhd \ \ulcorner \mathsf{False} \urcorner$

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	
5.	$\ulcorner \mathsf{False} \urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	false
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	
5.	$\ulcorner \mathsf{False} \urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	false
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	false
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	
5.	$\ulcorner\text{False}\urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner\text{False}\urcorner$	
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner\text{False}\urcorner$	
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner\text{False}\urcorner$	

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	false
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	false
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	false
5.	$\ulcorner \mathsf{False} \urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	false
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	false
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	false
5.	$\ulcorner \text{False} \urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	true
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner \text{False} \urcorner$	
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner \text{False} \urcorner$	
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner \text{False} \urcorner$	

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	false
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	false
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	false
5.	$\ulcorner \mathsf{False} \urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	true
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	false
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	false
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	false
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	false
5.	$\ulcorner \mathsf{False} \urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	true
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	false
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	true
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	false
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	false
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	false
5.	$\ulcorner \mathsf{False} \urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	true
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	false
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	true
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner \mathsf{False} \urcorner$	true

Heap implications: true or false?

1.	$(r \mapsto 3) * (s \mapsto 4) \rhd (s \mapsto 4) * (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 3)$	false
3.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 4)$	false
4.	$(s \mapsto 4) * (r \mapsto 3) \rhd (r \mapsto 3)$	false
5.	$\ulcorner \text{False} \urcorner * (r \mapsto 3) \rhd (s \mapsto 4) * (r \mapsto 4)$	true
6.	$(r \mapsto 4) * (s \mapsto 3) \rhd \ulcorner \text{False} \urcorner$	false
7.	$(r \mapsto 4) * (r \mapsto 3) \rhd \ulcorner \text{False} \urcorner$	true
8.	$(r \mapsto 3) * (r \mapsto 3) \rhd \ulcorner \text{False} \urcorner$	true

(4 helps ensure absence of memory leaks in some cases)

Instantiation of existentials and propositions

$$(r \mapsto 6) \rhd (\exists n.\, (r \mapsto n) * \ulcorner \mathsf{even}\, n \urcorner)$$

To prove the above, we exhibit an even number n for which $r \mapsto n$.

Rules:

$$\frac{H_1 \rhd H_2[v/x]}{H_1 \rhd (\exists x.\, H_2)}\ \textsc{exists-r} \qquad \frac{(H_1 \rhd H_2) \qquad P}{H_1 \rhd (H_2 * \ulcorner P \urcorner)}\ \textsc{prop-r}$$

Instantiation of existentials and propositions

$$(r \mapsto 6) \rhd (\exists n. (r \mapsto n) * \ulcorner \text{even}\, n \urcorner)$$

To prove the above, we exhibit an even number n for which $r \mapsto n$.

Rules:

$$\frac{H_1 \rhd H_2[v/x]}{H_1 \rhd (\exists x. H_2)}\ \textsc{exists-r} \qquad \frac{(H_1 \rhd H_2) \qquad P}{H_1 \rhd (H_2 * \ulcorner P \urcorner)}\ \textsc{prop-r}$$

Example:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(r \mapsto 6) \rhd (r \mapsto 6)}\ \textsc{refl} \qquad \dfrac{}{\text{even}\, 6}\ \textsc{math}}{(r \mapsto 6) \rhd (r \mapsto 6) * \ulcorner \text{even}\, 6 \urcorner}\ \textsc{prop-r}}{(r \mapsto 6) \rhd ((r \mapsto n) * \ulcorner \text{even}\, n \urcorner)\, [6/n]}\ \textsc{subst}}{(r \mapsto 6) \rhd \exists n. (r \mapsto n) * \ulcorner \text{even}\, n \urcorner}\ \textsc{exists-r}$$

Extraction of existentials and propositions

$$(\exists n. \ulcorner \mathsf{even}\, n \urcorner * (r \mapsto n)) \vartriangleright (\exists m. \ulcorner \mathsf{even}\, m \urcorner * (r \mapsto m + 2))$$

To prove the above, we show that for any even number n, we have:

$$(r \mapsto n) \vartriangleright \exists m. \ulcorner \mathsf{even}\, m \urcorner * (r \mapsto m + 2)$$

Extraction of existentials and propositions

$$(\exists n.\, \ulcorner \mathsf{even}\, n \urcorner * (r \mapsto n)) \;\rhd\; (\exists m.\, \ulcorner \mathsf{even}\, m \urcorner * (r \mapsto m + 2))$$

To prove the above, we show that for any even number n, we have:

$$(r \mapsto n) \;\rhd\; \exists m.\, \ulcorner \mathsf{even}\, m \urcorner * (r \mapsto m + 2)$$

Reasoning rules:

$$\frac{x \notin H_2 \qquad \forall x.\; (H_1 \rhd H_2)}{(\exists x.\, H_1) \rhd H_2}\;\textsc{exists-l} \qquad\qquad \frac{P \;\Rightarrow\; (H_1 \rhd H_2)}{(\ulcorner P \urcorner * H_1) \rhd H_2}\;\textsc{prop-l}$$

Heap implications: true or false?

1. $(r \mapsto 3) \rhd \exists n.\,(r \mapsto n)$
2. $\exists n.\,(r \mapsto n) \rhd (r \mapsto 3)$
3. $\exists n.\,(r \mapsto n) * \ulcorner n > 0 \urcorner \rhd \exists n.\,\ulcorner n > 1 \urcorner * (r \mapsto (n-1))$
4. $(r \mapsto 3) * (s \mapsto 3) \rhd \exists n.\,(r \mapsto n) * (s \mapsto n)$
5. $\exists n.\,(r \mapsto n) * \ulcorner n > 0 \urcorner * \ulcorner n < 0 \urcorner \rhd (r \mapsto m) * (r \mapsto m)$

Heap implications: true or false?

