Separation Logic 2/4

Jean-Marie Madiot

Inria Paris

February 5, 2025

part 2/4 : initial material 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:

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

Preservation of independent state

We have:

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

We also have:

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

More generally:

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

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:

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

Frame rule and allocation

We have:

$$\{\ulcorner\urcorner\}\ (\texttt{ref 3})\ \{\lambda r.\ (r \mapsto 3)\}$$

By the frame rule, we have:

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

Frame rule and allocation

We have:

$$\{\ulcorner\urcorner\}\ (\texttt{ref 3})\ \{\lambda r.\ (r \mapsto 3)\}$$

By the frame rule, we have:

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

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:

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

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:

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

We prove this specification by induction on L.

Verification of mlength: nil case

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

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

Same as:

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

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

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

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

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

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

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

// by induction hypothesis and framing $p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\}$

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

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

$\{p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \star p' \rightsquigarrow \mathsf{MList}\,L' \star \ulcorner n' = |L'| \urcorner \star \ulcorner n = 1 + n' \urcorner\}$

$\{p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\} \star p' \rightsquigarrow \mathsf{MList}\,L' \star \ulcorner n = 1 + |L'| \urcorner\}$

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

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

Exercise!

Instantiation of the frame rule

Induction hypothesis:

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

By the frame rule:

$$\{p' \leadsto \mathsf{MList}\, L' \ast p \leadsto \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\}\}$$
$$\big(\texttt{mlength p'}\big)$$
$$\{\lambda n.\ \ulcorner n = \mathsf{length}\, L' \urcorner \ast p' \leadsto \mathsf{MList}\, L' \ast p \leadsto \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\}\}$$

Verification of mlength: rocq

Rocq: `mlength_spec`.

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
```

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

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

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

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

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

$\{p \rightsquigarrow \{\!|\mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p'|\!\} \star p' \rightsquigarrow \mathsf{MList}\, L'\}$

```
list_incr p.tl
```

// by induction hypothesis and framing $p \rightsquigarrow \{\!|\mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p'|\!\}$

$\{p \rightsquigarrow \{\!|\mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p'|\!\} \star p' \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,L')\}$

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

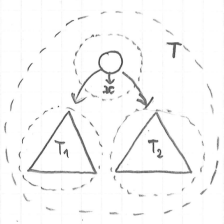
$\{p \rightsquigarrow \mathsf{MList}\,(\mathsf{map}\,(+1)\,(x :: L'))\}$

Verification of list_incr: rocq

Rocq: `list_incr_spec`.

Specification of tree copy

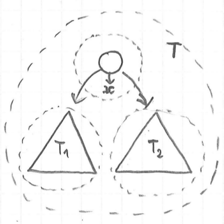
```
let rec copy (p:node) : node =
  if p == null then null else
  let p1' = copy p.left in
  let p2' = copy p.right in
  { item = p.item;
    left = p1';
    right = p2' }
```



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' }
```

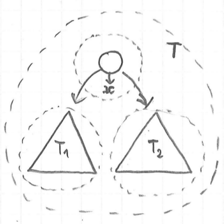


Exercise: specify the tree copy function.

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

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.

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

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
$p \mapsto \underline{\{\!\|x; p_1; p_2\|\!\}} \;*\; \underline{p_1 \rightsquigarrow \mathsf{Mtree}\, T_1} \;*\; p_2 \rightsquigarrow \mathsf{Mtree}\, T_2$ $\;*\; \underline{p_1' \rightsquigarrow \mathsf{Mtree}\, T_1}$	frame+induction
$p \mapsto \{\!\|x; p_1; p_2\|\!\} \;*\; p_1 \rightsquigarrow \mathsf{Mtree}\, T_1 \;*\; p_2 \rightsquigarrow \mathsf{Mtree}\, T_2$ $\;*\; p_1' \rightsquigarrow \mathsf{Mtree}\, T_1 \;*\; p_2' \rightsquigarrow \mathsf{Mtree}\, T_2$	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

Chapter 8

Small footprint specifications

Small footprint access to records

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

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

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

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

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

Specification of a write on the head field:

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

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:

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

Representation predicate for arrays

Representation predicate for C arrays:

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

where:

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

Representation predicate for arrays

Representation predicate for C arrays:

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

where:

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

Representation predicate for ML arrays:

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

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

Small footprint specifications of array operations

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

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

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

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

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

Small footprint specifications of array operations

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

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

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

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

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

$\forall p, i, v, n,$

$\{p[i] \mapsto v\}\ (\texttt{p.(i)})\ \{\lambda x.\ \ulcorner x = v \urcorner * p[i] \mapsto v\}$

