Separation Logic 2/4

Jean-Marie Madiot

Inria Paris

January 27, 2021

slides (mostly) from Arthur Charguéraud

Chapter 7

The Frame Rule

Preservation of independent state

We have:

$$\{r \mapsto 2\}\ (\texttt{incr r})\ \{\lambda\_.\ r \mapsto 3\}$$

We also have:

$$\{r \mapsto 2 \star s \mapsto 7\}\ (\texttt{incr r})\ \{\lambda\_.\ r \mapsto 3 \star s \mapsto 7\}$$

Preservation of independent state

We have:

$$\{r \mapsto 2\}\ (\texttt{incr r})\ \{\lambda\_.\ r \mapsto 3\}$$

We also have:

$$\{r \mapsto 2 \star s \mapsto 7\}\ (\texttt{incr r})\ \{\lambda\_.\ r \mapsto 3 \star s \mapsto 7\}$$

More generally:

$$\{r \mapsto 2 \star H\}\ (\texttt{incr r})\ \{\lambda\_.\ r \mapsto 3 \star H\}$$

The frame rule

Principle: a triple remains valid when both the pre-condition and the post-condition are extended with a same heap predicate.

General form:

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

Frame rule and allocation

We have:

$$\{\,[\,]\,\}\ (\texttt{ref}\ \ 3)\ \{\lambda r.\ \ (r \mapsto 3)\}$$

By the frame rule, we have:

$$\{s \mapsto 5\}\ (\texttt{ref}\ \ 3)\ \{\lambda r.\ \ (r \mapsto 3) \star (s \mapsto 5)\}$$

Frame rule and allocation

We have:

$$\{\,[\,]\,\}\ (\texttt{ref}\ \ 3)\ \{\lambda r.\ \ (r \mapsto 3)\}$$

By the frame rule, we have:

$$\{s \mapsto 5\}\ (\texttt{ref}\ \ 3)\ \{\lambda r.\ \ (r \mapsto 3) \star (s \mapsto 5)\}$$

This post-condition ensures $r \neq s$.

The reference cell r is thus guaranteed to be distinct from any cell that might exist prior to the allocation of r.

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:

$$\forall pL.\quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ [n = \mathsf{length}\, L] \star p \rightsquigarrow \mathsf{MList}\, L\}$$

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:

$$\forall pL.\quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ [n = \mathsf{length}\, L] \star p \rightsquigarrow \mathsf{MList}\, L\}$$

We prove this specification by induction on L.

Verification of mlength: nil case

Case $L = \textbf{nil}$. Then $p = \mathsf{null}$. Goal is:

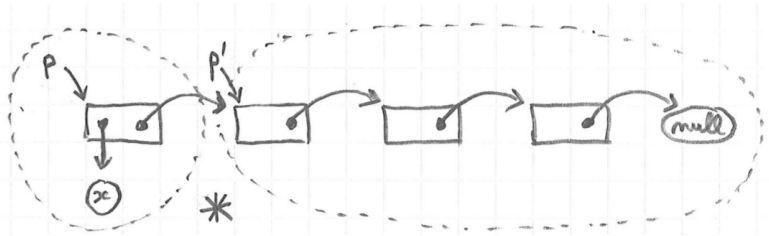
$$\{p \rightsquigarrow \mathsf{MList}\,\mathsf{nil}\}\ (0)\ \{\lambda n.\ [n = \mathsf{length}\,\mathsf{nil}] \star p \rightsquigarrow \mathsf{MList}\,\mathsf{nil}\}$$

Same as:

$$\{[p = \mathsf{null}]\}\ (0)\ \{\lambda n.\ [n = 0] \star [p = \mathsf{null}]\}$$

Verification of mlength: frame process

$\forall pL.\quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ [n = \mathsf{length}\, L] \star p \rightsquigarrow \mathsf{MList}\, L\}$



Assume $L = x :: L'$.

$p \rightsquigarrow \mathsf{MList}\, L$		pre-condition
$p \rightsquigarrow \{\!\|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'\|\!\}$	$\star\ p' \rightsquigarrow \mathsf{MList}\, L'$	by unfolding
	$p' \rightsquigarrow \mathsf{MList}\, L'$	frame begins
	$p' \rightsquigarrow \mathsf{MList}\, L' \star [n' = \lvert L' \rvert]$	by induction
$p \rightsquigarrow \{\!\|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'\|\!\}$	$\star\ p' \rightsquigarrow \mathsf{MList}\, L' \star [n' = \lvert L' \rvert]$	frame ends
$p \rightsquigarrow \mathsf{MList}\, L \star [n' + 1 = \lvert x :: L' \rvert]$		by folding
$p \rightsquigarrow \mathsf{MList}\, L \star [n = \lvert L \rvert]$		post-condition

Instantiation of the frame rule

Induction hypothesis:

$$\{p' \rightsquigarrow \mathsf{MList}\, L'\}$$
$$(\texttt{mlength p'})$$
$$\{\lambda n'.\ [n = \mathsf{length}\, L'] \star p' \rightsquigarrow \mathsf{MList}\, L'\}$$

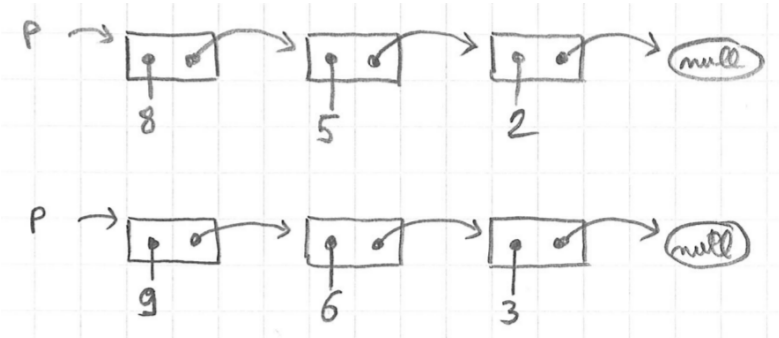
By the frame rule:

$$\{p' \rightsquigarrow \mathsf{MList}\, L' \star p \rightsquigarrow \{|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\}\}$$
$$(\texttt{mlength p'})$$
$$\{\lambda n.\ [n = \mathsf{length}\, L'] \star p' \rightsquigarrow \mathsf{MList}\, L' \star p \rightsquigarrow \{|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\}\}$$

Verification of mlength: coq

Coq demo.