1. $(r \mapsto 3) \rhd \exists n.\, (r \mapsto n)$ true
2. $\exists n.\, (r \mapsto n) \rhd (r \mapsto 3)$
3. $\exists n.\, (r \mapsto n) * \ulcorner n > 0 \urcorner \rhd \exists n.\, \ulcorner n > 1 \urcorner * (r \mapsto (n-1))$
4. $(r \mapsto 3) * (s \mapsto 3) \rhd \exists n.\, (r \mapsto n) * (s \mapsto n)$
5. $\exists n.\, (r \mapsto n) * \ulcorner n > 0 \urcorner * \ulcorner n < 0 \urcorner \rhd (r \mapsto m) * (r \mapsto m)$

Heap implications: true or false?

1.	$(r \mapsto 3) \rhd \exists n.\, (r \mapsto n)$	true
2.	$\exists n.\, (r \mapsto n) \rhd (r \mapsto 3)$	false
3.	$\exists n.\, (r \mapsto n) * \ulcorner n > 0 \urcorner \rhd \exists n.\, \ulcorner n > 1 \urcorner * (r \mapsto (n-1))$	
4.	$(r \mapsto 3) * (s \mapsto 3) \rhd \exists n.\, (r \mapsto n) * (s \mapsto n)$	
5.	$\exists n.\, (r \mapsto n) * \ulcorner n > 0 \urcorner * \ulcorner n < 0 \urcorner \rhd (r \mapsto m) * (r \mapsto m)$	

Heap implications: true or false?

1.	$(r \mapsto 3) \rhd \exists n. (r \mapsto n)$	true
2.	$\exists n. (r \mapsto n) \rhd (r \mapsto 3)$	false
3.	$\exists n. (r \mapsto n) * \ulcorner n > 0 \urcorner \rhd \exists n. \ulcorner n > 1 \urcorner * (r \mapsto (n - 1))$	true
4.	$(r \mapsto 3) * (s \mapsto 3) \rhd \exists n. (r \mapsto n) * (s \mapsto n)$	
5.	$\exists n. (r \mapsto n) * \ulcorner n > 0 \urcorner * \ulcorner n < 0 \urcorner \rhd (r \mapsto m) * (r \mapsto m)$	

Heap implications: true or false?

1.	$(r \mapsto 3) \rhd \exists n. (r \mapsto n)$	true
2.	$\exists n. (r \mapsto n) \rhd (r \mapsto 3)$	false
3.	$\exists n. (r \mapsto n) * \ulcorner n > 0 \urcorner \rhd \exists n. \ulcorner n > 1 \urcorner * (r \mapsto (n-1))$	true
4.	$(r \mapsto 3) * (s \mapsto 3) \rhd \exists n. (r \mapsto n) * (s \mapsto n)$	true
5.	$\exists n. (r \mapsto n) * \ulcorner n > 0 \urcorner * \ulcorner n < 0 \urcorner \rhd (r \mapsto m) * (r \mapsto m)$	

Heap implications: true or false?

1.	$(r \mapsto 3) \rhd \exists n.\, (r \mapsto n)$	true
2.	$\exists n.\, (r \mapsto n) \rhd (r \mapsto 3)$	false
3.	$\exists n.\, (r \mapsto n) * \ulcorner n > 0 \urcorner \rhd \exists n.\, \ulcorner n > 1 \urcorner * (r \mapsto (n-1))$	true
4.	$(r \mapsto 3) * (s \mapsto 3) \rhd \exists n.\, (r \mapsto n) * (s \mapsto n)$	true
5.	$\exists n.\, (r \mapsto n) * \ulcorner n > 0 \urcorner * \ulcorner n < 0 \urcorner \rhd (r \mapsto m) * (r \mapsto m)$	true

Proving heap entailment relations

Systematic approach to dealing with heap entailment:

1. extract from left hand side,
2. instantiate in right hand side,
3. cancel equal predicates on both sides.

Example:

$$\begin{array}{c}
\hline
a : \mathsf{int},\ a > 5 \ \vdash\ (r \mapsto 3) * (s \mapsto a) \ \rhd\ (r \mapsto 3) * (s \mapsto a) \\
\hline
a : \mathsf{int},\ a > 5 \ \vdash\ (r \mapsto 3) * (s \mapsto a) \ \rhd\ (r \mapsto 3) * (s \mapsto 3 + (a - 3)) \\
\hline
a : \mathsf{int},\ a > 5 \ \vdash\ (r \mapsto 3) * (s \mapsto a) \ \rhd\ \exists m.\,(r \mapsto 3) * (s \mapsto 3 + m) \\
\hline
a : \mathsf{int},\ a > 5 \ \vdash\ (r \mapsto 3) * (s \mapsto a) \ \rhd\ \exists nm.\,(r \mapsto n) * (s \mapsto n + m) \\
\hline
\varnothing \ \vdash\ \exists a.\,\ulcorner a > 5 \urcorner * (r \mapsto 3) * (s \mapsto a) \ \rhd\ \exists nm.\,(r \mapsto n) * (s \mapsto n + m) \\
\hline
\varnothing \ \vdash\ (r \mapsto 3) * \exists a.\,\ulcorner a > 5 \urcorner * (s \mapsto a) \ \rhd\ \exists nm.\,(s \mapsto n + m) * (r \mapsto n)
\end{array}$$

Structural rules

Frame rule

$$\frac{\{H_1\}\ t\ \{\lambda x.\, H_1'\}}{\{H_1 * H_2\}\ t\ \{\lambda x.\, H_1' * H_2\}}$$

Reformulation:

$$\frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 * H_2\}\ t\ \{Q_1 * H_2\}}\ \textsc{frame}$$

with the overloading:

$$Q * H \ \equiv\ \lambda x.\ (Q\, x * H)$$

Consequence rule

$$\frac{H \rhd H' \qquad \{H'\}\; t\; \{Q'\} \qquad Q' \rhd Q}{\{H\}\; t\; \{Q\}}\ \textsc{consequence}$$

with the overloading:

$$Q' \rhd Q \;\equiv\; \forall x.\, (Q'\, x \rhd Q\, x)$$

Consequence rule

$$\dfrac{H \rhd H' \qquad \{H'\}\; t\; \{Q'\} \qquad Q' \rhd Q}{\{H\}\; t\; \{Q\}}\ \text{CONSEQUENCE}$$

with the overloading:

$$Q' \rhd Q \;\equiv\; \forall x.\, (Q'\, x \rhd Q\, x)$$

Note that H and H' must cover the same set of memory cells, that is, no garbage collection is allowed here. Similarly for Q and Q'.