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

$\{p.\mathsf{length} \mapsto n\}\ (\texttt{Array.length p})\ \{\lambda x.\ \ulcorner x = n \urcorner * p.\mathsf{length} \mapsto n\}$

Small footprint specifications of array operations

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

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

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

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

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

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

$$p \leadsto \mathsf{Array}\, L \;=\; \begin{array}{l} p[i] \mapsto L[i] \\ \ast\ p.\mathsf{length} \mapsto |L| \\ \ast\ \circledast_{\substack{v \text{ at index } j \text{ in } L \\ \text{with } j \neq i}}\ p[j] \mapsto v \end{array}$$

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`.)

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:

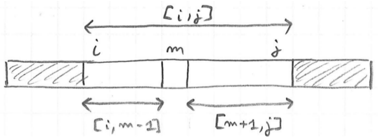
`p->hd = v`	compiles to	`*(p+hd)=v`
`p[i] = v`	compiles to	`*(p+i)=v`
`i[p] = v`	compiles to	`*(i+p)=v` (...)

Common small footprint specification for accessing a memory cell:

$$\{p \mapsto -\}\ (\texttt{*p = v})\ \{\lambda\_.\ p \mapsto v\}$$
$$\{p \mapsto v\}\ (\texttt{*p})\qquad \{\lambda x.\ \ulcorner x = v \urcorner * p \mapsto v\}$$

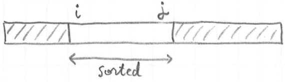
All other specifications for read and write operations are derivable.

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



$$
\begin{aligned}
\forall pLij.\quad & 0 \leqslant i \leqslant j < |L| \;\Rightarrow \\
& \{p \rightsquigarrow \mathsf{Array}\, L\} \\
& (\texttt{qsort i j p}) \\
& \{\lambda\_.\ \exists L'.\quad (p \rightsquigarrow \mathsf{Array}\, L') \\
& \qquad\qquad * \ulcorner \mathsf{sortedSeg}\, i\, j\, L' \urcorner \\
& \qquad\qquad * \ulcorner \mathsf{permut}\, L\, L' \urcorner \\
& \qquad\qquad * \ulcorner \forall a \in [0, i) \cup (j, |L|).\ L'[a] = L[a] \urcorner \quad \}
\end{aligned}
$$

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

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

Group of array cells

Definition

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

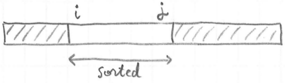
where M has type "map int A", for some A.

Conversions:

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

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

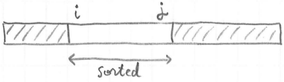
Small-footprint specification of quicksort



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

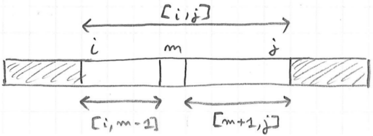


Exercise: give a small-footprint specification for quicksort.

- use $p \leadsto \mathsf{Cells}\, M$ 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.

$$
\begin{array}{ll}
\forall p M i j. & \mathsf{dom}\, M = [i, j] \;\Rightarrow \\
& \{p \leadsto \mathsf{Cells}\, M\} \\
& (\texttt{qsort i j p}) \\
& \{\lambda\_.\ \exists M'.\quad p \leadsto \mathsf{Cells}\, M' \\
& \qquad\qquad\quad * \ulcorner \mathsf{sortedSeg}\, i\, j\, M' \urcorner \\
& \qquad\qquad\quad * \ulcorner \mathsf{permut}\, M\, M' \urcorner \;\}
\end{array}
$$

Segment splitting in Quicksort



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

$$\begin{aligned} p \rightsquigarrow \mathsf{Cells}\, M \;\; = \;\; & \quad p \rightsquigarrow \mathsf{Cells}\,(M_{|\,[i,m-1]}) \\ & * \;\; p \rightsquigarrow \mathsf{Cells}\,(M_{|\,[m,m]}) \\ & * \;\; p \rightsquigarrow \mathsf{Cells}\,(M_{|\,[m+1,j]}) \end{aligned}$$

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:

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

$$\{p \rightsquigarrow \mathsf{Cells}\, M\}\ (\texttt{p.(i)})\ \{\lambda x.\ \ulcorner x = M[i] \urcorner \ast\ p \rightsquigarrow \mathsf{Cells}\, M\}$$

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

Summary

Representation of a full array using a list:

$$p \leadsto \mathsf{Array}\, L \quad \equiv \quad p.\mathsf{length} \mapsto |L| \; * \; \underset{v \text{ at index } i \text{ in } L}{\circledast} \; p[i] \mapsto v$$