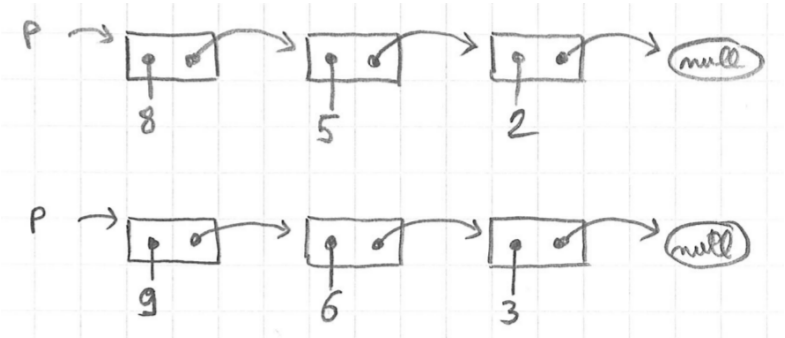
Verification of in-place increment




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

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

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
```

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

Exercise: describe the frame process for in-place increment.

Verification of in-place increment: frame process

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

Assume $L = x :: L'$.

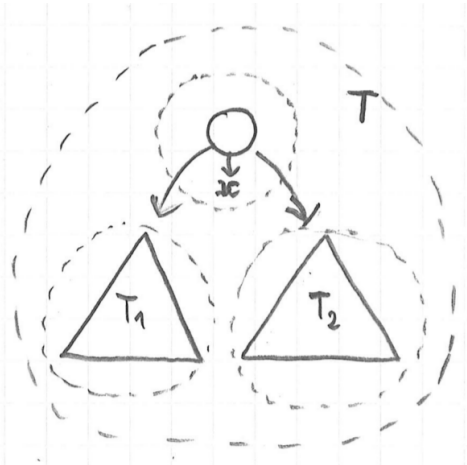
$p \rightsquigarrow \mathsf{MList}\, L$		pre-condition
$p \rightsquigarrow \{\!\lvert \mathsf{hd}{=}x;\ \mathsf{tl}{=}p' \rvert\!\}$	$\star\ p' \rightsquigarrow \mathsf{MList}\, L'$	by unfolding
$p \rightsquigarrow \{\!\lvert \mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p' \rvert\!\}$	$\star\ p' \rightsquigarrow \mathsf{MList}\, L'$	incrementing
	$p' \rightsquigarrow \mathsf{MList}\, L'$	frame begins
	$p' \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,L')$	by induction
$p \rightsquigarrow \{\!\lvert \mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p' \rvert\!\}$	$\star\ p' \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,L')$	frame ends
$p \rightsquigarrow \mathsf{MList}\,((x+1) :: (\mathsf{map}\,(+1)\,L'))$		by folding
$p \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,L)$		by rewriting
		post-condition

Verification of list_incr: coq

Coq.

Specification of tree copy

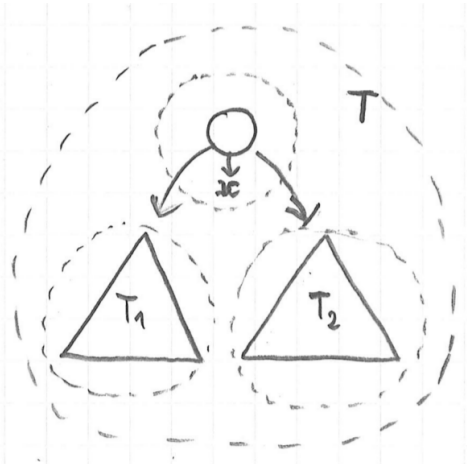
```
let rec copy (p:node) : node =
  if p == null then null else
  let p1' = copy p.left in
  let p2' = copy p.right in
  { item = p.item;
    left = p1';
    right = p2' }
```



Exercise: specify the tree copy function.

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' }
```

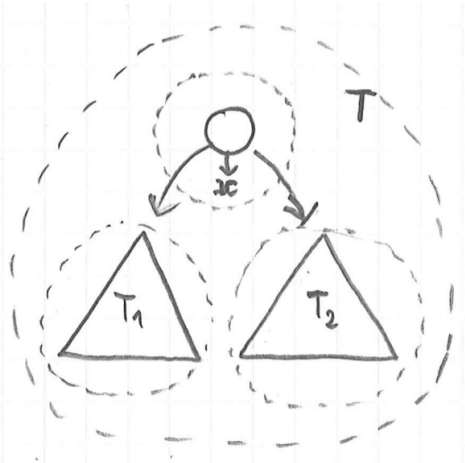


Exercise: specify the tree copy function.

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

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.

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

Exercise: describe the frame process for tree copy.

Verification of tree copy: frame process

$p \rightsquigarrow \mathsf{Mtree}\, T$	by pre-condition
$p \mapsto \{\!\|x; p_1; p_2\|\!\} \star p_1 \rightsquigarrow \mathsf{Mtree}\, T_1 \star p_2 \rightsquigarrow \mathsf{Mtree}\, T_2$	by unfolding
$p_1 \rightsquigarrow \mathsf{Mtree}\, T_1$	frame begins
$p_1 \rightsquigarrow \mathsf{Mtree}\, T_1$ $\star\ p_1' \rightsquigarrow \mathsf{Mtree}\, T_1$	by induction
$p \mapsto \{\!\|x; p_1; p_2\|\!\} \star p_1 \rightsquigarrow \mathsf{Mtree}\, T_1 \star p_2 \rightsquigarrow \mathsf{Mtree}\, T_2$ $\star\ p_1' \rightsquigarrow \mathsf{Mtree}\, T_1$	frame ends
$p_2 \rightsquigarrow \mathsf{Mtree}\, T_2$	frame begins
$p_2 \rightsquigarrow \mathsf{Mtree}\, T_2$ $\star\ p_2' \rightsquigarrow \mathsf{Mtree}\, T_2$	by induction
$p \mapsto \{\!\|x; p_1; p_2\|\!\} \star p_1 \rightsquigarrow \mathsf{Mtree}\, T_1 \star p_2 \rightsquigarrow \mathsf{Mtree}\, T_2$ $\star\ p_1' \rightsquigarrow \mathsf{Mtree}\, T_1 \star p_2' \rightsquigarrow \mathsf{Mtree}\, T_2$	frame ends
$p \mapsto \{\!\|x; p_1; p_2\|\!\} \star p_1 \rightsquigarrow \mathsf{Mtree}\, T_1 \star p_2 \rightsquigarrow \mathsf{Mtree}\, T_2$ $\star\ p' \mapsto \{\!\|x; p_1'; p_2'\|\!\} \star p_1' \rightsquigarrow \mathsf{Mtree}\, T_1 \star p_2' \rightsquigarrow \mathsf{Mtree}\, T_2$	by allocation
$p \rightsquigarrow \mathsf{Mtree}\, T$ $\star\ p' \rightsquigarrow \mathsf{Mtree}\, T$	by folding

Summary

$$\frac{\{H_1\}\ t\ \{\lambda x.\, H_1'\}}{\{H_1 \star H_2\}\ t\ \{\lambda x.\, H_1' \star H_2\}}\ \textsc{frame}$$

In-place mutable list increment, when $L = x :: L'$.

$p \rightsquigarrow \mathsf{MList}\, L$		
$p \rightsquigarrow \{\!\mid \mathsf{hd}{=}x;\ \mathsf{tl}{=}p' \mid\!\}$	$\star\ p' \rightsquigarrow \mathsf{MList}\, L'$	by unfolding
$p \rightsquigarrow \{\!\mid \mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p' \mid\!\}$	$\star\ p' \rightsquigarrow \mathsf{MList}\, L'$	incrementing
	$p' \rightsquigarrow \mathsf{MList}\, L'$	frame begins
	$p' \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L')$	by induction
$p \rightsquigarrow \{\!\mid \mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p' \mid\!\}$	$\star\ p' \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L')$	frame ends
$p \rightsquigarrow \mathsf{MList}\, ((x+1) :: (\mathsf{map}\, (+1)\, L'))$		by folding
$p \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L)$		by rewriting

Chapter 8

Small footprint specifications

Small footprint access to records

$$p \rightsquigarrow \{\!|\mathsf{hd}{=}v;\ \mathsf{tl}{=}q|\!\} \quad \equiv \quad p.\mathsf{hd} \mapsto v \ \star \ p.\mathsf{tl} \mapsto q$$

Specification of a write on the head field:

$$\{p \rightsquigarrow \{\!|\mathsf{hd}{=}w;\ \mathsf{tl}{=}q|\!\}\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \rightsquigarrow \{\!|\mathsf{hd}{=}v;\ \mathsf{tl}{=}q|\!\}\}$$