Recall the need for garbage collection

```
let myref x =
  let r = ref x in
  let s = ref r in
  r
```

From:

$$\{\ulcorner\urcorner\}\ (\texttt{myref x})\ \{\lambda r.\ r \mapsto x * \exists s.\, s \mapsto r\}$$

To:

$$\{\ulcorner\urcorner\}\ (\texttt{myref x})\ \{\lambda r.\ r \mapsto x\}$$

Recall the need for garbage collection

```
let myref x =
  let r = ref x in
  let s = ref r in
  r
```

From:

$$\{\ulcorner\urcorner\}\ (\texttt{myref x})\ \{\lambda r.\ r \mapsto x * \exists s.\, s \mapsto r\}$$

To:

$$\{\ulcorner\urcorner\}\ (\texttt{myref x})\ \{\lambda r.\ r \mapsto x\}$$

Can the following rule be used by choosing $H' = \exists s.\, s \mapsto r$?

$$\frac{\{H\}\ t\ \{Q * H'\}}{\{H\}\ t\ \{Q\}}\ \text{GC-POST'}$$

Garbage collection rules

Two rules: recall $\mathsf{GC} \equiv \exists H'.\, H'$

$$\frac{\{H\}\ t\ \{Q\}}{\{H * \mathsf{GC}\}\ t\ \{Q\}}\ \textsc{gc-pre} \qquad\qquad \frac{\{H\}\ t\ \{Q * \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post}$$

Garbage collection rules

Two rules: recall GC $\equiv \exists H'.\, H'$

$$\frac{\{H\}\ t\ \{Q\}}{\{H * \mathsf{GC}\}\ t\ \{Q\}}\ \textsc{gc-pre} \qquad \frac{\{H\}\ t\ \{Q * \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post}$$

Remarks:

- GC-PRE is derivable from GC-POST and FRAME (**Exercise**)
- no analog in Iris, where by default $P * Q \vdash P$.

Garbage collection rules

Two rules: recall GC $\equiv \exists H'.\, H'$

$$\frac{\{H\}\ t\ \{Q\}}{\{H * \mathsf{GC}\}\ t\ \{Q\}}\ \textsc{gc-pre} \qquad \frac{\{H\}\ t\ \{Q * \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post}$$

Remarks:

- $\textsc{gc-pre}$ is derivable from $\textsc{gc-post}$ and $\textsc{frame}$ (**Exercise**)
- no analog in Iris, where by default $P * Q \vdash P$.

Consequences:

$$\frac{\{H\}\ t\ \{Q\}}{\{H * H'\}\ t\ \{Q\}} \qquad \frac{\{H\}\ t\ \{\lambda x.\, Q\, x * H'\}}{\{H\}\ t\ \{Q\}}$$

In $\exists H'.\, H'$, the choice of H' may depend on the return value x. For $(\lambda r.\ r \mapsto x\ *\ \exists s.\, s \mapsto r)$, we may instantiate H' as $(\exists s.\, s \mapsto r)$.

Extraction of existentials and propositions

$$\{\exists n.\,(r \mapsto n) * \ulcorner \mathsf{even}\, n \urcorner\}\ (\mathtt{!r})\ \{\lambda x.\, ...\}$$

To prove the above, we need to show that:

$$\forall n.\ \mathsf{even}\, n \ \Rightarrow\ \{r \mapsto n\}\ (\mathtt{!r})\ \{\lambda x.\, ...\}$$

Rules:

$$\frac{x \notin t, Q \qquad \forall x.\ \{H\}\ t\ \{Q\}}{\{\exists x.\, H\}\ t\ \{Q\}}\ \textsc{exists} \qquad\qquad \frac{P\ \Rightarrow\ \{H\}\ t\ \{Q\}}{\{\ulcorner P \urcorner * H\}\ t\ \{Q\}}\ \textsc{prop}$$

Extraction of existentials and propositions

$$\{\exists n.\, (r \mapsto n) * \ulcorner \mathsf{even}\, n \urcorner\}\ (\texttt{!r})\ \{\lambda x. \ldots\}$$

To prove the above, we need to show that:

$$\forall n.\ \mathsf{even}\, n \ \Rightarrow\ \{r \mapsto n\}\ (\texttt{!r})\ \{\lambda x. \ldots\}$$

Rules:

$$\frac{x \notin t, Q \qquad \forall x.\ \{H\}\ t\ \{Q\}}{\{\exists x.\, H\}\ t\ \{Q\}}\ \textsc{exists} \qquad\qquad \frac{P \ \Rightarrow\ \{H\}\ t\ \{Q\}}{\{\ulcorner P \urcorner * H\}\ t\ \{Q\}}\ \textsc{prop}$$

Remark: why no rule introducing of $\exists x.$ and $\ulcorner P \urcorner$ in the postcondition?

Application: copying a tree with invariants

Specification of copy for binary trees:

$$\{p \rightsquigarrow \mathsf{Mtree}\, T\}\ (\texttt{copy p})\ \{\lambda p'.\ p \rightsquigarrow \mathsf{Mtree}\, T \ast p' \rightsquigarrow \mathsf{Mtree}\, T\}$$

Description of complete binary trees:

$$p \rightsquigarrow \mathsf{MtreeComplete}\, T \ \equiv\ \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \ast \ulcorner \mathsf{depth}\, n\, T \urcorner$$

Exercise: give a specification of `copy` in terms of MtreeComplete; which rules are used to derive this specification?