Small footprint accesses:

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

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

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

Chapter 9

Heap entailment

Definition of Hprop

Let:

$$\text{Hprop} \;\equiv\; \texttt{heap} \to \texttt{Prop}$$

For example:

$$\begin{aligned} \ulcorner\urcorner \quad &: \quad \text{Hprop} \\ l \mapsto v \quad &: \quad \text{Hprop} \\ H_1 * H_2 \quad &: \quad \text{Hprop} \end{aligned}$$

In particular:

$$(*) \quad : \quad \text{Hprop} \to \text{Hprop} \to \text{Hprop}$$

The separation algebra

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

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

Neutral element: $H * \ulcorner\urcorner \;=\; H$

Extrusion of existentials: $(\exists x.\, H_1) * H_2 \;=\; \exists x.\, (H_1 * H_2)$ (if $x \notin H_2$)

Extrusion of pure facts: $(\ulcorner P \urcorner * H)\, m \;=\; P \wedge (H\, m)$

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:

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

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:

$$\frac{}{H \rhd H}\ \textsc{Reflexivity} \qquad \frac{H_1 \rhd H_2 \qquad H_2 \rhd H_3}{H_1 \rhd H_3}\ \textsc{Transitivity} \qquad \frac{H_1 \rhd H_2 \qquad H_2 \rhd H_1}{H_1 = H_2}\ \textsc{Antisymmetry}$$

Frame property for heap entailment

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

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

(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:

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

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:

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

Example:

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

Extraction of existentials and propositions

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

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

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

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:

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

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:

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

Same with explicit proof contexts:

$$\frac{x \notin H_2 \qquad \Gamma,\; x : A \;\vdash\; H_1 \rhd H_2}{\Gamma \;\vdash\; (\exists (x : A).\, H_1) \rhd H_2} \qquad\qquad \frac{\Gamma,\; h : P \;\vdash\; H_1 \rhd H_2}{\Gamma \;\vdash\; (\ulcorner P \urcorner * H_1) \rhd H_2}$$

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. $\exists n.\, (r \mapsto n) * \ulcorner n > 0 \urcorner \rhd \exists n.\, \ulcorner n > 1 \urcorner * (r \mapsto (n-1))$
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:

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

Summary

$(*)$ is associative, commutative, and has $\ulcorner\urcorner$ as neutral element.

Existentials and pure facts may be extruded from stars.

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

"$\ulcorner\text{False}\urcorner \rhd H$" is always true.

"$(r \mapsto n) * (r \mapsto m)$" is equivalent to "$\ulcorner\text{False}\urcorner$".

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

Chapter 10

Structural rules

Frame rule

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

Reformulation:

$$\frac{\{H_1\}\ t\ \{Q_1\}}{\{H_1 * H_2\}\ t\ \{Q_1 \star H_2\}}\ \text{FRAME}$$

with the overloading:

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

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

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

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:

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

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 $H' = \exists s.\, s \mapsto r$?

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

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:

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

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

Garbage collection in the pre-condition

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

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

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

Garbage collection in the pre-condition

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

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

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

Proof:

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

Garbage collection in the pre-condition

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

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

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

Proof:

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

Remark: no analog in Iris, where by default $P \star Q \vdash P$.

Extraction of existentials and propositions

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

To prove the above, we need to show that:

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

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:

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

Description of complete binary trees:

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

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:

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

Description of complete binary trees:

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

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

$$\{p \rightsquigarrow \mathsf{MtreeComplete}\, T\}\ (\mathtt{copy\ \ p})\ \left\{\lambda p'.\ \begin{array}{l} p \rightsquigarrow \mathsf{MtreeComplete}\, T \\ *\ p' \rightsquigarrow \mathsf{MtreeComplete}\, T \end{array}\right\}$$

Proof of the derived specification

(1) By unfolding of MtreeComplete:

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

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:

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

Proof of the derived specification

(1) By unfolding of MtreeComplete:

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

(2) By the EXISTS and PROP rules:

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

(3) By the CONSEQUENCE rule:

$$p \leadsto \mathsf{Mtree}\, T \ast p' \leadsto \mathsf{Mtree}\, T \rhd \begin{array}{l} \quad \exists n.\ (p \leadsto \mathsf{Mtree}\, T) \ast \ulcorner \mathsf{depth}\, n\, T \urcorner \\ \ast\ \exists n.\ (p' \leadsto \mathsf{Mtree}\, T) \ast \ulcorner \mathsf{depth}\, n\, T \urcorner \end{array}$$

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

Proof of the derived specification

Rocq demo.