Small footprint access to records

$$p \rightsquigarrow \{\!|\mathsf{hd}{=}v;\ \mathsf{tl}{=}q|\!\} \quad \equiv \quad p.\mathsf{hd} \mapsto v \ \star\ p.\mathsf{tl} \mapsto q$$

Specification of a write on the head field:

$$\{p \rightsquigarrow \{\!|\mathsf{hd}{=}w;\ \mathsf{tl}{=}q|\!\}\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \rightsquigarrow \{\!|\mathsf{hd}{=}v;\ \mathsf{tl}{=}q|\!\}\}$$

Same, but with a smaller footprint:

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

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

Small footprint access to records

$$p \rightsquigarrow \{|\mathsf{hd}{=}v;\ \mathsf{tl}{=}q|\} \quad \equiv \quad p.\mathsf{hd} \mapsto v \ \star \ p.\mathsf{tl} \mapsto q$$

Specification of a write on the head field:

$$\{p \rightsquigarrow \{|\mathsf{hd}{=}w;\ \mathsf{tl}{=}q|\}\}\ (p.\mathsf{hd}\ \texttt{<-}\ v)\ \{\lambda\_.\ p \rightsquigarrow \{|\mathsf{hd}{=}v;\ \mathsf{tl}{=}q|\}\}$$

Same, but with a smaller footprint:

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

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

From small to large footprint using frame:

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

Representation predicate for arrays

Representation predicate for C arrays:

$$p \rightsquigarrow \mathsf{Array}\, L \quad \equiv \quad \circledast_{v \text{ at index } i \text{ in } L} \quad p[i] \mapsto v$$

where:

$$p[i] \mapsto v \quad \equiv \quad (p + i) \mapsto v$$

Representation predicate for arrays

Representation predicate for C arrays:

$$p \rightsquigarrow \mathsf{Array}\, L \quad \equiv \quad \circledast_{v \text{ at index } i \text{ in } L} \quad p[i] \mapsto v$$

where:

$$p[i] \mapsto v \quad \equiv \quad (p + i) \mapsto v$$

Representation predicate for ML arrays:

$$p \rightsquigarrow \mathsf{Array}\, L \quad \equiv \quad p.\mathsf{length} \mapsto |L| \star \circledast_{v \text{ at index } i \text{ in } L} \quad p[i] \mapsto v$$

where $p.\mathsf{length} \mapsto n$ and $p[i] \mapsto v$ are abstract definitions for the user.

Small footprint specifications of array operations

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

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

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

$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{Array.length p})\ \{\lambda n.\ [n = |L|] \star p \rightsquigarrow \mathsf{Array}\, L\}$$

Exercise: give small footprint specifications for array operations.
How to derive the large footprint specifications from them?

Small footprint specifications of array operations

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

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

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

$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{Array.length p})\ \{\lambda n.\ [n = |L|] \star p \rightsquigarrow \mathsf{Array}\, L\}$$

Exercise: give small footprint specifications for array operations.
How to derive the large footprint specifications from them?

$$\{p[i] \mapsto v\}\quad (\texttt{Array.get i p})\quad \{\lambda x.\ [x = v] \star p[i] \mapsto v\}$$

$$\{p[i] \mapsto -\}\quad (\texttt{Array.set i p v})\ \{\lambda \_.\ p[i] \mapsto v\}$$

$$\{p.\mathsf{length} \mapsto n\}\ (\texttt{Array.length p})\quad \{\lambda x.\ [x = n] \star p.\mathsf{length} \mapsto n\}$$

Small footprint specifications of array operations

$$i \in \mathsf{dom}\, L \Rightarrow$$
$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{Array.get i p})\ \{\lambda x.\ [x = L[i]] \star p \rightsquigarrow \mathsf{Array}\, L\}$$
$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{Array.set i p v})\ \{\lambda \_.\ p \rightsquigarrow \mathsf{Array}\, (L[i := v])\}$$
$$\{p \rightsquigarrow \mathsf{Array}\, L\}\ (\texttt{Array.length p})\ \{\lambda n.\ [n = |L|] \star p \rightsquigarrow \mathsf{Array}\, L\}$$

Exercise: give small footprint specifications for array operations.
How to derive the large footprint specifications from them?

$$\{p[i] \mapsto v\}\quad (\texttt{Array.get i p})\quad \{\lambda x.\ [x = v] \star p[i] \mapsto v\}$$
$$\{p[i] \mapsto -\}\quad (\texttt{Array.set i p v})\ \{\lambda \_.\ p[i] \mapsto v\}$$
$$\{p.\mathsf{length} \mapsto n\}\ (\texttt{Array.length p})\quad \{\lambda x.\ [x = n] \star p.\mathsf{length} \mapsto n\}$$

$$p \rightsquigarrow \mathsf{Array}\, L \;=\; p[i] \mapsto L[i] \star p.\mathsf{length} \mapsto |L| \star \circledast_{v \text{ at index } j \text{ in } L \text{ with } j \neq i} p[j] \mapsto v$$

Dynamic access of ML arrays

Dynamic checks in ocaml:

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

Dynamic access of ML arrays

Dynamic checks in ocaml:

```
# let v = Array.make 5 0 in Array.get 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 Array.get 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`.)

Access to a memory cell

In C, record and array accesses are treated uniformly:

`p->hd = v`	compiles to	`*(p+hd)=v`
`p[i] = v`	compiles to	`*(p+i)=v`

Access to a memory cell

In C, record and array accesses are treated uniformly:

`p->hd = v`	compiles to	`*(p+hd)=v`
`p[i] = v`	compiles to	`*(p+i)=v`
`i[p] = v`	compiles to	`*(i+p)=v` (...)

Access to a memory cell

In C, record and array accesses are treated uniformly:

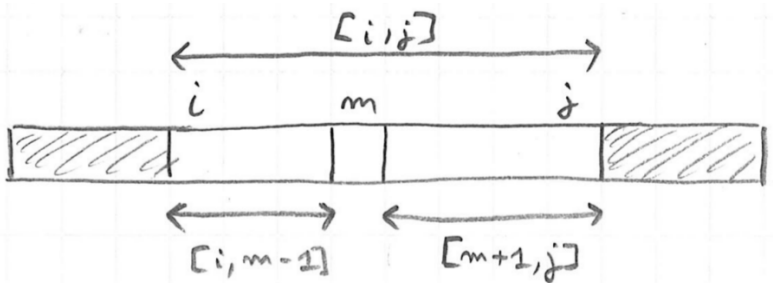
`p->hd = v`	compiles to	`*(p+hd)=v`
`p[i] = v`	compiles to	`*(p+i)=v`
`i[p] = v`	compiles to	`*(i+p)=v` (...)

Common small footprint specification for accessing a memory cell:

$$\{p \mapsto -\}\ (\texttt{*p = v})\ \{\lambda\_.\ p \mapsto v\}$$
$$\{p \mapsto v\}\ (\texttt{*p})\qquad \{\lambda x.\ [x = v] \star p \mapsto v\}$$

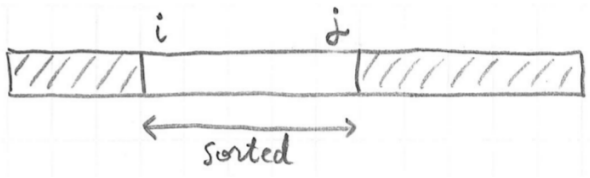
All other specifications for read and write operations are derivable.

Quicksort code