Application: copying a tree with invariants

Specification of copy for binary trees:

$$\{p \rightsquigarrow \mathsf{Mtree}\, T\}\ (\texttt{copy p})\ \{\lambda p'.\ p \rightsquigarrow \mathsf{Mtree}\, T \ast p' \rightsquigarrow \mathsf{Mtree}\, T\}$$

Description of complete binary trees:

$$p \rightsquigarrow \mathsf{MtreeComplete}\, T \ \equiv\ \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \ast \ulcorner \mathsf{depth}\, n\, T \urcorner$$

Exercise: give a specification of copy in terms of MtreeComplete; which rules are used to derive this specification?

$$\{p \rightsquigarrow \mathsf{MtreeComplete}\, T\}\ (\texttt{copy p})\ \{\lambda p'.\ \begin{array}{l} p \rightsquigarrow \mathsf{MtreeComplete}\, T \\ \ast\, p' \rightsquigarrow \mathsf{MtreeComplete}\, T \end{array}\}$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

$$\begin{array}{l}
\{\exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \ast \ulcorner \mathsf{depth}\, n\, T \urcorner\} \\
(\texttt{copy p}) \\
\{\lambda p'.\quad \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \ast \ulcorner \mathsf{depth}\, n\, T \urcorner \quad \} \\
\qquad\quad \ast\, \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\, T) \ast \ulcorner \mathsf{depth}\, n\, T \urcorner
\end{array}$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

$$\begin{array}{l}\{\exists n.\ (p \rightsquigarrow \mathsf{Mtree}\,T) * \ulcorner\mathsf{depth}\,n\,T\urcorner\} \\ (\texttt{copy p}) \\ \{\lambda p'.\quad \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\,T) * \ulcorner\mathsf{depth}\,n\,T\urcorner \quad\} \\ \qquad\quad * \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\,T) * \ulcorner\mathsf{depth}\,n\,T\urcorner\end{array}$$

(2) By the EXISTS and PROP rules:

$$\forall n.\ \mathsf{depth}\,n\,T \Rightarrow \begin{array}{l}\{p \rightsquigarrow \mathsf{Mtree}\,T\} \\ (\texttt{copy p}) \\ \{\lambda p'.\quad \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\,T) * \ulcorner\mathsf{depth}\,n\,T\urcorner \quad\} \\ \qquad\quad * \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\,T) * \ulcorner\mathsf{depth}\,n\,T\urcorner\end{array}$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

$$\begin{array}{l}\{\exists n.\ (p \rightsquigarrow \mathsf{Mtree}\,T) \ast \ulcorner \mathsf{depth}\,n\,T \urcorner\}\\ (\texttt{copy p})\\ \{\lambda p'.\quad \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\,T) \ast \ulcorner \mathsf{depth}\,n\,T \urcorner \quad \}\\ \qquad\quad \ast\ \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\,T) \ast \ulcorner \mathsf{depth}\,n\,T \urcorner\end{array}$$

(2) By the EXISTS and PROP rules:

$$\begin{array}{ll}\forall n.\ \mathsf{depth}\,n\,T \Rightarrow & \{p \rightsquigarrow \mathsf{Mtree}\,T\}\\ & (\texttt{copy p})\\ & \{\lambda p'.\quad \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\,T) \ast \ulcorner \mathsf{depth}\,n\,T \urcorner \quad \}\\ & \qquad\quad \ast\ \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\,T) \ast \ulcorner \mathsf{depth}\,n\,T \urcorner\end{array}$$

(3) By the CONSEQUENCE rule:

$$\begin{array}{ll}p \rightsquigarrow \mathsf{Mtree}\,T \ast p' \rightsquigarrow \mathsf{Mtree}\,T \rhd & \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\,T) \ast \ulcorner \mathsf{depth}\,n\,T \urcorner\\ & \ast\ \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\,T) \ast \ulcorner \mathsf{depth}\,n\,T \urcorner\end{array}$$

(4) Conclude using comm., assoc., extrusion, and EXISTS-R and PROP-R.

Proof of the derived specification

Rocq: `mtree.v` – use of exist/prop rules

Summary

Structural rules:

$$\frac{H \rhd H' \qquad \{H'\}\ t\ \{Q'\} \qquad Q' \rhd Q}{\{H\}\ t\ \{Q\}}\ \text{CONSEQUENCE}$$

$$\frac{\{H\}\ t\ \{Q * \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \text{GC-POST} \qquad \frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 * H_2\}\ t\ \{Q_1 * H_2\}}\ \text{FRAME}$$

$$\frac{\forall x.\ \{H\}\ t\ \{Q\}}{\{\exists x.\ H\}\ t\ \{Q\}}\ \text{EXISTS} \qquad \frac{P \Rightarrow \{H\}\ t\ \{Q\}}{\{\ulcorner P \urcorner * H\}\ t\ \{Q\}}\ \text{PROP}$$

Other structural rules are derivable.

Reasoning rules for terms — or break?

Reasoning rule for sequences

Example:

$$\frac{\{r \mapsto n\}\ (\texttt{incr r})\ \{\lambda\_.\, r \mapsto n+1\} \qquad \{r \mapsto n+1\}\ (\texttt{!r})\ \{\lambda x.\, \ulcorner x = n+1 \urcorner * r \mapsto n+1\}}{\{r \mapsto n\}\ (\texttt{incr r; !r})\ \{\lambda x.\, \ulcorner x = n+1 \urcorner * r \mapsto n+1\}}$$

Reasoning rule for sequences

Example:

$$\frac{\{r \mapsto n\}\ (\texttt{incr r})\ \{\lambda\_.\, r \mapsto n+1\} \qquad \{r \mapsto n+1\}\ (\texttt{!r})\ \{\lambda x.\, \ulcorner x = n+1 \urcorner * r \mapsto n+1\}}{\{r \mapsto n\}\ (\texttt{incr r; !r})\ \{\lambda x.\, \ulcorner x = n+1 \urcorner * r \mapsto n+1\}}$$

Exercise: complete the rule for sequences.

$$\frac{\{...\}\ t_1\ \{...\} \qquad \{...\}\ t_2\ \{...\}}{\{H\}\ (t_1\,;\,t_2)\ \{Q\}}$$

Reasoning rule for sequences

Solution 1:

$$\frac{\{H\}\ t_1\ \{\lambda\_.\, H'\} \qquad \{H'\}\ t_2\ \{Q\}}{\{H\}\ (t_1\,;\,t_2)\ \{Q\}}$$

Solution 2:

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

Remark: $Q' = \lambda\_.\, H'$ is equivalent to $Q'\,() = H'$.

Reasoning rule for let-bindings

Exercise: complete the reasoning rule for let-bindings.

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

Reasoning rule for let-bindings

Exercise: complete the reasoning rule for let-bindings.