Summary

Structural rules:

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

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

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

Other structural rules are derivable.

Chapter 11

Reasoning rules for terms

Reasoning rule for sequences

Example:

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

Reasoning rule for sequences

Example:

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

Exercise: complete the rule for sequences.

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

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:

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

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:

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

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.

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

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:

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

Transformation to A-normal form:

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

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 if t_1 or t_2 is not a value:

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

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 if t_1 or t_2 is not a value:

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

Remark: in general we have $\{H\}\; t'\; \{Q\} \Leftrightarrow \{H\}\; t\; \{Q\}$ for pure deterministic reductions, i.e. if $\forall m, <t, m> \to_{\mathsf{det}} <t', m>$ where

$$\frac{x \to y \qquad \forall z\; (x \to z) \Rightarrow y = z}{x \to_{\mathsf{det}} y}$$

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:

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

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: $Q' \equiv \lambda x. \ulcorner x = n \urcorner * (r \mapsto n)$.

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.

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.

Remark: again $v_1 v_2$ makes a pure deterministic reduction to $([f \to v_1]\,[x \to v_2]\,t)$

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:

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

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:

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

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:

$$\{p \leadsto \mathsf{MList}\, L\}\ (0)\ \{\lambda n.\ \ulcorner n = |L| \urcorner \ast\ p \leadsto \mathsf{MList}\, L\}$$

– Replace p with null.
– Rewrite null $\leadsto$ MList L to $\ulcorner L = \mathsf{nil} \urcorner$ in the pre and the post.
– By the PROP rule:

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

– Replace L with nil.

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

– Apply the VAL-FRAME rule.

$$\ulcorner\urcorner\ \rhd\ \ulcorner 0 = 0 \urcorner \ast \ulcorner \mathsf{nil} = \mathsf{nil} \urcorner$$

Verification of mlength: cons case (1/2)

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

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

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

$$p' \leadsto \mathsf{MList}\, L' \ *\ p \leadsto \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\}$$

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

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

– Apply the frame rule to remove: $p \leadsto \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\}$.
– Apply the let-binding rule with : $Q \equiv \lambda n'. \ulcorner n' = |L'| \urcorner * p' \leadsto \mathsf{MList}\, L'$.

Verification of mlength: cons case (2/2)

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

– First branch, exploit the induction hypothesis:

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

– Second branch:

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

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

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

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

Reasoning rule for local functions

Rule template:

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

Hypothesis about f:

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

Rule:

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

Summary

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

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

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

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

Summary of Course 2

Summary of chapter 7

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

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

$p \rightsquigarrow \mathsf{MList}\, L$		pre-condition
$p \rightsquigarrow \{\!|\mathsf{hd}{=}x;\ \mathsf{tl}{=}p'|\!\}$	$*\ p' \rightsquigarrow \mathsf{MList}\, L'$	by unfolding
$\underline{p \rightsquigarrow \{\!|\mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p'|\!\}}$	$*\ p' \rightsquigarrow \mathsf{MList}\, L'$	incrementing
$p \rightsquigarrow \{\!|\mathsf{hd}{=}x+1;\ \mathsf{tl}{=}p'|\!\}$	$*\ p' \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L')$	<u>frame</u>+induction
$p \rightsquigarrow \mathsf{MList}\, ((x+1) :: (\mathsf{map}\, (+1)\, L'))$		by folding
$p \rightsquigarrow \mathsf{MList}\, (\mathsf{map}\, (+1)\, L)$		by rewriting
		post-condition

Summary of chapter 8

Small footprint specification for C-style memory accesses:

$$\{p \mapsto w\}\ (\texttt{*p = v})\ \{\lambda\_.\ p \mapsto v\}$$
$$\{p \mapsto v\}\ \ (\texttt{*p})\qquad\quad \{\lambda x.\ \ulcorner x = v \urcorner\ *\ p \mapsto v\}$$

Representation of a full array using a list:

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

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

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

Summary of chapter 9

$(*)$ is associative, commutative, and has $\ulcorner\urcorner$ as neutral element.

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

"$\ulcorner\text{False}\urcorner \rhd H$" is always true.

"$(r \mapsto n) * (r \mapsto m)$" is equivalent to "$\ulcorner\text{False}\urcorner$".

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

Summary of chapter 10

Structural rules:

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

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

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

Other structural rules are derivable.

Summary of chapter 11

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

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

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

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

Exercises

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

Available from the webpage of the course.

The end!