```
let rec qsort i j t =
  if j - i > 1 then begin
    let m = pivot i j t in
    qsort i (m-1) t;
    qsort (m+1) j t;
  end
```

Large-footprint specification of quicksort



$$\forall p L i j.\quad 0 \leqslant i \leqslant j < |L| \;\Rightarrow$$

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

$$(\texttt{qsort i j p})$$

$$\{\lambda\_.\ \exists L'.\quad (p \rightsquigarrow \mathsf{Array}\, L') \qquad \}$$

$$\star\, [\mathsf{sortedSeg}\, i\, j\, L']$$

$$\star\, [\mathsf{permut}\, L\, L']$$

$$\star\, [\forall a \in [0, i) \cup (j, |L|).\ L'[a] = L[a]]$$

where: $\mathsf{sortedSeg}\, i\, j\, L' \;\equiv\; \forall a, b \in [i, j].\ a \leqslant b \Rightarrow L'[a] \leqslant L'[b].$

Group of array cells

Definition

$$p \rightsquigarrow \mathsf{Cells}\, M \quad \equiv \quad \circledast_{(i,v)\in M} \; p[i] \mapsto v$$

where M has type "map int A", for some A.

Group of array cells

Definition

$$p \rightsquigarrow \mathsf{Cells}\, M \quad \equiv \quad \circledast_{(i,v)\in M} \; p[i] \mapsto v$$

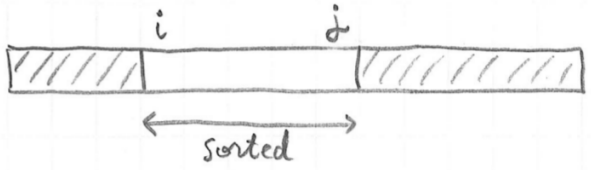
where M has type "map int A", for some A.

Conversions:

$$\begin{aligned}
p \rightsquigarrow \mathsf{Array}\, L &= p.\mathsf{length} \mapsto |L| \;\star\; p \rightsquigarrow \mathsf{Cells}\,(\mathsf{mapOfList}\, L) \\
p \rightsquigarrow \mathsf{Cells}\, \varnothing &= [\,] \\
p \rightsquigarrow \mathsf{Cells}\, \{(i,v)\} &= p[i] \mapsto v \\
p \rightsquigarrow \mathsf{Cells}\,(M_1 \uplus M_2) &= p \rightsquigarrow \mathsf{Cells}\, M_1 \;\star\; p \rightsquigarrow \mathsf{Cells}\, M_2 \qquad (\text{when } M_1 \perp M_2)
\end{aligned}$$

where: $\mathsf{mapOfList}\, L \equiv \{(i,v) \mid v \text{ at index } i \text{ in } L\}$.

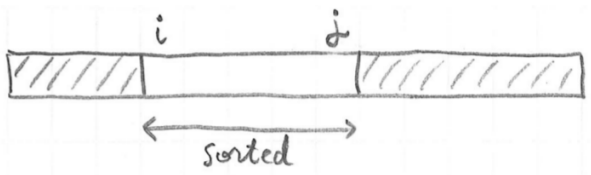
Small-footprint specification of quicksort



Exercise: give a small-footprint specification for quicksort.

- use $p \rightsquigarrow \mathsf{Cells}\, M$ with some constraints to describe a segment,
- use $\mathsf{sortedSeg}\, i\, j\, M$ to assert that a segment is sorted,
- use $\mathsf{permut}\, M\, M'$ to assert that values in a segment are permuted.

Small-footprint specification of quicksort

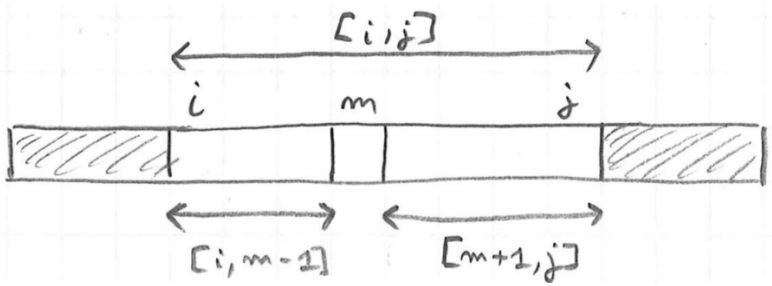


Exercise: give a small-footprint specification for quicksort.

- use $p \rightsquigarrow \mathsf{Cells}\, M$ with some constraints to describe a segment,
- use $\mathsf{sortedSeg}\, i\, j\, M$ to assert that a segment is sorted,
- use $\mathsf{permut}\, M\, M'$ to assert that values in a segment are permuted.

$$
\begin{array}{l}
\forall p M i j. \quad \mathsf{dom}\, M = [i, j] \;\Rightarrow \\
\quad \{p \rightsquigarrow \mathsf{Cells}\, M\} \\
\quad (\texttt{qsort i j p}) \\
\quad \{\lambda\_.\ \exists M'.\quad p \rightsquigarrow \mathsf{Cells}\, M' \qquad\quad \} \\
\qquad\qquad\qquad \star\, [\mathsf{sortedSeg}\, i\, j\, M'] \\
\qquad\qquad\qquad \star\, [\mathsf{permut}\, M\, M']
\end{array}
$$

Segment splitting in Quicksort



Assume dom $M = [i, j]$ with $j - i > 1$. Then:

$$p \rightsquigarrow \mathsf{Cells}\, M \;=\; p \rightsquigarrow \mathsf{Cells}\,(M_{|\,[i,m-1]}) \;\star\; p \rightsquigarrow \mathsf{Cells}\,(M_{|\,[m,m]}) \;\star\; p \rightsquigarrow \mathsf{Cells}\,(M_{|\,[m+1,j]})$$

Note that:

$$p \rightsquigarrow \mathsf{Cells}\,(M_{|\,[m,m]}) \;=\; p[m] \mapsto M[m]$$

In recursive calls, we frame over the cells that are not involved.

Operations on groups of cells

Convenient specifications for operating directly on groups of cells:

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

$$\{p \rightsquigarrow \mathsf{Cells}\, M\}\ (\texttt{Array.get i p})\ \{\lambda x.\ [x = M[i]] \star p \rightsquigarrow \mathsf{Cells}\, M\}$$

$$\{p \rightsquigarrow \mathsf{Cells}\, M\}\ (\texttt{Array.set i p v})\ \{\lambda \_.\ p \rightsquigarrow \mathsf{Cells}\, (M[i := v])\}$$

Summary

Representation of a full array using a list:

$$p \rightsquigarrow \mathsf{Array}\, L \quad \equiv \quad p.\mathsf{length} \mapsto |L| \star \circledast_{v \text{ at index } i \text{ in } L} p[i] \mapsto v$$

Small footprint accesses:

$$\begin{array}{lll} \{p[i] \mapsto v\} & (\texttt{Array.get i p}) & \{\lambda\_.\ p[i] \mapsto v\} \\ \{p[i] \mapsto -\} & (\texttt{Array.set i p v}) & \{\lambda x.\ [x = v] \star p[i] \mapsto v\} \\ \{p.\mathsf{length} \mapsto n\} & (\texttt{Array.length p}) & \{\lambda x.\ [x = n] \star p.\mathsf{length} \mapsto n\} \end{array}$$

Representation of a set of array cells using a finite map:

$$p \rightsquigarrow \mathsf{Cells}\, M \quad \equiv \quad \circledast_{(i,v) \in M} p[i] \mapsto v$$

Chapter 9

Heap entailment

Definition of Hprop

Let:

$$\text{Hprop} \quad \equiv \quad \texttt{heap} \rightarrow \texttt{Prop}$$

For example:

$$[\,] \quad : \quad \text{Hprop}$$

$$l \mapsto v \quad : \quad \text{Hprop}$$

$$H_1 \star H_2 \quad : \quad \text{Hprop}$$

In particular:

$$(\star) \quad : \quad \text{Hprop} \rightarrow \text{Hprop} \rightarrow \text{Hprop}$$

The separation algebra

Associativity: $$(H_1 \star H_2) \star H_3 \;=\; H_1 \star (H_2 \star H_3)$$

Commutativity: $$H_1 \star H_2 \;=\; H_2 \star H_1$$

Neutral element: $$H \star [\,] \;=\; H$$

Extrusion of existentials: $$(\exists x.\, H_1) \star H_2 \;=\; \exists x.\, (H_1 \star H_2) \qquad (\text{if } x \notin H_2)$$

Extrusion of pure facts: $$([P] \star H)\, m \;=\; P \;\wedge\; (H\, m)$$

Extensionality

Functional extensionality:

$$(\forall x.\, f\, x = g\, x) \;\Rightarrow\; (f = g) \qquad \text{(if } f,g{:}A \to B\text{)}$$

Propositional extensionality:

$$(P \Leftrightarrow Q) \;\Rightarrow\; (P = Q) \qquad \text{(if } P,Q{:}\texttt{Prop}\text{)}$$

Predicate extensionality:

$$(\forall x.\, P\, x \Leftrightarrow Q\, x) \;\Rightarrow\; (P = Q) \qquad \text{(if } P,Q{:}A \to \texttt{Prop}\text{)}$$

Heap predicate extensionality:

$$(\forall m.\, H\, m \Leftrightarrow H'\, m) \;\Leftrightarrow\; (H = H') \qquad \text{(if } H,H'{:}\mathsf{Hprop}\text{)}$$

Heap entailment

Definition:

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

For example:

$$(r \mapsto 6) \rhd \exists n.\, (r \mapsto n) \star [\mathsf{even}\, n]$$

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

Heap entailment as a partial order

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

The relation $(\rhd)$ defines a partial order on the type Hprop:

REFLEXIVITY

$$\frac{}{H \rhd H}$$

TRANSITIVITY

$$\frac{H_1 \rhd H_2 \qquad H_2 \rhd H_3}{H_1 \rhd H_3}$$

ANTISYMMETRY

$$\frac{H_1 \rhd H_2 \qquad H_2 \rhd H_1}{H_1 = H_2}$$

Frame property for heap entailment

$$\frac{H_1 \ \rhd \ H_1'}{H_1 \star H_2 \ \rhd \ H_1' \star H_2} \ \textsc{entail-frame}$$

For example, to prove:

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

it suffices to prove:

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

Heap implications: true or false?

1. $(r \mapsto 3) \star (s \mapsto 4) \rhd (s \mapsto 4) \star (r \mapsto 3)$
2. $(r \mapsto 3) \rhd (s \mapsto 4) \star (r \mapsto 3)$
3. $(s \mapsto 4) \star (r \mapsto 3) \rhd (r \mapsto 4)$
4. $(s \mapsto 4) \star (r \mapsto 3) \rhd (r \mapsto 3)$
5. $[\mathsf{False}] \star (r \mapsto 3) \rhd (s \mapsto 4) \star (r \mapsto 4)$
6. $(r \mapsto 4) \star (s \mapsto 3) \rhd [\mathsf{False}]$
7. $(r \mapsto 4) \star (r \mapsto 3) \rhd [\mathsf{False}]$
8. $(r \mapsto 3) \star (r \mapsto 3) \rhd [\mathsf{False}]$

Heap implications: true or false?

1. $(r \mapsto 3) \star (s \mapsto 4) \rhd (s \mapsto 4) \star (r \mapsto 3)$ — true
2. $(r \mapsto 3) \rhd (s \mapsto 4) \star (r \mapsto 3)$ — false
3. $(s \mapsto 4) \star (r \mapsto 3) \rhd (r \mapsto 4)$ — false
4. $(s \mapsto 4) \star (r \mapsto 3) \rhd (r \mapsto 3)$ — false
5. [False] $\star (r \mapsto 3) \rhd (s \mapsto 4) \star (r \mapsto 4)$
6. $(r \mapsto 4) \star (s \mapsto 3) \rhd$ [False]
7. $(r \mapsto 4) \star (r \mapsto 3) \rhd$ [False]
8. $(r \mapsto 3) \star (r \mapsto 3) \rhd$ [False]

Heap implications: true or false?

1.	$(r \mapsto 3) \star (s \mapsto 4) \rhd (s \mapsto 4) \star (r \mapsto 3)$	true
2.	$(r \mapsto 3) \rhd (s \mapsto 4) \star (r \mapsto 3)$	false
3.	$(s \mapsto 4) \star (r \mapsto 3) \rhd (r \mapsto 4)$	false
4.	$(s \mapsto 4) \star (r \mapsto 3) \rhd (r \mapsto 3)$	false
5.	$[\mathsf{False}] \star (r \mapsto 3) \rhd (s \mapsto 4) \star (r \mapsto 4)$	true
6.	$(r \mapsto 4) \star (s \mapsto 3) \rhd [\mathsf{False}]$	false
7.	$(r \mapsto 4) \star (r \mapsto 3) \rhd [\mathsf{False}]$	true
8.	$(r \mapsto 3) \star (r \mapsto 3) \rhd [\mathsf{False}]$	true

Instantiation of existentials and propositions

$$(r \mapsto 6) \rhd (\exists n.\, (r \mapsto n) \star [\mathsf{even}\, n])$$

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

Rules:

$$\frac{H_1 \rhd ([x \to v]\, H_2)}{H_1 \rhd (\exists x.\, H_2)}\ \textsc{exists-r} \qquad \frac{(H_1 \rhd H_2) \qquad P}{H_1 \rhd (H_2 \star [P])}\ \textsc{prop-r}$$

Instantiation of existentials and propositions

$$(r \mapsto 6) \rhd (\exists n.\, (r \mapsto n) \star [\mathsf{even}\, n])$$

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

Rules:

$$\frac{H_1 \rhd ([x \to v]\, H_2)}{H_1 \rhd (\exists x.\, H_2)}\ \textsc{exists-r} \qquad \frac{(H_1 \rhd H_2) \qquad P}{H_1 \rhd (H_2 \star [P])}\ \textsc{prop-r}$$

Example:

$$\dfrac{\dfrac{\dfrac{\dfrac{}{(r \mapsto 6) \rhd (r \mapsto 6)}\ \textsc{refl} \qquad \dfrac{}{\mathsf{even}\, 6}\ \textsc{math}}{(r \mapsto 6) \rhd (r \mapsto 6) \star [\mathsf{even}\, 6]}\ \textsc{prop-r}}{(r \mapsto 6) \rhd [n \to 6]\, ((r \mapsto n) \star [\mathsf{even}\, n])}\ \textsc{subst}}{(r \mapsto 6) \rhd \exists n.\, (r \mapsto n) \star [\mathsf{even}\, n]}\ \textsc{exists-r}$$

Extraction of existentials and propositions

$$(\exists n.\, [\mathsf{even}\, n] \star (r \mapsto n)) \rhd (\exists m.\, [\mathsf{even}\, m] \star (r \mapsto m + 2))$$

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

$$(r \mapsto n) \rhd \exists m.\, [\mathsf{even}\, m] \star (r \mapsto m + 2)$$

Extraction of existentials and propositions

$$(\exists n.\, [\mathsf{even}\, n] \star (r \mapsto n)) \rhd (\exists m.\, [\mathsf{even}\, m] \star (r \mapsto m+2))$$

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

$$(r \mapsto n) \rhd \exists m.\, [\mathsf{even}\, m] \star (r \mapsto m+2)$$

Reasoning rules:

$$\frac{\forall x.\ (H_1 \rhd H_2)}{(\exists x.\, H_1) \rhd H_2}\ \textsc{exists-l} \qquad \frac{P \Rightarrow (H_1 \rhd H_2)}{([P] \star H_1) \rhd H_2}\ \textsc{prop-l}$$

Extraction of existentials and propositions

$$(\exists n.\, [\mathsf{even}\, n] \star (r \mapsto n)) \;\rhd\; (\exists m.\, [\mathsf{even}\, m] \star (r \mapsto m+2))$$

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

$$(r \mapsto n) \;\rhd\; \exists m.\, [\mathsf{even}\, m] \star (r \mapsto m+2)$$

Reasoning rules:

$$\frac{\forall x.\;\; (H_1 \rhd H_2)}{(\exists x.\, H_1) \rhd H_2}\;\textsc{exists-l} \qquad\qquad \frac{P \;\Rightarrow\; (H_1 \rhd H_2)}{([P] \star H_1) \rhd H_2}\;\textsc{prop-l}$$

Same with explicit proof contexts:

$$\frac{\Gamma,\; x : A \;\vdash\; H_1 \rhd H_2}{\Gamma \;\vdash\; (\exists (x : A).\, H_1) \rhd H_2}\;(x \notin H_2) \qquad\qquad \frac{\Gamma,\; h : P \;\vdash\; H_1 \rhd H_2}{\Gamma \;\vdash\; ([P] \star H_1) \rhd H_2}$$

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. $\exists n.\, (r \mapsto n) \star [n > 0] \rhd \exists n.\, [n > 1] \star (r \mapsto (n-1))$
4. $(r \mapsto 3) \star (s \mapsto 3) \rhd \exists n.\, (r \mapsto n) \star (s \mapsto n)$
5. $\exists n.\, (r \mapsto n) \star [n > 0] \star [n < 0] \rhd (r \mapsto n) \star (r \mapsto n)$

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. $\exists n.\, (r \mapsto n) \star [n > 0] \rhd \exists n.\, [n > 1] \star (r \mapsto (n-1))$ — true
4. $(r \mapsto 3) \star (s \mapsto 3) \rhd \exists n.\, (r \mapsto n) \star (s \mapsto n)$ — true
5. $\exists n.\, (r \mapsto n) \star [n > 0] \star [n < 0] \rhd (r \mapsto n) \star (r \mapsto n)$ — 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:

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{}{a : \mathsf{int},\ a > 5 \ \vdash\ (r \mapsto 3) \star (s \mapsto a) \ \rhd\ (r \mapsto 3) \star (s \mapsto a)}}{a : \mathsf{int},\ a > 5 \ \vdash\ (r \mapsto 3) \star (s \mapsto a) \ \rhd\ (r \mapsto 3) \star (s \mapsto 3 + (a - 3))}}{a : \mathsf{int},\ a > 5 \ \vdash\ (r \mapsto 3) \star (s \mapsto a) \ \rhd\ \exists m.\, (r \mapsto 3) \star (s \mapsto 3 + m)}}{a : \mathsf{int},\ a > 5 \ \vdash\ (r \mapsto 3) \star (s \mapsto a) \ \rhd\ \exists nm.\, (r \mapsto n) \star (s \mapsto n + m)}}{\varnothing \ \vdash\ \exists a.\, [a > 5] \star (r \mapsto 3) \star (s \mapsto a) \ \rhd\ \exists nm.\, (r \mapsto n) \star (s \mapsto n + m)}}{\varnothing \ \vdash\ (r \mapsto 3) \star \exists a.\, [a > 5] \star (s \mapsto a) \ \rhd\ \exists nm.\, (s \mapsto n + m) \star (r \mapsto n)}$$

Summary

$(\star)$ is associative, commutative, and has $[\,]$ as neutral element.

Existentials and pure facts may be extruded from stars.

$(\rhd)$ is a partial order, satisfying the frame property.

"[False] $\rhd H$" is always true.

"$(r \mapsto n) \star (r \mapsto m)$" is equivalent to "[False]".

Strategy: extract from the left, instantiate on the right, then cancel out.

Chapter 10

Structural rules

Frame rule

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

Reformulation:

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

with the overloading:

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

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

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

with the overloading:

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

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:

$$\{[\,]\}\ (\texttt{myref x})\ \{\lambda r.\ r \mapsto x \star \exists s.\, s \mapsto r\}$$

To:

$$\{[\,]\}\ (\texttt{myref x})\ \{\lambda r.\ r \mapsto x\}$$

Recall the need for garbage collection

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

From:

$$\{[\,]\}\ (\texttt{myref x})\ \{\lambda r.\ r \mapsto x \star \exists s.\, s \mapsto r\}$$

To:

$$\{[\,]\}\ (\texttt{myref x})\ \{\lambda r.\ r \mapsto x\}$$

Can the following rule be used?

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

Garbage collection rule

$$\frac{\{H\}\ t\ \{Q \star \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post} \qquad \text{where: } \mathsf{GC} \equiv \exists H'.\, H'$$

Same as:

$$\frac{\{H\}\ t\ \{\lambda x.\ (Q\, x \star \exists H'.\, H')\}}{\{H\}\ t\ \{Q\}}$$

Observe that H' may depend on the return value x:
For $(\lambda r.\ r \mapsto x \star \exists s.\, s \mapsto r)$, we may instantiate H' as $(\exists s.\, s \mapsto r)$.

Garbage collection in the pre-condition

$$\frac{\{H\}\ t\ \{Q\}}{\{H \star \mathsf{GC}\}\ t\ \{Q\}}\ \textsc{gc-pre} \qquad \frac{\{H\}\ t\ \{Q\}}{\{H \star H'\}\ t\ \{Q\}}\ \textsc{gc-pre'}$$

Exercise: show that GC-PRE is derivable from GC-POST and FRAME.

$$\frac{\{H\}\ t\ \{Q \star \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post} \qquad \frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 \star H_2\}\ t\ \{Q_1 \star H_2\}}\ \textsc{frame}$$

Garbage collection in the pre-condition

$$\frac{\{H\}\ t\ \{Q\}}{\{H \star \mathsf{GC}\}\ t\ \{Q\}}\ \textsc{gc-pre} \qquad \frac{\{H\}\ t\ \{Q\}}{\{H \star H'\}\ t\ \{Q\}}\ \textsc{gc-pre'}$$

Exercise: show that GC-PRE is derivable from GC-POST and FRAME.

$$\frac{\{H\}\ t\ \{Q \star \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post} \qquad \frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 \star H_2\}\ t\ \{Q_1 \star H_2\}}\ \textsc{frame}$$

Proof:

$$\dfrac{\dfrac{\{H\}\ t\ \{Q\}}{\{H \star \mathsf{GC}\}\ t\ \{Q \star \mathsf{GC}\}}\ \textsc{frame}}{\{H \star \mathsf{GC}\}\ t\ \{Q\}}\ \textsc{gc-post}$$

Combined structural rule

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

$$\frac{\{H\}\ t\ \{Q \star \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post} \qquad \frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 \star H_2\}\ t\ \{Q_1 \star H_2\}}\ \textsc{frame}$$

Combined structural rule

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

$$\frac{\{H\}\ t\ \{Q \star \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post} \qquad \frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 \star H_2\}\ t\ \{Q_1 \star H_2\}}\ \textsc{frame}$$

$$\frac{H = H_1 \star H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 = Q}{\{H\}\ t\ \{Q\}}\ \textsc{frame'}$$

Combined structural rule

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

$$\frac{\{H\}\ t\ \{Q \star \mathsf{GC}\}}{\{H\}\ t\ \{Q\}}\ \textsc{gc-post} \qquad \frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 \star H_2\}\ t\ \{Q_1 \star H_2\}}\ \textsc{frame}$$

$$\frac{H = H_1 \star H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 = Q}{\{H\}\ t\ \{Q\}}\ \textsc{frame'}$$

$$\frac{H \rhd H_1 \star H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 \rhd Q \star \mathsf{GC}}{\{H\}\ t\ \{Q\}}\ \textsc{combined}$$

Extraction of existentials and propositions

$$\{\exists n.\, (r \mapsto n) \star [\mathsf{even}\, n]\}\ (\texttt{!r})\ \{\lambda x. \ldots\}$$

To prove the above, we need to show that:

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

Rules:

$$\frac{\forall x.\ \{H\}\ t\ \{Q\}}{\{\exists x.\, H\}\ t\ \{Q\}}\ \textsc{exists} \qquad \frac{P \Rightarrow \{H\}\ t\ \{Q\}}{\{[P] \star H\}\ t\ \{Q\}}\ \textsc{prop}$$

Application: copying a tree with invariants

Specification of copy for binary trees:

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

Description of complete binary trees:

$$p \rightsquigarrow \mathsf{MtreeComplete}\, T \ \equiv\ \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]$$

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:

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

Description of complete binary trees:

$$p \rightsquigarrow \mathsf{MtreeComplete}\, T \ \equiv\ \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]$$

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

$$\{p \rightsquigarrow \mathsf{MtreeComplete}\, T\}\ (\texttt{copy}\ \ \texttt{p})\ \{\lambda p'.\quad p \rightsquigarrow \mathsf{MtreeComplete}\, T\ \ \}$$
$$\star\ p' \rightsquigarrow \mathsf{MtreeComplete}\, T$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

$$\{\exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]\}$$
$$(\texttt{copy p})$$
$$\{\lambda p'.\quad \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T] \quad\}$$
$$\star\ \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

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

(2) By the EXISTS and PROP rules:

$$\begin{array}{lll}\forall n.\ \mathsf{depth}\, n\, T \Rightarrow & \{p \rightsquigarrow \mathsf{Mtree}\, T\} \\ & (\texttt{copy p}) \\ & \{\lambda p'.\quad \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]\ \ \} \\ & \qquad \star\ \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]\end{array}$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

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

(2) By the EXISTS and PROP rules:

$$\begin{array}{lll}\forall n.\ \mathsf{depth}\, n\, T \Rightarrow & \{p \rightsquigarrow \mathsf{Mtree}\, T\} \\ & (\texttt{copy p}) \\ & \{\lambda p'.\quad \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]\ \ \} \\ & \qquad \star\ \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]\end{array}$$

(3) By the CONSEQUENCE rule:

$$\begin{array}{lll}p \rightsquigarrow \mathsf{Mtree}\, T \star p' \rightsquigarrow \mathsf{Mtree}\, T \rhd & \exists n.\ (p \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T] \\ & \star\ \exists n.\ (p' \rightsquigarrow \mathsf{Mtree}\, T) \star [\mathsf{depth}\, n\, T]\end{array}$$

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

Summary

Structural rules:

$$\frac{H \rhd H_1 \star H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 \rhd Q \star \mathsf{GC}}{\{H\}\ t\ \{Q\}}\ \textsc{combined}$$

$$\frac{\forall x.\ \{H\}\ t\ \{Q\}}{\{\exists x.\ H\}\ t\ \{Q\}}\ \textsc{exists} \qquad \frac{P \Rightarrow \{H\}\ t\ \{Q\}}{\{[P] \star H\}\ t\ \{Q\}}\ \textsc{prop}$$

Other structural rules are derivable.

Chapter 11

Reasoning rules for terms

Reasoning rule for sequences

Example:

$$\frac{\begin{array}{c}\{r \mapsto n\}\ (\texttt{incr r})\ \{\lambda\_.\, r \mapsto n+1\} \\ \{r \mapsto n+1\}\ (\texttt{!r})\ \{\lambda x.\, [x = n+1] \star r \mapsto n+1\}\end{array}}{\{r \mapsto n\}\ (\texttt{incr r; !r})\ \{\lambda x.\, [x = n+1] \star r \mapsto n+1\}}$$

Reasoning rule for sequences

Example:

$$\frac{\begin{array}{c}\{r \mapsto n\}\ (\texttt{incr r})\ \{\lambda\_.\, r \mapsto n+1\} \\ \{r \mapsto n+1\}\ (\texttt{!r})\ \{\lambda x.\, [x = n+1] \star r \mapsto n+1\}\end{array}}{\{r \mapsto n\}\ (\texttt{incr r; !r})\ \{\lambda x.\, [x = n+1] \star r \mapsto n+1\}}$$

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:

$$\frac{\{H\}\ t_1\ \{Q'\} \qquad \{Q'\ ()\}\ t_2\ \{Q\}}{\{H\}\ (t_1\ ;\ t_2)\ \{Q\}}\ \text{SEQ}$$

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

Reasoning rule for let-bindings

Exercise: complete the reasoning rule for let-bindings.

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

Reasoning rule for let-bindings

Exercise: complete the reasoning rule for let-bindings.

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

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.

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

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:

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

Reasoning rule for values

Example:

$$\{\,[\,]\,\}\ 3\ \{\lambda x.\,[x = 3]\}$$

Rule:

$$\frac{}{\{\,[\,]\,\}\ v\ \{\lambda x.\,[x = v]\}}\ \text{VAL}$$

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

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

Reasoning rule for values

Example:

$$\{\,[\,]\,\}\ 3\ \{\lambda x.\,[x = 3]\}$$

Rule:

$$\frac{}{\{\,[\,]\,\}\ v\ \{\lambda x.\,[x = v]\}}\ \textsc{val}$$

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

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

Proof:

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

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:

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

Transformation to A-normal form:

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

Reasoning rule for top-level functions

Rule:

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

Transformation to A-normal form:

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

Verification of a simple function

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

Specification:

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

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:

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

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: $Q' \equiv \lambda x.\, [x = n] \star (r \mapsto n)$.

Reasoning rule for top-level recursive functions

Rule:

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

Specification of recursive functions may be established by induction.

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:

$$\forall pL.\quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ [n = |L|] \star p \rightsquigarrow \mathsf{MList}\, L\}$$

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:

$$\forall pL.\quad \{p \rightsquigarrow \mathsf{MList}\, L\}\ (\texttt{mlength p})\ \{\lambda n.\ [n = |L|] \star p \rightsquigarrow \mathsf{MList}\, L\}$$

We prove this specification by induction on L.
Consider p and L. Apply the "if" rule.

Verification of mlength: nil case

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

$$\{p \rightsquigarrow \mathsf{MList}\, L\}\ (0)\ \{\lambda n.\ [n = |L|] \star p \rightsquigarrow \mathsf{MList}\, L\}$$

– Replace p with null.