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

Solution:

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

Example of let-binding

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

Exercise: instantiate the rule for let-bindings on the following code.

$$\{r \mapsto 3\}\ \big(\texttt{let a = !r in a+1}\big)\ \{Q\}$$

Example of let-binding

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

Exercise: instantiate the rule for let-bindings on the following code.

$$\{r \mapsto 3\}\ (\texttt{let a = !r in a+1})\ \{Q\}$$

Solution:

$$\begin{array}{lcl} H & \equiv & (r \mapsto 3) \\ Q & \equiv & \lambda x.\ \ulcorner x = 4 \urcorner * (r \mapsto 3) \\ Q' & \equiv & \lambda y.\ \ulcorner y = 3 \urcorner * (r \mapsto 3) \end{array}$$

Bind rule

Logics often have **bind rule** in order to focus on an expression:

$$\frac{\{H\}\ e\ \{Q'\} \qquad \forall v\ \{Q'\,v\}\ C[v]\ \{Q\} \qquad C \text{ is an evaluation context}}{\{H\}\ C[e]\ \{Q\}}\ \textsc{Bind}$$

it is like applying *let expansion*:

$$C[e] \quad \approx \quad \texttt{let x} = e\ \texttt{in}\ C[x]$$

then applying the let rule.

A term that has been fully “let-expanded” is said to be in *A-normal form*

Reasoning rule for values

Example:

$$\{\ulcorner\urcorner\}\ 3\ \{\lambda x.\ \ulcorner x = 3 \urcorner\}$$

Rule:

$$\frac{}{\{\ulcorner\urcorner\}\ v\ \{\lambda x.\ \ulcorner x = v \urcorner\}}\ \text{VAL}$$

Exercise: state a reasoning rule for values using a heap implication.

$$\frac{\dots \rhd \dots}{\{H\}\ v\ \{Q\}}$$

Reasoning rule for values

Example:

$$\{\ulcorner\urcorner\}\ 3\ \{\lambda x.\, \ulcorner x = 3 \urcorner\}$$

Rule:

$$\frac{}{\{\ulcorner\urcorner\}\ v\ \{\lambda x.\, \ulcorner x = v \urcorner\}}\ \textsc{val}$$

Exercise: state a reasoning rule for values using a heap implication.

$$\frac{\ldots \rhd \ldots}{\{H\}\ v\ \{Q\}}$$

Solution:

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

Derivability of the val-frame rule

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

Proof:

$$\dfrac{\dfrac{\dfrac{}{\{\ulcorner\urcorner\}\ v\ \{\lambda x.\,\ulcorner x = v\urcorner\}}\ \textsc{val}}{\{H\}\ v\ \{\lambda x.\,\ulcorner x=v\urcorner * H\}}\ \textsc{frame} \qquad \dfrac{\dfrac{\dfrac{\dfrac{}{H \rhd Q\,v}\ \textsc{hypothesis}}{\forall x.\ x=v \Rightarrow (H \rhd Q\,x)}\ \textsc{subst}}{\forall x.\ (\ulcorner x=v\urcorner * H) \rhd (Q\,x)}\ \textsc{prop-l}}{(\lambda x.\,\ulcorner x=v\urcorner * H) \rhd Q}\ \textsc{def of}\ \rhd}{\{H\}\ v\ \{Q\}}\ \textsc{conseq}$$

Reasoning rule for conditionals

Rule:

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

Reasoning rule for conditionals

Rule:

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

When v is not a value, transformation to A-normal form:

$$(\mathsf{if}\, t_0\, \mathsf{then}\, t_1\, \mathsf{else}\, t_2) \quad \approx \quad (\mathsf{let}\, x = t_0\, \mathsf{in}\ (\mathsf{if}\, x\, \mathsf{then}\, t_1\, \mathsf{else}\, t_2))$$

Reasoning rule for functions

Rule:

$$\frac{v_1 \;=\; \lambda x.\, t \qquad \{H\}\; t[v_2/x]\; \{Q\}}{\{H\}\; (v_1\, v_2)\; \{Q\}}\;\textsc{fun}$$

Transformation to A-normal form if t_1 or t_2 is not a value:

$$(t_1\, t_2) \quad\approx\quad (\mathsf{let}\, f = t_1\, \mathsf{in}\, \mathsf{let}\, v = t_2\, \mathsf{in}\, (f\ v))$$

Reasoning rule for functions

Rule:

$$\frac{v_1 = \lambda x.\, t \qquad \{H\}\ t[v_2/x]\ \{Q\}}{\{H\}\ (v_1\, v_2)\ \{Q\}}\ \textsc{fun}$$

Transformation to A-normal form if t_1 or t_2 is not a value:

$$(t_1\, t_2) \quad \approx \quad (\mathsf{let}\ f = t_1\ \mathsf{in}\ \mathsf{let}\ v = t_2\ \mathsf{in}\ (f\ v))$$

Remark: in general we have $\{H\}\ t'\ \{Q\} \Leftrightarrow \{H\}\ t\ \{Q\}$ for pure deterministic reductions, i.e. if $t \rightarrow_{\mathsf{pure}} t'$, where

$$\frac{x \rightarrow y \qquad \forall z\ (x \rightarrow z) \Rightarrow y = z}{x \rightarrow_{\mathsf{det}} y} \qquad\qquad \frac{\forall m\ \langle t, m\rangle \rightarrow_{\mathsf{det}} \langle t', m\rangle}{t \rightarrow_{\mathsf{pure}} t'}$$

Verification of a simple function

```
let incr r =
  let a = !r in
  r := a+1
```

Specification:

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

Verification:
Fix r and n. We need to prove that the body satisfies the specification:

$$\{r \mapsto n\}\ (\texttt{let a = !r in r := a+1})\ \{\lambda\_.\ r \mapsto n+1\}$$

Verification of a simple function

```
let incr r =
  let a = !r in
  r := a+1
```

Specification:

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

Verification:
Fix r and n. We need to prove that the body satisfies the specification:

$$\{r \mapsto n\}\ (\texttt{let a = !r in r := a+1})\ \{\lambda\_.\ r \mapsto n+1\}$$

We conclude using the let-binding rule: $Q' \equiv \lambda x. \ulcorner x = n \urcorner * (r \mapsto n)$.

Reasoning rule for recursive functions

Rule:

$$\frac{v_1 \;=\; \mu f.\lambda x.t \qquad \{H\}\; t[v_1/f][v_2/x]\; \{Q\}}{\{H\}\; (v_1\, v_2)\; \{Q\}}\;\text{FIX}$$

Specification of recursive functions may be established by induction.

Reasoning rule for recursive functions

Rule:

$$\frac{v_1 \ = \ \mu f.\lambda x.t \qquad \{H\}\ t[v_1/f][v_2/x]\ \{Q\}}{\{H\}\ (v_1\, v_2)\ \{Q\}}\ \textsc{fix}$$

Specification of recursive functions may be established by induction.

Remark: again, $v_1 v_2 \to_{\mathsf{pure}} t[v_1/f][v_2/x]$

Verification of a recursive function

```
let rec mlength (p:'a cell) =
  if p == null
    then 0
    else let p' = p.tl in
         let n' = mlength p' in
         1 + n'
```

Specification:

$$\forall L p. \quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ \ulcorner n = |L| \urcorner * p \rightsquigarrow \mathsf{MList}\, L\}$$

Verification of a recursive function

```
let rec mlength (p:'a cell) =
  if p == null
    then 0
    else let p' = p.tl in
         let n' = mlength p' in
         1 + n'
```

Specification:

$$\forall L p. \quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ \ulcorner n = |L| \urcorner * p \rightsquigarrow \mathsf{MList}\, L\}$$

We prove this specification by induction on L.

Verification of mlength: nil case

Case $p = \textbf{null}$. Goal is:

$$p = \mathsf{null} \Rightarrow \{p \leadsto \mathsf{MList\,nil}\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * p \leadsto \mathsf{MList\,nil}\}$$

Verification of mlength: nil case

Case $p =$ **null**. Goal is:

$$p = \mathsf{null} \Rightarrow \{p \rightsquigarrow \mathsf{MList}\,\mathsf{nil}\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * p \rightsquigarrow \mathsf{MList}\,\mathsf{nil}\}$$

– Unfold to:

$$p = \mathsf{null} \Rightarrow \{\ulcorner p = \mathsf{null} \urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner p = \mathsf{null} \urcorner\}$$

Verification of mlength: nil case

Case $p =$ **null**. Goal is:

$$p = \mathsf{null} \Rightarrow \{p \rightsquigarrow \mathsf{MList}\,\mathsf{nil}\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}|\urcorner \ast p \rightsquigarrow \mathsf{MList}\,\mathsf{nil}\}$$

– Unfold to:

$$p = \mathsf{null} \Rightarrow \{\ulcorner p = \mathsf{null}\urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}|\urcorner \ast \ulcorner p = \mathsf{null}\urcorner\}$$

– Replace p with null.

$$\{\ulcorner \mathsf{null} = \mathsf{null}\urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}|\urcorner \ast \ulcorner \mathsf{null} = \mathsf{null}\urcorner\}$$

Verification of mlength: nil case

Case $p =$ **null**. Goal is:

$$p = \mathsf{null} \Rightarrow \{p \rightsquigarrow \mathsf{MList\,nil}\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * p \rightsquigarrow \mathsf{MList\,nil}\}$$

– Unfold to:

$$p = \mathsf{null} \Rightarrow \{\ulcorner p = \mathsf{null} \urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner p = \mathsf{null} \urcorner\}$$

– Replace p with null.

$$\{\ulcorner \mathsf{null} = \mathsf{null} \urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner \mathsf{null} = \mathsf{null} \urcorner\}$$

– Apply the VAL-FRAME rule.

$$\ulcorner \mathsf{null} = \mathsf{null} \urcorner \rhd \ulcorner 0 = |\mathsf{nil}| \urcorner * \ulcorner \mathsf{null} = \mathsf{null} \urcorner$$

Verification of mlength: nil case

Case $p = \textbf{null}$. Goal is:

$$p = \mathsf{null} \Rightarrow \{p \rightsquigarrow \mathsf{MList\,nil}\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * p \rightsquigarrow \mathsf{MList\,nil}\}$$

– Unfold to:

$$p = \mathsf{null} \Rightarrow \{\ulcorner p = \mathsf{null} \urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner p = \mathsf{null} \urcorner\}$$

– Replace p with null.

$$\{\ulcorner \mathsf{null} = \mathsf{null} \urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner \mathsf{null} = \mathsf{null} \urcorner\}$$

– Apply the VAL-FRAME rule.

$$\ulcorner \mathsf{null} = \mathsf{null} \urcorner \rhd \ulcorner 0 = |\mathsf{nil}| \urcorner * \ulcorner \mathsf{null} = \mathsf{null} \urcorner$$

Case $p \neq \textbf{null}$. Goal is unfolded to:

$$p \neq \mathsf{null} \Rightarrow \{\ulcorner p = \mathsf{null} \urcorner\}\ (\texttt{let p' = ...})\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner p = \mathsf{null} \urcorner\}$$

Verification of mlength: nil case

Case $p =$ **null**. Goal is:

$$p = \mathsf{null} \Rightarrow \{p \rightsquigarrow \mathsf{MList\,nil}\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * p \rightsquigarrow \mathsf{MList\,nil}\}$$

– Unfold to:

$$p = \mathsf{null} \Rightarrow \{\ulcorner p = \mathsf{null} \urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner p = \mathsf{null} \urcorner\}$$

– Replace p with null.

$$\{\ulcorner \mathsf{null} = \mathsf{null} \urcorner\}\ (0)\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner \mathsf{null} = \mathsf{null} \urcorner\}$$

– Apply the VAL-FRAME rule.

$$\ulcorner \mathsf{null} = \mathsf{null} \urcorner \rhd \ulcorner 0 = |\mathsf{nil}| \urcorner * \ulcorner \mathsf{null} = \mathsf{null} \urcorner$$