– Rewrite null $\rightsquigarrow \mathsf{MList}\, L$ to $[L = \mathsf{nil}]$ in the pre and the post.
– By the PROP rule:

$$L = \mathsf{nil} \ \Rightarrow\ \{[\,]\}\ (0)\ \{\lambda n.\ [n = |L|] \star [L = \mathsf{nil}]\}$$

– Replace L with nil.

$$\{[\,]\}\ (0)\ \{\lambda n.\ [n = |\mathsf{nil}|] \star [\mathsf{nil} = \mathsf{nil}]\}$$

– Apply the VAL-FRAME rule.

$$[\,] \rhd [0 = 0] \star [\mathsf{nil} = \mathsf{nil}]$$

Verification of mlength: cons case (1/2)

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

$$\{p \rightsquigarrow \mathsf{MList}\, L\}$$
$$(\texttt{let p' = p.tl in let n' = mlength p' in 1 + n'})$$
$$\{\lambda n.\ [n = |L|] \star p \rightsquigarrow \mathsf{MList}\, L\}$$

– Unfold MList in pre and post, and decompose L as $x :: L'$:

$$p' \rightsquigarrow \mathsf{MList}\, L' \star p \rightsquigarrow \{|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\}$$

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

$$\{p' \rightsquigarrow \mathsf{MList}\, L' \star p \rightsquigarrow \{|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\}\ \}$$
$$(\texttt{let n' = mlength p' in 1 + n'})$$
$$\{\lambda n.\ [n = |L|] \star p' \rightsquigarrow \mathsf{MList}\, L' \star p \rightsquigarrow \{|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\}\ \}$$

– Apply the frame rule to remove: $p \rightsquigarrow \{|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\}$.
– Apply the let-binding rule with : $Q \equiv \lambda n'.\, [n' = |L|'] \star p' \rightsquigarrow \mathsf{MList}\, L'$.

Verification of mlength: cons case (2/2)

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

– First branch, exploit the induction hypothesis:

$$\{p' \rightsquigarrow \mathsf{MList}\, L'\}\ (\texttt{mlength p'})\ \{\lambda n'.\ [n = |L|'] \star p' \rightsquigarrow \mathsf{MList}\, L'\}$$

– Second branch:

$$\{p' \rightsquigarrow \mathsf{MList}\, L' \star [n' = |L|']\}\ (\texttt{1 + n'})\ \{\lambda n.\ [n = |L|] \star p' \rightsquigarrow \mathsf{MList}\, L'\}$$

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

$$n' = |L'| \quad \Rightarrow \quad p' \rightsquigarrow \mathsf{MList}\, L' \ \rhd\ [1 + n' = |L|] \star p' \rightsquigarrow \mathsf{MList}\, L'$$

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

Reasoning rule for local functions

Rule template:

$$\frac{\forall f.\ (...) \Rightarrow \{H\}\ t\ \{Q\}}{\{H\}\ (\mathsf{let\,rec}\ f\ x = body\ \mathsf{in}\ t)\ \{Q\}}$$

Hypothesis about f:

$$\forall x H' Q'.\ \{H'\}\ body\ \{Q'\} \Rightarrow \{H'\}\ (f\ x)\ \{Q'\}$$

Rule:

$$\frac{\begin{array}{c}\forall f.\ Pf \Rightarrow \{H\}\ t\ \{Q\} \\ Pf = \left(\forall x H' Q'.\ \{H'\}\ body\ \{Q'\} \Rightarrow \{H'\}\ (f\ x)\ \{Q'\}\right)\end{array}}{\{H\}\ (\mathsf{let\,rec}\ f\ x = body\ \mathsf{in}\ t)\ \{Q\}}\ \text{FIX}$$

Summary

$$\frac{}{\{\,[\,]\,\}\ v\ \{\lambda x.\,[x = v]\}}$$

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

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

$$\frac{\forall f.\ \left(\forall x H' Q'.\ \{H'\}\ t_1\ \{Q'\} \Rightarrow \{H'\}\ (f\,x)\ \{Q'\}\right) \Rightarrow \{H\}\ t_2\ \{Q\}}{\{H\}\ (\mathsf{let\,rec}\, f\, x = t_1\, \mathsf{in}\, t_2)\ \{Q\}}$$

Summary of Course 2

Summary of chapter 7

$$\frac{\{H_1\}\ t\ \{\lambda x.\, H_1'\}}{\{H_1 \star H_2\}\ t\ \{\lambda x.\, H_1' \star H_2\}}\ \text{FRAME}$$

In-place mutable list increment, when $L = x :: L'$.

$p \rightsquigarrow \mathsf{MList}\, L$			
$p \rightsquigarrow \{\!\mid \mathsf{hd}{=}x;\ \mathsf{tl}{=}p' \mid\!\}$	$\star$	$p' \rightsquigarrow \mathsf{MList}\, L'$	by unfolding
$p \rightsquigarrow \{\!\mid \mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p' \mid\!\}$	$\star$	$p' \rightsquigarrow \mathsf{MList}\, L'$	incrementing
		$p' \rightsquigarrow \mathsf{MList}\, L'$	frame begins
		$p' \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L')$	by induction
$p \rightsquigarrow \{\!\mid \mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p' \mid\!\}$	$\star$	$p' \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L')$	frame ends
$p \rightsquigarrow \mathsf{MList}\, ((x+1) :: (\mathsf{map}\, (+1)\, L'))$			by folding
$p \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L)$			by rewriting

Summary of chapter 8

Small footprint specification for C-style memory accesses:

$$\{p \mapsto w\}\ (\texttt{*p = v})\ \{\lambda\_.\ p \mapsto v\}$$
$$\{p \mapsto v\}\ (\texttt{*p})\qquad \{\lambda x.\ [x = v] \star p \mapsto v\}$$

Representation of a full array using a list:

$$p \rightsquigarrow \mathsf{Array}\, L \quad \equiv \quad p.\mathsf{length} \mapsto |L| \star \circledast_{v \text{ at index } i \text{ in } L} p[i] \mapsto v$$

Representation of a set of array cells using a finite map:

$$p \rightsquigarrow \mathsf{Cells}\, M \quad \equiv \quad \circledast_{(i,v) \in M} p[i] \mapsto v$$

Summary of chapter 9

$(\star)$ is associative, commutative, and has $[\,]$ as neutral element.

$(\rhd)$ is a partial order, regular w.r.t. $(\star)$.

"$[\text{False}] \rhd H$" is always true.

"$(r \mapsto n) \star (r \mapsto m)$" is equivalent to "[False]".

Strategy: extract from the left, instantiate on the right, then cancel out.

Summary of chapter 10

Structural rules:

$$\frac{H \rhd H_1 \star H_2 \qquad \{H_1\}\ t\ \{Q_1\} \qquad Q_1 \star H_2 \rhd Q \star \mathsf{GC}}{\{H\}\ t\ \{Q\}}\ \textsc{combined}$$

$$\frac{\forall x.\ \{H\}\ t\ \{Q\}}{\{\exists x.\ H\}\ t\ \{Q\}}\ \textsc{exists} \qquad \frac{P \Rightarrow \{H\}\ t\ \{Q\}}{\{[P] \star H\}\ t\ \{Q\}}\ \textsc{prop}$$

Other structural rules are derivable.

Summary of chapter 11

$$\frac{}{\{\,[\,]\,\}\ v\ \{\lambda x.\,[x = v]\}}$$

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

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

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

Exercises

- Exam from 2015, Exercise 2: Operations on binary search trees.

Available from the webpage of the course.

The end!