Case $p \neq$ **null**. Goal is unfolded to:

$$p \neq \mathsf{null} \Rightarrow \{\ulcorner p = \mathsf{null} \urcorner\}\ (\texttt{let p' = ...})\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner p = \mathsf{null} \urcorner\}$$

– Use PROP to get the tautology:

$$p \neq \mathsf{null} \Rightarrow p = \mathsf{null} \Rightarrow \{\}\ (\texttt{let p' = ...})\ \{\lambda n.\ \ulcorner n = |\mathsf{nil}| \urcorner * \ulcorner p = \mathsf{null} \urcorner\}$$

Verification of mlength: cons case, null case

Case $p =$ **null**. Goal is:

$$p = \mathsf{null} \Rightarrow \{p \rightsquigarrow \mathsf{MList}\,(x :: L')\}\ \ldots\ \{\ldots\}$$

Derive a contradiction from $\mathsf{null} \rightsquigarrow \mathsf{MList}\,(x :: L') \rhd \ulcorner\mathsf{False}\urcorner$ and the consequence rule and $\textsc{Prop}$

Verification of mlength: cons case (1/2)

Case $p \neq$ **null**. Goal is:

$$\{p \rightsquigarrow \mathsf{MList}\,(x :: L')\}$$
$$(\texttt{let p' = p.tl in let n' = mlength p' in 1 + n'})$$
$$\{\lambda n.\ \ulcorner n = |x :: L'| \urcorner * p \rightsquigarrow \mathsf{MList}\,(x :: L')\}$$

Verification of mlength: cons case (1/2)

Case $p \neq$ **null**. Goal is:

$$\{p \rightsquigarrow \mathsf{MList}\,(x :: L')\}$$
$$(\texttt{let p' = p.tl in let n' = mlength p' in 1 + n'})$$
$$\{\lambda n.\ \ulcorner n = |x :: L'| \urcorner * p \rightsquigarrow \mathsf{MList}\,(x :: L')\}$$

– unfold MList, use $\textsc{Exists}$ to introduce p' and *choose* p' in post:

$$\{p' \rightsquigarrow \mathsf{MList}\,L' * p \mapsto (x, p')\}$$
$$(\texttt{let p' = p.tl in let n' = mlength p' in 1 + n'})$$
$$\{\lambda n.\ \ulcorner n = |x :: L'| \urcorner * p' \rightsquigarrow \mathsf{MList}\,L' * p \mapsto (x, p')\}$$

Verification of mlength: cons case (1/2)

Case $p \neq$ **null**. Goal is:

$$\{p \rightsquigarrow \mathsf{MList}\,(x :: L')\}$$
$$(\texttt{let p' = p.tl in let n' = mlength p' in 1 + n'})$$
$$\{\lambda n.\ \ulcorner n = |x :: L'| \urcorner * p \rightsquigarrow \mathsf{MList}\,(x :: L')\}$$

– unfold MList, use $\textsc{Exists}$ to introduce p' and *choose* p' in post:

$$\{p' \rightsquigarrow \mathsf{MList}\,L' * p \mapsto (x, p')\}$$
$$(\texttt{let p' = p.tl in let n' = mlength p' in 1 + n'})$$
$$\{\lambda n.\ \ulcorner n = |x :: L'| \urcorner * p' \rightsquigarrow \mathsf{MList}\,L' * p \mapsto (x, p')\}$$

– Apply the let-binding rule, and the read rule. Remains:

$$\{p' \rightsquigarrow \mathsf{MList}\,L' * p \mapsto (x, p')\}$$
$$(\texttt{let n' = mlength p' in 1 + n'})$$
$$\{\lambda n.\ \ulcorner n = |L| \urcorner * p' \rightsquigarrow \mathsf{MList}\,L' * p \mapsto (x, p')\}$$

– Apply the frame rule to remove: $p \mapsto (x, p')$.

– Apply the let-binding rule with : $Q \equiv \lambda n'.\ulcorner n' = |L'|\urcorner * p' \rightsquigarrow \mathsf{MList}\,L'$.

Verification of mlength: cons case (2/2)

There remains to prove the two premises of the let-rule.

– First branch, exploit the induction hypothesis:

$$\{p' \leadsto \mathsf{MList}\, L'\}\ (\texttt{mlength p'})\ \{\lambda n'.\ \ulcorner n' = |L'| \urcorner * p' \leadsto \mathsf{MList}\, L'\}$$

– Second branch:

$$\{p' \leadsto \mathsf{MList}\, L' * \ulcorner n' = |L'| \urcorner\}\ (\texttt{1 + n'})\ \{\lambda n.\ \ulcorner n = |L| \urcorner * p' \leadsto \mathsf{MList}\, L'\}$$

– Apply the PROP rule and the VAL-FRAME rule.

$$n' = |L'| \quad \Rightarrow \quad p' \leadsto \mathsf{MList}\, L' \ \vartriangleright\ \ulcorner 1 + n' = |L| \urcorner * p' \leadsto \mathsf{MList}\, L'$$

– Cancel equal parts, conclude using $|L| = |x :: L'| = 1 + |L'| = 1 + n'$.

Verification of mlength: rocq

Rocq: `mlength_spec` – rules for terms.

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

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 $a \leqslant b$:

$$\frac{\forall i \in [a, b]. \quad \{I\, i\}\; t\; \{\lambda\_.\, I\, (i+1)\}}{\{I\, a\}\; (\mathsf{for}\, i = a\, \mathsf{to}\, b\, \mathsf{do}\, t)\; \{\lambda\_.\, I\, (b+1)\}}$$

Reasoning rule for for-loops

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

$$\frac{\forall i \in [a,b]. \quad \{I\, i\}\; t\; \{\lambda\_.\, I\,(i+1)\}}{\{I\, a\}\; (\mathsf{for}\, i = a\, \mathsf{to}\, b\, \mathsf{do}\, t)\; \{\lambda\_.\, I\,(b+1)\}}$$

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

$$\frac{\forall i \in [a,b]. \quad \{I\, i\}\; t\; \{\lambda\_.\, I\,(i+1)\}}{\{I\, a\}\; (\mathsf{for}\, i = a\, \mathsf{to}\, b\, \mathsf{do}\, t)\; \{\lambda\_.\, I\,(\mathsf{max}\, a\,(b+1))\}}$$

or just the special case

$$\frac{a > b}{\{P\}\; (\mathsf{for}\, i = a\, \mathsf{to}\, b\, \mathsf{do}\, t)\; \{\lambda\_.\, P\}}$$

Reasoning rule for while loops using invariants

In general, need two invariants (I : Hprop) and (J : bool → Hprop)

- I after t_2, before t_1
- J specifying condition t_1's result, before t_2

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

- for total correctness: parameterize the invariant with a measure.

Reasoning rule for while loops using induction

We focus on a different approach that:

- supports **total correctness** through the meta-logic;
- allows to apply the **frame rule** during iterations.

Reasoning rule for while loops using induction

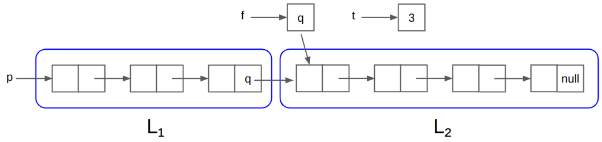
We focus on a different approach that:

- supports **total correctness** through the meta-logic;
- allows to apply the **frame rule** during iterations.

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

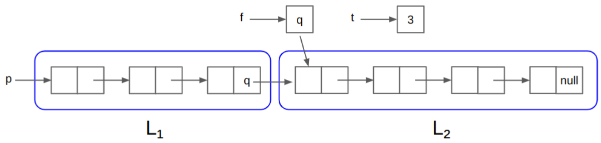
$$\frac{\{H\}\ (\mathsf{if}\, t_1\, \mathsf{then}\, (t_2\, ;\ (\mathsf{while}\, t_1\, \mathsf{do}\, t_2))\, \mathsf{else}\, ())\ \{Q\}}{\{H\}\ (\mathsf{while}\, t_1\, \mathsf{do}\, t_2)\ \{Q\}}$$

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 discard L_1 and prove by induction on L_2 that for all n and q:

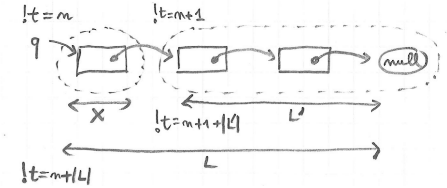
$$\{q \rightsquigarrow \mathsf{MList}\, L_2 \, * \, f \mapsto q \, * \, t \mapsto n\}$$
$$\big(\texttt{while !f != null do incr t; f := (!f).tl; done}\big)$$
$$\{q \rightsquigarrow \mathsf{MList}\, L_2 \, * \, f \mapsto \mathsf{null} \, * \, t \mapsto (n + \mathsf{length}\, L_2)\}$$

The loop unfolds to:

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

Exercise: give a proof sketch for the induction for `mlength`.

Length with a while loop: use of frame



$q \rightsquigarrow \mathsf{MList}\, L_2$	$* f \mapsto q$	$* t \mapsto n$	begin
$q \mapsto \{\mid x; q' \mid\} * q' \rightsquigarrow \mathsf{MList}\, L_2'$	$* f \mapsto q$	$* t \mapsto n$	unfold
$q \mapsto \{\mid x; q' \mid\} * q' \rightsquigarrow \mathsf{MList}\, L_2'$	$* f \mapsto q$	$* t \mapsto n + 1$	increment
$\underline{q \mapsto \{\mid x; q' \mid\}} * q' \rightsquigarrow \mathsf{MList}\, L_2'$	$* f \mapsto q'$	$* t \mapsto n + 1$	follow head
$q \mapsto \{\mid x; q' \mid\} * q' \rightsquigarrow \mathsf{MList}\, L_2'$	$* f \mapsto \mathsf{null}$	$* t \mapsto n + 1 + \lvert L_2' \rvert$	<u>frame</u>+Ind.hyp.
$q \rightsquigarrow \mathsf{MList}\, L_2$	$* f \mapsto \mathsf{null}$	$* t \mapsto n + \lvert L_2 \rvert$	fold

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:

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

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

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

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'.

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

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

Function repeat

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

Exercise: specify `repeat`, using an invariant I, of type int $\to$ 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.

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

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

Iteration over a pure (immutable) list

⚠ pure lists (`'a list`) $\neq$ mutable lists (`'a cell`) ⚠
They are allocated in memory but by design the language guarantees they behave like base values.

```
let rec iter (f : 'a -> unit) (l : 'a list) =
  match l with
  | [] -> ()
  | x::t -> f x; iter f t
```

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

Possible answers

Invariant on **past** elements:

$$\forall f l I \quad \frac{\forall x k\ \{I\,k\}\ (f\,x)\ \{\lambda\_.\ I\,(k\&x)\}}{\{I\ \mathsf{nil}\}\ \mathtt{iter}\,f\,l\ \{\lambda\_.\ I\,l\}}\ \textsc{Iter-past}$$

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

Invariant on **future** elements:

$$\forall f l I \quad \frac{\forall x k\ \{I\,(x :: k)\}\ (f\,x)\ \{\lambda\_.\ I\,k\}}{\{I\,l\}\ \mathtt{iter}\,f\,l\ \{\lambda\_.\ I\ \mathsf{nil}\}}\ \textsc{Iter-Future}$$

Possible answers

Invariant on **past** elements:

$$\forall f l I \quad \frac{\forall x k\ \{I\, k\}\ (f\, x)\ \{\lambda\_.\ I\, (k \& x)\}}{\{I\, \mathsf{nil}\}\ \texttt{iter}\, f\, l\ \{\lambda\_.\ I\, l\}}\ \textsc{Iter-past}$$

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

Invariant on **future** elements:

$$\forall f l I \quad \frac{\forall x k\ \{I\, (x :: k)\}\ (f\, x)\ \{\lambda\_.\ I\, k\}}{\{I\, l\}\ \texttt{iter}\, f\, l\ \{\lambda\_.\ I\, \mathsf{nil}\}}\ \textsc{Iter-Future}$$

Exercise: Give an example where $\textsc{Iter-Past}$ is insufficient

Exercise: Show $\textsc{Iter-Future}$ implies $\textsc{Iter-Past}$

The end!